

**UNIVERSIDAD DON BOSCO  
FACULTAD DE INGENIERÍA  
ESCUELA DE COMPUTACIÓN**



**ESTUDIO MONOGRÁFICO DE GRÁFICAS POR COMPUTADORA  
Y DESARROLLO DE SOFTWARE TUTOR WEB, IMÁGENES  
INTERACTIVAS Y EJEMPLOS GRÁFICOS.**

**TRABAJO DE GRADUACIÓN**

**PREPARADO POR LA FACULTAD DE INGENIERÍA**

**PARA OPTAR AL GRADO DE**

**INGENIERO EN CIENCIAS DE LA COMPUTACIÓN**

**PRESENTADO POR:**

**EVA MARÍA PÉREZ MAJANO  
JESSICA MARIEL JOVEL MEJÍA  
RODRIGO JOSÉ PLEITES CALLEJAS**

**ASESOR:**

**LIC. SANTIAGO ABARCA**

**CIUDELA DON BOSCO**

**SEPTIEMBRE 2006**

UNIVERSIDAD DON BOSCO

RECTOR

ING. FEDERICO MIGUEL HUGUET RIVERA

SECRETARIO GENERAL

LIC. MARIO RAFAEL OLMOS

DECANO DE LA FACULTAD DE INGENIERÍA

ING. ERNESTO GODOFREDO GIRÓN

ASESOR DEL TRABAJO DE GRADUACIÓN

LIC. SANTIAGO ABARCA

JURADO EVALUADOR

ING. ERICK PANAMEÑO

ING. CARLOS ROSALES VIANA

ING. RENÉ WILFREDO MÉNDEZ

TUTOR DEL TRABAJO DE GRADUACIÓN

ING. ANA MONTECINOS


UNIVERSIDAD DON BOSCO  
FACULTAD DE INGENIERÍA  
ESCUELA DE COMPUTACIÓN

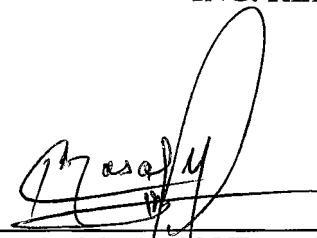


**ESTUDIO MONOGRÁFICO DE GRÁFICAS POR COMPUTADORA  
Y DESARROLLO DE SOFTWARE TUTOR WEB, IMÁGENES  
INTERACTIVAS Y EJEMPLOS GRÁFICOS**

  
LIC. SANTIAGO ABARCA  
*Asesor*

  
ING. ERICK PANAMEÑO  
*Jurado*

  
ING. RENÉ WILFREDO MÉNDEZ  
*Jurado*

  
ING. CARLOS ANTONIO ROSALES VIANA  
*Jurado*

## AGRADECIMIENTOS

Gracias a Dios y a la Virgen María, por permitirme llegar hasta el final de este proyecto. Gracias por concederme tener a mis padres Eva Majano y Agustín Pérez, quienes han sido los forjadores de lo que ahora soy, gracias a ellos que han luchado y sacrificado toda su vida por darme los estudios, el apoyo incondicional en los momentos difíciles y poniendo siempre en sus oraciones diarias mi futuro, mi trabajo y mi vida entera.

Gracias Dios por que mi hermano Carlos Agustín ha tenido siempre para mí el consejo, el abrazo y el apoyo en todo momento de felicidad y aflicción, además de ser para el ejemplo de honestidad y superación.

Gracias también por poner en mi camino a mis compañeros de tesis Jessica Jovel, Rodrigo Pleites y nuestro asesor Lic. Santiago Abarca, con quienes he compartido cada etapa del trabajo, y lo más importante la amistad que encontré en ellos y que llevaré siempre conmigo.



Gracias porque a través de mis estudios he conocido a mis amigos y en especial a mi novio Juan Fernando Romero, quien ha compartido conmigo cada etapa de mi carrera y me ha apoyado siempre en dificultades.

Pero en especial quiero agradecer y dedicar este proyecto a mi tío José, quien desde el cielo ha sido el ángel guía, y que desde el principio ha estado conmigo presente en mi corazón.

Gracias además por todos mis familiares y amigos que se han alegrado y comparten la finalización de mi carrera con gran regocijo a todos ellos muchísimas bendiciones.

F. Eva María Pérez Majano

## *Agradecimientos.*

*Agradezco a Dios Padre todo poderoso por guiarme en cada uno de los momentos difíciles, por darme sabiduría y fortaleza al emprender una etapa más de mi vida, por abrir nuevos senderos en mi camino, por todas y cada una de las bendiciones que el me brinda a diario y por enseñarme que con fe y esfuerzo todo se puede lograr.*

*Agradezco a mis padres Elizabeth y José Lorenzo por su amor, dedicación, esfuerzo, comprensión, por sus enseñanzas, por sus principios, por ese abrazo cálido lleno de afecto y de rigor que me invitan día a día a tratar de ser un buen ser humano y de enfrentar la vida con el espíritu emprendedor que ellos inculcaron en mí y es por ello que se convirtieron en el motivo mas importante por el que se debe luchar. Gracias por haberme dado la vida. Los amo.*

*Agradezco a mi hermano José Adalberto por compartir y llenar de alegría los momentos difíciles que se presentaron a lo largo de esta jornada, por ser mi apoyo, por ser uno de los motivos por los cuales luchar y por todo el cariño incondicional que el me brinda. Lo amo.*

*Agradezco a mis abuelos, tíos, primos y de mas familia por siempre dar un palabra de esperanza, por respaldar la idea de seguir adelante y siempre fueron voces de aliento que en momentos difíciles me hicieron seguir.*

*A mi novio German Martínez por ser mi apoyo incondicional, por sus consejos, por su comprensión, por ese amor que me brindó a lo largo de estos años y me dio la fortaleza para seguir adelante convirtiéndose en un motivo mas para lograr este sueño.*

*Gracias. Te amo.*

*A la familia Martínez por su comprensión, apoyo, cariño incondicional y los consejos que me dieron ánimo, para continuar sin desmayar y alcanzar este logro tan anhelado.*

*A mis amigos por su amistad, comprensión, paciencia, cariño, por estar a mi lado recorriendo este camino, tanto en las alegrías como en los momentos difíciles, brindando una voz de aliento y cariño*

*A mis compañeros de tesis Eva María Majano y José Pleitez, por su esfuerzo, tolerancia y cariño, por ser más que compañeros y convertirse en amigos y luchar juntos por este logro alcanzado.*

*A nuestro asesor Lic. Santiago Abarca por su tiempo brindado, consejos y conocimientos aportados para el desarrollo y culminación de este proyecto.*

*A mis maestros por compartir sus enseñanzas y ser una guía más para forjar mis principios y conocimientos a lo largo de esta jornada.*

*Agradezco a todas las personas que de forma directa o indirecta formaron parte de esta etapa de mi vida. Gracias.*

*Jessica Mariel Jovel Mejía*

## AGRADECIMIENTOS

En Agradecimiento, primeramente le agradezco a Dios por haberme dado fuerza e inteligencia estos años de universidad. También le estoy muy agradecido por haberme dado la oportunidad de seguir adelante, por las tantas veces que caí.

Les agradezco a mis padres haberme dado la oportunidad de superarme como persona. Por haberme permitido recibir una educación adecuada, por mantenerme siempre en el buen camino. Se que en los años de la universidad fueron momentos difíciles, porque nos olvidamos que tan difícil era el trabajo de cada uno, pero al final, nos entendíamos y salíamos de cada cosa juntos, como familia.

Por ultimo, pero no menos importante, le agradezco a Emilia, mi linda novia, el haberme dado fuerza cuando mas lo necesitaba; por estar allí cada segundo y estarlo por lo que resta de mi vida. Porque una persona como ella, me la puso Dios, para darme vida y crecer, no solo, sino con ella a mi lado.

Gracias, a todas personas que me apoyaron o que estuvieron conmigo, porque de cierta manera formaron parte de mi crecimiento profesional y como ser humano; y como digo: En todo camino siempre existe una pequeña partícula de luz, que es lo necesario para salir adelante, pero todas las personas que forma parte de mi vida, me hicieron un sol y gracias a ellos nunca me perdí. Espero yo, también ser una pequeña partícula para otros que lo necesiten.

Rodrigo José Pleites Callejas

## INDICE

I. INTRODUCCIÓN .....	I
1.1 ANTECEDENTES .....	iii
1.2 IMPORTANCIA DE LA INVESTIGACIÓN .....	v
1.3 DESCRIPCIÓN DEL PROYECTO .....	vii
1.4 OBJETIVOS .....	viii
1.5 ALCANCES .....	ix
1.6 LIMITACIONES .....	xiii
1.7 METODOLOGÍA DE LA INVESTIGACIÓN .....	xiv
1. Introducción a las Gráficas por Computadora .....	
1.1 Generalidades de las Gráficas por Computadora .....	1
1.1.1 Breve Historia .....	1
1.1.2 Aplicaciones Actuales .....	3
1.2 Fundamentos Matemáticos .....	13
1.2.1 Tipos de Vectores .....	16
1.2.2 Representación Gráfica de una Magnitud Vectorial .....	18
EJEMPLOS DE VECTORES .....	28
1.3 Matrices .....	36
1.3.1 Origen .....	36
1.3.2 Tipos de Matrices .....	41
1.3.3 Operaciones con Matrices .....	46
EJEMPLOS DE MATRICES.....	49
2. Graficado en Dos Dimensiones .....	
2.1 Introducción .....	52
2.2 Sistema de Coordenadas rectangulares .....	53
2.2.1 Trazado de Puntos .....	65
2.2.2 Trazado de Líneas .....	67
2.2.2.1 Método Algebráico .....	71
2.2.2.2 Analizador Diferencial Digital (DDA) .....	72
2.2.2.3 Algoritmo de Bresenham .....	74
2.2.2.4 Estilos de Líneas .....	82
2.2.3 Trazado de Circunferencias .....	87
2.2.3.1 Método Algebráico .....	87
2.2.3.2 Método Trigonométrico .....	94
2.2.3.3 Algoritmo de Bresenham para Curvas .....	96
2.2.4 Trazado de Curvas .....	101
2.2.4.2 Otras Curvas .....	108
2.3 Llenado de Región .....	113
2.3.1.1 Cálculo de la Lista de Línea de Exploración (LDR) .....	114
2.3.1.2 Pasos del Algoritmo de Línea de Exploración .....	115
2.3.2 Algoritmo de Inundación .....	116
2.3.3 Algoritmo de Inundación con Recursión Mínima .....	117
2.3.3.1 Llenado de triángulos y Polígonos .....	117
2.3.3.2 Llenado de Polígonos en OPENGL .....	118

2.3.4 Llenado de Circunferencias y Elipses .....	118
2.3.4.1 Llenado de Línea de Exploración de áreas de Frontera Curva .....	118
2.3.4.2 Algoritmo de Inundación con Fronteras Curvas .....	120
2.4 Transformaciones de Coordenadas .....	121
2.4.1 Traslación, Escalamiento y Rotación .....	121
2.4.2 Representaciones Matriciales .....	128
2.4.3 Transformaciones Compuestas y Eslabonamientos .....	131
EJEMPLOS GRAFICADO EN 2D .....	137
 3. Graficado en Tres Dimensiones	
3.1 Introducción .....	144
3.2 Representación Rectangular Tridimensional .....	145
3.3 Transformaciones de Coordenadas .....	155
3.4 Representaciones Matriciales .....	157
3.5 Proyecciones en un Plano .....	175
3.5.1 Proyección Paralela .....	176
3.5.2 Proyección Perspectiva .....	180
3.6 Supresión de líneas y Superficies Ocultas .....	183
3.7 Representación de Circunferencias Tridimensionales .....	195
EJEMPLOS GRAFICADO EN 3D .....	203
 4. Color, Iluminación, Sombreado y Textura	
4.1 Modelo de Color .....	214
4.1.1 Modelo RGB .....	225
4.2 Iluminación .....	227
4.2.1 Modelos Básicos de Iluminación .....	227
4.2.2 Algoritmos de Iluminación Global .....	245
4.3 Sombras .....	249
4.3.1 Introducción a sombras .....	249
4.3.2 Modelos de Sombreado Poligonal .....	251
4.3.2.1 Sombreado Plano (flat) .....	251
4.3.2.2 Sombreado Interpolativo y goraund .....	253
4.3.2.3 Sombreado Pong .....	255
4.3.2.4 Sombreado el Modelo de una Esfera .....	257
4.4 Detalles de una Superficie .....	259
4.4.1 Tipos de texturas .....	260
4.4.1.1 Textura 1D .....	261
4.4.1.2. Textura 2D .....	263
4.4.2 Dibujo de polígonos con Texturas .....	265
4.4.3 Texturas Multimapa .....	267
4.5 Transparencias .....	270
EJEMPLOS COLOR, ILUMINACIÓN, SOMBREADO Y TEXTURA.....	272

<b>5. ANIMACIÓN</b>	
5.1 Introducción.....	294
5.2 Conceptos Generales de Animación .....	295
5.3 Técnica de Animación de Síntesis por Ordenador .....	298
5.4 Animación en Tiempo Real .....	305
5.5 Control de Frecuencia de Refresco .....	316
5.6 Algoritmos Básicos de Animación .....	320
<b>EJEMPLOS DE ANIMACIÓN</b> .....	324
 <b>6. GRAFICADO EN WEB CON SWIFT 3D</b>	
6.1 Introducción a Swift 3D V4.....	349
6.1.1 Objetos en Swift .....	354
6.1.2 Luces e Iluminación en Swift .....	361
6.2 Animaciones .....	362
6.3 Graficado 3D .....	365
6.3.1 Modelado Vectorial y Poligonal .....	367
6.4 Implementación en la Web .....	368
<b>EJEMPLOS CON SWIFT 3D</b> .....	373
 <b>BIBLIOGRAFÍA</b> .....	399
<b>CONCLUSIONES</b> .....	405
<b>RECOMENDACIONES</b> .....	407
<b>ANEXOS</b>	

## INTRODUCCIÓN

En el mundo actual las computadoras han tenido un gran auge, se han convertido en algo indispensable para la vida cotidiana, empresarial, industrial, de entretenimiento y de más. Las empresas se enfrentan con situaciones que requieren individuos cada vez más creativos, capaces de encontrar las soluciones más convenientes.

El mundo de la informática es muy extenso y tiene diferentes áreas de estudio sin ser una menos importante que otra, entre ellas, las matemáticas es una de las ciencias que juegan un papel primordial en éste entorno.

Si bien se ha considerado a la matemática como una poderosa auxiliar de otras ciencias, hoy en día, se han ampliado tanto sus posibilidades, fundamentalmente por el aporte de las nuevas tecnologías, siendo uno de éstos el graficado por computadora.

Las gráficas por Computadora, son herramientas que permiten visualizar el comportamiento de fenómenos variables, o simplemente dan vida a imágenes u objetos.

El presente trabajo contiene una breve descripción al proyecto ESTUDIO MONOGRÁFICO DE GRAFICAS POR COMPUTADORA Y DESARROLLO DE TUTOR WEB, IMÁGENES INTERACTIVAS Y EJEMPLOS GRÁFICOS, proyecto que surge a partir de la necesidad del desarrollo de un estudio que dé soporte a la iniciación en ésta rama de la computación, abarcando un extenso contenido desde las bases fundamentales del graficado por computadora hasta cubrir y culminar con aplicaciones en la Web.



En este documento se presentan una introducción al proyecto definida por los antecedentes que representan la razón por la cual se desarrollará la investigación.

Objetivos tanto General como Específicos que permiten definir el enfoque y hacia dónde se verá dirigido el proyecto, los alcances y limitaciones que marcarán un margen del estudio y hasta dónde se llevará acabo.

Se define la proyección social, en donde se visualiza a quienes se beneficiará.

Se incluye además, un marco teórico, que forma las bases de la investigación como una breve introducción.

Finalizando este documento de anteproyecto, con el cronograma de actividades que encierra el tiempo en el que se desarrollaran la monografía y en software tutor en su totalidad.

## 1.1 ANTECEDENTES

Las gráficas por computadora, se encargan del estudio, diseño y trabajo del despliegue de imágenes en la pantalla de un computador a través de las herramientas proporcionadas por la física y la matemática<sup>1</sup>.

Esta rama de la computación es utilizada por: Ingenieros, Científicos, hombres de negocios, artistas y educadores.

En Ingeniería, las actividades pueden dividirse en cinco áreas: diseño, análisis, dibujo, fabricación/construcción/procesamiento y control de calidad, en donde existen software de simulación y diseño como lo son: ACAD –AUTOCAD, BLENDER, ARENA, SWIFT 3D 4, entre otros.

La aplicación en Ciencias, se ve reflejado en el estudio de la estructura de las sustancias químicas con modelaje del comportamiento de moléculas, cuerpo humano, animales, ADN.

En Arte, el artista siempre ha podido escoger los medios de expresión adecuados para su talento, como la creación de ilustraciones, imágenes de objetos reales y surrealistas y patrones repetitivos con pequeños cambios; para la animación de personajes de caricaturas, y para despliegues dinámicos o cambiantes.

---

<sup>1</sup> El matemático Francés René Descartes, revolucionó las matemáticas al unir dos de sus ramas Algebra y Geometría en 1637. Para mayor referencia consultar: Cálculo Sexta Edición Volumen 1, Larson/ Hostetler/ Edwards. Editorial Mc Graw Hill.

En educación y capacitación se diseñan aplicaciones especiales, como por ejemplo entre algunos sistemas especializados, podemos mencionar los simuladores para sesiones de práctica o capacitación de capitanes de barcos, pilotos, de avión, operadores de equipo pesado y el personal de control de tráfico aéreo, además el desarrollo de software gráficos para la educación de personas con capacidades especiales.

El desarrollo del proyecto **Estudio Monográfico de Gráficas por Computadora y Desarrollo de Tutor Web, Imágenes Interactivas y Ejemplos Gráficos**, representa una alternativa para soporte, no solo en recopilación de información sino también de una guía teórico-práctica que permita a docentes, estudiantes o personas con interés del conocimiento de esta rama de la ciencia obtener desde documentación básica, hasta cubrir con animación, además de incluir fundamentos científicos y matemáticos que respalden la investigación.

## 1.2. IMPORTANCIA DE LA INVESTIGACIÓN

### 1.2.1 JUSTIFICACIÓN

Las aplicaciones enfocadas a la creación de gráficas por computadora, son en la actualidad muy comunes tanto en el cine, juegos, software de ingeniería, medicina, aeronáutica, entre muchas otras aplicaciones de la vida cotidiana.

Éste estudio, representa una base fundamental en el área de informática, con éstas herramientas se pueden desarrollar una infinita gama de aplicaciones.

La importancia de un **Estudio Monográfico de Gráficas por Computadora y Desarrollo de Tutor Web, Imágenes Interactivas y Ejemplos Gráficos**, representa la oportunidad de aumentar la inquietud de estudiantes por cursar una asignatura que busca dar una introducción al futuro tecnológico.

El proyecto no solo esta enfocado a estudiantes sino también, a cualquier persona interesada en el aprendizaje del modelado 2D y 3D, para ello, se incluye el software tutor que contiene información teórica, ejercicios, ejemplos e imágenes, ayuda a facilitar la comprensión y la búsqueda de información por parte de los usuarios.

### **1.2.2 PROYECCIÓN SOCIAL**

Parte de la misión de la Universidad Don Bosco, es brindar servicios cualificados de educación superior científicos y tecnológicos. Con el desarrollo de un estudio respaldado con software tutor, se pretende ayudar a la población estudiantil que pretendan adquirir conocimientos de gráficas por computadora.

El Estudio Monográfico de Gráficas por Computadora y Desarrollo de Tutor Web, brinda muchos beneficios tanto a docentes como a estudiantes ya que significa un pilar importante para la exposición de la cátedra.

Los estudiantes pueden obtener información y visualizar de forma interactiva gráficos que permitan una mejor comprensión de algoritmos, técnicas y métodos que involucren la investigación.

## 1.3 DESCRIPCIÓN DEL PROYECTO

**Estudio Monográfico de Gráficas por Computadora y Desarrollo de Tutor Web, Imágenes Interactivas y Ejemplos Gráficos**, es un proyecto pensado para la realización de una monografía de gráficas por computadora en dos y tres dimensiones, basado en técnicas y métodos científicos y matemáticos.

El desarrollo de éste estudio, se enfoca a la recopilación y ordenamiento de la información, incluirá además la creación de un tutor web, el cual es una guía práctica con imágenes interactivas, ejemplos y ejercicios prácticos, para la complementación del estudio.

El tutor web esta desarrollado, de manera que estudiantes, docentes o personas interesadas en la rama del graficado por computadora, puedan obtener conocimientos básicos para la creación de aplicaciones.

## **1.4 OBJETIVOS**

### **1.4.1 OBJETIVO GENERAL**

Desarrollar un estudio monográfico de gráficas por computadora y respaldar la investigación con la creación de un software tutor.

### **1.4.2 OBJETIVOS ESPECÍFICOS**

1. Organizar y elaborar una investigación detallada de gráficas por computadora basada en diferentes fuentes científicas de consulta.
2. Definir los fundamentos de la graficación por computadora.
3. Identificar diferentes métodos para la creación de modelos en dos y tres dimensiones.
4. Analizar y Aplicar conceptos, algoritmos y técnicas para el desarrollo de gráficas por computadora en dos y tres dimensiones.
5. Aplicar técnicas de color, iluminación, sombreado y textura para la visualización de objetos en dos y tres dimensiones.
6. Investigar y simular algoritmos básicos de animación en dos y tres dimensiones.
7. Investigar y Aplicar técnicas para el desarrollo de gráficas en el entorno Web con la herramienta Swift 3D 4.

8. Elaborar un tutor Web, que contenga información teórica, gráficos, ejercicios, ejemplos descargables y exámenes iterativos. Se utilizara para el desarrollo de ejercicios la herramienta Visual C++ 6.0 con la librería gráfica OpenGL.
9. Desarrollar la interfase gráfica del tutor en un ambiente Web con las herramientas HTML, Macromedia Flash y Swish.

## **1.5 ALCANCES<sup>2</sup>**

- ✓ La realización del estudio monográfico, se basa en la investigación, recopilación y organización de la información sustentada con fuentes científicas de consulta, iniciando con las bases y fundamentos de gráficas por computadora, cubriendo desarrollo de gráficas en 2D y 3D, color, iluminación, sombreados y texturas hasta cubrir la investigación con conocimientos de animación. El estudio se desarrolla bajo las siguientes especificaciones<sup>3</sup>:

### **UNIDAD 1: INTRODUCCIÓN A LAS GRÁFICAS POR COMPUTADORA**

#### **1.1 Generalidades de las Gráficas por Computadora:**

##### **1.1.1 Breve historia**

##### **1.1.2 Aplicaciones actuales.**

#### **1.2 Hardware para Gráficas por Computadora**

#### **1.3 Fundamentos matemáticos:**

##### **1.3.1 Vectores y**

##### **1.3.2 Matrices.**

---

<sup>2</sup> Alcance: a) Alcance de Proyectos: es una expresión más asociada a la misión, metas y objetivos del proyecto. b) Alcance del Trabajo: se refiere a todos los elementos individuales del trabajo (colectivamente) que habrá que realizar para cumplir el proyecto. La Bibliografía Gestión de Proyectos. Autor: Gary R. Heerkens. Editorial McGraw-Hill Profesional 2002. Pag 122 Capítulo 7. Preparación de un Plan de Proyecto Detallado: Paso a Paso, define que el primer paso de la planificación consiste en identificar exactamente lo que hay que hacer, es decir el Alcance del proyecto. En esta etapa, habrá que identificar los principales elementos de trabajo y luego desglosarlos sistemáticamente en fragmentos más y más pequeños, hasta que cada uno de ellos adquiera una envergadura cómoda para estimar, ejecutar y supervisar".



## UNIDAD 2: GRAFICADO EN DOS DIMENSIONES.

- 2.1 Introducción.
- 2.2 Sistema de coordenadas rectangulares.
  - 2.2.1 Trazado de puntos.
  - 2.2.2 Trazado de líneas.
    - 2.2.2.1 Método algebraico.
    - 2.2.2.2 Analizador diferencial digital (DDA).
    - 2.2.2.3 Algoritmo de Bresenham.
    - 2.2.2.4 Estilos de líneas (continua, punteada).
  - 2.2.3 Trazado de circunferencias.
    - 2.2.3.1 Método algebraico.
    - 2.2.3.2 Método trigonométrico.
    - 2.2.3.3 Algoritmo de Bresenham.
  - 2.2.4 Trazado de otras curvas (elipses, arcos).
- 2.4 Llenado de región.
  - 2.4.1 Llenado de circunferencias y elipses.
  - 2.4.2 Algoritmo de exploración por línea.
  - 2.4.3 Algoritmo de inundación.
  - 2.4.4 Inundación con recursión mínima.
  - 2.4.5 Llenado de triángulos y polígonos.
- 2.5 Transformaciones de coordenadas.
  - 2.5.1 Traslación, Escalamiento y Rotación.
  - 2.5.2 Representaciones matriciales.
  - 2.5.3 Transformaciones compuestas y eslabonamientos. (P)

## UNIDAD 3: GRAFICADO EN TRES DIMENSIONES.

- 3.1 Introducción.
- 3.2 Representación rectangular tridimensional.
- 3.3 Transformaciones de coordenadas.

- 3.4 Representaciones matriciales
- 3.5 Proyecciones en un plano (en pantalla)
  - 3.5.1 Proyección Paralela.
  - 3.5.2 Proyección Perspectiva.
- 3.6 Supresión de líneas y superficies ocultas.

#### UNIDAD 4: COLOR, ILUMINACIÓN, SOMBREADO Y TEXTURA

- 4.1 Modelo de color
  - 4.1.1 Modelo RGB
- 4.2 Iluminación
  - 4.2.1 Modelos Básicos de Iluminación
  - 4.2.2 Algoritmos de Iluminación Global
- 4.3 Sombras
  - 4.3.1 Introducción a Sombras
  - 4.3.2 Modelo de Sombreado para Polígonos
- 4.4 Detalles de Superficie (Textura)
- 4.6 Transparencias

#### UNIDAD 5: ANIMACIÓN

- 5.1 Introducción.
- 5.2 Algoritmos básicos.

#### UNIDAD 6: GRAFICADO EN WEB CON SWIFT 3D 4

- 6.1 Introducción a Swift.
- 6.2 Gráficos 3D.
  - 6.2.1 Vectorial.
  - 6.2.2 Polígonos.
- 6.3 Animaciones.
- 6.4 Implementación en Paginas Web

✓ La investigación contiene elementos básicos para el desarrollo de graficas por computadoras, esto hace referencia a los fundamentos tanto generales como matemáticos, según está especificado en la unidad 1 del estudio monográfico.

✓ El estudio involucra el conocimiento de algoritmos de graficación tanto en 2D y 3D, para ello, los algoritmos que se desarrollan son los siguientes:

- Para Graficado en Dos y Tres Dimensiones:

- Analizador Diferencial Digital (DDA)
- Algoritmo de Bresenham para trazado de Líneas y puntos
- Algoritmo de Bresenham para trazado de Circunferencias
- Algoritmo de Exploración por Línea
- Algoritmo de Inundación

- Para Iluminación:

- Algoritmos de Iluminación Global

✓ El software tutor en la web, incluye información teórica que está basada en el estudio monográfico, se detalla de tal manera que los usuarios aprendan y analicen a partir de gráficos, imágenes, ejemplos y ejercicios que complementen la información.

✓ El tutor contiene imágenes interactivas, las cuales son creadas con las herramientas gráficas Macromedia Flash y Swish. Esto permite mostrar al usuario comportamientos vectoriales y matriciales de las gráficas por computadora en dos y tres dimensiones.

Para la realización de ejercicios se utilizan apartados obtenidos del estudio monográfico, dichos ejercicios pueden ser descargados según el ítem al cual se hace referencia.

Los elementos del tutor están diseñados con la herramienta HTML.

- ✓ Para la creación y desarrollo de ejemplos y ejercicios, se utiliza la herramienta Visual C++ 6.0 con la librería gráfica OpenGL.
- ✓ Al término de cada unidad se presentará un examen teórico iterativo, de manera que el usuario, compruebe con una evaluación el avance de sus conocimientos. El administrador del sistema tendrá la posibilidad de crear preguntas con cuatro respuestas alternativas, las cuales serán mostradas aleatoriamente.

## **1. 6 LIMITACIONES**

- ✓ El Explorador Web donde se cargue el tutor debe poseer plugins correspondientes a Macromedia Flash Player. Se recomienda instalar los plugins que el Explorador seleccionado establezca como necesarios para el funcionamiento o descargar los plugins gratuitos de la pagina Web de Macromedia, [www.macromedia.com](http://www.macromedia.com).
- ✓ La investigación Monográfica no incluye el estudio del tema: Realidad Virtual y Simulaciones.
- ✓ Para realizar los distintos ejemplos, que se desarrollan tanto en la monografía y en el tutor se necesita tener instalado un compilador de C junto con la librería gratuita de OpenGL.
- ✓ Para el desarrollo de la Unidad 6: GRAFICADO EN WEB CON SWIFT 4, se debe tener instalado el software Swift 3D 4 o superior.

## **1.7 METODOLOGÍA DE LA INVESTIGACIÓN**

El trabajo consiste en un estudio monográfico y un software tutor que respalde dicho estudio.

### **1.7.1 METODOLOGÍA DE LA INVESTIGACIÓN PARA EL ESTUDIO MONOGRÁFICO**

El estudio monográfico consiste en la recopilación de información de fuentes bibliográficas y manuales, de manera de formar un contenido específico.

La monografía que se ha creado con éste proyecto, involucra seis unidades específicas, según se ha definido en la determinación de alcances. Cada una de estas unidades, se ha cubierto dentro de un tiempo específico y han sido creadas en cuatro etapas:

1. Búsqueda y recopilación de información
2. Comprender y analizar la temática de cada unidad
3. Organizar la información y conformar la unidad en estudio, para así formar la monografía.
4. Realizar ejemplos y ejercicios que respalden cada unidad, para facilitar el aprendizaje del lector.

El desarrollo de cada unidad se contempla dentro del cronograma de actividades como una actividad, esto permite llevar un control organizado del proyecto.

### 1.7.2 METODOLOGÍA DE LA INVESTIGACIÓN PARA EL DESARROLLO DEL SOFTWARE TUTOR WEB

Como en todo proyecto informático, el desarrollo del software tutor web, se lleva a cabo, a partir de las siguientes etapas:

1. **Diseño del Tutor Web:** se definieron las imágenes que se utilizan, y el tipo de animaciones creadas para las diferentes unidades de la monografía. Además el diseño de las tablas de la base de datos del sistema. También en este punto se ha creado un formato de las páginas web que forman la interfaz del usuario.
- 2 **Selección de Información:** a partir de la monografía creada, se identificaron los datos que contendrá el software tutor, incluyendo ejemplos y ejercicios.
- 3 **Creación de imágenes:** a partir de los ejemplos, se crearon imágenes que ayuden a la mejor comprensión de los mismos, para mejorar la perspectiva del usuario con respecto a las gráficas 2D y 3D.
- 4 **Creación de Animación Interactiva:** con la ayuda de las herramientas Macromedia Flash y Swish, se han creado animaciones según lo requiera la información correspondiente a las unidades de la monografía.
- 5 **Programación HTML:** es básicamente la unión de todos los elementos, tanto información, imágenes, animaciones, ejemplos descargables y exámenes.

- 6 **Prueba del Software:** se ha comprobado en esta etapa, el funcionamiento correcto de cada uno de los elementos del tutor.
- 7 **Corrección de Errores:** se corrigen los módulos que han presentado errores en la etapa de prueba de software.

# MONOGRAFÍA

---



1

**INTRODUCCIÓN A LAS  
GRÁFICAS POR  
COMPUTADORA**

---

## **1.1 Generalidades de las Gráficas por Computadora:**

### **1.1.1 Breve historia**

Los gráficos por computadora comenzaron con el despliegado de datos en hardcopy, plotters y pantallas de tubos de rayos catódicos poco después de la introducción de las computadoras. Estos con el tiempo, se han desarrollado en la creación, almacenamiento y manipulación de modelos e imágenes de objetos<sup>1</sup>.

Los computadores personales, popularizaron el uso de gráficos **Bitmap**<sup>2</sup>. Una vez que el uso de éstos llegó a ser económico, pronto aparecieron aplicaciones gráficas debido a que se sabía que con ellos los sistemas se hacían fáciles de usar.

En los inicios, pocos hubieran imaginado que la potencia de la informática llegaría a lograr representaciones como las que se producen hoy en día. Durante su surgimiento, se logró dibujar líneas y curvas, y paulatinamente se incrementó la complejidad, así como la de los programas que los generaban y el hardware que lo soportaba. Luego se da el surgimiento de los sistemas CAD (Diseño Asistido por Computadora, en este caso las iniciales son las mismas en inglés y en español) y se llegó el momento en el que se pudo usar a las computadoras para diseñar, y abandonar de ésta manera los tableros de dibujo y hacerlo sobre una pantalla.

Las primeras aplicaciones en ciencia y en ingeniería tenían que basarse en equipos costosos y complicados.

---

<sup>1</sup> Donald Hearn, M. Pauline Baker. Gráficas por Computadora. Prentice-Hall Capítulo 1 pp 13.

<sup>2</sup> Bitmap: a) es la representación binaria en la cual un bit o conjunto de bits corresponde a alguna parte de un objeto como una imagen o fuente. b) Un tipo de imágenes para ordenador, en las que se almacena información sobre los puntos que las componen y el color de cada punto (al contrario que en las imágenes vectoriales). Libro: Norton. Introducción a la Computación 3ª edición. Editorial McGraw Hill. Capítulo 1 pp 18.

Actualmente se puede afirmar que las gráficas se utilizan rutinariamente en áreas diversas, por un lado la industria cinematográfica y del entretenimiento impulsa la imitación de la naturaleza en sus aspectos más espectaculares, como sucede en los efectos especiales o en los videojuegos. Por otro lado se perfeccionan herramientas digitales que imitan y simulan las de los artesanos, como, en el caso del diseño gráfico y los programas de fotoretoque y de ilustración digital.

El grado de perfección logrado ha agotado el interés de la comunidad de la gráfica por computadora (artistas, programadores, matemáticos) por la simple imitación de la naturaleza<sup>3</sup>: en la última década, gracias a una nueva interacción con las disciplinas científicas, se está redefiniendo el concepto de **arte digital** como la exploración de las posibilidades artísticas escondidas tanto en la programación y en la matemática como en los procesos internos de la vida y de la evolución.

Los nuevos artistas ya no buscan simplemente utilizar herramientas digitales que reemplacen a las técnicas tradicionales, sino crear procesos nuevos que les permiten lograr imágenes completamente diferentes tanto en su lógica como en su estructura.

---

<sup>3</sup> Plastock, Roy. Teoría y Problemas de Gráficas por Computadora—1987 Capítulo 9 pp 348.

## 1.1.2 Aplicaciones Actuales

### 1.1.2.1 Topografía y estudio del terreno.

Una de las aplicaciones prácticas más antigua es la visualización y medición del relieve terrestre mediante fotografías aéreas. Este método consiste en un aeroplano tome dos fotografías de una zona con una cierta distancia calculada entre ellas, se obtiene una imagen estereo<sup>4</sup>, que puede verse en relieve con un estereoscopio especial. Si las tomas se realizan con una precisión adecuada, permiten calcular elevaciones en el terreno. En la actualidad gracias a estaciones y software especialmente diseñados, como son Intergraph, Zeiss y Topko, se pueden obtener imágenes más precisas y datos más exactos de los relieves.

No solo la superficie terrestre puede ser aplicación para las imágenes 3D, también

puede ser utilizada para el análisis del relieve submarino, tal como USGS, United Status Geological Survey. Otro trabajo topográfico realizado en esta área es la del trasbordador espacial Endeavour, que permitió obtener mapas tridimensionales de los planetas.

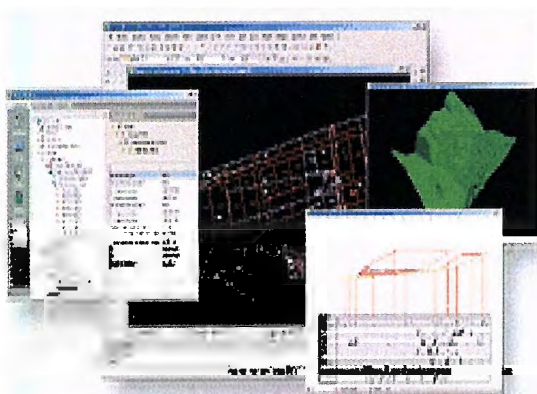


Figura No. 1. Imagen del software Topko.

---

<sup>4</sup> Imagen estereo: Dos fotografías están tomadas desde ángulos ligeramente diferentes y se observan a través de dos objetivos con lentes separadas e inclinadas para que coincidan y se fundan las dos imágenes en una tridimensional.

### **1.1.2.2 Estudio de la Tierra y otros Planetas**

NASA<sup>5</sup> ha obtenido numerosas vistas tridimensionales de fotografías de la Tierra obtenidas desde satélites, así como también de otros planetas de nuestro Sistema Solar. Las imágenes estereo de la superficie de Marte obtenidas por la sonda Pathfinder son otro ejemplo de aplicaciones para el estudio de otros planetas. La toma de estas no solo sirvió para ver la superficie de Marte en 3D, sino para calcular distancias y tamaños de las rocas y conducir con más seguridad el vehículo.

### **1.1.2.3. CAD (Diseño Asistido por Computador) Y CAE (ingeniería Asistida por Computador)**

Actualmente el mundo de la informática ha crecido tanto que ha facilitado el diseño de diferentes estructuras y objetos que son parte del diario vivir del ser humano, tales como aviones, carros, edificios, casas, maquinaria industrial, entre otros. Estas herramientas son poderosas para el diseño y visualización de prototipos, por ejemplo en la industria automovilística. Chrysler, Ford, Opel, Renault, Volvo y otros fabricantes ya usan estas técnicas, con un importante ahorro en tiempo y dinero durante el desarrollo. Los más importantes paquetes y estaciones de diseño por ordenador, como IBM, HP, DEC, Sun o Silicon Graphic, Autodesk<sup>6</sup>.

Una de las más poderosas herramientas para el diseño asistido por computadora es AutoCad, es un programa de diseño asistido por ordenador (DAC o en inglés CAD - Computer Aided Design)) para creación de objetos 2D y 3D. Actualmente está siendo desarrollado y comercializado por Autodesk.

---

<sup>5</sup> **NASA:** National Aeronautics and Space Administration, es la agencia gubernamental de los Estados Unidos responsable de los programas espaciales.

<sup>6</sup> **Autodesk:** una empresa de software totalmente diversificada que ofrece soluciones específicas para crear, gestionar y compartir activos digitales.

Es un programa donde se dibuja figuras básicas (líneas, circunferencias, rectángulos, y otros), y junto con poderosas herramientas de edición, se crea un gráfico más complejo. Puede organizar los objetos por medio de capas (layers) y bloques (blocks), a la vez permite plotear<sup>7</sup> con distintos espesores de línea. Es muy utilizado en proyectos y presentaciones de ingeniería.

#### 1.1.2.4. Medicina.

En la medicina donde las graficas 3D tienen su mayor uso, ya que no solo sirve para el análisis del cuerpo humano, sino que permite que salvar una vida humana sea mucho más eficiente y seguro. La estereoscopia<sup>8</sup> proporciona más ayuda para la enseñanza, la interpretación de imágenes para el diagnóstico o como ayuda en las intervenciones. Actualmente no es una novedad que lupas y



Figura No. 2. Gafas de Cristal Líquido (LCS)

microscopios de precisión cuentan con visión estereo. Firmas como Zeiss u Olympus disponen de diversos modelos según las aplicaciones, tal como el microscopio estéreo, que se le puede conectar dos cámaras de vídeo, ofreciendo una presentación 3D en un monitor o pantalla grande de vídeo, así como grabar las imágenes 3D.

En el campo de la microcirugía la imagen 3D ofrece grandes posibilidades. La empresa alemana Zeiss cuenta ya con sistemas de microcirugía tridimensional, como el MediLive 3D, del que ya existen referencias sobre sus ventajas aplicado a la oftalmología. También el VRex cuenta con un sistema de microcirugía orientado

<sup>7</sup> Trazador gráfico que permite imprimir documentos de gran formato (como por ejemplo mapas). El término castellano trazador casi no se usa y en su defecto es muy común utilizar el vocablo plotter en Español. Por asimilación, a la acción de realizar una impresión utilizando el plotter se le denomina plotear

<sup>8</sup> **Estereoscopia:** es una técnica capaz de grabar información visual en 3 dimensiones o crear la ilusión de una imagen profunda.

a la endodoncia. Estos sistemas usan un multiplexor para entrelazar las imágenes izquierda y derecha, y la visualización tridimensional se consigue con gafas de cristal líquido (LCS).

Una de las ventajas de este sistema es que todo el equipo quirúrgico puede observar en una gran pantalla y en 3D una intervención si está dotado de gafas para la visión estereoscópica. Además las imágenes tridimensionales pueden grabarse en un vídeo convencional para estudiarlas posteriormente o emplearlas en la docencia.

En la enseñanza tiene evidentes aplicaciones en la visualización de muestras y en la creación de programas multimedia de anatomía virtual.

Este tipo de imágenes se usan para visualizar imágenes o modelos del interior del cuerpo humano, bien artificiales, bien generados a partir de imágenes reales obtenidas por medio de TAC (Tomografía Asistida por Computador) o RMN (Resonancia Magnética Nuclear). Técnicas como la radiografía estereoscópica permiten situar claramente cuerpos extraños o anomalías en el interior del paciente.

Asimismo el diagnóstico de enfermedades oculares esta dentro de la aplicaciones del 3D, un ejemplo de esto es Kasha Software, Inc.

#### **1.1.2.5. Psicología<sup>9</sup>.**

La realidad virtual es una de las herramientas para tratar problemas psicológicos, como las fobias. El miedo a volar, a los insectos pequeños, la aracnofobia y la claustrofobia pueden ser tratados de manera segura, si exponer al paciente a peligros reales, porque estos sistemas permiten a los pacientes



Figura No. 3. Lentes LCD utilizados para tratamientos psicológicos

<sup>9</sup> IEEE, Computer Graphics and Applications Septiembre/Octubre, 2005.

interactuar con el entorno y elementos, por ejemplo los usuarios pueden tocar mesas donde el animal esta pasando, inclusive pueden mirar una animal mientras corre por sus pies o interactuar con elementos como matamoscas o insecticidas. Dichos programas permiten al psicólogo escoger el lugar y el tiempo, logrando así un tratamiento más efectivo.

#### **1.1.2.6. Ingeniería Molecular.**

En ingeniería molecular<sup>10</sup>, con la visualización estéreo en las estaciones de diseño es factible crear nuevas moléculas complejas. Esta aplicación es desarrollada por Widener University, en el departamento de Química.

#### **1.1.2.7. Videojuegos 3D<sup>11</sup>.**

Los juegos electrónicos cuyo desarrollo tiene lugar en la pantalla de una computadora o de una consola como xbox, PlayStation, GameCube, entre otros. Los programas van grabado en un disquete, un CD, DVD o un cartucho especial para juegos.



Figura No.4. Tomb Raider Legend, juego del 2006, aplica nuevas técnicas de graficado 3D.

Los videojuegos se dividen en distintas categorías: educativos, de aventuras y deportivos. Los más populares emplean sonidos reales y colores, además de rápidos efectos visuales. Los juegos deportivos, como el fútbol, el baloncesto o el hockey sobre hielo, adquirieron especial popularidad a finales de la década de 1980, cuando determinados equipos profesionales prestaron su nombre a estas versiones en vídeo de su deporte.

<sup>10</sup> Rafael Lahoz-Beltran. Bioinformática. Simulación, vida artificial e inteligencia artificial. 2004.

<sup>11</sup> Dr. Pere Marqués Graells. Los Videojuegos. 2001



Las críticas a los videojuegos parten del hecho de pasar demasiado tiempo ante la pantalla e inhibirse por completo en un mundo de fantasía puede tener en el desarrollo emocional de los niños. También se ha comprobado que la velocidad con que se mueven los gráficos puede provocar ataques en las personas que padecen diversos tipos de epilepsia.

Lo positivo de los videojuegos es enseñar a resolver problemas técnicos, estimular la habilidad de los jugadores y familiarizar a los niños con el uso de los equipos informáticos. Incluso algunas personas sostienen que mejoran la comunicación, cuando se juega en familia. Los videojuegos se emplean también como entretenimiento en clínicas y hospitales, así como en ciertas terapias de rehabilitación.



Figura No. 5: Imágenes de Half Life 2, uno de los primeros juegos, donde el usuario interactúa con objetos virtuales.

Desde 1993 estas dos compañías están realizando esfuerzos para controlar y establecer el contenido de los juegos. La iniciativa responde a las críticas, especialmente de los padres, preocupados por la intensificación de la violencia y la introducción de temas para adultos en los juegos infantiles.

#### 1.1.2.8 Simulación<sup>12</sup>.

Alternativos a los juegos, nació la simulación con un fin educacional. Esta es una imitación de cierto dispositivo verdadero o una situación. Este sistema procura representar ciertas características del comportamiento de un sistema físico o abstracto por el comportamiento de otro sistema.

<sup>12</sup> Rafael Lahoz-Beltran. Bioinformática. Simulación, vida artificial e inteligencia artificial. 2004.

La simulación se utiliza en muchos contextos, incluyendo modelar sistemas naturales, y sistemas humanos para ganar la penetración en la operación de esos sistemas; y simulación en la ingeniería de la tecnología y de seguridad donde está probar la meta un cierto panorama práctico del mundo real. Al utilizar un simulador o de otra manera la experimentación con una situación ficticia pueden demostrar los efectos verdaderos eventual de algunas condiciones posibles.

#### **1.1.2.9 Simulación física e interactiva.**

La simulación física se refiere a la simulación en la cual los objetos físicos se substituyen para la cosa verdadera, estos objetos físicos se elige a menudo porque son más pequeños o más baratos, que el objeto o el sistema real.

La simulación interactiva, que es una clase especial de simulación física, y designado a menudo ser humano en las simulaciones del lazo, es las simulaciones físicas que incluyen a seres humanos, tales como el modelo usado en un simulador de vuelo.

#### **1.1.2.10 Simulación en el entrenamiento.**

La simulación se utiliza a menudo en el entrenamiento del personal civil y militar. Esto ocurre generalmente cuando es costoso o simplemente demasiado peligroso permitir que los aprendices utilicen el equipo verdadero en el mundo verdadero. En tales situaciones pasarán tiempo que aprenden lecciones valiosas en un ambiente virtual seguro. La conveniencia de estos es a menudo permitir errores durante el entrenamiento para un sistema seguridad.

#### **1.1.2.11 Simuladores Médicos.**

Los simuladores médicos se están desarrollando y se están desplegando cada vez más para enseñar procedimientos terapéuticos y de diagnóstico así como

conceptos y la toma de decisión médicos al personal en las profesiones médicas. Los simuladores se han desarrollado para los procedimientos del entrenamiento que se extendían de los fundamentos tales como drenaje de la sangre, al cuidado laparoscópico de la cirugía y del trauma. Muchos simuladores médicos implican una computadora conectada con una simulación plástica de la anatomía relevante. Algunos contienen simulaciones de los gráficos de la computadora de las imágenes tales como radiografía u otras imágenes médicas. Algunos simuladores de pacientes emplean a maniquí de tamaño real que responde a las drogas inyectadas y puede ser programado para crear simulaciones de emergencias peligrosas para la vida.

#### **1.1.2.12 Simuladores De la Ciudad/Simulación Urbana.**

Un simulador de la ciudad puede ser un juego pero puede también ser una herramienta usada por los planificadores urbanos para entender cómo las ciudades son probables desarrollarse en respuesta a varias decisiones de política. UrbanSim (desarrollado en la universidad de Washington) e ILUTE (desarrollados en la universidad de Toronto) son ejemplos de los simuladores urbanos modernos, en grande diseñados para el uso por los planificadores urbanos. Los simuladores de la ciudad son generalmente simulaciones agente-basadas con las representaciones explícitas para la utilización del suelo y el transporte.

#### **1.1.2.13 Simuladores de vuelo.**

Un simulador de vuelo se utiliza para entrenar a pilotos en la tierra. Permite que un piloto se estrelle en un avión simulado sin ser lastimado. Estos se utilizan a menudo para entrenar a pilotos en situaciones extremadamente peligrosas, tales como aterrizajes sin los motores, o para determinar faltas eléctricas o hidráulicas. Los simuladores más avanzados tienen sistemas visuales de alta fidelidad y sistemas hidráulicos del movimiento. El simulador es normalmente más barato que enseñar en un avión verdadero.

#### **1.1.2.14 Simulación de la ingeniería.**

La simulación es una característica importante al dirigir sistemas. Por ejemplo en la ingeniería eléctrica, puede ser utilizado simular la propagación retrasa y desplazamiento de fase causado por una línea real de la transmisión. Semejantemente, las cargas simuladas se pueden utilizar para simular impedancia sin la simulación de la propagación, y se utilizan en situaciones donde está indeseada la propagación. Un simulador puede imitar solamente algunos las operaciones y las funciones de la unidad que simula de un circuito. Un ejemplo de estos sistemas puede ser el pspice

La mayoría de las simulaciones de la ingeniería exigen modelar matemático e investigación asistida computadora. Hay muchos casos, sin embargo, donde no está confiable el modelar matemático. La simulación de los problemas fluidos de la dinámica requiere a menudo simulaciones matemáticas y físicas. En estos casos los modelos físicos requieren el similitud dinámico.

#### **1.1.2.15 Simulación en la educación.**

Las simulaciones en la educación son algo como simulaciones del entrenamiento. Se centran en tareas específicas. En el pasado, el vídeo se ha utilizado para los profesores y los estudiantes a observar, problema de la educación solucionan y juego del papel; sin embargo, un uso más reciente de simulaciones en la educación incluye los vignettes narrativos animados (ANV). ANVs son narrativas en video de historias hipotéticas y realidades basadas en la enseñanza y aprender del salón de clases. ANVs se ha utilizado para determinar conocimiento, habilidades el solucionar de problema y disposiciones de niños, y a profesores.

Otra forma de simulación ha estado encontrando favor en la educación del negocio en años recientes. Las simulaciones del negocio que incorporan un

modelo dinámico permiten la experimentación con estrategias de negocio en un ambiente libre del riesgo y proporcionan una extensión útil a las discusiones del estudio de caso.

## 1.2 Fundamentos matemáticos:

### 1.2.1 Vectores

El graficado de imágenes se relaciona de una u otra forma con posiciones en el espacio<sup>13</sup>, la descripción matemática del movimiento de un objeto requiere un método para escribir la ubicación de éste en diferentes tiempos. Esto se hace mediante coordenadas cartesianas, donde los ejes horizontales y verticales se cruzan en un punto que se considera el origen.

### 1.2.1 Magnitudes Escalares y Vectoriales<sup>14</sup>

En física y matemática, se encuentran ciertas magnitudes que pueden ser especificadas completamente mediante un número y unidad de medida. Estas magnitudes reciben el nombre de **Magnitudes Escalares** y entre éstas se pueden citar el tiempo, la longitud, la masa, la temperatura, entre otras.

Las magnitudes escalares pueden someterse a las operaciones fundamentales de la aritmética (suma, resta, multiplicación y división). Por ejemplo:

$$30 \text{ Kg} + 17 \text{ Kg} + 29 \text{ Kg} = 76 \text{ Kg}$$

$$25.4 \text{ cm} - 14.8 \text{ cm} = 10.6 \text{ cm}$$

$$20.0 \text{ m} \times 16.0 \text{ m} = 320 \text{ m}^2$$

$$\frac{60 \text{ m}}{10 \text{ s}} = 6.0 \text{ m/s}$$

$$10 \text{ s}$$

Otras magnitudes físicas, tales como el desplazamiento, la velocidad, la aceleración y la fuerza, requieren de un número y las respectivas unidades, la

---

<sup>13</sup> Serway, Beichner. Física Para Ciencias e Ingeniería. McGraw-Hill tomo 1 5ta edición Capítulo 3 pp 58-67

<sup>14</sup> Vector: todo segmento de recta dirigido en el espacio. Villalta Roberto Herrera Carlos Matemática Básica, 1998. Capítulo 4 pp 91.

especificación de una dirección y un sentido. A éstas se les denomina **Magnitudes Vectoriales**.

La dirección de una magnitud vectorial (por ejemplo la velocidad) se da haciendo referencia a direcciones convencionales ya establecidas tales como el Norte, Sur, Este y Oeste, las líneas imaginarias horizontal y vertical, o en general haciendo uso de los ejes coordenados (Eje X y Eje Y).

**El sentido** es aquel que indica hacia adónde se produce, se manifiesta o actúa la magnitud. Un desplazamiento puede darse hacia la derecha, izquierda, arriba o abajo. Para efectos operacionales en una misma dirección se consideran dos sentidos: Positivo (arriba y derecha) y Negativo (abajo e izquierda).

**Las magnitudes vectoriales** obedecen las leyes del álgebra Vectorial. Esta establece las operaciones entre elementos llamados vectores.

Para hacer referencia a un vector se hace uso de una letra en negrilla o de una pequeña flecha sobre ella, por ejemplo:

**A** o  $\vec{A}$   
**B** o  $\vec{B}$   
**C** o  $\vec{C}$

**El módulo de un vector** se representa haciendo uso del símbolo para valor absoluto o simplemente por la letra sin el distintivo de vector. Así:

$|A|$  o  $|\vec{A}|$  o A

**Componentes de Un vector**<sup>15</sup>: son proyecciones a lo largo de los ejes de un sistema de coordenadas. Cualquier vector puede describirse por completo mediante sus componentes, los signos de éstas dependen del cuadrante en que se localice el vector. (Ver Imagen No.3).

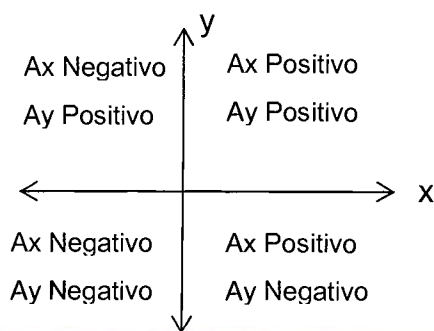


Imagen No. 3: Los signos de los componentes de l vector dependen del cuadrante en el cual se ubique el vector

Aritméticamente, las componentes se definen a partir de fórmulas:

$$\begin{aligned} (ec1) \quad A_x &= A \cos \theta \\ A_y &= A \sin \theta \end{aligned}$$

Éstas forman dos lados de un triángulo rectángulo, con una hipotenusa de longitud A.

Así se deduce que la magnitud y dirección de A se relacionan con sus componentes por medio de las expresiones:

$$(ec2) \quad A = \sqrt{A_x^2 + A_y^2}$$

$$(ec3) \quad \theta = \tan^{-1} (A_y/A_x)$$

<sup>15</sup> Serway, Beichner. Física Para Ciencias e Ingeniería. McGraw-Hill tomo 1 5ta edición Capítulo 3 pp 64-65



### 1.2.1 Tipos de vectores<sup>16</sup>

Los tipos de vectores son los siguientes:

- Vector unitario.
- Vector posición.
- Vector desplazamiento
- Vector Nulo o Cero

**Vector unitario:** es un vector sin dimensiones que tienen una magnitud

exactamente igual a uno, estos se utilizan para especificar una dirección determinada y no tienen otro significado físico, se usan solo por conveniencia en la descripción de una dirección en el espacio, se pueden representar de la siguiente forma  $\mathbf{i}$ ,  $\mathbf{j}$  y  $\mathbf{k}$  para que estos apunten en las direcciones  $\mathbf{x}$ ,  $\mathbf{y}$  y  $\mathbf{z}$  positivas formando un conjunto de vectores perpendiculares entre si.

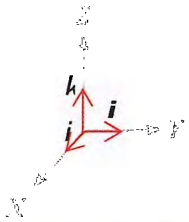


Figura No.4: Los vectores unitarios  $\mathbf{i}$ ,  $\mathbf{j}$ ,  $\mathbf{k}$ , están dirigidos a lo largo de los ejes  $\mathbf{x}$ ,  $\mathbf{y}$  y  $\mathbf{z}$ , respectivamente

**Vector posición:** es un vector cuyo origen es el del sistema de referencia y cuyo

extremo es la posición. En forma de vector unitario la posición, esta dada por  $\mathbf{r} = x\mathbf{i} + y\mathbf{j}$ . Es decir las componentes de  $\mathbf{r}$  son las coordenadas  $x$  y  $y$ .

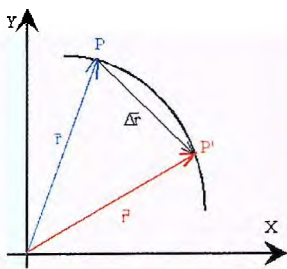


Figura No.5: Representación del vector posición

<sup>16</sup> Resnick, Halliday, Krane. Física Volumen 1 4ta Edición, Capítulo 3 pp 41-44

**Vector Desplazamiento:** el vector desplazamiento es un segmento recto dirigido desde el punto inicial al final, independientemente de que la trayectoria o camino que siguió la partícula es curva.(ver imagen No.6).

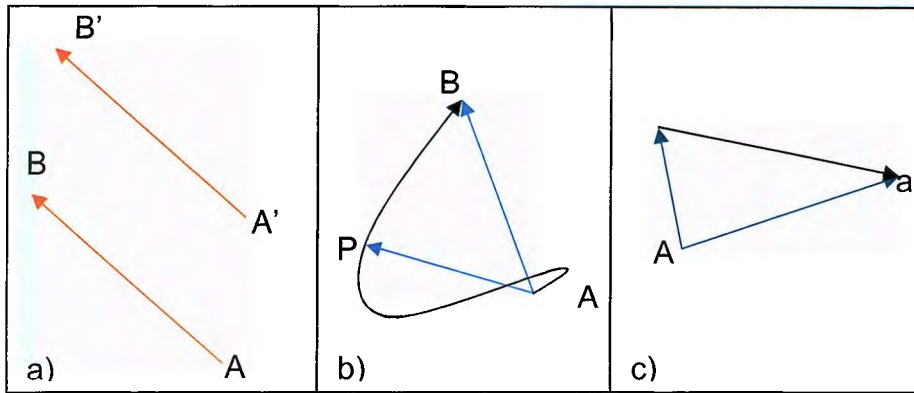


Figura No. 6: Vectores de desplazamiento. a) los vectores  $AB$ , y  $A'B'$  son idénticos, puesto que tienen la misma longitud y apuntan a la misma dirección. b) la trayectoria real de la partícula al moverse de  $A$  a  $B$ , puede ser la curva que se muestra; el desplazamiento es el vector  $AB$ . En el punto intermedio  $P$ , el desplazamiento es el vector  $AP$ . c) Después del desplazamiento  $AB$ . La partícula experimenta otro  $BC$ . El efecto neto de los dos desplazamientos es el vector  $AC$ .

**Vector Nulo ó Vector Cero:** el origen coincide en su extremo, éste vector tiene magnitud nula, pero no tiene orientación.

Todos los vectores nulos se consideran iguales, porque tienen componentes nulas; se representan por  $O$  y se escribe:  $O = [0,0,0]$ .

## 1.2.2 Representación Gráfica de una Magnitud Vectorial

Un vector puede ser representado gráficamente mediante un segmento de recta dirigido y orientado (como una flecha) y cuya longitud es proporcional a su módulo<sup>17</sup>. Por ejemplo si se dice que una fuerza de 50 N es aplicada en un punto de un cuerpo mediante una cuerda que se tensa hacia arriba formando un ángulo de  $30^\circ$  con la horizontal, esto se puede representar tal como se muestra en la figura No. 7

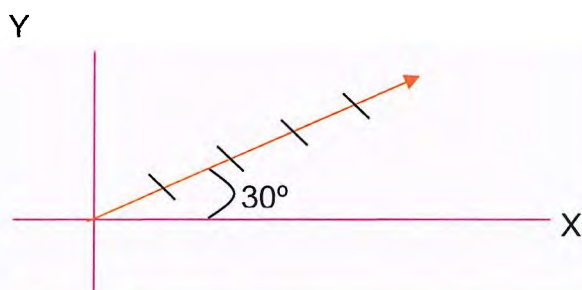


Figura No. 7: representación gráfica de una fuerza

### 1.2.2.1 Propiedades de Vectores<sup>18</sup>

#### Igualdad de vectores

Se dice que dos o más vectores son iguales si tienen el mismo módulo, dirección y Sentido<sup>19</sup>.

Es decir  $A = B$  solo si  $A = B$  y si  $A$  y  $B$  apuntan en la misma dirección a lo largo de líneas paralelas aun teniendo diferentes puntos de inicio por ejemplo:

<sup>17</sup> Módulo: Es la longitud o tamaño de un vector. Para hallarla es preciso conocer el origen y el extremo del vector, pues para saber cuál es el módulo del vector, debemos medir desde su origen hasta su extremo. Vector: todo segmento de recta dirigido en el espacio. Villalta Roberto Herrera Carlos Matemática Básica, 1998. Capítulo 4 pp

<sup>18</sup> Serway, Beichner. Física Tomo 1, 5ta Edición, McGraw-Hill, Capítulo 3 pp 61-64.

<sup>19</sup> Igualdad de Vectores o Vectores Equipolentes, para mejor referencia consultar: D.C Murdoch Geometría Analítica Con Vectores y Matrices. Limusa Noriega 1990. Capítulo 4 pp 78.

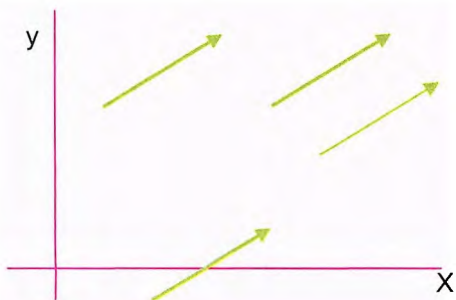


Figura No.8: Estos cuatro vectores son iguales porque tienen la misma longitud y apuntan en la misma dirección

## Suma o Adición de Vectores

Al igual que en la suma y resta de [escalares](#), solo se pueden sumar o restar vectores que corresponden a la misma magnitud física. Una fuerza solo se puede sumar o restar a otra fuerza, un desplazamiento con otro desplazamiento, una velocidad con otra velocidad.

La suma de vectores se describen adecuadamente con métodos geométricos, para sumar el vector B al A se dibuja primero el vector A con su magnitud

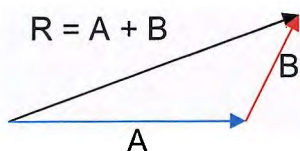


Imagen No.9: El vector resultante  $R = A + B$  es el vector dibujado desde el origen de A hasta el punto de finalización de B. Esto se conoce como el método de adición del triángulo.

representada por una escala conveniente sobre un papel para graficas y luego se dibuja el vector B en la misma escala y con su origen comenzando en la punta del vector A.

Teorema para la Suma de Vectores<sup>20</sup>:

Sean

$$\begin{aligned} X_1 &= [x_1, y_1, z_1] \text{ y} \\ (\text{ec4}) \quad X_2 &= [x_2, y_2, z_2], \end{aligned}$$

dos vectores cualesquiera. Se define la Suma de  $X_1 + X_2$  como:

<sup>20</sup> Kaplan, Wilfred, Cálculo y Algebra Lineal : Vectores del plano y cálculo de una variable Vol. I. Limusa 1978

(ec5)  $X_1 + X_2 = [x_1 + x_2, y_1 + y_2, z_1 + z_2]$ .

Las construcciones geométricas pueden utilizarse para sumar mas de dos vectores formando figuras como polígonos y paralelogramo.

Cuando se suman vectores el total es independiente del orden de la adición, pero cuando estos se multiplican este orden es importante.

La suma de vectores puede efectuarse a través de dos métodos:

- a) Gráfico
- b) Analítico

### **a) Método Gráfico o Geométrico**

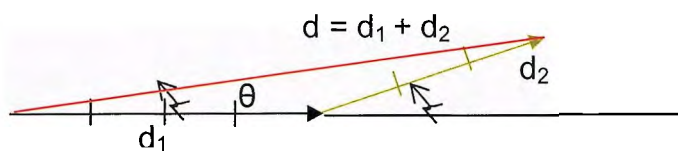
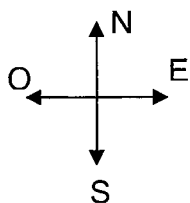
Para sumar dos vectores se reconocen dos reglas

1. Del Triángulo
2. Del Paralelogramo

#### **1. Del Triángulo:**

Usando una escala adecuada se dibujan los vectores uno seguido del otro y de acuerdo a la dirección y sentido indicados. Por definición la suma de los vectores es igual al vector que completa el triángulo, es decir, debe dibujarse desde el origen del primer vector hasta el extremo del segundo vector.

Considérese el siguiente ejemplo: sea  $d_1$ , el desplazamiento de una persona al caminar 40 m directamente hacia el este y  $d_2$ , el desplazamiento cuando cambia de rumbo caminando otros 30 m hacia el norte a  $30^\circ$  del este. El desplazamiento total se expresa como  $d = d_1 + d_2$ , y gráficamente se puede representar en la forma que se ilustra en la figura No.10

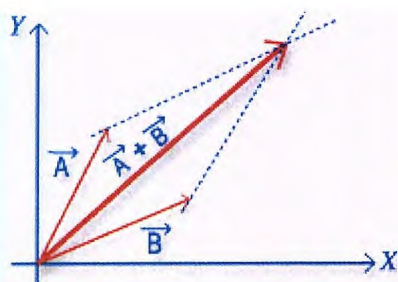


Escala de 10 m

Figura No. 10: Suma de desplazamientos

La dirección  $\theta$  del desplazamiento total  $d$ , se determina por medio de un transportador y su módulo  $d$  midiendo con la misma escala.

## 2. Del Paralelogramo:



Siempre a escala, se dibujan los vectores, haciendo coincidir su origen y luego completando el paralelogramo. El Vector suma es trazado a lo largo de la diagonal que parte del origen común. (ver figura No.11)

Figura No.11: Representación del método del paralelogramo

Esta regla se entiende al considerar, por ejemplo, dos fuerzas  $F_1$  y  $F_2$ , aplicadas a un mismo punto  $O$  formando un ángulo de  $135^\circ$  una con respecto a la otra y cuyos valores son:  $F_1=60$  N y  $F_2= 40$  N. en la figura No.12 se representa la situación y la fuerza resultante  $F= F_1 +F_2$ .

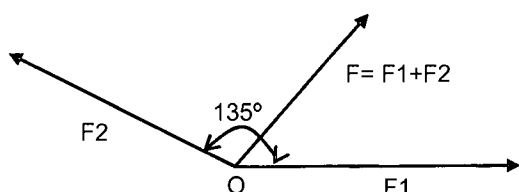


Figura No.12: Regla del paralelogramo

TEOREMA: Si A, B y C son tres puntos cualesquiera, entonces  $\vec{AC} = \vec{AB} + \vec{BC}$ .

**Demostración:** Escoja un sistema de referencia rectangular con origen en A y suponga que las coordenadas de B y C, respectivamente son:  $(x_1, y_1, z_1)$  y  $(x_2, y_2, z_2)$ , entonces:

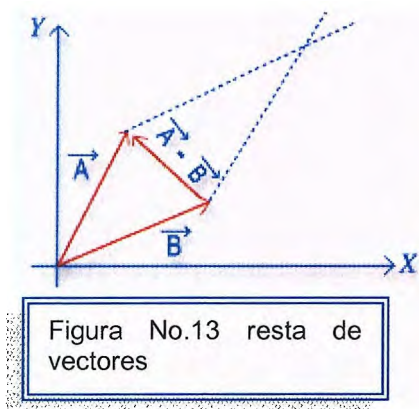
$$\begin{aligned} \vec{AB} &= [x_1, y_1, z_1] \\ \text{(ec6)} \quad \vec{BC} &= [x_2 - x_1, y_2 - y_1, z_2 - z_1] \\ \vec{AC} &= [x_2, y_2, z_2] = \vec{AB} + \vec{BC} \end{aligned}$$

Corolario No.1. Si  $\vec{AB}$  y  $\vec{AC}$  son vectores con el mismo origen A, entonces  $\vec{AB} + \vec{AC} = \vec{AD}$  donde D es el cuarto vértice del paralelogramo del cual  $\vec{AB}$  y  $\vec{AC}$  son lados adyacentes.

Corolario No.2: Si  $\vec{AB}$  y  $\vec{AC}$  son vectores que tienen el mismo origen, entonces:  $\vec{AC} - \vec{AB} = \vec{BC}$

### Sustracción o resta de Vectores

Ésta emplea la definición del negativo de un vector. Se define la operación  $A - B$  como el vector  $-B$  sumado al vector A, por ejemplo  $A - B = A + (-B)$ .



El método del paralelogramo puede aplicarse también en la resta de vectores.

Aquí el método consiste en desplazar el vector B al final del vector A y unir el origen con el final del vector B (el método es similar para la resta de vectores  $[A - B]$ , sólo debe cambiarse el sentido del

vector B a  $-B$  y sumar este último al vector A. (ver figura No. 13).

Definiciones de Leyes aplicadas a la suma y resta de vectores:

1. La suma de dos vectores cualesquiera es un vector definido en forma única.
2. Ley Conmutativa:  $X_1 + X_2 = X_2 + X_1$
3. ley Asociativa:  $(X_1 + X_2) + X_3 = X_1 + (X_2 + X_3)$
4. vector nulo. Existe un vector  $O$ . tal que  $X + O = X$  para todo vector  $X$ .
5. todo vector  $X$ , tiene un negativo  $-X$  tal que  $X + (-X) = 0$ ; la suma de  $X + (-Y)$  se escribe  $X - Y$  y se llama la diferencia de  $X$  y  $Y$



## Multiplicación de Vectores

Como los escalares, los vectores de diferentes clases pueden multiplicarse por otro para generar cantidad de dimensiones físicas nuevas. A causa de que los vectores tienen tanto dirección como magnitud, el vector de multiplicación no puede seguir exactamente las mismas reglas algebraicas de la multiplicación escalar, para ello se definen las siguientes:

### Multiplicación de un Vector por un Escalar

Una cantidad escalar ( $m$ ) esta especificada por un valor con la unidad apropiada y no tiene dirección, éste se puede multiplicar con un vector donde el producto  $m.A$  es un vector que tiene la misma dirección que  $A$  y la magnitud  $m.A$ ,

Si se multiplica una cantidad escalar negativa ( $-m$ ) el producto será  $-m.A$ , y esta dirigido en sentido opuesto que  $A$ .

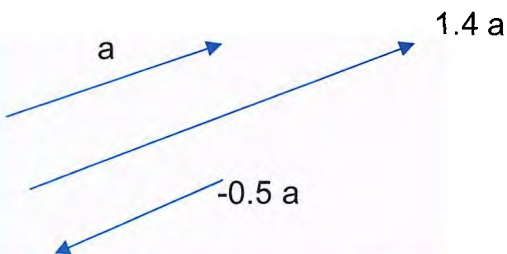


Figura No.14 La multiplicación de un Vector  $A$  por un escalar  $C$ , da un vector  $CA$ , cuya magnitud es  $C$  veces la magnitud de  $A$ , el vector  $CA$ , tendrá la misma dirección de  $A$  si  $C$  es positivo y opuesta si  $C$  es negativo. Por ejemplo  $C = +1.4$ , y  $C = -0.5$ .

## Multiplicación de Dos Vectores para dar por Resultado un Escalar

El producto escalar de dos Vectores A y B, escrito como  $A \cdot B$ , se define como  $a \cdot b = a \cdot b \cos \Phi$ , donde  $a$  es la magnitud del vector A,  $b$  es la magnitud del vector B, y  $\cos \Phi$  es el coseno del ángulo  $\Phi$  entre los dos vectores.

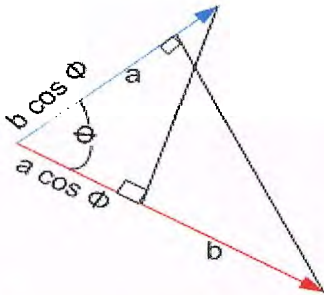


Figura No.15: El producto escalar  $a \cdot b$  ( $= ab \cos \Phi$ ) es el producto de la magnitud de un vector (por ejemplo,  $a$ ) y la componente del otro vector en la dirección del primero ( $b \cos \Phi$ )

A causa de la notación,  $a \cdot b$  se llama también PRODUCTO PUNTO de  $a$  y  $b$ , y se le dice “ $a$  punto  $b$ ”.

El producto punto es independiente de la elección de los ejes coordenados.

Puesto que  $a$  y  $b$  son escalares y  $\cos \Phi$  es un número puro, el producto escalar de dos vectores es un escalar. Además puede ser visto como el producto de la magnitud de un vector y la componente del otro vector en la

dirección del primero. Otra forma de representar el producto punto, es  $a(b \cos \Phi)$  o como  $b(a \cos \Phi)$ .

La siguiente nomenclatura explica de forma más clara el producto punto:

$$\mathbf{a} \cdot \mathbf{b} = a \cdot b \cos \varphi = a_x b_x + a_y b_y$$

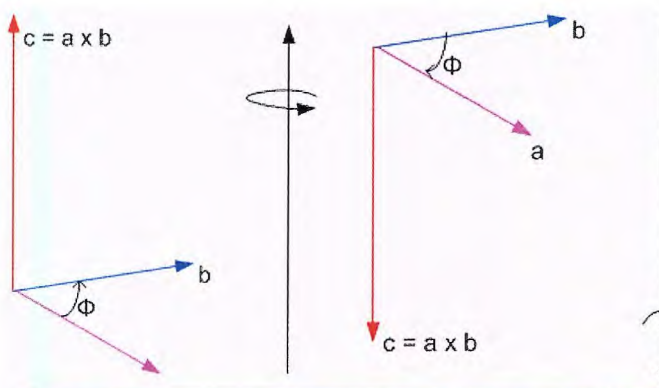
Puede decirse además que  $\mathbf{a} \cdot \mathbf{b}$  es cualquier operación, por ejemplo,  $a^{1/3} b^{1/4} \tan(\Phi/2)$ .

## Multiplicación de Dos Vectores para dar por Resultado otro Vector

El producto vectorial de dos vectores **a** y **b**, se escribe como **a x b** y es otro vector **c**, donde **c = a x b**. La magnitud de **c** se define como: **c = |a x b| = ab sen Φ**.

Donde Φ es el ángulo (más pequeño) entre **a** y **b**. Existen dos ángulos diferentes entre **a** y **b**: Φ y 2π-Φ. En la multiplicación vectorial, siempre se elige el más pequeño de los ángulos.

A causa de la notación **a x b**, se denomina PRODUCTO CRUZ o PRODUCTO VECTORIAL, y se dice "**a cruz b**".



La dirección de **c**, el producto vectorial de **a** y **b** se define como perpendicular al plano formado por **a** y **b**. Para especificar el sentido del vector **c** nos referimos a la figura No. 16

Figura No.16: vector **c**, resultado del producto cruz **a x b**

El vector **c** debe contener (ver figura No.17):

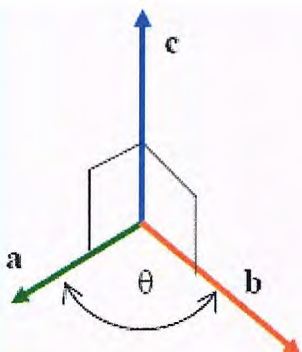


Figura No.17 Representación del vector **c**

- i) el módulo igual al producto de los módulos de **a** y **b** por el seno del ángulo que forman.
- ii) la dirección perpendicular al plano determinado por las direcciones de los vectores **a**, **b**;
- iii) el sentido tal que el triedro formado por los vectores **a**, **b**, **c** tenga la misma orientación que el espacio.

## Propiedades del producto vectorial<sup>21</sup>

i) Si en vez de  $a \wedge b = c$  se tiene  $b \wedge a$ , el módulo y dirección del vector producto no cambian; pero el sentido deberá ahora ser tal que el triedro  $b, a, c$  tenga la orientación del fundamental  $x, y, z$ ; por lo tanto el nuevo vector producto  $c$  deberá tener sentido contrario al anterior:  $a \wedge b = -b \wedge a$ . O sea, el producto vectorial es *anticonmutativo*.

ii) siendo  $\lambda$  un escalar, se verifica que:  $\lambda (a \wedge b) = \lambda a \wedge b = a \wedge \lambda b$

iii) El producto vectorial tiene la propiedad distributiva, o sea:

$$(a + b) \wedge c = a \wedge c + b \wedge c$$

Para demostrar dicha propiedad, puede observarse que si se dividen ambos miembros de la igualdad anterior por el módulo  $c$  del vector  $c$ , se obtiene:

$$(a + b) \wedge c_0 = a \wedge c_0 + b \wedge c_0$$

Donde  $c_0$  es el vector de igual dirección y sentido que  $c$ .

iv) La condición necesaria y suficiente para que dos vectores tengan la misma dirección (con sentidos iguales u opuestos) es que su producto vectorial sea nulo. En efecto, dos vectores tendrán la misma dirección cuando el ángulo que formen sea cero o  $\pi$ ; en estos casos el seno del ángulo es nulo y, por lo tanto, también es nulo el producto vectorial.

---

<sup>21</sup> Mosquera Carlos, Magnitudes Escalares y Vectoriales, Editorial CECSA pp 8-15.

## EJEMPLOS DE VECTORES

**Ejemplo 1.1:** Si B se suma a A ¿en que condición el vector resultante A+B tiene una magnitud igual a A + B? ¿Bajo que condiciones el vector resultante es igual a cero?

Respuesta: La resultante tiene una magnitud A +B cuando A se orienta en la misma dirección que B. El vector resultante  $A + B = 0$  cuando A se orienta en la dirección opuesta a B, y cuando  $A = B$

**Ejemplo 1.2:** Si una componente de un vector no es cero ¿su magnitud puede ser cero?

Respuesta: No. La magnitud de un vector A es igual a  $\sqrt{A_x^2 + A_y^2 + A_z^2}$ , por tanto si una componente es cero A no puede ser cero.

**Ejemplo 1.3:** Considere los puntos  $A = (1,1,1)$ ,  $B = (1,2,1)$ ,  $C = (-1,2,0)$  determine:

- a) El área del triángulo ABC
- b) Los ángulos del triángulo ABC
- c) Las magnitudes de los lados del triángulo ABC.
- d) Las alturas del triángulo ABC

Solución:

Los vectores con magnitud y dirección los lados del triángulo pueden escribirse:

$$\vec{c} = \vec{AB} = (1,2,1) - (1,1,1) = (0,1,0)$$

$$\vec{a} = \vec{BC} = (-1,2,0) - (1,2,1) = (-2,0,-1)$$

$$\vec{b} = \vec{CA} = (1,1,1) - (-1,2,0) = (2,-1,1)$$

De manera que:

$$\vec{c} \times \vec{a} = (0,1,0) \times (-2,0,-1) = (-1,0,2)$$

$$\vec{b} \times \vec{a} = (2,-1,1) \times (0,1,0) = (-1,0,2)$$

$$\vec{a} \times \vec{b} = (-2,0,-1) \times (2,-1,1) = (-1,0,2)$$

entonces el área del triángulo es:

$$A = \frac{1}{2} |(-1,0,2)| = \frac{1}{2} \sqrt{5}.$$

Las magnitudes de los lados son:

$$|\vec{c}| = |(0,1,0)| = 1$$

$$|\vec{b}| = |(2,-1,1)| = \sqrt{6}$$

$$|\vec{a}| = |(-2,0,-1)| = \sqrt{5}$$

Los ángulos están dados por:

$$\sin \alpha = \frac{(|\vec{b} \times \vec{c}|)}{(|\vec{b}| |\vec{c}|)} = \frac{\sqrt{5}}{\sqrt{6}}$$

$$\sin \beta = \frac{(|\vec{c} \times \vec{a}|)}{(|\vec{a}| |\vec{c}|)} = \frac{\sqrt{5}}{\sqrt{5}} = 1$$

$$\sin \omega = \frac{(|\vec{b} \times \vec{a}|)}{(|\vec{a}| |\vec{b}|)} = \frac{\sqrt{5}}{\sqrt{5}\sqrt{6}} = \frac{1}{\sqrt{6}}$$

Las alturas del triángulo se calculan de acuerdo a:

$$h_c = |\vec{b}| \sin \alpha = \sqrt{5}$$

$$h_b = |\vec{a}| \sin \omega = \frac{\sqrt{5}}{\sqrt{6}}$$

$$h_a = |\vec{c}| \sin \beta = 1$$

**Ejemplo 1.4:** Considere un paralelogramo donde se dan tres vértices  $A = (0,1,1)$ ,  
 $B = (1,0,1)$ ,  $C = (1,1,0)$

- a) Determine el cuarto vértice
- b) Determine el área del paralelogramo
- c) Determine las longitudes de las diagonales

Solución:

Construir los vectores:

$$\vec{AC} = \vec{OC} - \vec{OA} = (1,0,-1)$$

$$\vec{AB} = \vec{OB} - \vec{OA} = (1,-1,0)$$

De manera que:

$$\vec{AD} = \vec{AB} + \vec{AC} = (2,-1,-1)$$

Entonces el cuarto vértice está en la posición (esta es una solución de otras posibles)

$$\vec{OD} = \vec{OA} + \vec{AD} = (2,0,0)$$

El área del paralelogramo será:

$$A = |\vec{AB} \times \vec{AC}| = |(1,1,1)| = \sqrt{3}$$

Donde las longitudes de las diagonales serán:

$$|\vec{AB} + \vec{AC}| = |(2,-1,-1)| = \sqrt{6}$$

$$|\vec{AB} - \vec{AC}| = |(0,-1,1)| = \sqrt{2}$$

**Ejemplo 1.5:** Escriba la ecuación de un plano que es perpendicular a la dirección  $\hat{n} = (1, -1, 1)/\sqrt{3}$  y que pasa a distancia 3 del origen.

Solución:

La ecuación resulta:

$$\hat{n} \cdot \vec{r} = 3$$

o sea

$$x - y + z = 3\sqrt{3}$$

**Ejemplo 1.6:** Sea una recta:  $x = 2t+1$ ;  $y = -t+2$ ;  $z = 3t-1$ , siendo  $t$  un parámetro. Determine su distancia del origen.

Solución:

La distancia de un punto arbitrario de la recta al origen es:

$$d = \sqrt{x^2 + y^2 + z^2}$$

$$\text{esto es: } d = \sqrt{(2t+1)^2 + (-t+2)^2 + (3t-1)^2} \Rightarrow d = \sqrt{14t^2 - 6t + 6}$$

La cantidad subradical, polinomio de segundo grado, tiene un mínimo justo en el punto medio entre sus dos raíces que son:

$$t_1 = 3/14 + (5/14)\sqrt{3}, \quad t_2 = 3/14 - (5/14)\sqrt{3}$$

$$\text{y el punto medio es: } t = (1/2)(6/14) = 3/14$$

y para ese valor "d" es la distancia de la recta al origen, cuyo valor resulta:

$$d = (5/14)\sqrt{42} = 2.315$$



**Ejemplo 1.7:** Sean  $\vec{a}=(1,1,0)$ ,  $\vec{b} = (-1,1,1)$  dos vectores. Determine la ecuación de un plano que pase por el origen y que contenga los vectores  $\vec{a}$  y  $\vec{b}$ .

Solución:

Si los dos vectores  $a$  y  $b$  están sobre el plano, entonces un vector normal al plano es  $\vec{N}=\vec{a} \times \vec{b}$

Calculando resulta:

$$\vec{N}= (1,1,0) \times (-1,1,1) = (1,-1,2)$$

La ecuación del plano en general es:  $\vec{r} \cdot \vec{N} = \text{constante}$ ,

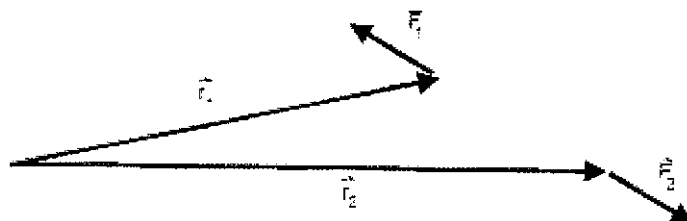
Y si pasa por el origen entonces es:  $\vec{r} \cdot \vec{N} = 0$

Calculando  $(x,y,z) \cdot (1,-1,2) = x - y + 2z$  de modo que la ecuación del plano es:

$$x - y + 2z = 0$$

**Ejemplo 1.8:** Con relación a la figura, demuestre que si  $\vec{F}_1 = -\vec{F}_2$  entonces

$$\vec{r}_1 \times \vec{F}_1 + \vec{r}_2 \times \vec{F}_2 = \vec{0}$$



Solución:

Podemos escribir:

$$\vec{r}_1 \times \vec{F}_1 + \vec{r}_2 \times \vec{F}_2 = 0$$

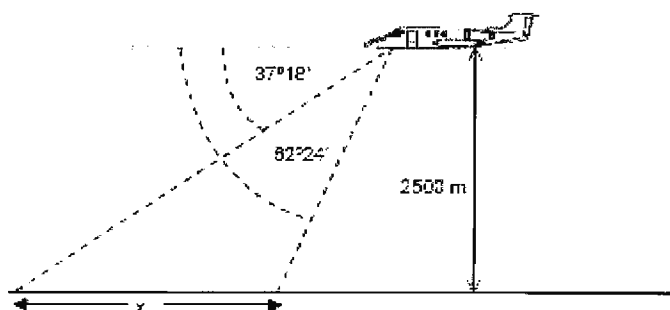
$$\vec{r}_1 \times \vec{F}_1 - \vec{r}_2 \times \vec{F}_1 = 0$$

$$(\vec{r}_1 - \vec{r}_2) \times \vec{F}_1 = 0,$$

Porque  $F_1$  es paralela a  $(\vec{r}_1 - \vec{r}_2)$

**Ejemplo 1.9:** Desde un avión de reconocimiento que vuela a una altura de 2500m, el piloto observa dos embarcaciones que se encuentran en un mismo plano vertical con ángulos de depresión de  $62^{\circ} 24'$  y  $37^{\circ} 18'$  respectivamente.

Encuentre, la distancia  $x$  entre las embarcaciones:



**Solución:**

Con Decimales los ángulos son:  $18/60 = 0.3$  y  $24/60 = 0.4$

Entonces:  $62.4^{\circ}$  y  $37.3^{\circ}$

Si  $d$  es la distancia horizontal entre el avión y la embarcación más cercana, se tiene:

$$\frac{X + d}{2500} = \tan (90 - 37.3)$$

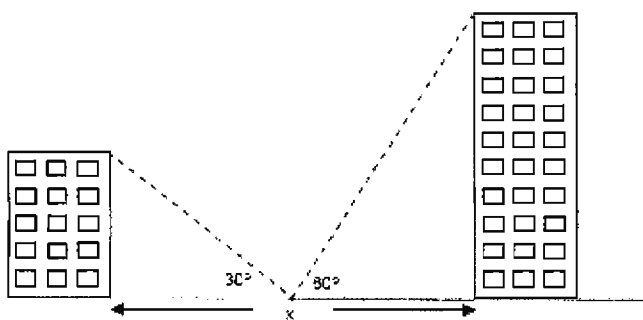
$$2500$$

$$\frac{d}{2500} = \tan (90 - 62.4)$$

$$2500$$

Y restando se obtiene:  $d = 2500 (\cot 37.3 - \cot 62.4) = 1974.751\text{m}$

**Ejemplo 1.10:** Una persona se encuentra en la mitad de la distancia que separa dos edificios y observa la parte más alta de éstos con ángulos de elevación de  $30^{\circ}$  y  $60^{\circ}$  respectivamente. Demuestre que la altura de los edificios está en la relación 1:3.



Solución:

Si las alturas son llamadas  $h_1$  y  $h_2$  respectivamente tenemos que:

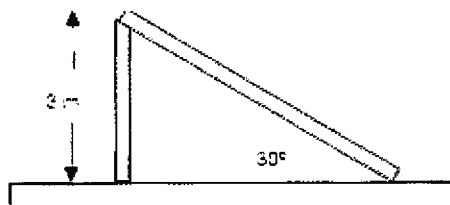
$$\tan 30 = \frac{h_1}{x/2}$$

$$\tan 60 = \frac{h_2}{x/2}$$

de donde,

$$\frac{h_1}{h_2} = \frac{\tan 30}{\tan 60} = \frac{(1/3)\sqrt{3}}{\sqrt{3}} = 1/3$$

**Ejemplo 1.11:** un mástil por efecto del viento se ha quebrado en dos partes, la parte que quedó vertical en el piso mide 3m y la parte de arriba quedó atada al extremo superior de la parte vertical, formando un ángulo de  $30^\circ$  con el piso. Encontrar la altura del mástil.



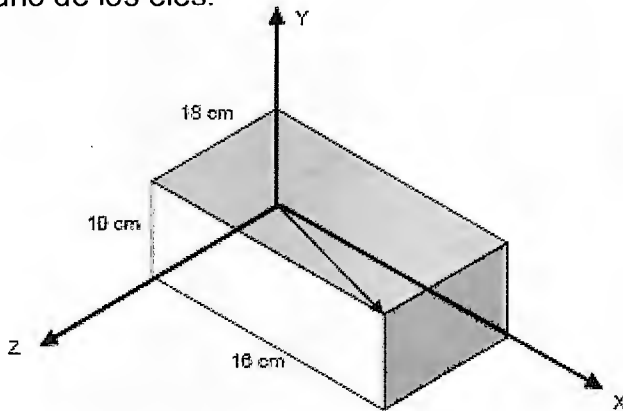
Solución:

La hipotenusa  $c$ , será dada por  $3/c = \sin 30 = 1/2$

Entonces:  $c = 6$

Por lo tanto la altura del mástil era :  $h = 3 + 6 = 9\text{m}$

**Ejemplo 1.12:** Una caja tiene 16cm de largo, 18 cm de ancho y 10 cm de alto. Encuentre la longitud de la diagonal de la caja y el ángulo que ésta forma con cada uno de los eies.



Solución:

El vector que representa la diagonal es:

$$\mathbf{r} = 16\hat{i} + 18\hat{j} + 10\hat{k}$$

y entonces su longitud es:

$$r = \sqrt{16^2 + 18^2 + 10^2} = 26.077 \text{ cm}$$

los ángulos están dados por:

$$\cos \alpha = \frac{\mathbf{r} \cdot \hat{i}}{r} = \frac{16}{26.077}$$

$$\cos \beta = \frac{\mathbf{r} \cdot \hat{j}}{r} = \frac{18}{26.077}$$

$$\cos \gamma = \frac{\mathbf{r} \cdot \hat{k}}{r} = \frac{10}{26.077}$$

De donde:

$$\alpha = 52.152^\circ$$

$$\beta = 46.349^\circ$$

$$\gamma = 67.4501^\circ$$

Note que:  $\cos^2 \alpha + \cos^2 \beta + \cos^2 \gamma = 1$

## **1.3.2 Matrices.**

### **1.3.2.1 Origen**

Las matrices aparecen por primera vez hacia el año 1850, introducidas por J.J. Sylvester<sup>22</sup>. El desarrollo inicial de la teoría se debe al matemático W.R. Hamilton en 1853.

En 1858, A. Cayley introduce la notación matricial como una forma abreviada de escribir un sistema de  $m$  ecuaciones lineales con  $n$  incógnitas.

Las matrices se utilizan en el cálculo numérico, en la resolución de sistemas de ecuaciones lineales, de las ecuaciones diferenciales y de las derivadas parciales. Además de su utilidad para el estudio de sistemas de ecuaciones lineales, las matrices aparecen de forma natural en geometría, estadística, economía, informática, física entre otras ramas.

La utilización de matrices (arrays) constituye actualmente una parte esencial en los lenguajes de programación, ya que la mayoría de los datos se introducen en los ordenadores como tablas organizadas en filas y columnas: hojas de cálculo, bases de datos,...

---

<sup>22</sup> Matemático Inglés (1814-1897) Creó el término matriz, con el propósito de distinguir entre matrices y determinantes. Su idea, era que matriz significara <madre de los determinantes. Ivora Castillo. Análisis Matemático, pp 9.

### 1.3.2.2 Definición

Una matriz<sup>23</sup> es un conjunto de elementos de cualquier naturaleza aunque, en general, suelen ser números ordenados en filas y columnas.

Se llama Matriz de orden " $m \times n$ " a un conjunto rectangular de elementos  $a_{ij}$  dispuestos en  $m$  filas y en  $n$  columnas. El orden de una éstas también se denomina dimensión o tamaño, siendo  $m$  y  $n$  números naturales.

Los arreglos<sup>24</sup> se denotan con letras mayúsculas: A, B, C, ... y los elementos de las mismas con letras minúsculas y subíndices que indican el lugar ocupado: a, b, c, ... Un elemento genérico que ocupe la fila  $i$  y la columna  $j$  se escribe  $a_{ij}$ .

Si el elemento genérico aparece entre paréntesis también representa a toda la matriz :  $A = (a_{ij})$

En un sistema de ecuaciones lineales<sup>25</sup> con  $n$  variables o incógnitas  $x_1, x_2 \dots x_n$ , de la forma:

$$\begin{aligned} & \text{(ec7)} \quad \begin{array}{l} 1x_1 + a_{12}x_2 + .....a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + .....a_{2n}x_n = b_2 \\ ..... \\ a_{21}x_1 + a_{22}x_2 + .....a_{2n}x_n = b_n \end{array} \end{aligned}$$

Pueden diferenciarse tres clases de elementos: Incógnitas, coeficientes de las incógnitas y términos independientes de cada ecuación del sistema. Estos elementos están representados por:

$X_j$  = la  $j$ -ésima incógnita o variable  $j=1, \dots, n$

$a_{ij}$  = el coeficiente que aparece en la  $i$ -ésima ecuación multiplicando a la  $j$ -ésima incógnita con  $i=1, \dots, m$  y  $j=1, \dots, n$ .

<sup>23</sup> Definición de matriz; capítulo 8 pp191 D.C Murdoch. Geometría Analítica con Vectores y Matrices.

<sup>24</sup> Arreglo: Un arreglo puede definirse como un grupo o una colección finita, homogénea y ordenada de elementos. Los arreglos pueden ser de una, dos o tres dimensiones. Norton, Meter. *Introducción a la Computación*. Mc-Graw Hill 2000. capítulo 5 pp 188.

<sup>25</sup> Rosa Barbolla, Paloma Sanz. *Álgebra Lineal y Teoría de Matrices*. Capítulo 2, pp 41-43.

$b_i$  = el término independiente de la  $i$ -ésima ecuación con  $i=1, \dots, m$ .

Se trata de agrupar estos elementos de alguna forma, que permita escribir el sistema de forma más compacta y operativa. Si se hace uso de la operación de producto escalar para dos vectores de  $\mathbf{R}^n$ , cada ecuación del sistema se puede expresar como el producto escalar de un vector cuyas componentes son los coeficientes que aparecen en la ecuación y un vector que se nota por  $x$  constituido por las  $n$  incógnitas. El resultado de esta operación es el término independiente de la ecuación correspondiente. Así pues, la ecuación  $i$ -ésima del sistema:

$$a_{i1}x_1 + a_{i2}x_2 + \dots + a_{in}x_n = b_i$$

Atendiendo a la definición de producto escalar es: (ver página no \_\_\_\_ de este documento)

$$(ec8) \quad a_i \cdot X = b_i$$

Con  $a_i = (a_{i1}, a_{i2}, \dots, a_{in})$  Y  $X = (x_1, x_2, \dots, x_n)$ , lo que permite expresar el sistema en la forma:

$$(ec9) \quad \left. \begin{array}{l} a_1 \cdot X = b_1 \\ a_2 \cdot X = b_2 \\ \dots \\ a_m \cdot X = b_m \end{array} \right\}$$

Si por  $b$  se denota al vector cuyas componentes son los términos independientes  $b_i$ , con  $i=1, \dots, m$  y por:

$$\begin{pmatrix} a_1 \\ a_2 \\ \dots \\ a_m \end{pmatrix} \cdot x$$

Los primeros miembros de las  $m$  ecuaciones del sistema, se tiene:

$$(ec10) \quad \begin{pmatrix} a_1 \blacksquare \\ a_2 \blacksquare \\ \dots \\ a_m \blacksquare \end{pmatrix} X = b$$

Y como para cada  $i=1, \dots, n$ ,  $a_i \blacksquare = (a_{i1}, \dots, a_{in})$  se puede escribir

$$(ec11) \quad \begin{pmatrix} a_1 \blacksquare \\ a_2 \blacksquare \\ \dots \\ a_m \blacksquare \end{pmatrix} X = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} X = b$$

Este bloque rectangular de  $m$  por  $n$  números ordenados es una matriz de orden  $m \times n$ , pues está constituido por  $m$  filas y  $n$  columnas.

La definición formal de este concepto es la siguiente:

Dados  $a_{ij} \in K$  con  $i=1, \dots, m$   $j=1, \dots, n$  al rectángulo de  $m \times n$  "números" ordenados en una tabla con  $m$  filas y  $n$  columnas de la forma:

$$(ec12) \quad A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \dots & \cdot \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

Se le denomina *matriz* de orden  $m \times n$ .



Se dice que dos matrices son iguales cuando son del mismo tipo y los números ocupan iguales posiciones, en ambas matrices y son idénticos.

Una matriz tipo  $(n,n)$  recibe el nombre de *Matriz Cuadrada* de orden  $n$ ..

El vector o arreglo  $[x_1, x_2, x_3]$  es una matriz del tipo  $(1,3)$ .

Los números que forman una matriz, reciben el nombre de *elementos de la matriz*; en general, vamos a tratar solamente con matrices de elementos reales; sin embargo, los elementos de una matriz también podrían ser números complejos o aun polinomios.

A toda matriz  $M$  se le asocia otra  $M^T$ , llamada: La Traspuesta de  $M$ , que se obtiene cambiando las líneas de  $M$  en columnas; así, si  $A$  y  $B$  son las dos matrices anteriores, se tiene:

$$(ec13) \quad A = \begin{pmatrix} 2 & -6 \\ 3 & 4 \end{pmatrix} \quad A^T = \begin{pmatrix} 2 & 3 \\ -6 & 4 \end{pmatrix}$$

$$B = \begin{pmatrix} 5 & 1 & 0 & 3 \\ 2 & 6 & -1 & 4 \\ 3 & -5 & 7 & 0 \end{pmatrix} \quad B^T = \begin{pmatrix} 5 & 2 & 3 \\ 1 & 6 & -5 \\ 0 & -1 & 7 \\ 3 & 4 & 0 \end{pmatrix}$$

Es claro que si  $M$  es del tipo  $(m,n)$ ,  $M^T$  es del tipo  $(n,m)$ .

Un vector de  $V_n$  (que suele designarse con el nombre de vector de orden  $n$ ), tal como  $X = [x_1, x_2, \dots, x_n]$ , se llama *Vector de línea*, para distinguirlo de su traspuesto.

$$X^T = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ \dots \\ \dots \\ x_n \end{pmatrix}$$
 Que recibe el nombre de *vector de Columna*. Las líneas de una matriz, consideradas como vectores, se llaman *Vectores de línea*; análogamente, las columnas se llaman *vectores de columna*.

Se acostumbra usar la notación inicial para designar los elementos de una matriz; por ejemplo, si  $A$  es una matriz cualquiera de tipo  $(m,n)$ , el elemento situado en la línea  $i$  y la columna  $j$  de  $A$ , se representa por  $a_{ij}$  (ó  $a_{ij}$ ,  $b_j$ ,  $c_{ij}$ , ...), esta definición hace referencia a la fila y columna respectivamente, en la cual esté situado el elemento.

### 1.3.2.2 Tipos de Matrices<sup>26</sup>

Tipo de Matriz	Definición	Ejemplo
Fila	Aquella matriz que tiene una sola fila, siendo su orden $1 \times n$ .	$A_{1 \times 3} = (7 \quad 2 \quad -5)$
Columna	Aquella matriz que tiene una sola columna, siendo su orden $m \times 1$	$A_{3 \times 1} = \begin{pmatrix} -7 \\ 1 \\ 6 \end{pmatrix}$
Rectangular	Aquella matriz que tiene distinto número de filas que de columnas, siendo su orden $m \times n$ , $m \neq n$ .	$A_{3 \times 4} = \begin{pmatrix} 1 & 3 & 2 & 9 \\ 5 & 7 & -1 & 8 \\ 0 & 3 & 5 & 1 \end{pmatrix}$

<sup>26</sup> Para mejor referencia puede consultarse Barbolla, Rosa, Algebra Lineal y Teoría de Matrices. Prentice Hall 1998.

Opuesta	La matriz opuesta de una dada es la que resulta de sustituir cada elemento por su opuesto. La opuesta de A es -A.	$B = \begin{pmatrix} 1 & 3 \\ 5 & -7 \\ -6 & 4 \end{pmatrix} \quad -B = \begin{pmatrix} -1 & -3 \\ -5 & 7 \\ 6 & -4 \end{pmatrix}$
Nula	Si todos sus elementos son cero. También se denomina matriz cero y se denota por $0_{m \times n}$ .	$0_{3 \times 4} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$
Cuadrada	<p>Aquella matriz que tiene igual número de filas que de columnas, <math>m = n</math>, diciendose que la matriz es de orden n.</p> <p>Diagonal principal : son los elementos <math>a_{11}, a_{22}, \dots, a_{nn}</math></p> <p>Diagonal secundaria : son los elementos <math>a_{ij}</math> con <math>i+j = n+1</math></p> <p>Traza de una matriz cuadrada : es la suma de los elementos de la diagonal principal A</p>	$B_{2 \times 2} = \begin{pmatrix} 1 & 2 \\ 4 & 3 \end{pmatrix}$ <p>Diagonal Principal:</p> $A = \begin{pmatrix} 5 & 1 & 5 & -6 \\ 1 & 2 & 0 & 3 \\ 7 & -3 & 4 & 11 \\ 1 & 9 & 3 & 8 \end{pmatrix}$ <p>Diagonal Secundaria:</p> $A = \begin{pmatrix} 5 & 1 & 5 & -6 \\ 1 & 2 & 0 & 3 \\ 7 & -3 & 4 & 11 \\ 1 & 9 & 3 & 8 \end{pmatrix}$

Simétrica	Es una matriz cuadrada que es igual a su traspuesta. $A = A^t$ , $a_{ij} = a_{ji}$	$A_3 = \begin{pmatrix} 1 & 9 & -6 \\ 9 & 2 & 1 \\ -6 & 1 & 5 \end{pmatrix}$
Antisimétrica	Es una matriz cuadrada que es igual a la opuesta de su traspuesta. $A = -A^t$ , $a_{ij} = -a_{ji}$ Necesariamente $a_{ii} = 0$	$A_3 = \begin{pmatrix} 0 & 9 & -6 \\ 9 & 0 & 1 \\ -6 & 1 & 0 \end{pmatrix}$
Diagonal	Es una matriz cuadrada que tiene todos sus elementos nulos excepto los de la diagonal principal	$A = \begin{pmatrix} 7 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & -2 \end{pmatrix}$
Escalar	Es una matriz cuadrada que tiene todos sus elementos nulos excepto los de la diagonal principal que son iguales.	$A = \begin{pmatrix} 7 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 7 \end{pmatrix}$
Identidad	Es una matriz cuadrada que tiene todos sus elementos nulos excepto los de la diagonal principal que son iguales a 1. También se denomina matriz unidad.	$I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Triangular	<p>Es una matriz cuadrada que tiene todos los elementos por encima (por debajo) de la diagonal principal nulos.</p>	$A = \begin{pmatrix} 1 & 3 & 5 \\ 0 & 4 & -1 \\ 0 & 0 & 9 \end{pmatrix}$ <p>T. superior</p> $A = \begin{pmatrix} 1 & 0 & 0 \\ 5 & 4 & 0 \\ 2 & 8 & 7 \end{pmatrix}$ <p>T. inferior</p>
Ortogonal	<p>Una matriz ortogonal es necesariamente cuadrada e invertible : <math>A^{-1} = A^T</math></p> <p>La inversa de una matriz ortogonal es una matriz ortogonal.</p> <p>El producto de dos matrices ortogonales es una matriz ortogonal.</p> <p>El determinante de una matriz ortogonal vale +1 ó -1.</p>	$A \cdot A^T = A^T \cdot A = I$ $\begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{pmatrix} \cdot \begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{pmatrix}$ $= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
Normal	<p>Una matriz es normal si conmuta con su traspuesta. Las matrices simétricas, antisimétricas u ortogonales son necesariamente normales.</p>	$A = \begin{pmatrix} 5 & 4 \\ -4 & 5 \end{pmatrix}$ $A \cdot A^T = A^T \cdot A$

Inversa	<p>Decimos que una matriz cuadrada <math>A</math> tiene inversa, <math>A^{-1}</math>, si se verifica que :</p> $A \cdot A^{-1} = A^{-1} \cdot A = I$	$A = \begin{pmatrix} 2 & 3 \\ 1 & 1 \end{pmatrix}$ $A^{-1} = \begin{pmatrix} -1 & 3 \\ 1 & -2 \end{pmatrix}$
---------	--	--

### 1.3.2.3 Operaciones con Matrices<sup>27</sup>

#### Suma y Diferencia de Matrices:

Como en aritmética y en álgebra elemental, en el álgebra matricial se utiliza la operación de adición. Es posible la adición o sustracción de dos matrices sólo cuando son del mismo orden. Se dice que tales matrices tienen *conformidad para la adición o sustracción*.<sup>28</sup>

La suma o diferencia de dos matrices **A** y **B**, de orden  $m \times n$ , es otra matriz **C** de orden  $m \times n$ , en la que cada elemento se obtiene como la suma o resta de los correspondientes elementos de **A** y **B**. Por lo tanto, la suma o resta de **A** y **B** representada como la matriz **C** se puede escribir como: **C** = **A**  $\pm$  **B**

En donde  $c_{ij} = a_{ij} \pm b_{ij}$  para  $i = 1, 2, \dots, m$  y  $j = 1, 2, \dots, n$ . Por ejemplo:

$$\begin{aligned} C = A + B &= \begin{pmatrix} 4 & 3 & 2 \\ 8 & 1 & -1 \end{pmatrix} + \begin{pmatrix} 0 & 2 & -2 \\ -2 & 7 & 0 \end{pmatrix} = \begin{pmatrix} 4 & 5 & 0 \\ 6 & 8 & -1 \end{pmatrix} \\ (ec14) \quad C = A - B &= \begin{pmatrix} 4 & 3 & 2 \\ 8 & 1 & -1 \end{pmatrix} - \begin{pmatrix} 0 & 2 & -2 \\ -2 & 7 & 0 \end{pmatrix} = \begin{pmatrix} 4 & 1 & 4 \\ 10 & -6 & -1 \end{pmatrix} \end{aligned}$$

#### Propiedades de la suma y resta de matrices

1.  $A + (B + C) = (A + B) + C$  (propiedad asociativa)
2.  $A + B = B + A$  (propiedad conmutativa)
3.  $A + 0 = A$  (0 es la matriz nula)
4. La matriz  $-A$ , que se obtiene cambiando de signo todos los elementos de **A**, recibe el nombre de matriz opuesta de **A**, ya que  $A + (-A) = 0$ .

<sup>27</sup> Corcobado Cartes Juan Luís MATEMÁTICA 1, Editorial CEI Cáceres, capítulo 2 pp 23-33

<sup>28</sup> Dorf Richard C. Introducción al Álgebra de Matrices, Capítulo 1 pp 51 – 53 Se define el término conformidad como apropiado o adecuado.

## Producto de una Matriz por un Escalar<sup>29</sup>:

Si  $A = (a_{ij})$  es una matriz de  $m \times n$ , y si  $k$  es un escalar, entonces la matriz  $k.A$  de  $m \times n$ , está dada por:

$$(ec15) \quad kA = (ka_{ij}) = \begin{pmatrix} ka_{11} & ka_{12} & \dots & ka_{1n} \\ ka_{21} & ka_{22} & \dots & ka_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ ka_{m1} & ka_{m2} & \dots & ka_{mn} \end{pmatrix}$$

Ejemplo:

$$A = \begin{pmatrix} 5 & -5 & 0 \\ 4 & 2 & 6 \end{pmatrix}, \Rightarrow 3A = \begin{pmatrix} 3 \times 5 & 3 \times (-5) & 3 \times 0 \\ 3 \times 4 & 3 \times 2 & 3 \times 6 \end{pmatrix} = \begin{pmatrix} 15 & -15 & 0 \\ 12 & 6 & 18 \end{pmatrix}$$

## Producto de Matrices

Para empezar, se tiene la multiplicación de un vector renglón  $x$  por un vector columna  $y$ . Es necesario que tengan conformabilidad para la multiplicación, que implica que ambas contienen  $n$  elementos. Por lo tanto, la multiplicación de matrices se escribe como sigue:

$$(ec16) \quad x.y = [x_1, x_2, \dots, x_n] \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

La multiplicación de matrices de un vector renglón  $1 \times n$ , y un vector columna  $n \times 1$ , se define como una matriz  $C$ , resultante de orden  $1 \times 1$ , tal que:

$$C = [x_1 y_1 + x_2 y_2 + \dots + x_n y_n]$$

Es decir, el resultado, de la multiplicación de un renglón por una columna es un número escalar.

<sup>29</sup> Kaplan Matemáticas avanzadas Capítulo 5 pp 242.



Por ejemplo:

$$xy = \begin{bmatrix} 2 & 5 & -2 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ -3 \\ 4 \end{bmatrix} = [2*(3) + 5*(-3) + (-2)*(4)] = -17$$

### Propiedades del Producto de Matrices por un Escalar

1.  $k(A + B) = kA + kB$  (propiedad distributiva)
2.  $(k + W)A = kA + WA$  (propiedad distributiva respecto a suma con escalares)
3.  $(kW)A = k(WA)$  (propiedad pseudo asociativa)
4. Elemento unidad: Existe  $1 \in \mathbb{R}$  tal que  $1.A = A$  para cualquier  $A \in M_{m \times n}$ .

### Propiedades del Producto de Matrices

1. *No es conmutativo*
2.  $A(BC) = (AB)C$  (propiedad asociativa)
3. Existen matrices  $I_m$  e  $I_p$  tales que para todo  $A \in M_{m \times n}$   
 $I_m A = A$        $A I_p = A$
4.  $B(C+D) = BC + BD$ . (Propiedad distributiva respecto a la suma)

## Ejemplos de Matrices

**Ejemplo1.11:** Efectuar las siguientes operaciones con matrices:

$$a) \begin{pmatrix} 1 & 2 & -1 & 0 \\ 4 & 0 & 2 & 1 \\ 2 & -5 & 1 & 2 \end{pmatrix} + \begin{pmatrix} 3 & -4 & 1 & 2 \\ 1 & 5 & 0 & 3 \\ 2 & -2 & 3 & -1 \end{pmatrix} = \begin{pmatrix} (1+3) & (2+(-4)) & (-1+1) & (0+2) \\ (4+1) & (0+5) & (2+0) & (1+3) \\ (2+2) & (-5+(-2)) & (1+3) & (2+(-1)) \end{pmatrix} = \begin{pmatrix} 4 & -2 & 0 & 2 \\ 5 & 5 & 2 & 4 \\ 4 & -7 & 4 & 1 \end{pmatrix}$$

$$b) \begin{pmatrix} 1 & 2 & -1 & 0 \\ 4 & 0 & 2 & 1 \\ 2 & -5 & 1 & 2 \end{pmatrix} - \begin{pmatrix} 3 & -4 & 1 & 2 \\ 1 & 5 & 0 & 3 \\ 2 & -2 & 3 & -1 \end{pmatrix} = \begin{pmatrix} (1-3) & (2-(-4)) & (-1-1) & (0-2) \\ (4-1) & (0-5) & (2-0) & (1-3) \\ (2-2) & (-5-(-2)) & (1-3) & (2-(-1)) \end{pmatrix} = \begin{pmatrix} 2 & -6 & -2 & -2 \\ 3 & -5 & 2 & -2 \\ 0 & -3 & -2 & 3 \end{pmatrix}$$

$$c) 3 \begin{pmatrix} 1 & 2 & -1 & 0 \\ 4 & 0 & 2 & 1 \\ 2 & -5 & 1 & 2 \end{pmatrix} = \begin{pmatrix} (3.1) & (3.2) & (3.(-1)) & (3.0) \\ (3.4) & (3.0) & (3.2) & (3.1) \\ (3.2) & (3.(-5)) & (3.1) & (3.2) \end{pmatrix} = \begin{pmatrix} 3 & 6 & -3 & 0 \\ 12 & 0 & 6 & 3 \\ 6 & 15 & 3 & 6 \end{pmatrix}$$

$$d) - \begin{pmatrix} 1 & 2 & -1 & 0 \\ 4 & 0 & 2 & 1 \\ 2 & -5 & 1 & 2 \end{pmatrix} = \begin{pmatrix} -1 & -2 & 1 & 0 \\ -4 & 0 & -2 & -1 \\ -2 & -5 & -1 & -2 \end{pmatrix}$$

**Ejemplo1.12:** Dadas las matrices  $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$  Y  $B = \begin{pmatrix} -3 & -2 \\ 1 & -5 \\ 4 & 3 \end{pmatrix}$ , hallar  $D = \begin{pmatrix} p & q \\ r & s \\ t & u \end{pmatrix}$  de manera que  $A + B - D = 0$

Solución:

$$\text{La matriz } A + B - D = \begin{pmatrix} 1-3-p & 2-2-q \\ 3+1-r & 4-5-s \\ 5+4-t & 6+3-u \end{pmatrix} = \begin{pmatrix} -2-p & -q \\ 4-r & -1-s \\ 9-t & 9-u \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Con lo que  $-2-p=0$  por lo tanto  $p = -2$

$4-r=0$  por lo tanto  $r = 4$

$9-t=0$  por lo tanto  $t = 9$

$-q = 0$

$-1-s = 0$  por lo tanto  $s = -1$

$9-u = 0$  por lo tanto  $u = 9$

Entonces la matriz  $D = \begin{pmatrix} -2 & 0 \\ 4 & -1 \\ 9 & 9 \end{pmatrix} = A + B$

**Ejemplo No. 1.13:** Efectuar los siguientes productos

a)  $[4 \ 5 \ 6] \cdot \begin{pmatrix} 2 \\ 3 \\ -1 \end{pmatrix} = [4(2) + 5(3) + 6(-1)] = [17]$

b)  $\begin{pmatrix} 2 \\ 3 \\ -1 \end{pmatrix} \cdot [4 \ 5 \ 6] = \begin{pmatrix} 2(4) & 2(5) & 2(6) \\ 3(4) & 3(5) & 3(6) \\ -1(4) & -1(5) & -1(6) \end{pmatrix} = \begin{pmatrix} 8 & 10 & 12 \\ 12 & 15 & 18 \\ -4 & -5 & -6 \end{pmatrix}$

c)  $[1 \ 2 \ 3] \cdot \begin{pmatrix} 4 & -6 & 9 & 6 \\ 0 & -7 & 10 & 7 \\ 5 & 8 & -11 & -8 \end{pmatrix} = [ \{1(4)+2(0)+3(5)\} \quad \{1(-6)+2(-7)+3(8)\} \\ \{1(9)+2(10)+3(-11)\} \quad \{1(6)+2(7)+3(-8)\} ]$   
 $= [19 \ 4 \ -4 \ -4]$

d)  $\begin{pmatrix} 2 & 3 & 4 \\ 1 & 5 & 6 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 2(1) + 3(2) + 4(3) \\ 1(1) + 5(2) + 6(3) \end{pmatrix} = \begin{pmatrix} 20 \\ 29 \end{pmatrix}$

**Ejemplo No. 1.14:** Demostrar que  $A = \begin{pmatrix} 1 & 1 & 3 \\ 5 & 2 & 6 \\ -2 & -1 & -3 \end{pmatrix}$  es una matriz nilpotente<sup>30</sup> de orden 3

$$A^2 = \begin{pmatrix} 1 & 1 & 3 \\ 5 & 2 & 6 \\ -2 & -1 & -3 \end{pmatrix} \begin{pmatrix} 1 & 1 & 3 \\ 5 & 2 & 6 \\ -2 & -1 & -3 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 3 & 3 & 9 \\ -1 & -1 & -3 \end{pmatrix} \quad \text{y} \quad A^3 = A^2 \cdot A = \begin{pmatrix} 0 & 0 & 0 \\ 3 & 3 & 9 \\ -1 & -1 & -3 \end{pmatrix} \begin{pmatrix} 1 & 1 & 3 \\ 5 & 2 & 6 \\ -2 & -1 & -3 \end{pmatrix}$$

$$A^3 = 0$$

<sup>30</sup> Decimos que una matriz cuadrada A de orden n es **NILPOTENTE** si y sólo si se verifica que  $A \cdot A = 0_n$ , es decir  $A^2 = 0_n$ . (MATRIZ NULA). Referencia Ayres jr. Frank. Matrices Editorial McGraw-Hill 1993, Capítulo 2. Matrices especiales.

**Ejemplo No. 1.15:** Demostrar que  $\begin{pmatrix} 2 & -2 & -4 \\ -1 & 3 & 4 \\ 1 & -2 & -3 \end{pmatrix}$  es una matriz idempotente<sup>31</sup>.

$$A^2 = \begin{pmatrix} 2 & -2 & -4 \\ -1 & 3 & 4 \\ 1 & -2 & -3 \end{pmatrix} \begin{pmatrix} 2 & -2 & -4 \\ -1 & 3 & 4 \\ 1 & -2 & -3 \end{pmatrix} = \begin{pmatrix} 2 & -2 & -4 \\ -1 & 3 & 4 \\ 1 & -2 & -3 \end{pmatrix} = A$$

**Ejemplo No. 1.16:** Demostrar que  $(AB)' = B'A'$ .

Sean  $A = [a_{ij}]$  una matriz de orden  $m \times n$  y  $B = [b_{ij}]$  de orden  $n \times p$ ; entonces  $C = AB = [c_{ij}]$  es de orden  $m \times p$ . El elemento de la fila  $i$  y columna  $j$  de  $AB$  es

$$c_{ij} = \sum_{k=1}^n a_{ik} \cdot b_{kj} \quad \text{que también pertenece a la fila } j \text{ y columna } i \text{ de } (AB)'$$

Los elementos de la fila  $j$  de  $B'$  son  $b_{1j}, b_{2j}, \dots, b_{nj}$ , y los de la columna  $i$  de  $A$  son

$a_{i1}, a_{i2}, \dots, a_{in}$ . Por tanto, el elemento de la fila  $j$  y columna  $i$  de  $B'A'$  es:

$$\sum_{k=1}^n b_{kj} \cdot a_{ik} = \sum_{k=1}^n a_{ik} \cdot b_{kj} = c_{ij}.$$

Luego,  $(AB)' = B'A'$ .

**Ejemplo No. 1.17:** Encontrar la traspuesta de la matriz  $B = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$

Solución:

$$B' = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

<sup>31</sup> Decimos que una matriz cuadrada  $A$  de orden  $n$  es IDEMPOTENTE si y sólo si se verifica que  $A.A=A$ , es decir  $A^2=A$ . Referencia Ayres jr. Frank. Matrices Editorial McGraw-Hill 1993, Capítulo 2. Matrices especiales...

2

**GRAFICADO EN DOS  
DIMENSIONES**

---

## 2.1 Introducción.

Se llaman gráficos a todos los objetos que se pueden dibujar en una escena, sean éstos puntos, segmentos, curvas, arcos, entre otros.

Todo aquello que se dibuja en un sistema gráfico está formado por puntos llamados píxeles<sup>1</sup>; dichos puntos se representan mediante coordenadas, estas son un conjunto de números que corresponden a cada uno de los ejes, por ejemplo: un punto 2D se representa mediante dos coordenadas  $(x,y)$ , así un punto 3D se representa con tres coordenadas  $(x,y,z)$ .

Si queremos realizar efectos en los trazos (rotación, reflexión, traslación, escala, etc.), hay que aplicar cambios a todo el sistema gráfico; esto se logra modificando los factores que definen el sistema: ángulos de los ejes, ubicación del origen, factor de escala, etc. A estos cambios les llamamos transformaciones del sistema.

---

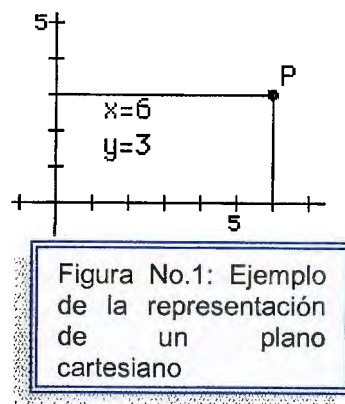
<sup>1</sup> Pixel: punto mínimo de luz que pasa a formar parte de una imagen.

## 2.2 Sistema de coordenadas rectangulares.

En un sistema de coordenadas rectangulares<sup>2</sup> o cartesiano se puede localizar un punto con una sola pareja de puntos (x,y) estos valores son las distancias dirigidas, partiendo del origen, desde los ejes x e y respectivamente. El origen es el punto donde se intersectan los dos ejes coordenados.

Las ecuaciones de los ejes **X** e **Y** son respectivamente **y=0** y **x=0**, rectas<sup>3</sup> que se cortan en el origen 0 cuyas coordenadas son, obviamente, (0,0).

La distancia en un eje se llama "x" y en el otro "y". Dado un punto P se dibujan, desde él, líneas paralelas a los ejes y los valores de "x" e "y" definen totalmente el punto. En honor a Descartes<sup>4</sup>, esta forma de designación de los puntos se conoce como sistema cartesiano y los dos números (x, y) que definen la posición de cualquier punto son sus coordenadas cartesianas (ver figura No.1).



Los ejes dividen el espacio en cuatro cuadrantes en los que los signos de las coordenadas alternan de positivo a negativo; así por ejemplo las coordenadas de cualquier punto A serán ambas positivas, mientras que las del B serán ambas negativas.

<sup>2</sup> Bourne, D.E. Análisis Vectorial y Tensores Cartesianos, editorial Limusa 1980

<sup>3</sup> La recta es la línea más corta que une dos puntos, y el lugar geométrico de los puntos del plano (o el espacio) en una misma dirección. Es uno de los entes geométricos fundamentales, junto al punto y el plano. Son considerados conceptos primitivos, o sea que no es posible definirlos en base a otros elementos ya conocidos. Pita Ruiz, Claudio. Cálculo Vectorial. Prentice Hall 1998.

<sup>4</sup> René Descartes, filósofo y matemático francés (1596-1650) fundamenta su pensamiento filosófico en la necesidad de tomar un punto de partida sobre el que construir todo el conocimiento: **Pienso luego existo**. En matemáticas es el creador de la geometría analítica, construida también tomando un punto de partida y dos rectas perpendiculares que se cortan en ese punto, es el denominado sistema de referencia cartesiano.

Las coordenadas de un punto cualquiera vendrán dadas por las proyecciones del segmento entre el origen y el punto sobre cada uno de los ejes.

### Coordenadas cartesianas rectangulares<sup>5</sup>

El sistema cartesiano  $(x, y)$  puede extenderse hacia las tres dimensiones añadiendo una tercera coordenada  $z$ . Si  $(x, y)$  es un punto en una hoja, entonces el punto  $(x, y, z)$  en el espacio se consigue situándose en  $(x, y)$  y elevándose una distancia  $z$  sobre el papel (los puntos por debajo del papel tienen  $z$  negativa).

Es simple y claro, una vez que se toma la decisión de en qué lado de la hoja es positiva  $z$ . Por común acuerdo, las ramas positivas de los ejes  $(x, y, z)$ , (ver figura No.2) siguen el pulgar y los dos primeros dedos de la mano derecha, en el mismo orden, cuando se extienden de tal forma que formen el mayor ángulo entre ellos.

La posición de un punto  $P$  con respecto a un sistema dado de ejes cartesianos rectangulares, se especifica de la manera siguiente: se dibujan las perpendiculares  $PL$ ,  $PM$  y  $PN$  que van desde  $P$  a los planos  $yz$ ,  $zx$  y  $xy$  respectivamente, como en la figura No.2

Siendo de signo positivo cuando  $P$  y  $Ox$  están situados en el mismo lado del plano  $yz$ , y de signo negativo en caso contrario. Similarmente, se definen:

$Y = \pm$  la longitud  $PM$

$Z = \pm$  la longitud  $PN$ ,

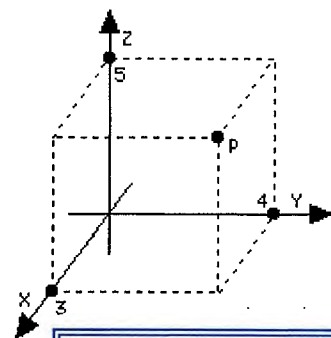


Figura No.2: P es el punto  $(3,4,5)$  donde  $X=3$ ,  $Y=4$  y  $Z=5$

Se toma el signo positivo o negativo para  $y$  dependiendo de  $P$  y  $Oy$  estén situados sobre el mismo lado o sobre el opuesto del plano  $Zx$ ; y el signo positivo o negativo para  $Z$  según  $P$  y  $OZ$  están situados sobre el mismo lado o sobre el opuesto del

<sup>5</sup> Bourne-Kendall. Análisis Vectorial y Tensores Cartesianos. Editorial Limusa Capítulo 1, páginas 14-28



plano xy. A los números  $x$ ,  $y$  y  $z$  se les llama coordenada  $X$ , coordenada  $y$  y coordenada  $z$  de  $P$ . A  $P$  se le denomina punto  $(x,y,z)$ .

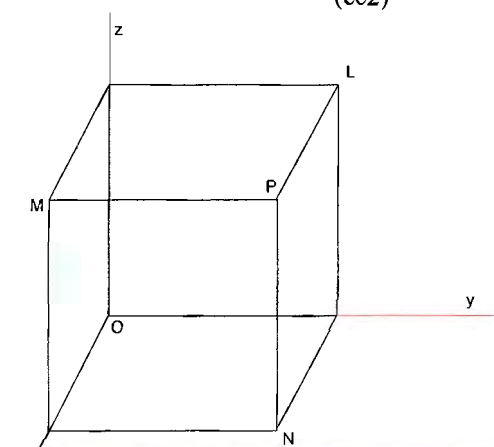
Una observación elemental es que, cuando se dan  $x,y,z$  la posición de  $P$  con respecto a los ejes queda determinada de manera única. Recíprocamente, cuando se da un punto  $P$ , éste determina una triada única de coordenadas. En otras palabras, hay una correspondencia de uno a uno entre los puntos  $P$  y las triadas de números reales  $(x,y,z)$ .

*Distancia al origen*<sup>6</sup>. Para hallar la distancia desde  $P$  al origen  $O$ , se construye paralelepípedo rectangular que tenga como lados a  $PL$ ,  $PM$  y  $PN$ . Por el teorema de Pitágoras se tiene que:

(ec1)  
(ec2)

$$OP^2 = ON^2 + PN^2$$

$$OP^2 = PL^2 + PM^2 + PN^2$$



Puesto que las distancias perpendiculares desde  $P$  a los planos coordenados son:  $|y|$ ,  $|x|$ ,  $|z|$ , se deduce que:

$$OP = \sqrt{x^2 + y^2 + z^2}$$

*Distancia entre Puntos*. La distancia entre los puntos  $P(x,y,z)$  y  $P'(x',y',z')$  se obtienen de la manera siguiente. Se dibujan tres ejes coordenados nuevos  $PX$ ,  $PY$ , y  $PZ$  que pasen por  $P$  y que sean paralelos a los ejes originales  $OX$ ,  $OY$ ,  $OZ$ , como se muestra en la figura 4. Si  $X$ ,  $Y$ ,  $Z$  son las coordenadas de  $P'$  con respecto a éstos nuevos ejes entonces es fácil ver que:

$$X = x' - x, \quad Y = y' - y, \quad Z = z' - z$$

Figura No.3: Representación de planos  $YZ$ ,  $ZX$ , y  $XY$

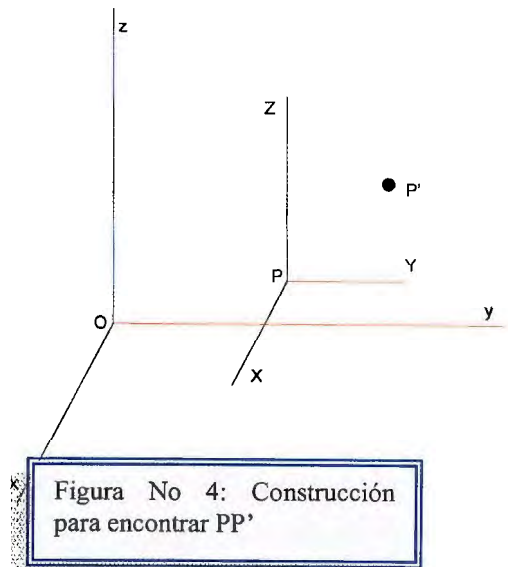
<sup>6</sup> Krasnov M.L. Análisis Vectorial. BCR Editorial. 1999 Capítulo 3. Sistemas de Coordenadas Cartesianas. Páginas 115 – 161.

En base a el teorema:  $OP = \sqrt{x^2 + y^2 + z^2}$

$$(ec3) \quad PP' = \sqrt{X^2 + Y^2 + Z^2}$$

Y en términos de las coordenadas con respecto a los ejes originales

$$(ec4) \quad PP' = \sqrt{(x' - x)^2 + (y' - y)^2 + (z' - z)^2}$$



Sobre cada uno de los ejes se definen vectores unitarios (*i* y *j*) como aquellos paralelos a los ejes y de módulo (longitud) la unidad.

En forma vectorial, la posición del punto A se define respecto del origen con las componentes del vector **OA**. (ver imagen No.5)

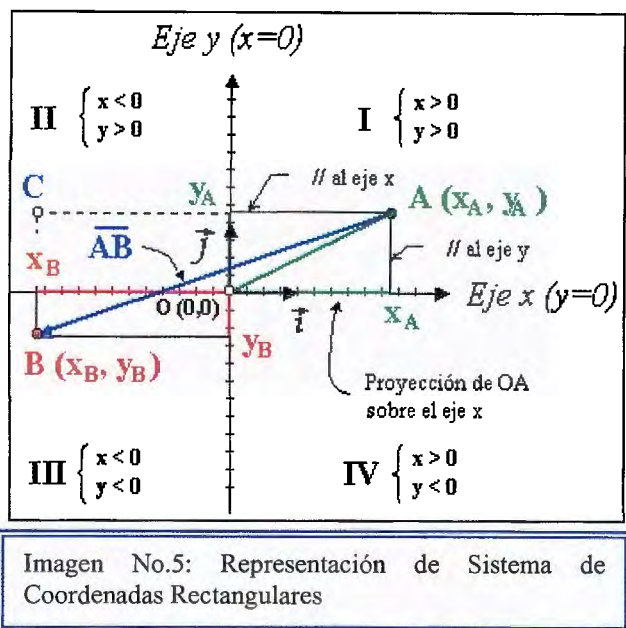
$$\mathbf{OA} = x_A \cdot \mathbf{i} + y_A \cdot \mathbf{j} \equiv (x_A, y_A) = A$$

Nótese que la lista de coordenadas puede expresar tanto la posición de un punto como las componentes de un vector en notación matricial<sup>7</sup>.

La distancia entre dos puntos cualesquiera vendrá dada por la expresión:

$$d_{AB} = [(x_B - x_A)^2 + (y_B - y_A)^2]^{1/2}$$

Lo cual conlleva a la aplicación del teorema de Pitágoras<sup>8</sup> al triángulo rectángulo **ABC**.



<sup>7</sup> Notación Matricial: Forma abreviada de escribir un sistema de *m* ecuaciones lineales con *n* incógnitas *A*. Cayley en 1858 quién define éste cálculo, novedoso en su tiempo. Fundamentos de Matemáticas para Física e Ingeniería. Editorial Limusa. Páginas 14-19.

<sup>8</sup> Teorema de Pitágoras: En un triángulo rectángulo, el cuadrado de la hipotenusa es igual a la suma de los cuadrados de los catetos. Luís Sántalo. Enfoques hacia una Didáctica Humanística de la Matemática. Páginas 45-48

Un vector cualquiera AB se definirá restando, coordenada a coordenada, las del punto de origen de las del punto de destino:

$$\mathbf{AB} = (x_B - x_A) \cdot \mathbf{i} + (y_B - y_A) \cdot \mathbf{j}$$

Evidentemente, el módulo del vector AB será la distancia  $d_{AB}$  entre los puntos A y B antes calculada.

Las coordenadas<sup>9</sup> de un punto cualquiera vendrán dadas por las proyecciones del segmento entre el origen y el punto sobre cada uno de los ejes.

Sobre cada uno de los ejes se definen vectores unitarios ( $\mathbf{i}$  y  $\mathbf{j}$ ) como aquellos paralelos a los ejes y de módulo (longitud) la unidad. En forma vectorial, la posición del punto A se define respecto del origen con las componentes del vector **OA**.

Las gráficas usan ese sistema, al igual que algunos mapas. Funciona bien en una hoja de papel plana, pero el mundo real es tridimensional y a veces es necesario designar los puntos en dicho espacio tridimensional.

### **Cosenos directores y Razones de Dirección<sup>10</sup>**

*Cosenos Directores.* Sea OP una recta orientada desde O (el origen), a un punto P, y representándose por  $\alpha, \beta, \gamma$  los ángulos que OP forma con Ox, Oy y Oz (ver figura No.6). Los cosenos directores de OP se definen como:

$\cos \alpha, \cos \beta$  y  $\cos \gamma$ .

$L = \cos \alpha, m = \cos \beta, n = \cos \gamma$ .

<sup>9</sup> Las Coordenadas son grupos de números que describen una posición a lo largo de una línea, en una superficie o en el espacio.

<sup>10</sup> Scala Estrella Juan José. Análisis Vectorial Vol1. capítulo 2 Cosenos Directores. Páginas 45-53.

Por ejemplo, los cosenos directores del eje x son 1,0,0.

Indique por N el pie de la perpendicular desde P al eje x, sea  $OP=r$ , y suponga que las coordenadas de P son  $(x,y,z)$ . Por el triángulo OPN, se tiene que  $ON=|x|=r|\cos \alpha|$ . También si  $\alpha$  es un ángulo agudo, tanto  $\cos \alpha$  y x son positivos, mientras que si  $\alpha$  es un ángulo obtuso  $\cos \alpha$  y x son negativos.

De aquí se deduce que  $x= r \cos \alpha$ , y análogamente se demuestra que  $y= r \cos \beta$  y  $z=r \cos \gamma$ .

Por tanto los cosenos directores de OP son:

$$(ec5) \quad L=x/r \quad m=y/r \quad n=z/r$$

Puesto que  $r^2 = x^2 + y^2 + z^2$ , se tiene:

$$(ec6) \quad L^2 + m^2 + n^2 = 1$$

Esto demuestra que los cosenos directores de una recta no son independientes: deben satisfacer ésta última ecuación.

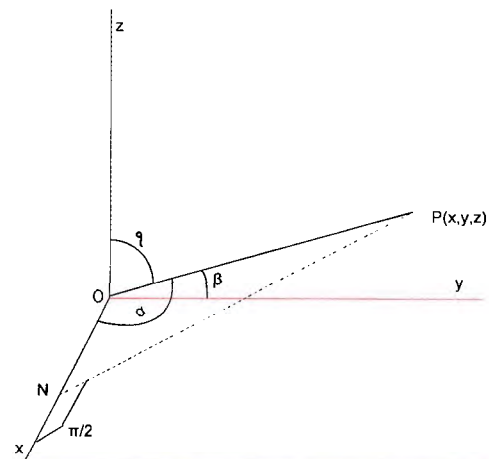


Figura No.6 La recta OP forma ángulos  $\alpha$ ,  $\beta$ ,  $\gamma$  con los ejes

Por definición los cosenos directores de una recta L que no pasa por el origen son los mismos que los de recta paralela trazada desde el origen con el mismo sentido que L.

**Razones de Dirección.** Cuando tres números cualesquiera a, b, y c son tales que

$$(ec7) \quad a:b:c= L:m:n$$

se les llama razones de dirección de OP. Si se verifica dicha ecuación, se tiene:

$$L= a/d \quad m= b/d \quad n=c/d$$

Que al sustituir se tiene:

$$(ec8) \quad d= \pm\sqrt{(a^2 + b^2 + c^2)}$$

La elección del signo indica que hay dos conjuntos posibles de cosenos directores correspondientes a cada conjunto dado de razones de dirección. Estos conjuntos de cosenos directores corresponden a rectas paralelas con direcciones opuestas.

### Ángulos entre Rectas que Pasan por el Origen<sup>11</sup>

Considere dos rectas OA y OA', con cosenos directores L, m, n y L', m', n'. Para hallar el ángulo  $\Theta$  entre ellas, represente por B y B' a los puntos OA y OA' (si es necesario en su prolongación) tal que OB = OB' = 1 (ver la figura No.7)

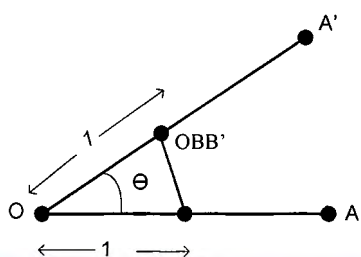


Figura No.7: Representación, de ángulo que pasa por el origen

Entonces de (ec5) cuando  $r=1$  las coordenadas de B y B' son  $(L, m, n)$  y  $(L', m', n')$ . Al aplicar la ley de los cosenos al triángulo OBB', se tiene:

$$(ec9) \quad \cos \Theta = \frac{OB^2 + OB'^2 - BB'^2}{2OB \cdot OB'}$$

Pero por (ec4), se tiene:

$$(ec10) \quad BB'^2 = (L' - L)^2 + (m' - m)^2 + (n' - n)^2$$

$$(ec11) \quad BB'^2 = (L'^2 + m'^2 + n'^2) + (L^2 + m^2 + n^2) - 2(LL' + mm' + nn')$$

Usando los resultados  $L^2 + m^2 + n^2 = 1$  y  $L'^2 + m'^2 + n'^2 = 1$ , se obtiene:

$$(ec12) \quad \cos \Theta = LL' + mm' + nn'$$

Obsérvese que, cuando se toma el ángulo entre OA y OA' igual a  $2\pi - \Theta$ , también se obtiene (ec12), porque  $\cos(2\pi - \Theta) = \cos \Theta$ .

**Condición para rectas perpendiculares.** Dos rectas que pasan por el origen son perpendiculares si y sólo si:  $LL' + mm' + nn' = 0$

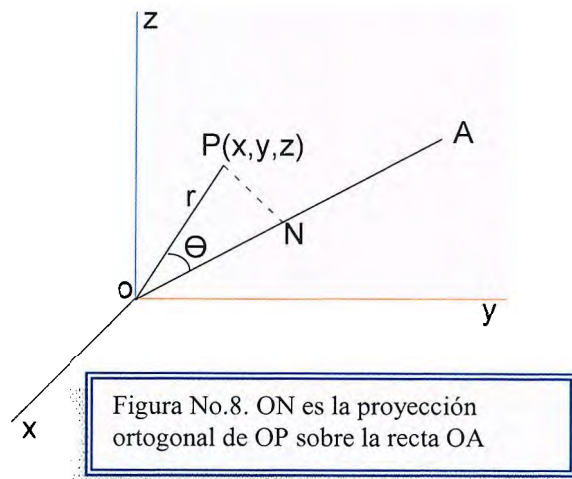
<sup>11</sup> Higdon Archie. Estática Vectorial Tomo1. páginas 38-46

Demostración: Las dos rectas OA y OA' son perpendiculares si y sólo si  $\Theta = \frac{1}{2}\pi$  ó  $\Theta = \frac{3}{2}\pi$ , es decir si y sólo si  $\cos \Theta = 0$ . El resultado se deduce de (ec12).

### Proyección Ortogonal de una Recta sobre Otra

Sean dos rectas OP y OA unidas por un ángulo  $\Theta$ . Entonces se define proyección ortogonal de OP sobre OA como  $OP \cos \Theta$ . (ver figura No.8).

Nótese que si N es el pie de la perpendicular desde P a OA, entonces  $ON = OP |\cos \Theta|$ .



En (ec4) demuestra que las proyecciones OP sobre los ejes cartesianos rectangulares con origen O son las coordenadas x, y, z de P con respecto a estos ejes. Este resultado se extiende ahora para hallar la proyección de OP sobre una recta OA la cual no necesariamente es parte de uno de los ejes coordenados.

Sea L, m, n los cosenos directores de OA y P el punto (x,y,z). Entonces la proyección ortogonal de OP sobre OA es:

$$(ec13) \quad Lx + my + nz$$

*Demostración.* Por (ec5), los cosenos directores de OP son  $x/r$ ,  $y/r$ ,  $z/r$ , donde  $r = OP$ . De ahí que, por la fórmula (ec12) el ángulo  $\Theta$  entre OP y OA esté dado por:

$$\cos \Theta = (Lx + my + nz)/r.$$

La expresión (ec13) se deduce inmediatamente de la definición de proyección ortogonal de OP sobre OA.

## Rotación de Ejes<sup>12</sup>

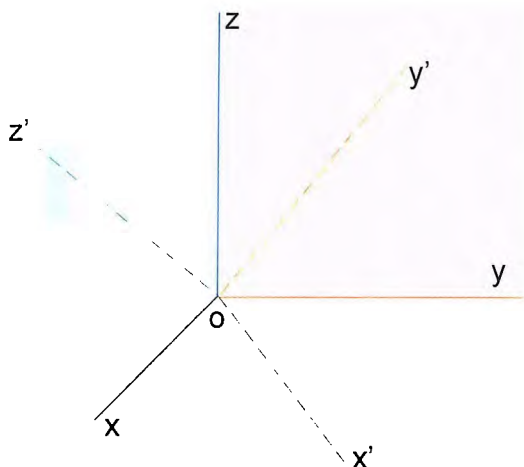


Figura No.9: Rotación de ejes

La matriz de Transformación y sus propiedades. Considere dos sistemas de ejes cartesianos rectangulares derechos Oxyz y Ox'y'z'. Es fácil de ver que, por un movimiento continuo adecuado con respecto a O, el sistema de ejes Oxyz se puede poner en coincidencia con el sistema Ox'y'z'. A tal movimiento se le llama una *Rotación de ejes*.

Obsérvese que si un sistema de ejes derecho e izquierdo es imposible ponerlos en coincidencia por una rotación.

Es conveniente denominar a Oxyz como ejes originales de ox'y'z', como ejes nuevos. (ver figura No.9)

Sean  $L_{11}$ ,  $L_{12}$  y  $L_{13}$  los cosenos directores de  $Ox'$  con respecto a los ejes Oxyz. Además indique por  $L_{21}$ ,  $L_{22}$ ,  $L_{23}$ , y  $L_{31}$ ,  $L_{32}$ ,  $L_{33}$  los cosenos directores de  $Oy'$  y  $Oz'$ . Esto se resume adecuadamente en la disposición siguiente.

(m1)	O	x	y	z
	x'	$L_{11}$	$L_{12}$	$L_{13}$
	y'	$L_{21}$	$L_{22}$	$L_{23}$
	z'	$L_{31}$	$L_{32}$	$L_{33}$

<sup>12</sup>Cuando se realiza una rotación de los ejes, los vectores deben conservar su módulo, dirección y sentido. Por ello, deben transformarse de acuerdo con la ley de  $X_j' = C_{ij}X_i$  eje. Alfa Con Estándares 10.Editorial Norma. Capítulo 5 Geometría Analítica Páginas 172-199

Éste es un arreglo, los cosenos directores de  $Ox'$  con respecto a los ejes  $Oxyz$ , aparecen en la primera fila, los de  $Oy'$  en la segunda y los de  $Oz'$  en la tercera. Además, si se leen hacia abajo las tres columnas a su vez, es fácil ver que se obtienen los cosenos directores de los ejes  $Ox$ ,  $Oy$ ,  $Oz$  con respecto a los ejes  $Ox'y'z'$ . La disposición de cosenos directores se llama *matriz de transformación*<sup>13</sup>.

Puesto que los ejes  $Ox'$ ,  $Oy'$ ,  $Oz'$  son perpendiculares entre sí, entonces:

$$\begin{aligned} L_{11}L_{21} + L_{12}L_{22} + L_{13}L_{23} &= 0 \\ \text{(ec14)} \quad L_{21}L_{31} + L_{22}L_{32} + L_{23}L_{33} &= 0 \\ L_{31}L_{11} + L_{32}L_{12} + L_{33}L_{13} &= 0 \end{aligned}$$

Se sabe que la suma de cuadrados de los cosenos directores son iguales a la unidad, y así se tiene:

$$\begin{aligned} L_{11}^2 + L_{12}^2 + L_{13}^2 &= 1 \\ \text{(ec15)} \quad L_{21}^2 + L_{22}^2 + L_{23}^2 &= 1 \\ L_{31}^2 + L_{32}^2 + L_{33}^2 &= 1 \end{aligned}$$

A estas seis ecuaciones se les llama *Relaciones de Ortonormalidad*. Obsérvese cómo se establecen en la disposición (m1). Puesto que los elementos de las columnas forman los cosenos directores de los ejes  $Ox$ ,  $Oy$ ,  $Oz$  con respecto a los ejes  $Ox'y'z'$ , se deduce por argumentos similares que:

$$\begin{aligned} L_{11}L_{21} + L_{12}L_{22} + L_{13}L_{23} &= 0 \\ \text{(ec16)} \quad L_{21}L_{31} + L_{22}L_{32} + L_{23}L_{33} &= 0 \\ L_{31}L_{11} + L_{32}L_{12} + L_{33}L_{13} &= 0 \end{aligned} \quad y \quad \begin{aligned} L_{11}^2 + L_{21}^2 + L_{31}^2 &= 1 \\ L_{12}^2 + L_{22}^2 + L_{32}^2 &= 1 \\ L_{13}^2 + L_{23}^2 + L_{33}^2 &= 1 \end{aligned} \quad \text{(ec17)}$$

<sup>13</sup>Las formulas de escalado, rotación y traslación corresponden a transformaciones lineales las cuales se llevan a cabo por medio de aplicaciones con matrices. Corcobado Cartes Juan Luis. Matemática 1. Editorial Cáceres páginas 135-137.



La matriz de transformación satisfacen una condición más, la cual aparece por el hecho de que tanto el sistema Oxyz, como el sistema Ox'y'z' son derechos. Considere de ésta forma el determinante:

$$T = \begin{vmatrix} L_{11} & L_{12} & L_{13} \\ L_{21} & L_{22} & L_{23} \\ L_{31} & L_{32} & L_{33} \end{vmatrix}$$

Si se indica el determinante transpuesto de T por T', se tiene: (para mejor información sobre este procedimiento el lector puede hacer referencia a la Unidad de Matrices de ésta monografía)

$$T = TT' = \begin{vmatrix} L_{11} & L_{12} & L_{13} \\ L_{21} & L_{22} & L_{23} \\ L_{31} & L_{32} & L_{33} \end{vmatrix} \times \begin{vmatrix} L_{11} & L_{21} & L_{31} \\ L_{12} & L_{22} & L_{32} \\ L_{13} & L_{23} & L_{33} \end{vmatrix}$$

De ahí que al multiplicar los dos determinantes y usar las condiciones de **ortonormalidad**.

$$(ec18) \quad T^2 = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} = 1.$$

Si los ejes se hacen girar de coincidencia, los cosenos directores  $L_{ij}$  variarán durante la rotación de manera continua, y como el determinante T es la suma de los productos de los cosenos directores, su valor también variará continuamente. Pero en todos los pasos de rotación  $T=1$  ó  $-1$ , y así sucesivamente, T debe tomar el valor de 1 ó  $-1$  a través de la rotación para que no ocurra discontinuidad en el valor. Puesto que  $T=1$  cuando los dos sistemas coordenados coinciden, se deduce que en todas las posiciones

$$(ec19) \quad T = \begin{vmatrix} L_{11} & L_{12} & L_{13} \\ L_{21} & L_{22} & L_{23} \\ L_{31} & L_{32} & L_{33} \end{vmatrix} = 1$$

Esta es la condición adicional que debe satisfacer la matriz de transformación.

Se ha visto que, si las componentes de la disposición (m1) son los cosenos directores de los ejes nuevos con respecto a los ejes originales, las condiciones (ec14), (ec15) y (ec17) necesariamente se satisfacen. Estas condiciones son también suficientes para la disposición que represente una rotación de los ejes derechos Oxyz. En primer lugar, si se satisfacen las ecuaciones (ec14), los ejes Ox', oy', Oz' son perpendiculares entre sí, segundo, si se satisfacen las ecuaciones (ec15), las filas en la matriz de transformación representan a los cosenos directores de Ox', Oy', Oz'; y finalmente si se satisface (ec19), el sistema Ox', Oy', Oz' es derecho.

*Transformación de Coordenadas.* Sea un punto P de coordenadas (x,y,z) y (x',y',z') con respecto a los ejes Oxyz y Ox'y'z' respectivamente. Las coordenadas x',y',z' de P son las proyecciones de OP sobre Ox', Oy', Oz'. De ahí que, si se emplea (ec13) para calcularlas, se obtiene:

$$(ec20) \quad \begin{aligned} x' &= L_{11}^2 x + L_{12}^2 y + L_{13}^2 z = 1 \\ y' &= L_{21}^2 x + L_{22}^2 y + L_{23}^2 z = 1 \\ z' &= L_{31}^2 x + L_{32}^2 y + L_{33}^2 z = 1 \end{aligned}$$

Estas ecuaciones demuestran cómo las coordenadas de P se transforman de acuerdo a la rotación de los ejes; note también que estas expresiones son establecidas por la disposición (m1).

Por supuesto, se podría considerar a los ejes Ox'y'z' como el sistema original y a los ejes Oxyz como el sistema nuevo y determinar las coordenadas (x,y,z) en términos de (x',y'z'). Recuerde que los elementos de las columnas en (m1) son los cosenos directores de los ejes x,y,z con respecto a los ejes Ox'y'z' y empleando (ec13) se deduce que:

$$(ec21) \quad \begin{aligned} x &= L_{11}x' + L_{12}y' + L_{13}z' \\ y &= L_{21}x' + L_{22}y' + L_{23}z' \\ z &= L_{31}x' + L_{32}y' + L_{33}z' \end{aligned}$$

### 2.2.1 Trazado de puntos.

Todo aquello que se dibuja en un sistema gráfico está formado por puntos llamados píxeles<sup>14</sup>; dichos puntos se representan mediante coordenadas, estas son un conjunto de números que corresponden a cada uno de los ejes, por ejemplo: un punto 2D se representa mediante dos coordenadas (x,y), así un punto 3D se representa con tres coordenadas (x,y,z).

El trazo de un punto se realiza al convertir una posición particular de coordenadas en el dispositivo de salida en uso (en este caso el monitor del computador). Un sistema (vectorial) de rastreo aleatorio almacena instrucciones para el trazo de puntos en la lista de despliegue y los valores de las coordenadas de estas instrucciones se convierten en las posiciones de la pantalla que se deben trazar.

#### Puntos en OPENGGL:

OpenGL<sup>15</sup> tiene únicamente unas pocas primitivas geométricas: puntos, líneas, polígonos. Todas ellas se describen en términos de sus respectivos vértices. Un vértice está caracterizado por 2 o 3 puntos flotantes, las coordenadas cartesianas del vértice son (x, y) en 2D y (x, y, z) en 3D.

Como en OpenGL todos los objetos geométricos son finalmente descritos como un conjunto ordenado de vértices, existe una familia de rutinas donde su sintaxis es:

---

<sup>14</sup> Wong Benjamín. Dibujo Técnico Digital. Editorial Gustavo GILI SA. Capítulo Puntos y trazos páginas 25-27

<sup>15</sup> OpenGL es sin lugar a dudas la API que prevalece en la industria para desarrollar aplicaciones gráficas 2D y 3D. Se le puede considerar el sucesor a la formidable IRIS GL-library de Silicon Graphics que hizo tan popular las estaciones de trabajo SGI como plataforma predilecta para desarrollo científico, de ingeniería y de efectos especiales. With Richard S. OPENGGL. Capítulo 2 Uso de OpenGL. Páginas 215-218

```
void glVertex{234}{sifd}[v](TYPE coords);
```

Las llaves indican parte del nombre de la rutina, las rutinas pueden tomar 2, 3 o 4 parámetros de tipo short, long, float o double. Opcionalmente, estos parámetros se pueden proporcionar en forma de vector, en este caso deberemos usar la rutinas del tipo v.

Aquí hay algunos ejemplos:

```
void glVertex2s(1, 3);  
void glVertex2i(23L, 43L);  
void glVertex3f(1.0F, 1.0F, 5.87220972F);  
float vector[3];  
void glVertex3fv(vector);
```

El contexto se declara mediante el par de rutinas glBegin(GLenum mode) y glEnd(), toda sentencia glVertex\* ejecutada entre estas dos se interpreta según el valor de mode, por ejemplo:

```
glBegin(GL_POINTS);  
    glVertex2f(0.0, 0.0);  
    glVertex2f(1.0, 0.0);  
    glVertex2f(0.0, 1.0);  
    glVertex2f(1.0, 1.0);  
    glVertex2f(0.5, 0.5);  
glEnd();
```

Esta rutina dibuja 5 puntos en 2D con las coordenadas especificadas. GL\_POINTS, la cual es una de las etiquetas definidas en el fichero cabecera de OpenGL <GL/gl.h>.

## 2.2.2 Trazado de líneas.

La ecuación de una recta puede enunciarse en la forma:  $y = mx + b$ , con  $m$  como pendiente de la recta y  $b$  como intercepción  $y$ . Dado que los dos extremos de un segmento rectilíneo se especifican como  $(x_1, y_1)$  y  $(x_2, y_2)$ . Véase figura No.10

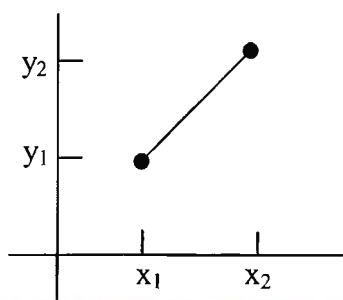


Figura No.10: Segmento de línea, especificado por extremos coordenadas  $(x_1, y_1)$  y  $(x_2, y_2)$

Puede determinar valores de la pendiente  $m$  y de la intersección  $y$ ,  $b$ , con los siguientes cálculos<sup>16</sup>.

$$m = \frac{y_2 - y_1}{x_2 - x_1} \quad b = y_1 - mx_1$$

Los algoritmos<sup>17</sup> para desplegar líneas rectas se basan en la ecuación de la recta y en éstos cálculos.

Para cualquier intervalo  $\Delta x$  de  $x$  a lo largo de una recta, podemos calcular el intervalo  $\Delta y$  de  $y$  correspondiente

a la ecuación  $m = \frac{y_2 - y_1}{x_2 - x_1}$

Esta ecuación forma la base para determinar tensiones de reflexión en dispositivos analógicos. La variación en la tensión de deflexión vertical se hace proporcional al valor de  $\Delta y$  calculado a partir de la ecuación  $\Delta y = m\Delta x$ . Estas deflexiones se usan después para generar una línea con pendiente  $m$  entre los extremos que se especifican.

En base a éste principio, el trazo de líneas se efectúa mediante el cálculo de posiciones intermedias a lo largo de la trayectoria de la línea entre dos posiciones extremas específicas. Un dispositivo de salida se dirige para llenar estas posiciones entre los extremos.

<sup>16</sup> Hearn Donald Graficado por Computadora. Editorial Prentice Hall. Capítulo 3. páginas 59-63

<sup>17</sup> Un algoritmo es un conjunto finito de instrucciones o pasos que sirven para ejecutar una tarea o resolver un problema. La palabra algoritmo deriva del nombre del matemático árabe Al Juarismi, que vivió entre los siglos VIII y IX.

Los dispositivos digitales despliegan un segmento de línea recta al trazar puntos discretos entre los dos extremos.

Las posiciones de coordenadas discretas a lo largo de la trayectoria de la línea se calculan a partir de la ecuación de la recta.

Para un despliegue de video de rastreo, el color (intensidad) de la línea se carga entonces en el búfer de estructura en las coordenadas de píxel correspondientes.

A continuación, al leer el búfer de estructura, el controlador de video “traza” los píxeles de la pantalla. Las posiciones en la pantalla se expresan como valores enteros, de modo que las posiciones trazadas sólo puedan aproximarse a las posiciones de línea reales entre dos extremos específicos.

Por ejemplo, una posición de línea calculada de (10.48, 20.51) se convertiría a la posición de píxel (10, 21). Este redondeo de valores de coordenadas a enteros provoca que las líneas se desplieguen con una apariencia de escalones, como se representa en la figura No.11.

La forma de escalón característica de las líneas de rastreo se percibe, de manera particular, en sistemas con baja resolución, y podemos mejorar su apariencia en cierto grado al desplegarlas en sistemas de alta resolución.



Figura No.11: representación de línea en forma de escalón

Las técnicas más efectivas se basan en el ajuste de las intensidades de píxel a lo largo de las trayectorias de línea.

Para los algoritmos relativos a dispositivo de gráficas de rastreo, las posiciones de objetos se especifican directamente en coordenadas de dispositivo enteras.

Por el momento, supondremos que se hace referencia a las posiciones de píxel de acuerdo con el número de línea de rastreo y el número de columna (posición de píxel a lo largo de una línea de rastreo).

En la figura No.12. se ilustra este esquema de dirección Las líneas de rastreo se numeran de manera consecutiva desde 0, comenzando en la parte inferior de la pantalla; y las columnas de píxel se numeran a partir de 0, de izquierda a derecha a lo largo de cada línea de rastreo.

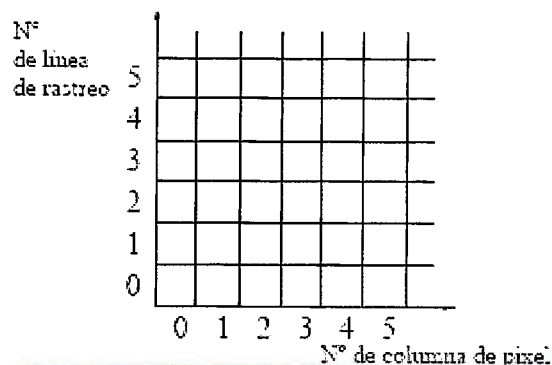


Figura No.12: Posiciones de píxel a las que se hace referencia por el número de línea de rastreo y el n° de columna

Para cargar un valor de intensidad en el búfer de estructura en una posición correspondiente a la columna x a lo largo de la línea de rastreo y, supondremos que tenemos un procedimiento de bajo nivel disponible.

(En OpenGL "glBegin(GL\_POINTS)").

### Algoritmo para Trazado de Líneas<sup>18</sup>

En sistemas de rastreo, las líneas se trazan con píxeles y los tamaños de paso en las direcciones horizontal y vertical se limitan por las separaciones de píxel. Es decir, debemos "efectuar un muestreo" de una línea en posiciones discretas y con determinar el píxel más cercano a la línea en cada posición sometida al muestreo.

Este proceso de conversión de rastreo para líneas rectas se ilustra en la figura 4, para una línea cercana a la horizontal con posiciones de muestra discretas a lo largo del eje de las x.

<sup>18</sup> Wong Benjamín. Dibujo Técnico Digital. Capítulo 3 Líneas Rectas páginas 35-38

En sistemas de rastreo, las líneas se trazan con píxeles y los tamaños de paso en las direcciones horizontal y vertical se limitan por las separaciones de píxel.

Es decir, debemos “efectuar un muestreo” de una línea en posiciones discretas y con determinar el píxel más cercano a la línea en cada posición sometida al muestreo.

Este proceso de conversión de rastreo para líneas rectas se ilustra en la figura No. 13, para una línea cercana a la horizontal con posiciones de muestra discretas a lo largo del eje de las x.

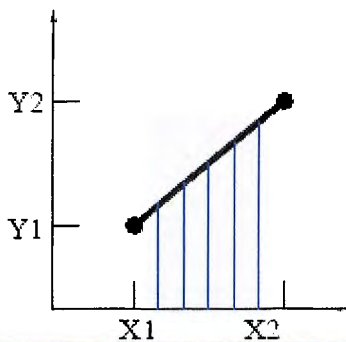


Figura No.13.Segmento de línea recta con cinco posiciones de muestreo a lo largo del eje x entre x1, y1



### 2.2.2.1 Método algebraico.

La ecuación de intersección de la pendiente cartesiana de una línea recta es:

(ec1)  $y = m x + b$  Donde  $m$  representa la pendiente de la línea y  $b$  la intersección de  $y$ .

Dado que los dos extremos de un segmento de línea se especifica en las posiciones  $(x_1, y_1)$  y  $(x_2, y_2)$ , se puede determinar valores para la pendiente y la intersección de  $y$  en  $b$  con los siguientes cálculos:

(ec22)  $m = (y_2 - y_1) / (x_2 - x_1)$

(ec23)  $b = y_1 - m x_1$



- Si  $|m| < 1$ , la línea es mas horizontal que vertical.  
Según tiende a cero  $m$ , la línea tiende a ser horizontal.
- Si  $|m| > 1$ , la línea es mas vertical que horizontal.  
Según tiende a infinito  $m$ , la línea tiende a ser vertical.

Para cualquier  $x$  dentro del intervalo  $\Delta x$  a lo largo de una línea, se puede calcular el intervalo correspondiente a  $\Delta y$  de  $y$  a partir de la ecuación (ec2) como

(ec24)  $\Delta y = m \Delta x$

De modo similar se puede obtener el intervalo  $\Delta x$  de  $x$  correspondiente a una  $\Delta y$  específica como:

(ec25)  $\Delta x = (\Delta y / m)$ .

El algoritmo básico o método algebraico, sería utilizar la ecuación (ec22) para calcular  $y$  en termino de  $x$  (o viceversa).

Esto sería bastante lento ya que requeriría siempre una multiplicación más una suma:

$$y = m x + b$$

### 2.2.2.2 Analizador diferencial digital (DDA).

El analizador diferenciador digital<sup>19</sup> (DDA - Digital Differential Analyzer) es un algoritmo de conversión de rastreo<sup>20</sup> que se basa en el cálculo ya sea de  $\Delta y$  o  $\Delta x$  por medio de las ecuaciones (ec24) o (ec25).

El Algoritmo DDA es un algoritmo de línea de conversión de rastreo que se basa en el cálculo ya sea en el incremento de X o en el incremento de Y. La finalidad de este algoritmo es determinar los valores enteros correspondientes más próximos a la trayectoria de la línea para la otra coordenada.

Se efectúa un muestreo de la línea en intervalos unitarios en una coordenada y se determina los valores enteros correspondientes mas próximos a la trayectoria de la línea para la otra coordenada.

Tomemos una línea con pendiente positiva, si la pendiente  $|m| \leq 1$ , se hace el muestreo en x en intervalos unitarios  $\Delta x = 1$  y  $\Delta y = m$  dado que  $m = \Delta y / \Delta x$  y se calcula cada valor sucesivo de y como:

$$(ec26) \ y_{k+1} = y_k + m$$

#### Código en OpenGL para DDA

```
#define ROUND(a) (int)(a+0.5)
void lineDDA (int xa, int ya, int xb, int yb)
{
    int dx = xb - xa, dy = yb - ya, steps, k;
    float xIncrement, yIncrement, x = xa, y = ya;
    if (abs (dx) > abs (dy)) steps = abs (dx);
    else steps = abs (dy);
    xIncrement = dx / (float) steps;
    yIncrement = dy / (float) steps;
    drawDot(ROUND(x),ROUND(y)); for (k=0; k<steps;
    k++) {
        x += xIncrement;
        y += yIncrement;
```

<sup>19</sup> Hearn Donal. Gráficas por Computadora. Editorial Prentice Hall Capítulo 3 páginas 61-63

<sup>20</sup> Rastreo: término que en gráficas por computadora es sinónimo de Trazado.

```
drawDot (ROUND(x),ROUND(y));  
}  
}
```

Este algoritmo se resume en el procedimiento siguiente, que acepta como entrada las dos posiciones de píxel de los extremos. Las diferencias horizontal y vertical entre las posiciones de los extremos se asignan a los parámetros  $dx$  y  $dy$ .

La diferencia con la mayor magnitud determina el valor del parámetro  $steps$ . Al iniciar con la posición de píxel  $(x_a, y_a)$ , determinamos la compensación necesaria para generar la posición del píxel siguiente a lo largo de la trayectoria de la línea.

Realizamos el ciclo de este proceso  $steps$  veces. Si la magnitud de  $dx$  es mayor que la de  $dy$  y  $x_a$  es menor que  $x_b$ , los valores de los incrementos en las direcciones de  $x$  e  $y$  son  $1$  y  $m$ , respectivamente. Si la variación más alta es en la dirección de  $x$ , pero  $x_a$  es mayor que  $x_b$ , entonces los decrementos  $-1$  y  $-m$  sirven para generar cada nuevo punto en la línea. De otra manera, utilizamos un incremento (o decremento) unitario en la dirección de  $y$  y un incremento (o decremento) de  $1/m$  en la dirección de  $x$ .

El algoritmo DDA es un método para calcular posiciones de píxel, que es más rápido que la aplicación directa de la ecuación 1, al utilizar características de rastreo, de manera que se aplican incrementos adecuados en la dirección de  $x$  e  $y$  para pasar a las posiciones de píxel a lo largo de la trayectoria de la línea. Sin embargo, para segmentos de línea largos, la acumulación de errores de redondeo en adiciones sucesivas del incremento de punto flotante pueden provocar que las posiciones de píxel calculadas se desvíen de la trayectoria real de la línea.

Se puede mejorar el desempeño del algoritmo DDA al separar los incrementos  $m$  y  $1/m$  en partes enteras y fraccionarias, de forma tal que todos los cálculos se reduzcan a operaciones de enteros.

### 2.2.2.3 Algoritmo de Bresenham<sup>21</sup>.

Es un algoritmo de línea más efectivo para determinar las posiciones del píxel, éste encuentra las coordenadas enteras más próximas a la trayectoria real de la recta utilizando solamente aritmética entera. Convierte mediante rastreo las líneas al utilizar sólo cálculos incrementales con enteros que se pueden adaptar para desplegar circunferencias y otras curvas.

Las figuras 14 a) y b), ilustran secciones de una pantalla de despliegue donde se deben trazar segmentos de línea recta.

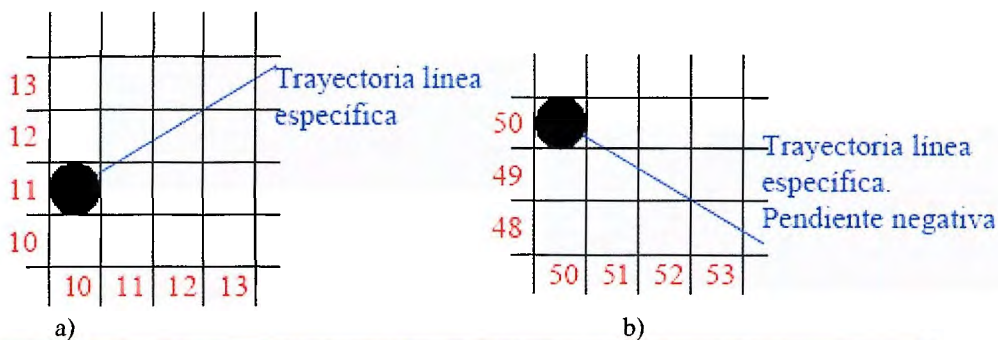


Figura No.14: Sección de una pantalla de despliegue donde se traza una línea recta. Pendiente Positiva y Negativa.

Los ejes verticales muestran las posiciones de línea de rastreo y los ejes horizontales identifican columnas de píxel. Al realizar un muestreo de x en intervalos unitarios en estos ejemplos, necesitamos decidir cuál de dos posibles posiciones de píxel está más próxima a la trayectoria de la línea en cada paso del muestreo.

Al iniciar desde el extremo izquierdo que se presenta en la figura 11 a), debemos determinar en la siguiente posición del muestreo si trazamos el píxel en la posición (11,11) o en (11,12). De manera similar, en la figura 11 b) aparece una trayectoria

<sup>21</sup> Donal Hearn Gráficas por Computadora. Editorial Prentice Hall. Capítulo 3 Primitivas de Salida, páginas 63-64.

de línea con pendiente negativa que inicia a partir del extremo izquierdo en la posición de píxel (50, 50). En este caso, ¿seleccionamos la siguiente posición de píxel como (51, 50) o como (51, 49)? El algoritmo de línea de Bresenham responde estas preguntas al probar el signo de un parámetro entero, cuyo valor es proporcional a la diferencia entre las separaciones de las dos posiciones de píxel de la trayectoria real de la línea.

Para ilustrar el planteamiento de Bresenham, primero consideramos el proceso de conversión de rastreo para líneas con pendiente positiva menor que 1. Las posiciones de píxel a lo largo de la trayectoria de una línea se determinan entonces al efectuar un muestreo de x en intervalos unitarios.

Si iniciamos desde el extremo izquierdo ( $x_0, y_0$ ) de una línea determinada, pasamos a cada columna sucesiva (posición de x) y trazamos el píxel cuyo valor de y de la línea de rastreo se aproxima más a la trayectoria de la línea de rastreo.

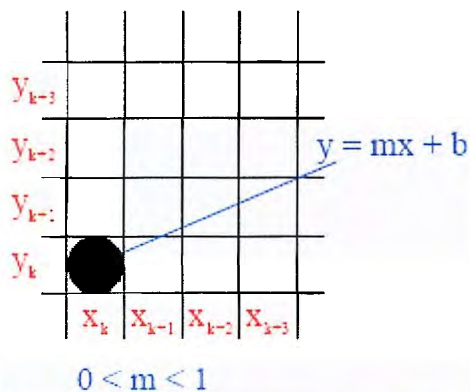


Figura No.15: Sección de una cuadrícula de pantalla que presentan un píxel en la columna  $X_k$

La figura No.15 demuestra el k-ésimo paso en este proceso. Si suponemos que determinamos que se debe desplegar el píxel en  $(x_k, y_k)$ , a continuación necesitamos decidir qué píxel se va a trazar en la columna  $x_{k+1}$ .

Nuestras alternativas son los píxeles en las posiciones  $(x_{k+1}, y_k)$  y  $(x_{k+1}, y_{k+1})$ .

Al realizar el muestreo en la posición  $X_{k+1}$ , designamos las separaciones de píxel verticales de la trayectoria de la línea matemática como  $d_1$  y  $d_2$  (ver la figura 8).

La coordenada de y en la línea matemática en la posición de la columna de píxel  $X_k + 1$  se calcula como:

$$(ec27) \quad y = m(x_k + 1) + b$$

entonces

$$(ec28) \quad d_1 = y - y_k = m(x_k + 1) + b - y_k$$

y

$$(ec29) \quad d_2 = (y_k + 1) - y = y_k + 1 - m(x_k + 1) + b$$

La diferencia entre éstas dos separaciones es:

$$(ec30) \quad d_1 - d_2 = 2m(x_k + 1) - 2y_k + 2b - 1$$

Ahora definiremos un parámetro que ofrece una medida de las distancias relativas de dos pixeles de la posición actual sobre una recta dada. De la sustitución de  $m = \Delta y / \Delta x$  se puede reescribir la ecuación (ec30) de manera que comprenda solamente aritmética entera:

$$(ec31) \quad P_k = \Delta x(d_1 - d_2)$$

$$(ec32) \quad P_k = 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c$$

El signo de  $P_k$  es el mismo que el de  $d_1 - d_2$ , puesto que  $\Delta x > 0$  para éste ejemplo. El parámetro  $c$  es constante y tiene el valor  $2\Delta y + \Delta x(2b-1)$ , que es el independiente del píxel si  $y_k$  está más cerca de la trayectoria de la línea que el píxel  $y^k + 1$  (es decir,  $d_1 < d_2$ ), entonces el parámetro de decisión  $P_k$  es negativo.

En ese caso, trazamos el píxel inferior; de otro modo, trazamos el píxel superior.

Los cambios de coordenadas a lo largo de la línea ocurren en pasos unitarios ya sea en la dirección de  $x$  o en la de  $y$ .

## Algoritmo del Punto Medio para Circunferencias<sup>27</sup>

Al igual que en el algoritmo de trazo de líneas, se efectúa un muestreo en intervalos unitarios y se determina la posición del píxel más cercano a la trayectoria específica de la circunferencia en cada paso.

Para un radio  $r$  determinado y una posición central en la pantalla  $(x_c, y_c)$ , podemos establecer primero nuestro algoritmo para calcular las posiciones de píxel alrededor de una trayectoria circular centrada en el origen de coordenadas  $(0,0)$ .

Así, cada posición calculada  $(x, y)$  se mueve a su posición propia en la pantalla al sumar  $X_c + X$  e  $y_c + Y$ .

A lo largo de la sección circular de  $X = 0$  a  $X = Y$  en el primer cuadrante, la pendiente de la curva varía entre 0 y -1. Por tanto, podemos tomar pasos unitarios en la dirección positiva de  $X$  en este octante y utilizar un parámetro de decisión para determinar cuál de las dos posiciones posibles de  $y$  está más próxima a la trayectoria de la circunferencia en cada paso.

Las posiciones de los otros siete octantes se obtienen entonces por simetría.

Para aplicar el método del punto medio, definimos una función de circunferencia como:

$$F(x,y) = x^2 + y^2 - r^2$$

Cualquier punto  $(x, y)$  en la frontera de la circunferencia con radio  $r$  satisface la ecuación  $F(x,y) = 0$ .

Si el punto está en el interior de la circunferencia, la función de circunferencia es negativa;

Y si está en su exterior, es positiva.

---

<sup>27</sup> Raghavachary Saty. Rendering for Beginners. Editorial Focal Press. Capítulo 4. Geometric Primitives. Páginas 76-88

Por tanto, es posible obtener los valores de parámetros de decisión sucesivos al utilizar cálculos con incrementos de enteros. En el paso  $k + 1$ , el parámetro de decisión se evalúa con base en la ecuación (ec32) como

$$(ec33) \quad P_{k+1} = 2\Delta y \cdot x_{k+1} - 2\Delta x \cdot y_{k+1} + c$$

Restando la ecuación (ec32), se tiene:

$$(ec34) \quad P_{k+1} - P_k = 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k)$$

Pero  $x_{k+1} = x_k + 1$ , de modo que:

$$(ec35) \quad P_{k+1} = P_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k)$$

Esta ecuación nos da de manera calculada el valor de cada parámetro sucesivo a partir del anterior. El primer parámetro,  $P_1$ , se obtiene de la evaluación de la ecuación (ec32) con  $(x_1, y_1)$  como punto extremo inicial y  $m = \Delta y / \Delta x$ :

$$P_1 = 2\Delta y - \Delta x$$

Las etapas del algoritmo de Bresenham se resumen de la siguiente manera:

Para una recta con pendiente positiva menor que 1. Como las constantes  $2\Delta y$ ,  $\Delta x$  y  $2(\Delta y - \Delta x)$ , necesitan ser evaluadas y almacenadas solo una vez, la aritmética comprende únicamente la adición y la ilustración de enteros

### **Algoritmo de Bresenham para el trazo de líneas para $|m| < 1$ .<sup>22</sup>**

1. Se capturan los dos extremos de la línea y se almacena el extremo izquierdo en  $(x_0, y_0)$ .
2. Se carga  $(x_0, y_0)$  en el búfer de estructura; es decir, se traza el primer punto.
3. Se calculan las constantes  $\Delta x$ ,  $\Delta y$ ,  $2\Delta y$ ,  $2\Delta y - 2\Delta x$  y se obtiene el valor inicial para el parámetro de decisión como  $P_0 = 2\Delta y - \Delta x$ .

---

<sup>22</sup> D.Foley James. Computer Graphics: Principles and Practice. Editorial Wesley. Capítulo 1. Large scale Computations. Páginas 66-78



4. En cada  $x_k$  a lo largo de la línea, que inicia en  $k=0$ , se efectúa la prueba siguiente: si  $p_k < 0$ , el siguiente punto que debe trazarse es  $(x_{k+1}, y_k)$  y  

$$P_{k+1} = P_k + 2\Delta y$$
De otro modo, el siguiente punto que se debe trazar es  $(x_{k+1}, y_{k+1})$  y  

$$P_{k+1} = P_k + 2\Delta y - 2\Delta x$$
5. Se repite el paso 4  $D_x$  veces.

### Código en OpenGL para Bresenham

```
#include <GL/glut.h>
// drawline
// dibuja una línea usando el algoritmo de Bresenham
// desde (ax,ay) hasta (bx,by)
// Asume:
// (1)  $bx \geq ax$ 
// (2) Pendiente de la línea en  $[0,1]$ .

void drawline(const int ax, const int ay, const int bx, const int by)
{
    const int w = bx - ax; // ancho (horizontal)
    const int h = by - ay; // alto (vertical)

    const int rightstep = 2 * h; // "f" aumenta para mover rt
    const int diagstep = 2 * (h - w); // "f" aumenta para mover rt y arriba

    int x; // x coord de pt actual
    int y = ay; // y coord de pt actual
    int f = 2 * h - w; // "f": valor de prueba de Bresenham

    glBegin(GL_POINTS);

    for (x = ax; x <= bx; ++x) // Ciclo principal de Bresenham
    {
        glVertex2i(x, y); // Plot del punto
        if (f < 0)
        {
            f += rightstep; // a la derecha
        }
        else
        {
            ++y;
            f += diagstep; // derecha y arriba
        }
    }
}
```

```

    }

    glEnd();
}

// display
// rutina principal de dibujo
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0); // R
    drawline(5,15, 90,95); // Dibuja
    glFlush();
}

// init
// Inicializa estados de GL, stacks...

void init()
{
    glClearColor (0.0, 0.0, 0.0, 0.0);

    glPointSize(5.0); // Dibuja Puntos "grandes" (cuadros 5x5)

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0, 100, 0, 100, -1.0, 1.0);
    // La ventana va de 0 a 100 en ambas coords.
}

int main(int argc, char** argv)
{
    // Inicializa OpenGL/GLUT
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

    // Ventana
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(50, 50);
    glutCreateWindow("Algoritmo de Bresenham (líneas)");

    // Estados de GL y callback de registros
    init();
    glutDisplayFunc(display);

    glutMainLoop();
    return 0;
}

```

El algoritmo de Bresenham se generaliza para líneas con una pendiente arbitraria al considerar la simetría entre los diversos octantes y cuadrantes del plano de  $xy$ .

Para una línea con una pendiente positiva mayor que 1, intercambiamos las direcciones de "x" y "y". Es decir, se pasa a lo largo de la dirección de y en pasos unitarios y calculamos los valores sucesivos de x que se aproximen más a la trayectoria de la línea. Asimismo, podemos revisar el programa para trazar píxeles iniciando desde cualquier extremo. Si la posición inicial para una línea con pendiente positiva es el extremo derecho, tanto x como y disminuyen conforme pasamos de derecha a izquierdas. Con el fin de asegurarnos de que los mismos píxeles se tracen sin que tenga importancia el extremo en que se inicie, siempre seleccionando superior (o inferior) de los dos candidatos toda vez que las dos separaciones de la trayectoria de la línea son iguales ( $d_1 = d_2$ ).

En el caso de pendientes negativas, los procedimientos son similares excepto que ahora, una coordenada decrece conforme la otra aumenta. Por último, es posible manejar los casos especiales por separado. Las líneas horizontales ( $\Delta y = 0$ ), las líneas verticales ( $\Delta x = 0$ ) y las diagonales con  $|\Delta x| = |\Delta y|$  se pueden cargar en forma directa en el búfer de estructura sin procesarlas mediante el algoritmo para el trazo de líneas.

Esta función presenta una desventaja, dado que cada bit en la máscara corresponde a una iteración y no a una distancia unitaria a lo largo de la recta, la longitud de los patrones variará con la inclinación de la recta.

Para dibujos en ingeniería, esta variación es inaceptable, por lo que las discontinuidades de la recta han de ser tratadas y convertidas como segmentos de línea independientes.

En último caso, las líneas gruesas son tratadas como pequeños rectángulos equiespaciados cuyos vértices son calculados exactamente en función del estilo de línea seleccionado.

## GROSOR DE LÍNEA

La implantación de las opciones de anchura de las líneas depende del tipo de dispositivo de salida que se utilice. Una línea ancha en un monitor de video podría trazarse como líneas paralelas adyacentes, mientras que una graficadora Palm podría requerir cambio de pluma de escritura.

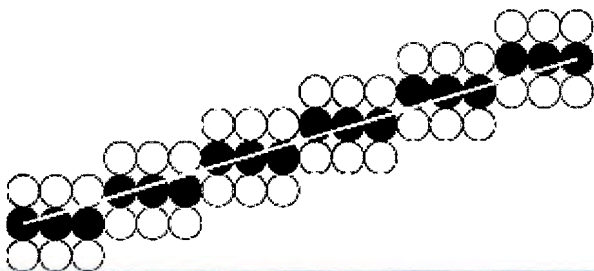


Figura No.17: ancho de línea mediante replicación en columna.

El método más sencillo para aumentar el ancho o grosor de la línea consiste en activar los píxeles que se encuentran en torno a los de la recta resultante de la rasterización (ver figura No.17).

Si la inclinación de la recta  $m \in [-1,1]$ , lo que se hará es añadir más píxeles por

columna a ambos lados de la recta.

Si la inclinación de la recta estuviera fuera del intervalo antes mencionado, se añadirían más píxeles por fila.

Este método es muy rápido, pero presenta la desventaja de que el principio y final de las rectas es siempre un corte horizontal o vertical (dependiendo de la

#### 2.2.2.4 Estilos de líneas (continua, punteada...)

Los atributos del estilo de la línea determina la forma en que se desplegará una línea por medio de una rutina de trazo de líneas. Los atributos comunes de una línea son su tipo, su anchura y su color. Las rutinas para el trazo de líneas deben estructurarse para producir líneas con las características especificadas.

##### TIPOS DE LÍNEAS<sup>23</sup>

Diferentes instancias del atributo de tipo de línea puede producir líneas continuas, punteadas y con líneas. (ver figura No.16) Para permitir asignar este tipo de atributo a las rectas que se estén renderizando, es necesario modificar los algoritmos de forma que se permita definir la longitud y el espaciado de las secciones sólidas de la línea.

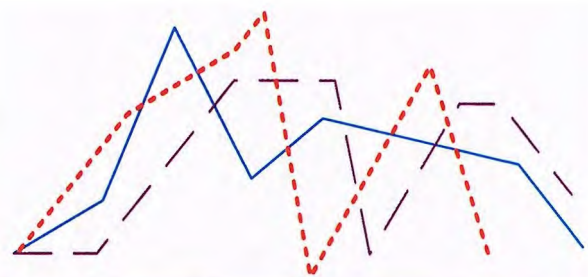


Figura No.16: Trazo de tres conjuntos de datos con tres diferentes estilos de líneas

Una forma de poder especificar el tipo de línea es mediante la utilización de patrones almacenados como cadenas de bits, por ejemplo, patrones de 16-bits, que permitirían establecerlos de manera que se repetirían cada 16 píxeles.

En los algoritmos puede definirse una función de cambio de píxel llamada *dibuja\_pixel(x,y)*, en donde los parámetros de entrada son los puntos cartesianos.

<sup>23</sup> Referencia 1: Tajadura Zapirain J.A / J.López Fernández. Autocad 2000 Avanzado. Editorial Mc.Graw Hill  
Capítulo 3 Creación y edición de Dibujos páginas 143-192  
Referencia 2: D.Foley James. Computer Graphics: Principles and Practice. Editorial Wesley. Capítulo 3. Points and Lines, páginas 133-145



inclinación) y, además, el ancho de las rectas así generadas depende de la inclinación de las mismas.

En síntesis si la pendiente de la recta es menor que 1, para cada posición de x se pinta una sección vertical de píxeles, tantos como ancho de línea queramos, por igual a cada lado.

Si la pendiente es mayor que 1, se usan secciones horizontales.

Hay que tener en cuenta que el ancho de las líneas horizontales y verticales será  $\sqrt{2}$  veces más grueso que las diagonales.

### Problema en Las terminaciones

Se ha de tener una especial atención a los extremos de las líneas a la hora de generar líneas con grosor. Existe un problema en los bordes finales de las líneas: son siempre horizontales o verticales. Para ello decimos que:

1. Para los extremos sueltos de la línea: se pueden implementar diferentes tipos de terminaciones: acabado normal, redondeado o rectangular. (ver la figura No.18)

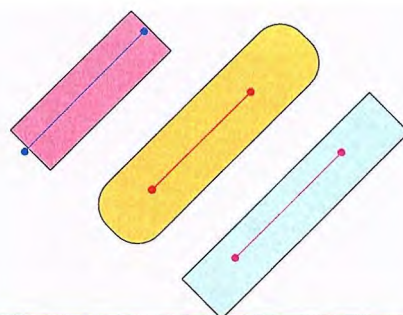


Figura No.18: tipos de finales de línea, normal, redondeado y rectangular

2. Para la unión entre rectas en una poli línea o prolongación de los límites de cada una de las dos rectas hasta que se juntan, unión mediante una sección circular cuyo diámetro es igual al ancho de la línea y unión de las rectas mediante acabado normal rellenando los huecos que se generan en el punto de unión (ver figura 19).

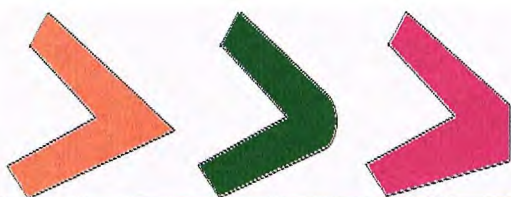


Figura No.19: tipos de uniones entre líneas. Punta, redondeada y plana

## Color de Línea<sup>24</sup>

Varias opciones de color y nivel de intensidad pueden ponerse a disposición de un usuario, según las capacidades y objetivos de diseño de un sistema determinado.

Algunos sistemas ofrecen una amplia selección de colores; otros solo tienen unas cuantas opciones. Los sistemas de rastreo con rastreador, por ejemplo, a menudo se diseñan para desplegar muchos colores, mientras que los monitores de rastreo al azar comúnmente ofrecen solo unas cuantas selecciones de color. A los códigos de color por lo general se les asigna valores numéricos que varían desde 0 a través de enteros positivos. En el caso de un CRT, estos códigos de color se convierten en ubicaciones de nivel de intensidad para los haces de electrones en monitores de máscara de sombra o de penetración del haz.

Con una graficadora de color, los códigos podrían controlar los depósitos de chorro o los cambios de la pluma.

### Tablas de Color

Un esquema simple para almacenar selecciones de código de color en el buffer de estructura de un sistema rastreador se muestra en la figura No.20. Cuando hacen las especificaciones para un código de color determinado en un programa de aplicación, el valor binario correspondiente se almacena en el buffer de estructura para cada píxel componente en las primitivas de salida que se

Código de Color	Valores de Color Almacenados en el Buffer de Cuadros			Color Desplegado
	Rojo	Verde	Azul	
0	0	0	0	Negro
1	0	0	1	Azul
2	0	1	0	Verde
3	0	1	1	Azul - verde
4	1	0	0	Rojo
5	1	0	1	Magenta
6	1	1	0	Amarillo
7	1	1	1	Blanco

Figura No.20: Códigos de Color almacenados en un buffer de estructura con tres bits por píxel.

<sup>24</sup> Donald Hearn. Gráficas por Computadora. Capítulo 4 Atributos de Salida páginas 87-89

desplegarán en ese color. El esquema que se da en la tabla admite 8 selecciones de color en 3 bits por el píxel de almacenamiento. Cada una de las tres posiciones de los bits se utiliza para controlar el nivel de intensidad (ya sea encendido o apagado) del disparador de electrones correspondiente a un monitor RGB.

El bit de más a la izquierda controla el disparador rojo, el del medio el verde y el de la derecha el azul. Si se agregan más bits por píxel el buffer de estructura aumenta el número de alternativas de color.

En un rastreador con 6 bits por píxel, pueden utilizarse 2 bits por cada disparador. Esto ofrece niveles de intensidad por cada disparador de color (rojo, verde, azul) y podrían almacenarse 64 diferentes códigos de color.



## 2.2.3 Trazado de circunferencias.

### 2.2.3.1 Método algebraico.

Como la circunferencia es un componente que se utiliza con frecuencia en imágenes y gráficas, la mayor parte de los paquetes de gráficas incluye un procedimiento para generar ya sea circunferencias completas o arcos circulares.

De modo más general, se puede ofrecer un solo procedimiento para desplegar ya sea curvas circulares o elípticas.

#### Procedimiento de las circunferencias<sup>25</sup>

Una circunferencia se define como un conjunto de puntos que se encuentran, en su totalidad, a una distancia determinada  $r$  de una posición central<sup>26</sup>  $(x_c, y_c)$  ver figura No.21. Esta relación de distancia se expresa por medio del teorema de Pitágoras en coordenadas cartesianas como

$$(ec36) \quad (x - x_c)^2 + (y - y_c)^2 = r^2$$

Podríamos utilizar esta ecuación para calcular la posición de los puntos de una circunferencia pasando a lo largo del eje de las  $x$  en pasos

unitarios de  $x_c - r$  a  $x_c + r$ , y calcular los valores correspondientes de  $y$  en cada posición como

$$(ec37) \quad y = y_c \pm \sqrt{(r^2 - (x_c - x)^2)}$$

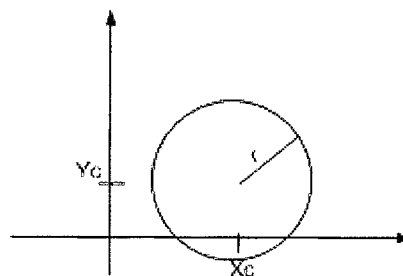


Figura No.21: Circunferencia con coordenadas centrales  $(X_c, Y_c)$  y radio  $r$ .

Sin embargo, éste no es el mejor método para generar una circunferencia.

<sup>25</sup>McClelland Decae. Drawing on the Pc. Editorial J.Lightell. Capítulo 2 Basic Drawing Theory. Páginas 44-76

<sup>26</sup> Definición de Circunferencia, mejor referencia en Matemática Básica Preuniversitaria, Autor Mendoza William. UCA editores.

Un problema es que considera cálculos considerables en cada paso. El espacio entre las posiciones de píxel trazadas no es uniforme. Para ajustar el espacio al intercambiar “x” y “y” (pasar por los valores de “y”, calcular los valores de “x”) siempre que el valor absoluto de la pendiente de la circunferencia sea mayor que 1.

Pero esto sólo incrementa el cálculo y el procesamiento que el algoritmo requiere. Otra manera de eliminar el espacio irregular consiste en calcular los puntos a lo largo de la frontera circular utilizando las coordenadas polares  $r$  y  $\Theta$  ver figura No.22. Al expresar la ecuación de la circunferencia en forma polar paramétrica, se obtiene el par de ecuaciones

$$(ec38) \quad x = x_c + r \cos \Theta$$

$$(ec39) \quad y = y_c + r \sin \Theta$$

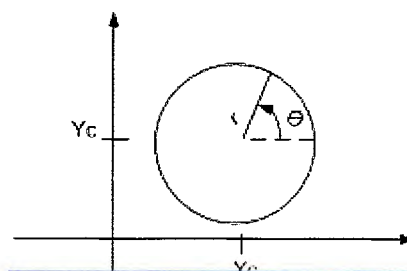


Figura No.22: Relación entre coordenadas Cartesianas y polares

Cuando un despliegue se genera con estas ecuaciones utilizando un tamaño de paso angular fijo, una circunferencia se traza con puntos equidistantes a lo largo de la misma. El tamaño de paso seleccionado para  $\alpha$  depende de la aplicación o del dispositivo de despliegue.

Las separaciones angulares más grandes a lo largo de la circunferencia se pueden unir con segmentos de línea recta a fin de aproximarse a la trayectoria circular. En el caso de una frontera más continua, podemos establecer el tamaño de paso como  $1/r$ . Trazando posiciones de píxel que están aproximadamente una unidad aparte.

Es posible reducir el cálculo al considerar la simetría de las circunferencias.

La forma de la circunferencia es similar en cada cuadrante. Se puede generar la sección circular del segundo cuadrante del plano de  $xy$  al notar que las dos

secciones circulares son simétricas con respecto del eje de las  $y$ . Y las secciones circulares del tercero y el cuarto cuadrante se pueden obtener a partir de las secciones del primero y el segundo cuadrantes considerando su simetría en relación con el eje de las  $x$ .

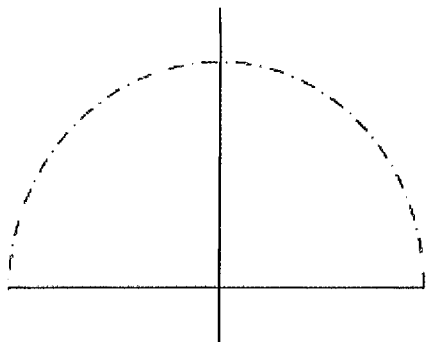


Figura No.23: Semicircunferencia Positiva trazada con (ec37) y  $(X_c, Y_c) = (0, 0)$

También hay simetría entre octantes. Las secciones circulares en octantes adyacentes dentro de un cuadrante son simétricas con respecto de la línea de  $45^\circ$  que divide los dos octantes.

Los algoritmos de circunferencias más efectivos se basan en el cálculo en incremento de los parámetros de decisión, como en el algoritmo de línea de Bresenham, que sólo implica operaciones

simples de enteros. Un método para la comparación directa de la distancia consiste en probar la posición central entre los dos píxeles para determinar si este punto medio se halla adentro o afuera de la frontera circular.

La determinación de las posiciones de píxeles a los largo de una circunferencia mediante el uso de las ecuaciones (ec37) o (ec38) y (ec39) requiere una cantidad de considerable de tiempo de cómputo. El enfoque del teorema de Pitágoras implica multiplicaciones y extracción de raíces cuadradas, mientras que las ecuaciones paramétricas contienen cálculos trigonométricos y multiplicaciones. Podemos depurar la eficiencia de la generación de circunferencias aplicando un método que reduce los cálculos, lo más posible, a aritmética entera.

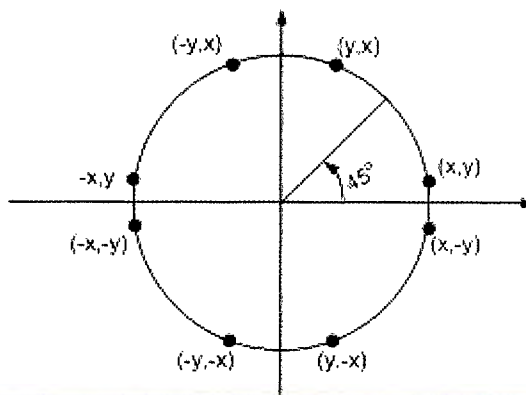


Figura No.24: Simetría de una circunferencia. El cálculo de un punto  $(x,y)$  situado en el segmento de la circunferencia del primer octante también produce los siete puntos adicionales que se muestran

Para resumir, la posición relativa se puede determinar al verificar el signo de la función de circunferencia:

- Si  $F(x,y) < 0$ , está dentro de la frontera
- Si  $F(x,y) = 0$ , está en el frontera
- Si  $F(x,y) > 0$ , está fuera de la frontera

Las pruebas de función de circunferencia en éstas condiciones se realizan para las posiciones medias entre los pixeles cercanos a la trayectoria de la circunferencia en cada paso del muestreo.

Así, la función de circunferencia es el parámetro de decisión en el algoritmo de punto medio y podemos determinar un cálculo incrementable para esta función.

La figura No.25 muestra el punto medio entre los dos pixeles candidatos en la posición de muestreo  $X_{k+1}$ .

Suponiendo que se acaba de trazar el píxel en  $(x_k, y_k)$  en seguida se necesita determinar si el píxel en la posición  $(x_k + 1, y_k)$  o aquel en la posición  $(x_k + 1, y_{k-1})$  está más cerca de la circunferencia.

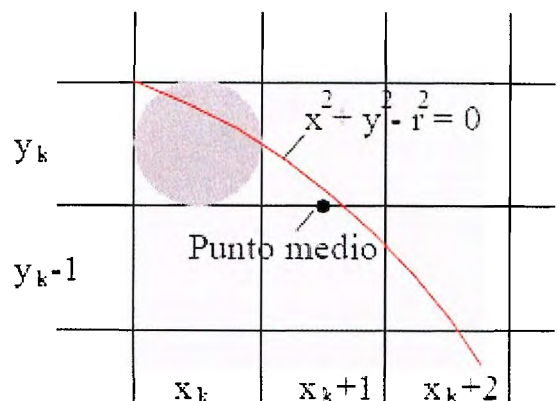


Figura No.25: Punto medio entre píxeles candidatos en la posición de muestreo  $x_k + 1$  a lo largo de la trayectoria

Nuestro parámetro de decisión es la función de circunferencia evaluada en el punto medio entre dos pixeles.

$$P_k = F(X_k + 1, Y_k - 1/2) = (X_k + 1)^2 + (y_k - 1/2)^2 - r^2$$

Si  $P_k < 0$ , este punto medio está adentro de la circunferencia y el píxel en la línea de rastreo  $y_k$  está más próximo a la frontera de la circunferencia.

De otro modo, la posición media se localiza afuera de la frontera de la circunferencia o en ésta y seleccionamos el píxel en la línea de rastreo  $y_k - 1$ .

Los parámetros de decisión sucesivos se obtienen al utilizar cálculo incremental.

Obtenemos una expresión recursiva para el siguiente parámetro de decisión cuando evaluamos la función de circunferencia en la posición de muestreo:

$$X_{k+1} + 1 = X_k + 2$$

$$P_k = F(X_k + 1, Y_k - 1/2) = [(X_k + 1) + 1]^2 + (y_k - 1/2)^2 - r^2$$

O

$$P_{k+1} = P_k + 2(X_k + 1) + (Y_{k+1}^2 - Y_k^2) - (Y_{k+1} - Y_k) + 1$$

donde  $y_{k+1}$  es ya sea  $y_k$  o  $y_{k-1}$  dependiendo del signo de  $P_k$ .

Los incrementos para obtener  $P_{k+1}$  son ya sea  $2x_{k+1}+1$  (si  $P_k$  es negativa) o  $2x_{k+1}+1-2y_{k+1}$ .

La evaluación de los términos  $2x_{k+1}$  y  $2y_{k+1}$  también pueden efectuarse de modo incremental como:

$$2X_{k+1} = 2X_k + 2$$

$$2Y_{k+1} = 2Y_k + 2$$

En la posición de inicio  $(0, r)$ , estos dos términos tienen los valores 0 y 2, en forma respectiva. Cada valor sucesivo se obtiene al sumar 2 al valor previo de  $2x$  y sustrayendo dos del valor previo de  $2y$ .

El parámetro de decisión inicial se obtiene al evaluar la función de circunferencia en la posición de inicio  $(x_0, y_0) (0, r)$ :

$$P_0 = F(1, r - 1/2) = 1 + (r - 1/2)^2 - r^2$$

O simplemente:  $P_0 = 5/4 - r$

Si el radio  $r$  se especifica como un entero, se puede redondear simplemente  $P_0$  a

$$P = 1 - r \cdot 0 \text{ (para } r \text{ como un entero)}$$

Al igual que el algoritmo para el trazo de líneas de Bresenham, el método del punto medio calcula las posiciones de píxel a lo largo de una circunferencia utilizando adiciones y sustracciones de enteros, si se supone que los parámetros de la circunferencia se especifican en coordenadas enteras de pantalla.

Se pueden resumir los pasos del algoritmo de la circunferencia de punto medio como sigue:

### Pasos del Algoritmo del Punto Medio

1. Se capturan el radio  $r$  y el centro de la circunferencia  $(x_c, y_c)$  y se obtiene el primer punto de una circunferencia centrada en el origen como  $(x_0, y_0) = (0, r)$
2. Se calcula el valor inicial del parámetro de decisión como  $p_0 = 5/4 - r$
3. En cada  $X_k$  posición, al iniciar en  $k = 0$ , se realiza la prueba siguiente. Si  $P_k < 0$ , el siguiente punto a lo largo de la circunferencia centrada en  $(0, 0)$  es  $(x_{k+1}, y_k)$  y

$$P_{k+1} = P_k + 2X_{k+1} + 1$$

De otro modo, el siguiente punto a lo largo de la circunferencia es  $(x_{k+1}, y_{k+1})$

$$y_{k+1} = P_k + 2X_{k+1} + 1 - 2y_{k+1}$$

donde  $(2X_{k+1} = 2X_k + 2)$  y  $(2y_{k+1} = 2y_k + 2)$

4. Se determinan puntos de simetría en los otros siete octantes.
5. Se mueve cada posición de píxel calculada  $(x, y)$  a la trayectoria circular centrada en  $(x_c, y_c)$  y trazamos los valores de las coordenadas:

$$x = x + x_c ; \quad y = y + y_c$$

6. Se repiten los pasos 3 a 5 hasta que  $x \leq y$ .

### Función en Código C para Algoritmo del Punto Medio<sup>28</sup>

```
void circleMidpoint (int xCenter, int yCenter, int radius)
{
    int x = 0;
    int y = radius;
    int p = 1 - radius;
    void circlePlotPoints (int, int, int, int);
    /* Plot first set of points */
    circlePlotPoints (xCenter, yCenter, x, y);
    while (x < y){
        x++;
        if (p < 0)
            p += 2*x + 1;
        else {
            y--;
            p += 2*(x - y) + 1;
        }
        circlePlotPoints (xCenter, yCenter, x, y);
    }
}

void circlePlotPoints (int xCenter, int yCenter, int x, int y)
{
    drawDot (xCenter + x, yCenter + y);
    drawDot (xCenter - x, yCenter + y);
    drawDot (xCenter + x, yCenter - y);
    drawDot (xCenter - x, yCenter - y);
    drawDot (xCenter + y, yCenter + x);
    drawDot (xCenter - y, yCenter + x);
    drawDot (xCenter + y, yCenter - x);
    drawDot (xCenter - y, yCenter - x);
}
```

---

<sup>28</sup> Dave shreiner. OpenGL Programming Guide. Editorial Addison Wesley Capítulo 3 Summary of Commands and Routines páginas 30-45



### 2.2.3.2 Método trigonométrico.

Circunferencia, en geometría significa curva plana cerrada en la que cada uno de sus puntos equidista de un punto fijo, llamado centro.

No se deben confundir los términos círculo y circunferencia<sup>29</sup>, aunque ambos conceptos están estrechamente relacionados. La circunferencia pertenece a la clase de curvas conocidas como cónicas, pues una circunferencia se puede definir como la intersección de una superficie cónica con un plano perpendicular a su eje.

Cualquier segmento rectilíneo que pasa por el centro y cuyos extremos están en la circunferencia se denomina diámetro (ver figura No.26). Un radio es un segmento que va desde el centro hasta la circunferencia. Una cuerda es un segmento rectilíneo cuyos extremos son dos puntos de la circunferencia (ver figura No. 27). Un arco de circunferencia es la parte de ésta que está delimitada por dos puntos. Un ángulo central es un ángulo cuyo vértice es el centro y cuyos lados son dos radios.



Figura No.26: Principales componentes de una circunferencia



Figura No.27: representación de Cuerda, tangente y Secante de una circunferencia

La proporción entre la longitud de la circunferencia y su diámetro es una constante, representada por el símbolo  $\pi$  o pi.

El centro de la circunferencia es centro de simetría, y cualquier diámetro es eje de simetría.

<sup>29</sup> Diferencia entre círculo y circunferencia, referencia Thompson. Trigonometría. Aprenda Usted Mismo. Editorial Limusa 1993. Capítulo 1: Ángulos, Circunferencias y triángulos. Páginas 7-13



## Posiciones Relativas de una Recta y una Circunferencia

Una recta y una circunferencia pueden ser exteriores, si no se cortan (no tienen ningún punto en común), tangentes, si sólo se tocan en un punto (punto de tangencia), y secantes si tienen dos puntos comunes.

Una recta tangente a una circunferencia es perpendicular al radio que une el centro con el punto de tangencia.

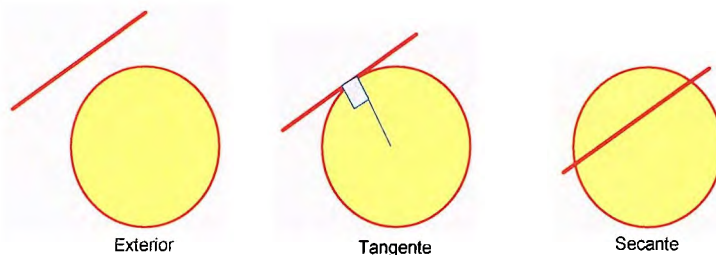


Figura No. 28: Posiciones relativas de una recta y una circunferencia

El radio correspondiente a un arco puede calcularse a partir de la longitud  $L$  de la cuerda y la distancia  $D$  que va del centro de la cuerda al punto más

cercano al círculo por medio de :

- Método geométrico:  $\text{Radio} = ((L / 2)^2 + D^2) / 2D$
- Método trigonométrico:  $\text{Radio} = L / (\sin \pi - 2 \tan^{-1}(L/D))$

### 2.2.3.3 Algoritmo de Bresenham<sup>30</sup>.

Como sucede en el algoritmo de líneas, las posiciones enteras a lo largo de una trayectoria circular pueden obtenerse determinando cuál de los dos pixeles está más próximo a la circunferencia en cada etapa. Para simplificar los enunciados del algoritmo, primero se considera una circunferencia con centro en el origen coordenado ( $x_c = 0$   $y_c = 0$ ). También se calculan los puntos de un octavo segmento de una circunferencia suponiendo que se obtendrán los puntos restantes por simetría para almacenarse en un rastreador. Se toman etapas unitarias en el sentido x, comenzado desde  $x=0$  y terminando cuando  $x=y$ . La coordenada inicial de nuestro algoritmo es por tanto  $(0,r)$ .

En la figura No.29 se muestra la situación en alguna etapa arbitraria del algoritmo. Se supone que la posición  $(x_i, y_i)$  se ha determinado como más próxima a la trayectoria de la circunferencia. La siguiente posición de  $y$  en la trayectoria, se determina como:

$$(ec40) \quad y^2 = r^2 - (x_i + 1)^2$$

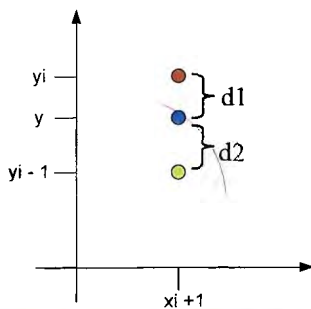


Figura No.30: Diferencias coordenadas entre los centros de los pixeles y la posición  $y$  en una circunferencia en  $x_i + 1$

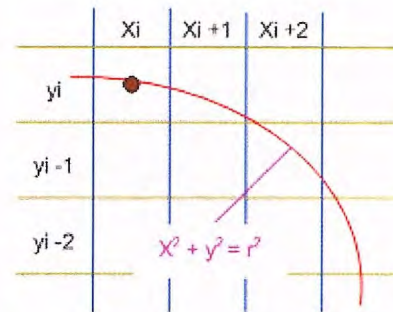


Figura no.29: Sección de una retícula de la pantalla donde se desplegará una circunferencia que pasa por el punto  $(x_i, y_i)$

La figura No. 30 ilustra la relación entre  $y$  y los valores coordenados enteros,  $y_i$  y  $y_i - 1$ . Una medida de la diferencia en las posiciones coordenadas puede definirse en términos del cuadrado de los valores de  $y$  como:

$$d1 = y_i^2 - y^2$$

$$d1 = y_i^2 - r^2 + (x_i + 1)^2$$

$$(ec41) \quad d2 = y_i^2 - (y_i - 1)^2$$

<sup>30</sup> D.Foley James. Computer Graphics: Principles and Practice. Editorial Wesley. Capítulo 1. Large scale Computations. Páginas 88-123

$$(ec42) \quad d2 = r^2 - (x_i + 1)^2 - (y_i - 1)^2$$

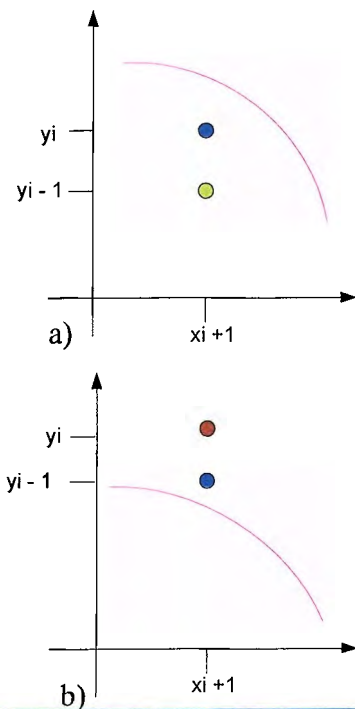
Ahora se crea un parámetro para determinar la siguiente posición coordenada como la diferencia entre  $d1$  y  $d2$ :

$$(ec43) \quad p_i = d1 - d2$$

$$(ec44) \quad p_i = 2(x_i + 1)^2 + y_i^2 + (y_i - 1)^2 - 2r^2$$

Si  $p_i$  es negativa, se selecciona el píxel en la posición  $y_i$ . De lo contrario, se selecciona el píxel situado en la localidad  $y_i - 1$ .

La prueba de la selección del siguiente píxel se cumple si la trayectoria real pasa sobre  $y_i$  o bien debajo de  $y_i - 1$ , como se muestra en la figura No. 31



En el primer caso, la figura 20a), se tiene  $d1 < 0$ ,  $d2 > 0$  y  $p_i < 0$ , de manera que el punto en  $y_i$  sería el seleccionado. En el segundo caso,  $d1 > 0$  y  $d2 < 0$ . Ahora  $p_i > 0$  y el punto  $y_i - 1$  es el que se selecciona.

Una forma recursiva del parámetro  $p$  se obtiene evaluando  $p_{i+1}$  en términos de  $p_i$ :

$$p_{i+1} = 2[(x_i + 1) + 1]^2 + (y_{i+1})^2 + (y_{i+1} - 1)^2 - 2r^2$$

Esta expresión puede escribirse en términos de la ecuación (ec44)

$$(ec45) \quad p_{i+1} = p_i + 4x_i + 6 + 2(y_{i+1}^2 - y_i^2) - 2(y_{i+1} - y_i)$$

Figura No. 31: Posibles posiciones de píxeles: a) Ambos centros de los píxeles están debajo de la trayectoria  
b) Ambos centros de los píxeles están sobre la trayectoria

La posición  $y$ ,  $y_{i+1}$  es la misma que  $y_i$  o bien la misma que  $y_i - 1$ , según el valor de  $p_i$ . Comenzando desde  $p_1$ , el algoritmo determina cada parámetro  $p$  sucesivo desde  $0$  a partir del anterior. Se obtiene  $p_1$  haciendo  $(x_1, y_1) = (0, r)$  en la ecuación (ec44), de la siguiente forma:

$$(ec46) \quad p_1 = 3 - 2r$$

### Algoritmo de Bresenham para el trazo de Circunferencias:

1. Selecciona la primera posición para el despliegue como  $(x_1, y_1) = (0, r)$
2. Calcule el primer parámetro como  $p_1 = 3 - 2r$   
 Si  $p_1 < 0$ , la siguiente posición es  $(x_1 + 1, y_1)$ . De lo contrario, la siguiente posición es  $(x_1 + 1, y_1 - 1)$ .
3. Continúe por incrementar la coordenada  $x$  en pasos unitarios y calcule cada parámetro sucesivo  $p$  a partir del anterior. Si para el parámetro anterior se halló que  $p_i < 0$ , entonces  $p_{i+1} = p_i + 4x_i + 6$   
 En Caso contrario (para  $p_i \geq 0$ ),  $p_{i+1} = p_i + 4(x_i - y_i) + 10$   
 Por lo tanto, si  $p_{i+1} < 0$ , el siguiente punto seleccionado es  $(x_i + 2, y_{i+1})$ . De lo contrario, el siguiente punto es  $(x_i + 2, y_{i+1} - 1)$ .  
 La coordenada  $y$  es  $y_{i+1} = y_i$ , si  $p_i < 0$  o bien  $y_{i+1} = y_i - 1$ , si  $p_i \geq 0$ .
4. Repita los procedimientos del paso 3 hasta que las coordenadas  $x$  y  $y$  sean iguales.

### Código de Procedimiento Algoritmo de Bresenham <sup>31</sup>

```
#include <GL/glut.h>
// drawcirc
// Dibuja circunferencia de radio r centrada en el origen
// Utiliza Bresenham para calcular los puntos enteros
// en un arco de (0,r) "clockwise" hasta la recta x = y.
// Dibuja los otros puntos por simetrías
```

<sup>31</sup> Dave shreiner. OpenGL Programming Guide. Editorial Addison Wesley Capítulo 3 Summary of Commands and Routines páginas 30-45

```

void drawcirc(const int r)
{
    int x;          // x coord de punto actual
    int y = r;      // y coord (idem)
    int f = 5 - 4 * r; // valor de testing

    glBegin(GL_POINTS);

    for (x = 0; x <= y; ++x) // Ciclo Pal. Bresenham
    {
        glVertex2i( x, y); // Plot de un punto
        glVertex2i( y, x); // Plot de los otros por simetrías
        glVertex2i(-x, y); // (Es ligeramente ineficiente: puntos de inicio y final dos
veces)
        glVertex2i( y,-x);
        glVertex2i( x,-y);
        glVertex2i(-y, x);
        glVertex2i(-x,-y);
        glVertex2i(-y,-x);
        if (f < 0)
        {
            f += 8 * x + 12;      // A la derecha
        }
        else
        {
            --y;                  // A la derecha y abajo
            f += 8 * (x - y) + 20;
        }
    }

    glEnd();
}

// display
// Rutina pal. de dibujo
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0); // "R"
    drawcirc(42); // Dibuja círculo de radio 42 centrado al origen
    glFlush();
}

// init
// Inicializa estados GL, matrices, stacks, etc...

```



```

void init()
{
    glClearColor (0.0, 0.0, 0.0, 0.0);

    glPointSize(5.0); // Punto grandes

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-50, 50, -50, 50, -1.0, 1.0);
    // De 0 a 100 en ambos ejes
}

int main(int argc, char** argv)
{
    // Inicializa OpenGL/GLUT
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

    // Ventana
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(50, 50);
    glutCreateWindow("CS 381 - Bresenham's Algorithm: Circle");

    // Inicializa estados y callbacks
    init();
    glutDisplayFunc(display);

    glutMainLoop();

    return 0;
}

```

## 2.2.4 Trazo de otras curvas (elipses, arcos).

### 2.2.4.1. ELIPSES<sup>32</sup>

Las elipses<sup>33</sup> pueden ser vistas como círculos alargados, como consecuencia, pueden ser generadas mediante la modificación de los procedimientos de conversión de círculos para tener en cuenta las diferentes dimensiones a lo largo de sus ejes mayor y menor.

En el algoritmo que se ve a continuación, la elipse de entrada se supone que estará centrada en el origen de coordenadas (0,0), con un eje principal paralelo al eje X y el otro paralelo al eje Y.

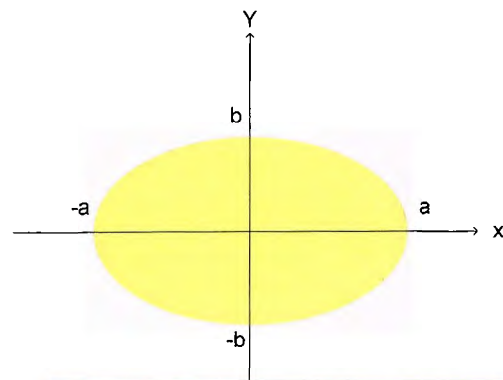


Figura No. 32: elipse estándar centrada en el origen

Esta restricción no impide generar cualquier tipo de elipses, ya que bastaría con generarla cumpliendo las restricciones y después, mediante matrices de transformación, llevarla a la posición e inclinación que se indique.

Un algoritmo de trazo de circunferencias puede ampliarse para dibujar elipses. En la figura No.33, se muestra una orientación con r1 que marca el eje semimayor y r2, como el eje semimenor. La forma estándar de la ecuación elíptica es:

$$(ec46) \quad \left( \frac{x - x_c}{r1} \right)^2 + \left( \frac{y - y_c}{r2} \right)^2 = 1$$

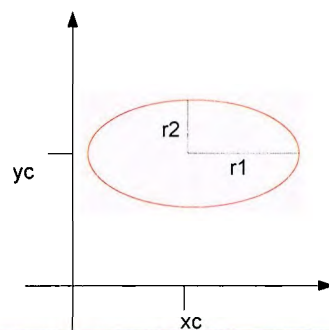


Figura No.33: elipse centrada en (xc,yc) con eje semimayor r1 y eje semimenor r2

<sup>32</sup> Donald Hearn. Gráficas por Computadora. Capítulo 3 Primitivas de Salida, Páginas 74 -76

<sup>33</sup> Elipse:a) Se llama elipse al lugar geométrico de los puntos del plano tales que la suma de las distancias a dos puntos fijos llamados focos es una cantidad constante. MARAVALL CASESNOVES, D Diccionario de Matemática Moderna. editorial Ra-Ma b) Una elipse es el lugar geométrico de los puntos del plano equidistantes de una circunferencia y de un punto interior.

Mediante el uso de las coordenadas polares  $r$  y  $\Theta$ , también se pueden escribir las ecuaciones elípticas en forma paramétrica:

$$(ec47) \quad \begin{aligned} x &= x_c + r1.\cos\Theta \\ y &= y_c + r2.\sen\Theta \end{aligned}$$

El algoritmo de Bresenham para circunferencias puede modificarse para generar formas elípticas utilizando éstas ecuaciones. Es decir, para una elipse con centro en el origen, se pueden expresar valores de  $y$  en la forma

$$(ec48) \quad y^2 = r^2.(1 - x^2 / r1^2)$$

La única diferencia está en la forma de los parámetros  $p$ . Una elipse se traza en una posición arbitraria agregando desplazamiento a los valores de  $x$  y  $y$  de salida, como en la generación de posiciones de la circunferencia.

### **Simetría de Cuatro Puntos**

En la rasterización del círculo, el problema de las discontinuidades debido al cambio de inclinación de la curva que lo define, se solucionaba mediante la utilización de la simetría del mismo.

Esta permitía la división del círculo en octantes, de modo que, únicamente calculando el primer octante, se podía dibujar todo el círculo.

Esto permitía no sólo acelerar el algoritmo de conversión de círculos sino que, dado que en el primer octante la inclinación está entre  $[-1,1]$  hacía desaparecer las discontinuidades.

Las simetrías provocan permiten dibujar una elipse completa a partir de un cuadrante completo, deduciéndose el resto de la elipse a partir de las simetrías existentes.



Utilizando esta propiedad en el algoritmo de la fuerza bruta lo aceleraría, pero no haría desaparecer las discontinuidades.

La no desaparición de las discontinuidades se debe fundamentalmente a la existencia de dos regiones de inclinaciones diferentes en un mismo cuadrante (ver figura No.34) La *Región 1* con una inclinación dentro del intervalo  $[-1,1]$  y que, por lo tanto, para su dibujado se podría utilizar la ecuación positiva dada para el algoritmo de la fuerza bruta. La *Región 2*, cuya inclinación está fuera del intervalo  $[-1,1]$  y que para su dibujado se tendría que obtener una ecuación que obtuviera el

valor  $X$  en función de incrementos unitarios de  $Y$  es decir:

$$x = r_x \sqrt{1 - \left(\frac{y}{r_y}\right)^2}$$

Para detectar el paso de la region 1 a la region 2 se puede utilizar el vector gradiente de la elipse.

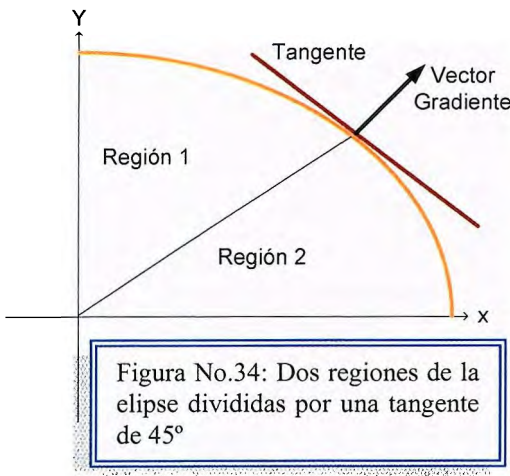


Figura No.34: Dos regiones de la elipse divididas por una tangente de 45°

Este vector, en un punto de su perímetro, es perpendicular a la curva de la elipse en ese

punto, y viene definido de la siguiente forma:

$$eq = [(x/r_x)^2 + (y/r_y)^2]$$

$$\text{grad}(eq) = [(\partial eq) / \partial x] \hat{i} + [(\partial eq) / \partial y] \hat{j} = 2 \cdot r_y^{-2} \cdot x \cdot \hat{i} + 2 \cdot r_x^{-2} \cdot y \cdot \hat{j}$$

El cambio de gradiente se dará en el punto donde la inclinación de la curva sea exactamente -1.

Esto ocurre cuando en el vector gradiente, el valor de la componente  $\hat{i}$  y el valor de la componente  $\hat{j}$  coinciden. Por tanto, la región 1 se puede caracterizar como aquella en la que:

$$2 \cdot r_y^{-2} \cdot x \cdot \hat{i} \leq 2 \cdot r_x^{-2} \cdot y \cdot \hat{j}$$

Mientras que la región 2 se puede caracterizar como aquella en la que:

$$2.r_y^2 .x.\hat{i} \geq 2.r_x^2 y.\hat{j}$$

### Algoritmo detalle del Punto Medio para Elipses

El algoritmo del punto medio esta basado en la subdivisión en cuadrantes descrita anteriormente, de modo que, para rasterizar una elipse sólo forma el primer cuadrante, obteniendo el resto por simetrías.

La función principal es:  $F(x,y) = r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2$

Donde  $F(x,y)$  valdrá 0 cuando el punto  $(x,y)$  se encuentre sobre el perímetro de la elipse.

- $F(x,y) > 0$  cuando el punto  $(x,y)$  se encuentre por encima de la curva (fuera del área delimitada por la elipse)
- $F(x,y) < 0$  cuando el punto  $(x,y)$  se encuentre por debajo de la curva (dentro del área delimitada por la elipse)
- El paso de la región 1 a la región 2 en el algoritmo del punto medio, se sabe cuando el siguiente punto medio a comprobar cumple la desigualdad:

$$r_y^2 (y_k - 1/2) \leq (x_k + 1)$$

### Región 1

Si el píxel actual es  $(X_k, Y_k)$ , la función de decisión  $d_k = F(X_k + 1, Y_k - 1/2)$ , donde  $(X_k + 1, Y_k - 1/2)$ , es el punto medio entre E y SE (ver figura No.33). Lo ideal es poder calcular la variable de decisión actual en función de la anterior. De este modo, si el punto elegido de movimiento en función de  $d_k$ , fuera E, la variable de decisión  $d_k + 1$ , tendría la siguiente forma:

$$(ec49) \quad d_{k+1} = F(x_k + 2y_k - 1/2) = r_y^2 (x_k + 2)^2 + r_x^2 (y_k - 1/2)^2 - r_x^2 r_y^2$$

Por tanto:  $d_{k+1} = dx_k + r_y^2 (2x_k + 3)$  (ec50)

Ahora bien, si el punto elegido de movimiento en función de  $d_k$  fuera SE, la variable de decisión tendría la siguiente forma:

$$(ec50) \quad d_{k+1} = F(x_k + 2y_k - 3/2) = r_y^2 (x_k + 2)^2 + r_x^2 (y_k - 3/2) - r_x^2 r_y^2$$

Por tanto:  $(ec51) \quad d_{k+1} = d_k + r_y^2 (2x_k + 3) + r_x^2 (-2y_k + 2)$

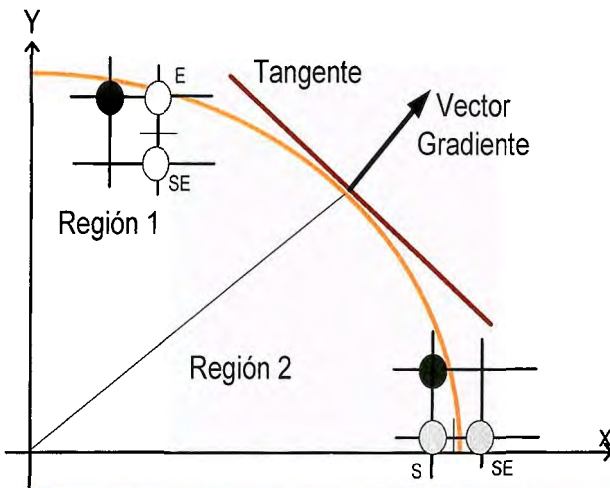


Figura No.35: Dos regiones representadas, mostrando puntos de evaluación E y SE

### Región 2:

En la región 2, la variable de decisión  $d_k = F(X_k + 1/2, Y_k - 1)$ , donde  $(X_k + 1/2, Y_k - 1)$  es el punto medio entre S y SE. Siguiendo el mismo razonamiento que para la región 1, si es en función de  $d_k$ , se elige el punto S, y el valor de  $d_{k+1}$  será:

$$(ec52) \quad d_{k+1} = d_k + r_y^2 (2x_k + 2) + r_x^2 (-2y_k + 3)$$

## Inicialización de Variable de Decisión

Hasta ahora se ha obtenido la fórmula para determinar la variable de decisión actual en función de la anterior. Lo único que queda por definir es el valor inicial de dicha variable de decisión, tanto para la región 1 como para la región 2.

En la región 1, dado que el primer punto que se dibuja es el  $(0, r_y)$ , el primer punto medio a calcular será  $(1, b-1/2)$ , y por tanto el valor inicial de la variable de decisión será:

$$(ec53) \quad d_0 = r_y^2 + r_x^2(-r_y + 1/4)$$

En la región 2, la inicialización de la variable de decisión se lleva a cabo justo en el momento del cambio de región. De este modo, si  $(x, y)$  es el punto a partir del cual se cumple la desigualdad que marca el cambio de región, el valor inicial que se le asignará a la variable de decisión de esta región será:

$$(ec54) \quad d_0 = F(x + 1/2, y-1) = r_y^2 (x + 1/2)^2 + r_x^2 (y - 1) - r_x^2 r_y^2$$

Teniendo en cuenta todo lo dicho, el algoritmo del punto medio que permitiría rasterizar una elipse de radios  $r_x$  y  $r_y$  tendría la siguiente forma:

```
x= 0
y= 0
d= ry2 + rx2( -ry + 1/4)

Dibuja_4simetricos (x,y)

Mientras rx2( ry - 1/2) > ry2 (xk +1) { región 1}
    Si d < 0
        d= d+ ry2 (2x +3)
        x= x+1
    Sino
        d= d+ ry2 (2x +3) + rx2 (-2y +2)
        x= x+1
        y= y-1
    Fin Si
    Dibuja_4simetricos (x,y)
Fin Mientras {región}
```

$$d = r_y^2 (x + \frac{1}{2})^2 + r_x^2 (y - 1)^2 - r_x^2 r_y^2$$

```

Mientras y > 0                                {región 2}
    Si d < 0
        d= d+ r_y^2 (2x +2) + r_x^2 (-2y +3)
        x= x + 1
        y= y -1
    Sino
        d= d+ r_x^2 (-2y +3)
        y= y -1
    Fin Si
    Dibuja_4simetricos (x,y)
Fin Mientras                                {region 2}

```

Donde Dibuja\_4simetricos () toma como entrada un punto (x, y) del primer cuadrante de la elipse y dibuja ese punto y sus tres simétricos, por ejemplo: (x, -y), (-x,-y) y (-x,y)

## 2.2.4.2. OTRAS CURVAS<sup>34</sup>

Los procedimientos para desplegar varias curvas utilizan métodos similares a aquellas que generan circunferencias y elipses. Las curvas que se muestran incluyen funciones seno, funciones exponenciales, polinomiales, distribuciones de probabilidad y funciones spline.

Si se puede expresar una curva en forma funcional,  $y = f(x)$ , los valores de  $y$  pueden calcularse y trazarse con respecto a un intervalo especificado de  $x$ . Los valores pueden llenarse con puntos individuales o segmentos rectilíneos. En muchas aplicaciones, las curvas aproximadas con segmentos de rectas son más adecuadas.

Un método de trazo de puntos deja brechas en la curva en áreas donde la magnitud de la pendiente es mayor que 1. Evitar las brechas con un método de trazo de puntos significa que se debe obtener la función inversa,  $x = f^{-1}(y)$  y calcular los valores de  $x$  para valores dados de  $y$  siempre que la magnitud de la pendiente se vuelva grande.

Las consideraciones de simetría mejora la eficiencia de algunos algoritmos de generación de curvas. Muchas curvas tienen patrones repetidos, de manera que puede ser posible obtener más de un punto sobre la curva con un solo cálculo.

Las parábolas y la distribución normal de probabilidad son simétricas con respecto a un punto central, mientras que todos los puntos que se encuentran en un ciclo de una curva seno pueden generarse a partir de los puntos de un intervalo de  $90^\circ$ .

Para una curva definida por un conjunto de datos de puntos coordinados discretos, se debe graficar la curva en otras formas. Un método consiste

---

<sup>34</sup> Donald Hear. Gráficas por Computadora. Capítulo 3 Primitivas de Salida páginas 75-77.

simplemente en trazar los puntos de datos individuales y conectarlos con segmentos rectilíneos.

Una curva tiene un sólo punto de partida, una longitud y un punto final. Realmente se trata de una línea que ondula en el espacio. Una superficie tiene además ancho y largo, y por tanto un área.

Cuando pensamos en líneas rectas, podemos pensar en esta ecuación:  $Y = mX + b$

Aquí  $m$  representa la pendiente de la línea y  $b$  es la intersección  $Y$  de la línea.

Otra forma de representar la ecuación de una curva o línea es la ecuación paramétrica, que expresa tanto  $X$  como  $Y$  en términos de otra variable que varía a lo largo de un rango de valores predefinido, que no es parte explícita de la curva.

Cuando definimos una curva en OpenGL, también la definimos como una ecuación paramétrica. El parámetro de la curva, que llamamos  $u$ , y su rango de valores, constituyen el dominio de la curva. Las superficies se describen con dos parámetros  $u$  y  $v$ .

## **PUNTOS DE CONTROL**

Las curvas están representadas por un número de puntos de control que influyen en la forma de la curva. Para las curvas de Bezier, los puntos de control primero y último son parte de la curva. Los otros actúan como imanes, atrayendo la curva hacia ellos.

La figura No. 36 muestra algunos ejemplos de este concepto, con distintos números de puntos de control.

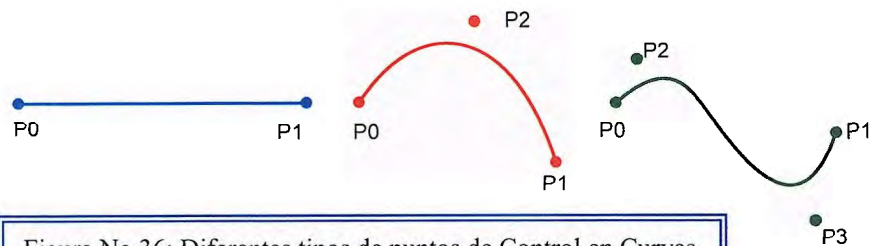


Figura No.36: Diferentes tipos de puntos de Control en Curvas

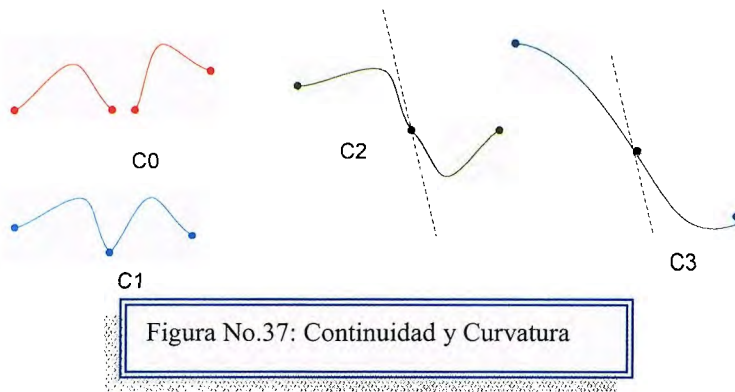
El orden de la curva está representado por el número de puntos de control usados en su descripción. El grado es uno menos el orden de la curva. En la figura anterior, la curva B se llama curva cuadrática (de grado 2), y la curva C se llama cúbica (de grado 3). Estas últimas son las más típicas. Teóricamente podemos definir una curva de cualquier orden, pero las curvas de órdenes altos comienzan a oscilar de manera incontrolable y pueden cambiar mucho con la más leve modificación en los puntos de control.

## CONTINUIDAD

Si dos curvas situadas cara con cara comparten el punto final (denominado punto de rotura), juntas forman una curva continua. La continuidad de estas curvas en su punto de rotura describe la suavidad de la transición entre ellas. Hay cuatro categorías de continuidad:

- Continuidad nula ( $C_0$ ): las curvas ni se tocan.
- Continuidad de posiciones ( $C_1$ ): las curvas se encuentran y comparten el punto final.
- Continuidad tangencial ( $C_2$ ): las curvas tienen la misma tangente en el punto de rotura.
- Continuidad de curvatura ( $C_3$ ): las tangentes de las curvas tienen el mismo ratio de cambio en el punto de rotura.





OpenGL contiene funciones que facilitan el dibujo de curvas y superficies de Bezier especificando los puntos de control y el rango de los parámetros  $u$  y  $v$ .

Entonces, con la función evaluadora apropiada, se generarán los puntos que definen la curva o superficie.

Un ejemplo 2D que especifica cuatro puntos de control para una curva de Bezier y después genera la curva usando un evaluador:

```
GLint nNumPoints = 4;    //El número de puntos de control de esta curva
```

```
GLfloat ctrlPoints[4][3]= {{ -4.0f, 0.0f, 0.0f},    // Punto final
                           {-6.0f, 4.0f, 0.0f},    // Punto de control
                           {6.0f, -4.0f, 0.0f},    // Punto de control
                           {4.0f, 0.0f, 0.0f}};    // Punto final
```

```
...
...
```

```
//Esta función se usa para superponer los puntos de control sobre la curva
void DrawPoints(void)
{
    int i;    //variable de cuenta

    // Selecciona el punto de mayor tamaño para hacerlo más visible
    glPointSize(5.0f);

    // Bucle sobre todos los puntos de control para este ejemplo
    glBegin(GL_POINTS);
        for(i = 0; i < nNumPoints; i++)
            glVertex2fv(ctrlPoints[i]);
    glEnd();
```

```

    }

// Cambia el volumen de visualización y la vista. Invocado al redimensionar la
ventana
void ChangeSize(GLsizei w, GLsizei h)
{
    // Previene una división por cero
    if(h == 0)
        h = 1;

    // Define una vista con las dimensiones de la ventana
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    gluOrtho2D(-10.0f, 10.0f, -10.0f, 10.0f);

    // Reinicia la matriz del modelador
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

//Invocado para dibujar la escena
void RenderScene(void)
{
    //int i;

    // Limpia la ventana con el color de borrado actual
    glClear(GL_COLOR_BUFFER_BIT);

    //Define la Bezier
    // Sólo necesitamos ejecutar ésto una vez y puede estar en la función de
inicio
    glMap1f(GL_MAP1_VERTEX_3, // Tipo de dato generado
    0.0f, // Rango menor de u
    100.0f, // Rango superior de u
    3, // Distancia entre los puntos almacenados
    nNumPoints, // numero de puntos de control
    &ctrlPoints[0][0]); // Matriz de puntos de control

    // Activa el evaluador
    glEnable(GL_MAP1_VERTEX_3);

    // Usa una franja de líneas para "conectar los puntos"
    glBegin(GL_LINE_STRIP);

```

## 2.3. Llenado de región<sup>35</sup>.

El llenado<sup>36</sup> de una región consiste en cubrir un área delimitada por líneas, puede ser un círculo, triángulo, cuadrado entre otros elementos (ver Figura No. 38).

Una manera de llenar una área consiste en determinar los intervalos de traslape para las líneas de rastreo que cruzan el área. Otro método para el llenado de áreas es iniciar desde una posición inicial determinada y pintar hacia afuera desde este punto

hasta encontrar las condiciones de frontera específicas. El planteamiento de la línea de rastreo

se utiliza, por lo regular, en paquetes de gráficas generales para llenar polígonos, circunferencias, elipses y otras curvas simples. Los métodos de llenado que empiezan desde un punto interior son útiles con fronteras más complejas y en sistemas interactivos de pintura.



Figura No. 38. Diferentes formas que pueden ser

### 2.3.1. Algoritmo de Exploración por línea.

El algoritmo exploración por línea se caracteriza por utilizar dos listas: la lista de líneas de exploración (LDR), y la lista de aristas activas (LAA). La lista de líneas de rastreo tendrá una entrada por cada una de las líneas que atraviesen el objeto cuya área se desea rellenar.

Cada una de estas entradas contendrá la lista de aristas que son intersectadas por primera vez y, dicha intersección, es con la línea de exploración asociada a dicha entrada.

<sup>35</sup> HEARN, D., y PAULINE BAKER, Gráficas por Computadora, 1995.

<sup>36</sup> Llenado: consiste en cubrir o cambiar de color áreas delimitadas por líneas.

Cada una de las entradas en las listas de aristas corresponde a una arista diferente que contendrá la siguiente información:

- N° de líneas de exploración que atraviesan dicha arista.
- Coordenada x de la intersección con la línea de exploración actual.
- $\Delta x$  por cada incremento unitario de y en dicha arista.

### 2.3.1.1. Cálculo de la Lista de línea de Exploración (LDR).

Para el cálculo de las líneas de rastreo a un objeto se debe aplicar los siguientes pasos.

- Trazar líneas de rastreo por cada fila del objeto y calcular las intersecciones de cada una de las líneas de rastreo del elemento cuya área se rellenara.
- Ordenar las intersecciones por orden creciente de las x.
- Agrupar las intersecciones por pares consecutivos.
- Rellenar los píxeles que caen entre estos pares de intersecciones

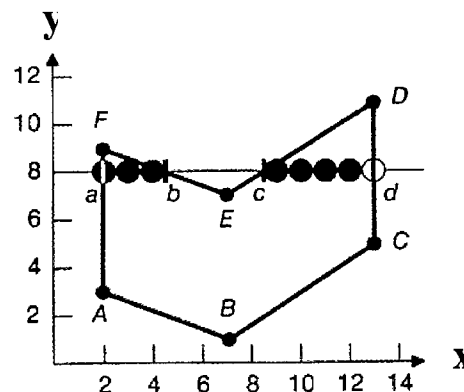
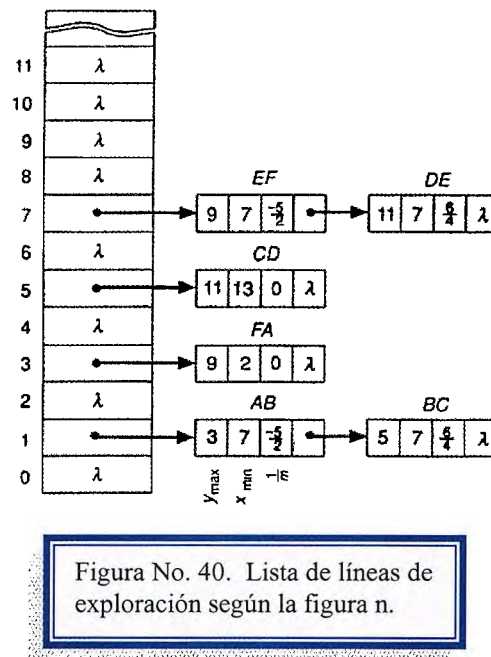


Figura No. 39. Trazo de línea de exploración.

El resultado final después de trazar todas las líneas de exploración que pasan por el objeto es el área interna del elemento rellena.

A continuación se crea la lista según el número de líneas de exploración utilizadas para rellenar el objeto, según la figura n, los datos se mostraran así:



### 2.3.1.2. Pasos del Algoritmo de Línea de exploración.

Tomando en cuenta la información del apartado anterior, el algoritmo de exploración por línea, se desarrolla de la siguiente forma:

1. Calcular LDR de la forma antes mencionada.
2. Inicializar LAA a vacía.
3. Asignar a LAA la lista de aristas contenida en la primera entrada de LDR no vacía.
4. Mientras la LAA no esté vacía.

- 4.1. Ordenar los componentes de la LAA en orden creciente valor  $x$ .
  - 4.2. Agrupar por pares las entradas de LAA.
  - 4.3. Rellenar entre los pares agrupados.
  - 4.4. Por cada una de las entradas de LAA.
    - 4.4.1. Decrementar el nº de líneas de exploración en una unidad.
    - 4.4.2. Incrementar la coordenada  $x$  de la intersección actual en función del valor  $Dx$ .
  - 4.5. Fin Por
  - 4.6. Si alguna entrada de LAA tiene su nº de líneas de rastreo a cero, eliminar dicha entrada de LAA.
  - 4.7. Se pasa a la siguiente línea de rastreo y se inserta en LAA la lista de aristas a la que apunta esta línea de rastreo en LDR.
5. Fin mientras.

### **2.3.2. Algoritmo de inundación<sup>37</sup>.**

Este algoritmo se caracteriza por estar delimitadas por un conjunto de píxeles con un determinado color, a estos se les denomina píxeles frontera. En el caso de este tipo de algoritmo, la única restricción que se ha de cumplir es que no existan píxeles internos en el área a rellenar. El incumplimiento de esta restricción puede generar un mal funcionamiento. El algoritmo iterativo que permitiría rellenar este tipo de áreas seguiría el siguiente esquema:

- $P$  = píxel semilla.
- Apilar  $P$  en la pila de semillas.
- Mientras la pila de semillas no esté vacía.
  - $S$  = cabeza pila semillas.
  - Desapila la cabeza de la pila de semillas.

---

<sup>37</sup> Trias Pairó Joan. Geometría para la Informática Gráfica. Editorial Alfa y Omega. Capítulo 3 Objetos geométricos lineales y algoritmos elementales



### **2.3.3. Algoritmo de inundación con recursión mínima.**

En el algoritmo se rellena áreas que estén caracteriza por estar definidas por un conjunto de píxeles con un atributo común (color). A continuación se describe el algoritmo que permite sustituir el color antiguo de un área por un color nuevo, mediante recursividad para relleno de regiones 4-conexas (para regiones 8-conexas la extensión es inmediata):

- Procedimiento *rellena(x, y, colorviejo, colornuevo)*.
- Si el color del píxel de coordenada (x,y) es colorviejo.
  - Asigna el colornuevo al píxel (x,y).
  - *rellena(x+1, y, colorviejo, colornuevo)*.
  - *rellena(x-1, y, colorviejo, colornuevo)*.
  - *rellena(x, y+1, colorviejo, colornuevo)*
  - *rellena(x, y-1, colorviejo, colornuevo)*
- Fin Si.
- Fin Procedimiento.

### **2.3.4. Llenado de Triángulos y polígonos.**

Para el llenado de áreas con forma de polígonos, se utilizan dos métodos de rellanado, el algoritmo de exploración por líneas<sup>38</sup> y por inundación<sup>39</sup>. Para llenar

---

<sup>38</sup> Ver el tema “2.2.3. Trazado de circunferencias”, con el sub-tema “2.2.3.1 Método Trigonométrico”, en la pagina No.

<sup>39</sup> Tomar como referencia el tema “2.3.2 Algoritmo de inundación”, desde la página No.

un rectángulo con un color sólido, se asigna a cada píxel sobre una misma línea de exploración desde el borde izquierdo al borde derecho el mismo valor de píxel; o sea llenamos cada intervalo de  $X_{min}$  a  $X_{max}$ .

### **2.3.4.1. Llenado de polígonos en OpenGL.**

Para el relleno de polígonos se hace la llamada a `glColor` entre la definición de cada polígono. El siguiente código representa el relleno de dos triángulos.

#### **Código en OpenGL<sup>40</sup>**

```
glBegin(GL_TRIANGLES);
    glColor3f(1.0f, 0.0f, 0.0f); // Rellenado Color Rojo
    glVertex(0.0f, 0.0f, 0.0f);
    glVertex(2.0f, 0.0f, 0.0f);
    glVertex(1.0f, 1.0f, 0.0f);

    glColor3f(0.0f, 1.0f, 0.0f); // Rellenado de Color Verde
    glVertex(-1.0f, 0.0f, 0.0f);
    glVertex(-3.0f, 2.0f, 0.0f);
    glVertex(-2.0f, 0.0f, 0.0f);
glEnd();
```

### **2.3.5. Llenado de circunferencias y elipses.**

#### **2.3.5.1. Llenado de línea de exploración de áreas de frontera curva.**

En llenado de línea de exploración con fronteras curvas requiere más trabajo que el llenado de polígonos porque los cálculos de intersecciones implican fronteras no lineales. Para curvas simples como circunferencias o elipses, realizar un llenado de línea de exploración es un proceso directo, sólo se calculan las dos intersecciones de la línea de rastreo en lados opuestos de la curva. Esto es lo

---

<sup>40</sup> Dave shreiner. OpenGL Programming Guide. Editorial Addison Wesley Capítulo 3 Summary of Commands and Routines



- Se rellena la fila de píxeles donde se encuentra  $V$  hasta llegar a los píxel frontera.
  - $S$  píxeles más a la izquierda de la última fila rellena.
  - Si el píxeles por debajo de  $S$  no es píxeles frontera ni al nuevo color.
  - Se apila el píxeles por debajo de  $S$  en la pila de semillas.
  - Si el píxeles por encima de  $S$  no es píxeles frontera ni al nuevo color.
  - Se apila el píxeles por encima de  $S$  en la pila de semillas.
  - Se apilan también todos los otros píxeles que se encuentran por debajo de la última fila rellena que no estén al nuevo color y a su izquierda haya un píxel frontera.
  - Se apilan también todos los otros píxeles que se encuentran por encima de la última fila rellena que no estén al nuevo color y a su izquierda haya un píxel frontera.
- o Fin Mientras.

A continuación se muestran las cuatro primeras iteraciones de este algoritmo donde el píxel blanco es la semilla.

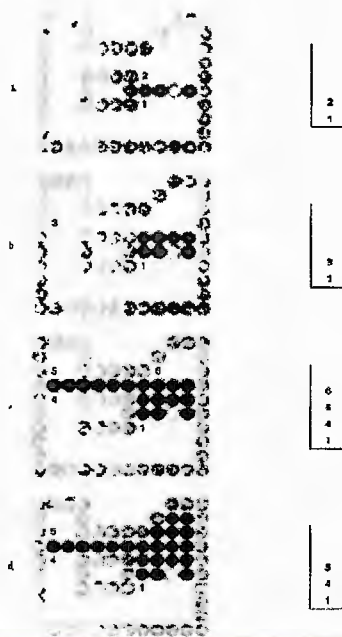


Figura No. 41. Algoritmo por inundación.

mismo que generar posiciones de píxel a lo largo de la frontera de la curva y podemos efectuarlo con el método del punto medio<sup>41</sup>, así, sólo se llena las extensiones de píxeles horizontales entre los puntos de la frontera en lados opuestos de la curva. Se emplea simetría entre cuadrantes (solo en circunferencias) para reducir los cálculos de fronteras<sup>42</sup>.

### **2.3.5.2. Algoritmo de inundación de áreas con fronteras curvas.**

Para el llenado de áreas circulares, consiste en iniciar en un punto adentro de una región y pintar el interior hacia afuera en dirección de la frontera. Si la frontera se especifica en un color particular, el algoritmo de llenado procede hacia afuera píxel por píxel hasta que se encuentra el color de la frontera. Un procedimiento para llenar fronteras acepta como entrada la coordenada de un punto interior (x,y), un color de llenado y un color de frontera. Al iniciar desde (x,y), el procedimiento prueba posiciones cercanas para determinar si son del color de la frontera. Si no es así, se pintan con el color de llenado y se prueban sus posiciones cercanas. Este procedimiento continúa hasta probar todos los píxeles hasta de frontera para el área. Se puede establecer tanto la frontera interior como la exterior para especificar un área. Los algoritmos por inundación quizá no llenen las regiones de manera correcta si algunos píxeles interiores ya sea despliegan en el color de llenado<sup>43</sup>.

---

<sup>41</sup> Ver el tema “2.2.3. Trazado de circunferencias”, con el sub-tema “2.2.3.1 Método Trigonométrico”, en la pagina No.

<sup>42</sup> Hacer referencia al tema “2.3.1. Algoritmo de exploración de línea”, desde la pagina No.

<sup>43</sup> Tomar como referencia el tema “2.3.2 Algoritmo de inundación”, desde la página No.

## **2.4. Transformaciones de coordenadas<sup>44</sup>.**

### **2.4.1. Traslación, Escalamiento y Rotación.**

Una de las mayores virtudes de los gráficos generados por ordenador es la facilidad con que se pueden realizar algunas modificaciones sobre las imágenes. Un arquitecto puede ver un edificio desde distintos puntos de vista. Un cartógrafo puede cambiar la escala de un mapa. Un animador puede modificar la posición de un personaje.

Conociendo ya las herramientas Básicas de construcción de figuras geométricas en dos dimensiones, el siguiente paso es las transformaciones bidimensionales, que no es más que otra alternativa de manipular las figuras de diferente forma. Estos cambios son fáciles de realizar porque la imagen gráfica ha sido codificada en forma de números susceptibles a las operaciones matemáticas.

Para realizar este tipo de transformaciones se debe conocer conceptos, afinidades en dos dimensiones y de considerar algunas propiedades de las transformaciones isometría. Esto se logra modificando los factores que definen el sistema de coordenadas: ángulos de los ejes, ubicación del origen, factor de escala, etc.

Las transformaciones isométricas básicas son:

- Traslación: cambios en la posición
- Rotación: cambios en la orientación
- Escalado: cambios en el tamaño

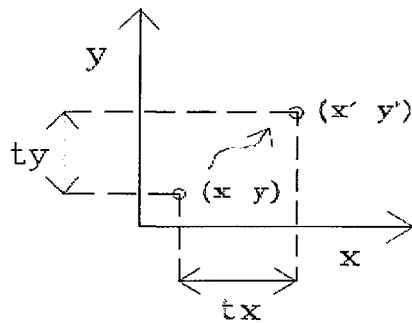
---

<sup>44</sup> Transformaciones de coordenadas: son las que permiten alterar de una forma uniforme toda la imagen, definiendo el espacio en forma numérica. Lengyel Eric. *Mathematics for 3D Game Programming And Computer Graphics*. Capítulo 3 Transforms, páginas 54-55.

## Traslación<sup>45</sup>

Consiste en moverse de un punto determinado a otra localización, Cada punto  $P(x,y)$  se traslada  $t_x$  unidades paralelas al eje  $x$  y  $t_y$  unidades paralelas al eje  $y$ , se obtiene un nuevo punto  $P'(x',y')$ .

Las ecuaciones de la transformación son:  $x' = x + t_x$        $y' = y + t_y$  (ec55)



En este caso,  $(x',y')$  denota las nuevas coordenadas del objeto trasladado en el sistema de coordenadas original.

El par de distancia de traslación  $(t_x, t_y)$  se denomina también vector traslación.

Figura No. 42. Desplazamiento de  $(x,y)$  a un punto  $(x',y')$

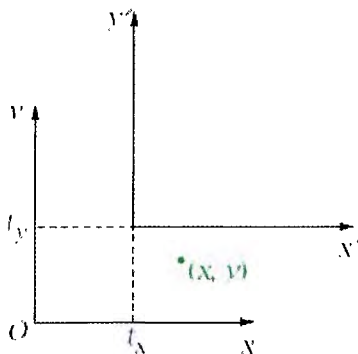


Figura No. 43. Desplazamiento hacia posición original.

En este caso,  $(x',y')$  denota las coordenadas del objeto antiguo en el sistema de coordenadas trasladadas

$$(ec56) \quad X = x' - t_x \quad Y = y' - t_y$$

Se puede aplicar la fórmula de traslación a todos los puntos de un objeto y extremos (válido también para rotaciones y escalados), cuando se trasladan líneas rectas solo se debe desplazar sus

extremos, lo contrario de los polígonos ya que se mueve su vértice y se redibuja, para cambiar la posición de una circunferencia o elipse, se trasladan las

<sup>45</sup> Traslación: es el movimiento en línea recta de cuerpo rígido trasladándolo a de una posición a otra. Hearn Donald Graficado por Computadora. Editorial Prentice Hall. Capítulo 5. página 116

coordenadas centrales y se vuelve a trazar la figura en la nueva localidad.(ver figura No.44)

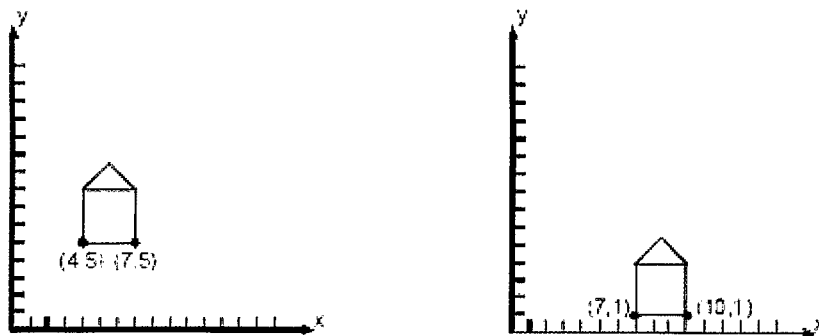


Figura No. 44. Traslación mueve la figura en el espacio pero no puede cambiarla de ninguna manera.

Las distancias de traslación pueden especificarse como cualquier número real (positivo, negativo o cero). Si un objeto se traslada mas allá de los límites del despliegue en coordenadas del dispositivo, el sistema podría retornar un mensaje de error **suprimir partes del objeto que sobrepasa presentar una imagen distorsionada.**

## Rotación<sup>46</sup>.

Para entender en totalidad que es rotación se debe conocer brevemente algunos conceptos trigonométricos. Sea un punto  $p_1=(x_1, y_1)$  y lo giramos alrededor del origen un ángulo<sup>47</sup>  $\theta$  para pasar a una nueva posición  $p_2=(x_2, y_2)$ . Queremos encontrar la transformación que convierte  $(x_1, y_1)$  en  $(x_2, y_2)$ . Pero, antes de comprobar si alguna transformación es la adecuada, debemos saber primero que  $(x_2, y_2)$  debe escribirse en función de  $(x_1, y_1)$  y  $\theta$ . Para esto es necesario recordar las razones trigonométricas<sup>48</sup> de seno y coseno.

<sup>46</sup> Rotación: giro de los puntos, un ángulo  $\theta$  respecto del origen.

<sup>47</sup> Ángulo: es la figura formada por dos lados con un punto común, llamado vértice.

<sup>48</sup> Razones trigonométricas: son las razones trigonométricas de un ángulo, que se denominan respectivamente: seno, coseno, tangente, cotangente, secante y cosecante. Larson.Hosteller. Cálculo. Sexta Edición Volumen1.

A la vista de la Figura No.45. Sabemos que:

$$(ec57) \quad \text{Sen}\Theta = \frac{y}{\sqrt{x^2 + y^2}} \quad \cos\Theta = \frac{x}{\sqrt{x^2 + y^2}}$$

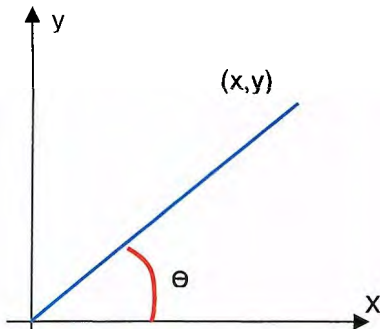


Figura No. 45. Definición de ángulo: figura formada por dos lados con un punto común, llamado vértice.

Es importante señalar que cuando la longitud del segmento es la unidad

$$(ec58) \quad \text{Sen}\Theta = y \quad \cos\Theta = x$$

También emplearemos las siguientes relaciones trigonométricas para determinar como gira un punto:

$$(ec59) \quad \begin{aligned} \cos(\Theta + \emptyset) &= \cos\emptyset \cos\Theta - \text{sen}\emptyset \text{sen}\Theta \\ \text{Sen}(\Theta + \emptyset) &= \cos\emptyset \text{sen}\Theta + \text{sen}\emptyset \cos\Theta \end{aligned}$$

Los valores positivos de  $\theta$  en estas ecuaciones indican una rotación contraria a la del reloj y los valores negativos hacen girar los objetos en el sentido del reloj

Ahora bien, se aplica una rotación bidimensional en un objeto al cambiar su posición a lo largo de la trayectoria de una circunferencia en el plano de x y. Para generar una rotación especificamos un ángulo de rotación y la posición  $Xr'$   $Yr'$  del punto de rotación o punto pivote en torno al cual se gira el objeto.

Cuando se rota líneas rectas solo se debe desplazar sus extremos, lo contrario de los polígonos ya que giran su vértice y se redibuja.

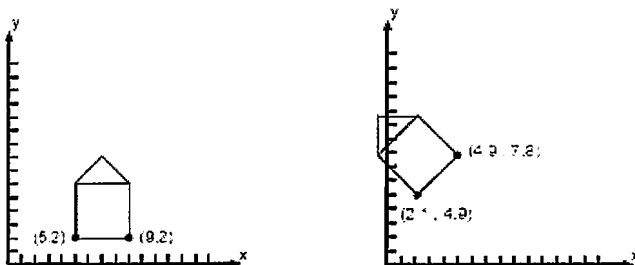


Figura No. 46. Rotación de la figura en el espacio pero sin cambia de ninguna su forma.

Determinación de rotación de un punto alrededor del origen, según la figura No.47

A la vista de esta figura tenemos:

$$(ec60) \quad \text{sen}\Theta = y/L \quad \text{Cos}\Theta = x/L$$

Donde  $L$  es la distancia del punto al origen de coordenadas. Por otro lado:

$$(ec61) \quad \text{sen}(\Theta + \emptyset) = y_2/L = \cos\Theta + \text{sen}\emptyset\cos\Theta$$

que nos lleva a:

De forma análoga:

$$y_2 = \cos\Theta\text{sen}\emptyset L + \text{sen}\emptyset\cos\Theta L = \cos\Theta y_1 + \text{sen}\emptyset x_2$$

$$\text{Cos}(\Theta + \emptyset) = x_2 = \cos\emptyset\cos\Theta - \text{sen}\emptyset\text{sen}\Theta$$

$$\text{Dando:}(ec63) \quad x_2 = \cos\emptyset\cos\Theta L - \text{sen}\emptyset\text{sen}\Theta L = \cos\Theta x_1 + \text{sen}\emptyset y_1$$

Se debe tener en cuenta que esto es cuando se realiza respecto al origen, estas se convierten en las nuevas coordenadas. Sin embargo se puede crear cierta distorsión en ese caso las nuevas coordenadas son:

$$(ec64) \quad x' = x + y y^{**}a \quad y' = y + x x^{**}b$$

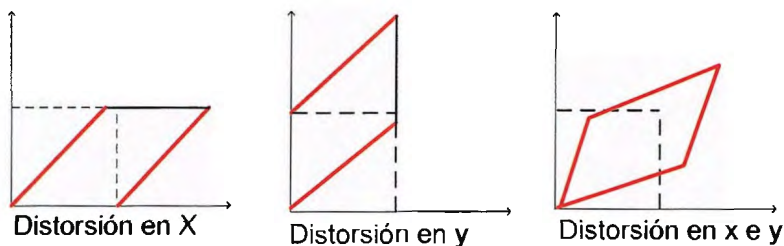


Figura No. 48. Muestra un ejemplo de distorsión en los ejes (x,y)



## Escalamiento<sup>49</sup>.

Es una transformación afines llamada así debido a que modifica cuerpos rígidos, además de otras transformaciones que preservan las líneas rectas, su función es cambiar el tamaño y posición del objeto en relación a un punto, si se realiza respecto al origen las nuevas coordenadas son :

$$(ec65) \quad x' = x \cdot S_x \quad y' = y \cdot S_y$$

La operación para polígonos se da al multiplicar los valores de coordenadas (x,y) de cada vértice por los factores de escalación  $s_x$  y  $s_y$  para producir las coordenadas transformadas ( $x'$  ,  $y'$ ).

El factor de escalado  $s_x$  escala objetos en la dirección de x, mientras que el factor de escalación  $s_y$  lo hace en la dirección de y.

Cuando se asignan el mismo valor a  $s_x$  y  $s_y$  se genera una escala uniforme, al asignar valores distintos a  $s_x$  y  $s_y$  se obtiene una escala diferencial.

Podemos encontrar la localización de un objeto escalonado al seleccionar una posición llamada punto fijo, que debe permanecer sin cambio después de la transformación de escalado.

Para realizar el escalado se considera lo siguiente:

- Si  $|S_x| > 1$  y  $|S_y| > 1$  aumenta el tamaño, Si  $|S_x| < 1$  y  $|S_y| < 1$  disminuye
- Si  $S_x = S_y$  escalado uniforme, Si  $S_x \neq S_y$  escalado no uniforme
- Si  $S_x < 0$  el objeto se refleja respecto al eje Y
- Si  $S_y < 0$  el objeto se refleja respecto al eje X
- El escalamiento por un número mayor que 1 aleja un objeto del origen.
- El escalamiento por un número fraccional acerca un objeto al origen

---

<sup>49</sup> Escalado: transformación que cambian el tamaño y la proporción de la imagen.



En la figura No.49. Se puede observar la función que realiza el escalado.

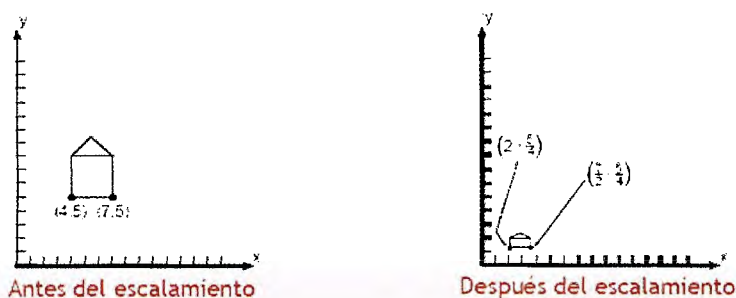


Figura No. 49. La imagen muestra tanto la modificación de tamaño como de posición. Escala 1/2 en el eje x y 1/4 en el eje y el escalado se efectúa con respecto al origen.

Cuando un objeto se vuelve a trazar con las ecuaciones de escalación la longitud de cada línea de la figura se hace a escala de acuerdo con los valores asignados a  $S_x$  y  $S_y$ , la distancia de cada vértice al origen también se hace a escala, se puede controlar la localidad de un objeto a escala eligiendo una posición llamada punto fijo<sup>50</sup>. Las coordenadas del punto fijo ( $X_f$ ,  $Y_f$ ) pueden elegirse como uno de los vértices, el centro del objeto o cualquier otra posición.

Para un vértice ( $x$ ,  $y$ ) las coordenadas a escala ( $x'$ ,  $y'$ ) se calculan como:

$$(ec66) \quad X' = x_f + (x - x_f) S_x, \quad Y' = y_f + (y - y_f) S_y$$

Se pueden reacomodar los términos de estas ecuaciones para obtener transformaciones de escalón relativas a un punto fijo seleccionado como:

$$(ec67) \quad X' = x \cdot S_x + (1 - S_x)x_f \quad Y' = y \cdot S_y + (1 - S_y)y_f$$

Donde los términos  $(1 - S_x)x_f$  y  $S_y + (1 - S_y)y_f$  son constantes para todos los puntos contenidos en el objeto.

<sup>50</sup> Punto fijo: es un punto que permanece inalterable después de la transformación de escalación.

Cuando se escalan líneas rectas solo se recorren sus extremos, lo contrario de los polígonos ya que escalamos su vértice y se redibuja, una circunferencia se hace a escala ajustando el radio y posiblemente reposicionando el centro de la circunferencia.

### **2.4.2. Representaciones Matriciales<sup>51</sup>.**

Las transformaciones anteriores traslación, rotación y escalamiento pueden representar de manera matricial para mayor eficiencia en el manejo de estas, incluyendo las constantes en una matriz cuadrada.

Muchas aplicaciones incluyen secuencias de transformaciones geométricas: una animación requiere que los objetos se trasladen y roten en cada fotograma, un diseño CAD requiere muchas transformaciones hasta obtener el resultado final en forma directa a partir de las coordenadas iniciales básicas en forma de matriz.

Desafortunadamente, la forma expuesta anteriormente de cómo describir la traslación, rotación y escalación no hace uso de matrices, por lo tanto no podría ser combinada con las otras transformaciones mediante una simple multiplicación de matrices. Tal combinación sería deseable; por ejemplo, hemos visto que la rotación alrededor de un punto que no sea el origen puede realizarse mediante una traslación, una rotación u otra traslación.

Sería deseable combinar estas tres transformaciones en una sola transformación por motivos de eficacia y elegancia. Una forma de hacer esto es emplear matrices 3x3 en vez de matrices 2x2, introduciendo una coordenada auxiliar  $w$ . Este método recibe el nombre de coordenadas homogéneas<sup>52</sup>.

---

<sup>51</sup> McGREGOR, Jim. THE ART OF MICROCOMPUTER GRAPHICS. Editorial Addison Wesley. Capítulo 1. Representaciones Vectoriales y Matriciales.

<sup>52</sup> Coordenada Homogénea: consiste en escribir ecuaciones de transformación en forma de matriz Hearn Donald Graficado por Computadora. Editorial Prentice Hall. Capítulo 5. página 119.

En estas coordenadas, los puntos están definidos por tres coordenadas y no por dos. Así un punto  $(x, y)$  estará representado por la tripleta  $(xw, yw, w)$ . Las coordenadas  $x$  e  $y$  se pueden recuperar fácilmente dividiendo los dos primeros números por el tercero respectivamente. En dos dimensiones su valor suele ser 1 para simplificar por tanto son útiles en tres dimensiones Sin embargo, se debe entender de forma general para tener conocimientos en cualquier caso a presentar.

Al utilizar este tipo de representaciones se debe tener en cuenta de forma muy eficiente toda la secuencia de transformaciones por ejemplo:

- Cada transformación puede representarse como  $P' = P M1 + M2$
- La matriz  $M1$  contiene la información de ángulos y factores de escala
- La matriz  $M2$  contiene los términos de traslación asociados al punto fijo y al centro de rotación
- Para producir una secuencia de transformaciones hay que calcular las nuevas coordenadas en cada transformación.

$$P'' = P' M3 + M4 = \dots = P M1 M3 + M2 M3 + M4$$

- Buscamos una solución más eficiente que permita combinar las transformaciones para obtener directamente las coordenadas finales a partir de las iniciales

### Matrices para traslación:

Las ecuaciones de traslación se convierten en:

Mediante el uso de esta notación, se puede escribir la forma matricial de las ecuaciones de traslación en forma más compacta como:

$$(ec67) \quad \begin{pmatrix} X' \\ Y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad T(t_x, t_y) = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

$P' = P \cdot T(T_x, T_y)$  donde:

$P' = [x' \ y' \ 1]$  y  $P$  son las matrices 1 por 3 en los cálculos matriciales.

### Matrices para rotación:

La transformación de rotación se convierte en:

$$(ec68) \quad \begin{pmatrix} X' \\ Y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad R\Theta = \begin{pmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Mediante el uso de esta notación, se puede escribir la forma matricial de las ecuaciones de rotación de la forma:  $P' = P \cdot R(\theta)$ .

### Matrices para escalamiento:

La transformación de rotación se convierte en

$$(ec69) \quad \begin{pmatrix} X' \\ Y' \\ 1 \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad S(S_x, S_y) = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

De forma matricial se escribe así:  $P' = P \cdot S(S_x, S_y)$

## 2.4.3. Transformaciones compuestas y eslabonamientos.

### Transformaciones compuestas.<sup>53</sup>

Para cualquier secuencia de transformaciones, podemos calcular la matriz de transformación compuesta, calculando el producto de las transformaciones individuales. La formación de productos de producto de matrices de transformación por lo general se conoce como eslabonamiento o composición de matrices

#### - En el caso de traslaciones sucesivas de un objeto.

Se efectúa primero las matrices de traslación, aplicando después la matriz compuesta a los puntos coordenados. Especificando las dos distancias de traslación sucesivas como  $(T_{x1}, T_{y1})$  y  $(T_{x2}, T_{y2})$  se calcula la matriz compuesta así:

$$(ec70) \quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_{x1} & T_{y1} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_{x2} & T_{y2} & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_{x1}+T_{x2} & T_{y1}+T_{y2} & 1 \end{pmatrix}$$

---

<sup>53</sup> Nemirovsky, The three-dimensional world – manipulations. Capítulos 8 y 9.

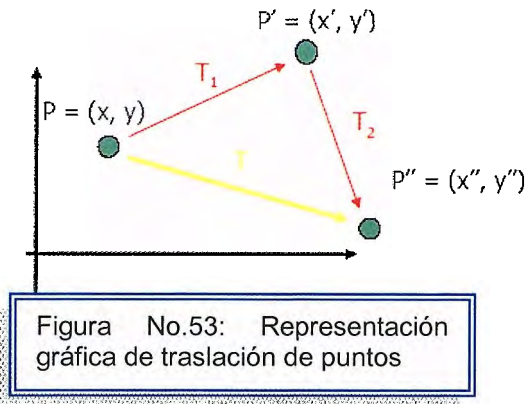
Esto demuestra que dos traslaciones son aditivas. Matricialmente la transformación de puntos se escribe

así:

$$P' = P.T(t_{x1}, t_{y1})$$

$$P'' = P'.T_2(t_{x2}, t_{y2})$$

$$(ec71) \quad P'' = P'.T_2 = P.T_1.T_2 = P.T$$



o de la siguiente forma:

$$(ec72) \quad P' = P.T(T_{x1} + T_{y2}, T_{y1} + T_{y2})$$

- En rotaciones sucesivas se establece así:

$$R(\theta) \cdot R(\Phi) = R(\theta + \Phi)$$

Matricialmente es así:

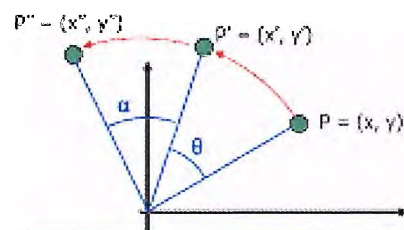
Como sucede en el caso de las traslaciones, las rotaciones sucesivas son aditivas.

También se puede escribir de siguiente la forma según la Figura No.54 :

$$P' = P.R(\theta)$$

$$P'' = P'.R(\alpha)$$

$$(ec73) \quad P'' = P'.R(\alpha) = P.R(\theta)R(\alpha) = P.R$$



$$(ec74) \quad \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\Phi & \sin\Phi & 0 \\ -\sin\Phi & \cos\Phi & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos(\theta + \Phi) & \sin(\theta + \Phi) & 0 \\ -\sin(\theta + \Phi) & \cos(\theta + \Phi) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- En escalados sucesivos se establece así :

Matricialmente es:  $S(S_{x1}, S_{y1}) \cdot (S_{x2}, S_{y2}) = S(S_{x1} \cdot S_{x2}, S_{y1} \cdot S_{y2})$

$$\begin{aligned} P' &= P \cdot S_1(S_{x1}, S_{x2}) \\ P'' &= P' \cdot S_2(S_{x2}, S_{y2}) \\ \text{(ec75)} \quad P'' &= P' \cdot S_2 = P \cdot S_1 \cdot S_2 \end{aligned}$$

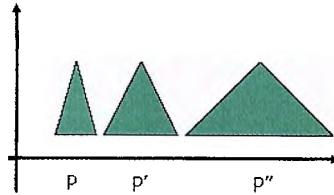


Figura No. 55. Representación de la secuencia de escalados.

donde:

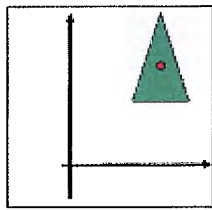
$$\text{(ec76)} \quad S = \begin{pmatrix} S_{x1} & 0 & 0 \\ 0 & S_{y1} & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} S_{x2} & 0 & 0 \\ 0 & S_{y2} & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} S_{x1}S_{x2} & 0 & 0 \\ 0 & S_{y1}S_{y2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

### - Escalación relativa a un punto fijo.

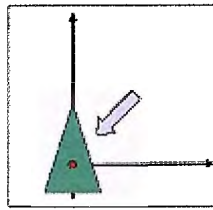
Utilizando las matrices de transformación de la traslación y reduciendo a escala se puede obtener la matriz compuesta para la escalación con respecto a un punto fijo  $(x_f, y_f)$  considerando una secuencia de tres transformaciones.

Primero, todas las coordenadas se trasladan de modo que el punto fijo se mueva al origen coordenado. Segundo, las coordenadas se reducen a escalas con respecto al origen. Tercero, las coordenadas se trasladan de manera que el punto fijo se devuelve a su posición original (ver figura No.56). La multiplicación de matrices de esta secuencia produce:

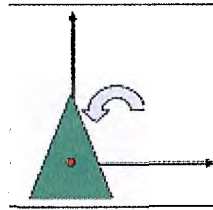
$$\text{(ec77)} \quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_1 & -y_1 & 1 \end{pmatrix} \cdot \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_f & y_f & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ T_{x1}+T_{x2} & T_{y1}+T_{y2} & 1 \end{pmatrix}$$



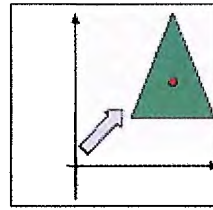
(a) posición original del objeto.



(b) traslación del objeto de manera que el punto fijo este en el origen



(c) Objeto a escala respecto al origen.



(d) traslación de punto fijo de manera que se devuelva a la posición original.

Figura No. 56. Secuencia de transformación que se necesita para reducir a escala un objeto con respecto al origen

### Rotación alrededor de un punto fijo.

Una sucesión de transformaciones para obtener la matriz compuesta de la rotación con respecto a un punto fijo  $(x_r, y_r)$  es: Primero, el objeto se traslada de manera que el punto pivote coincida con el origen. Segundo, se hace girar el objeto alrededor del origen. Tercero, el objeto se traslada de manera que el punto fijo coincida con su posición original.

$$(ec78) \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_r & -y_r & 1 \end{pmatrix} \cdot \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_r & y_r & 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ (1-\cos\theta)x_r + y_r\sin\theta & (1-\cos\theta)y_r - x_r\sin\theta & 1 \end{pmatrix}$$



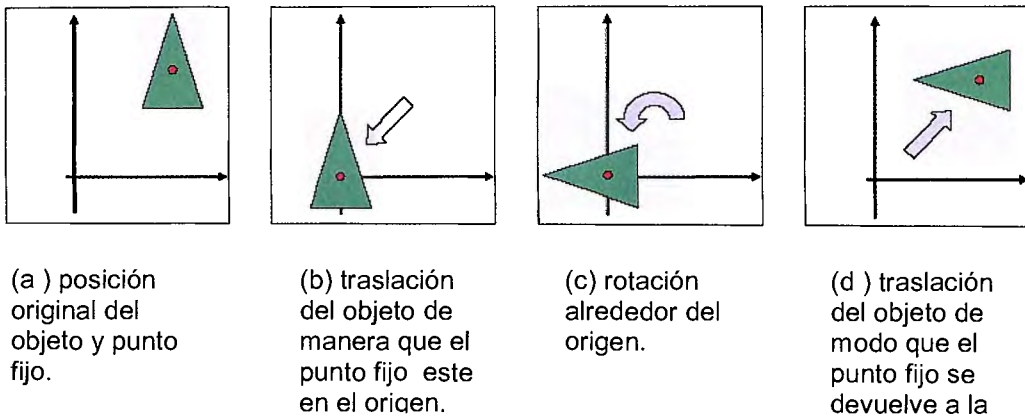


Figura No. 57. Secuencia de transformaciones que se necesitan para hacer girar un objeto alrededor de un punto fijo mediante matrices de transformación.

## Eslabonamiento.

Es la unión o concatenación de las diferentes transformadas para darle mas complejidad a la imagen o el diseño. Estas uniones deben cumplir un orden y algunas consideraciones como:

1. El producto de matrices no es conmutativo  $M1 \cdot M2 \neq M2 \cdot M1$ , por lo tanto, en general la aplicación de transformaciones tampoco lo es.
2. Transformaciones que si son conmutativas Traslación-Traslación, Escalado-Escalado, Rotación-Rotación, Escalado Uniforme-Rotación.
3. Transformaciones que no son conmutativas Traslación-Escalado, Traslación-Rotación, Escalado No Uniforme-Rotación.
4. Si se utiliza un vector fila, el orden de aplicación cambia las matrices que se utilizan son las traspuestas.

Por ejemplo: Rotación respecto a un punto cualquiera (  $x_c$  ,  $y_c$  ) en tres pasos:  
 Traslación ( -  $x_c$  , -  $y_c$  ) Rotación y Traslación ( )  $x_c$  ,  $y_c$  ) , Como las matrices son  
 cuadradas se obtiene una única matriz  $P3 = T( x_c , y_c ) ** R ** T(- x_c , - y_c ) ** P$

$$(ec79) \quad \begin{pmatrix} X_3 \\ Y_3 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & x_c \\ 0 & 1 & y_c \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\Phi & -\sin\Phi & 0 \\ \sin\Phi & \cos\Phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & -x_c \\ 0 & 1 & -y_c \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} \cos\Phi & -\sin\Phi & (x_c - \cos\Phi \cdot X_c + \sin\Phi \cdot y_c) \\ \sin\Phi & \cos\Phi & (y_c - \sin\Phi \cdot X_c - \cos\Phi \cdot y_c) \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$$

## EJEMPLOS GRAFICADO EN 2D

**Ejemplo 2.1:** a) Realizar la rotación en 2D de un punto  $P(x,y)$  en torno al origen y en sentido antihorario, donde  $C$  es el ángulo de rotación y  $P'(x',y')$  es el punto después de aplicarle la rotación, b) expresarlo en forma matricial.

Solución:

a)  $x = r \cos a$   
 $y = r \sin a$

$$x' = r \cos b$$
$$y' = r \sin b$$

$$b = a + c$$

$$x' = r \cos(a+c)$$
$$y' = r \sin(a+c)$$

Sustituyendo  $\cos(a+c)$  y  $\sin(a+c)$

$$x' = r (\cos a \cos c - \sin a \sin c)$$
$$y' = r (\sin a \cos c + \cos a \sin c)$$

$$x' = r \cos a \cos c - r \sin a \sin c$$
$$y' = r \sin a \cos c + r \cos a \sin c$$

Reemplazando por  $r \cos a = x$  y  $r \sin a = y$ , obtenemos finalmente las ecuaciones de rotación de un punto en el plano.

$$R// \quad x' = x \cos c - y \sin c$$
$$y' = y \cos c + x \sin c$$

b) En forma de una matriz de  $2 \times 2$   $\text{Rot}(c)$ :

$$\begin{vmatrix} \cos(c) & \sin(c) \\ -\sin(c) & \cos(c) \end{vmatrix}$$

Aplicando  $\text{Rot}(c)$  a un punto  $|x, y|$ , se obtiene el punto transformado  $|x', y'|$

$$|x', y'| = |x, y| * \begin{vmatrix} \cos(c) & \sin(c) \\ -\sin(c) & \cos(c) \end{vmatrix}$$

$$| -\sin(c) \cos(c) |$$

$$|x', y'| = |x \cos(c) - y \sin(c), x \sin(c) + y \cos(c)|$$

Donde

$$R// \quad x' = x \cos(c) - y \sin(c)$$

$$y' = x \sin(c) + y \cos(c)$$

**Ejemplo 2.2:** Desarrolle una función del algoritmo de Bresenham para líneas en 2D, con la herramienta de OpenGL:

Solución:

```
#include <GL/glut.h>

void drawline(const int ax, const int ay, const int bx, const int by)
{
    const int w = bx - ax; // ancho (horizontal)
    const int h = by - ay; // alto (vertical)
    const int rightstep = 2 * h; // "f" aumenta para mover rt
    const int diagstep = 2 * (h - w); // "f" aumenta para mover rt y arriba

    int x; // x coord de pt actual
    int y = ay; // y coord de pt actual
    int f = 2 * h - w; // "f": valor de prueba de Bresenham

    glBegin(GL_POINTS);

    for (x = ax; x <= bx; ++x) // Ciclo principal de Bresenham
    {
        glVertex2i(x, y); // Plot del punto
        if (f < 0)
        {
            f += rightstep;
        }
        else
        {
            ++y;
            f += diagstep;
        }
    }

    glEnd();
}
```

```

}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0); // R
    drawline(5,15, 90,95); // Dibuja
    glFlush();
}

void init()
{
    glClearColor (0.0, 0.0, 0.0, 0.0);
    glPointSize(5.0); // Dibuja Puntos "grandes" (cuadros 5x5)
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0, 100, 0, 100, -1.0, 1.0);
    // La ventana va de 0 a 100 en ambas coords.
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);

    glutInitWindowSize(500, 500);
    glutInitWindowPosition(50, 50);
    glutCreateWindow("Algoritmo de Bresenham (lineas)");

    init();
    glutDisplayFunc(display);

    glutMainLoop();

    return 0;
}

```

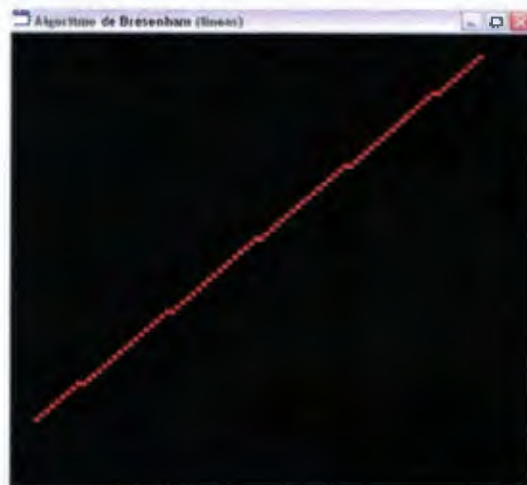


Figura No. 58: Salida del Programa

**Ejemplo 2.3:** Una circunferencia se define como un conjunto de puntos que se encuentran, en su totalidad, a una distancia determinada  $r$  de una posición central  $(x_r, y_r)$ . Realizar un programa con la herramienta OpenGL que muestre como salida un círculo 2D.

Solución:

```
#include <GL/glut.h>
#include <math.h>

circunferencia(int x, int y, int radio){
    int i,puntos=90;
    glBegin(GL_POINTS);
    for(i=0;i<puntos;i++){
        glVertex3f(x+radio*cos((2*3.14*i)/puntos),y+radio*sin((2*3.14*i)/puntos),0);
    }
    glEnd();
}

void dibuja() {
    glClear(GL_COLOR_BUFFER_BIT);
    circunferencia(0,0,100);
    glFlush();
}

int main(int argc, char** argv) {
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_RGB);
    glutInitWindowPosition(0,0);
    glutInitWindowSize(500,500);
    glutCreateWindow("Circunferencia");
    glOrtho(-200,200,-200,200,-200,200);
    glClearColor(0,0,0,0);
    glColor3f(1,1,1);
    glutDisplayFunc(dibuja);
    glutMainLoop();
    return 0;
}
```

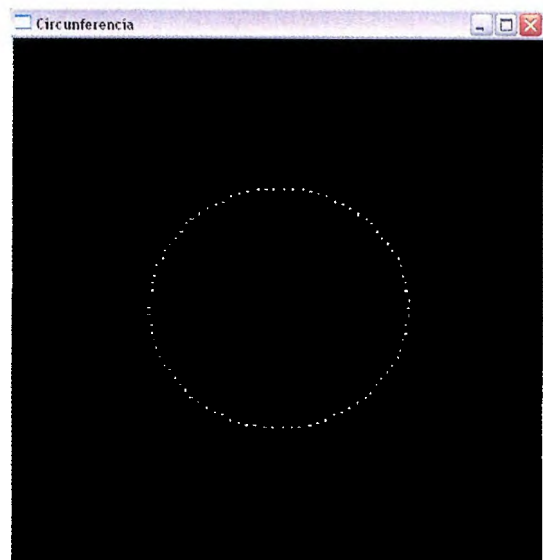


Figura No.59: Salida del Programa

**Ejemplo 2.4:** Calcule la rotación de una matriz de 3x3, con un ángulo de rotación de 30° sobre los ejes x, y, z:

Solución:

a) Sobre el eje x:

$$R_x(\Theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\Theta & -\sin\Theta \\ 0 & \sin\Theta & \cos\Theta \end{pmatrix}$$

Donde  $\Theta = 30^\circ$ , por lo tanto:

$$\cos \Theta = 0.866 = \sqrt{3} / 2 \quad \sin \Theta = 1/2$$

$$R_x(\Theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \sqrt{3}/2 & -1/2 \\ 0 & 1/2 & \sqrt{3}/2 \end{pmatrix}$$

b) Sobre el eje y:

$$R_y(\Theta) = \begin{pmatrix} \cos\Theta & 0 & \sin\Theta \\ 0 & 1 & 0 \\ -\sin\Theta & 0 & \cos\Theta \end{pmatrix}$$

$$R_y(\Theta) = \begin{pmatrix} \sqrt{3}/2 & 0 & 1/2 \\ 0 & 1 & 0 \\ -1/2 & 0 & \sqrt{3}/2 \end{pmatrix}$$

c) Sobre el eje z:

$$R_z(\Theta) = \begin{pmatrix} \cos\Theta & -\sin\Theta & 0 \\ \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$R_z(\Theta) = \begin{pmatrix} \sqrt{3}/2 & -1/2 & 0 \\ 1/2 & \sqrt{3}/2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$



**Ejemplo 2.5:** Realizar la función del Algoritmo DDA para líneas con pendiente positiva, con ayuda de la herramienta de OpenGL.

Solución:

```
#include <GL/glut.h>
#include <math.h>

inea (int x0, int x1, int y0, int y1, int valor)
{
    /* x varía de x0 a x1 en incrementos unitarios */
    float m = (y1-y0)/(x1-x0);
    float x = x0;
    float y = y0;
    while (x < x1){
        putpixel(round(x),round(y),valor); /*Asignar el píxel igual al valor */
        y += m; /* incrementar y por la pendiente m */
        x++;
    } //fin del while
}

Void main() {
    glutInitWindowPosition(0,0);
    glutInitWindowSize(500,500);
    glutCreateWindow("Algoritmo DDA para Lineas con m positiva");
    glOrtho(-200,200,-200,200,-200,200);
    glClearColor(0,0,0,0);
    glColor3f(3,3,3);
    glutDisplayFunc(inea);
    glutMainLoop();
    return 0;
}
```



**Ejemplo 2.6:** Cambiar la escala del punto  $P = [x_a \ y_a]$ . A partir de la matriz:

$$M = \begin{pmatrix} 4 & 0 \\ 0 & 7 \end{pmatrix}$$

Solución:

Se define el punto  $P$  como una matriz  $1 \times 2$ , y determinamos otro punto multiplicando por la matriz  $M$ :

$$[x_{a_2} \ y_{a_2}] = P_2 = PM$$

$$\text{Por lo tanto} \quad [x_{a_2} \ y_{a_2}] = [x_a \ y_a] \begin{pmatrix} 4 & 0 \\ 0 & 7 \end{pmatrix}$$

Realizamos la multiplicación de matrices y obtenemos el resultado de la nueva escala del punto  $P$

$$\mathbf{R//} \quad [x_{a_2} \ y_{a_2}] = [4x_a \ 7y_a]$$

# 3

## **GRAFICADO EN TRES DIMENSIONES**

---

## **3.1 Introducción**

Hasta hoy hemos enfocado nuestra atención en gráficas bidimensionales. Pero realmente vivimos en un mundo tridimensional, y en muchas aplicaciones de diseño debemos manejar y describir objetos tridimensionales.

Si un arquitecto deseara ver el aspecto real de la estructura, entonces un modelo tridimensional le permitiría observarla desde diferentes puntos de vista. Un diseñador de aviones podría desear analizar el comportamiento de la nave bajo fuerzas y tensiones tridimensionales. En este caso se necesita también una descripción tridimensional. Algunas aplicaciones de simulación, como el aterrizaje de un avión, también exigen una definición tridimensional del mundo.

En este capítulo generalizaremos nuestro sistema para manejar modelos de objetos tridimensionales. Extenderemos las transformaciones que hemos visto para permitir traslaciones y rotaciones en el espacio tridimensional. Ya que la superficie de visualización es sólo bidimensional, debemos considerar la forma de proyectar nuestros objetos en esta superficie plana para formar la imagen.

## 3.2 Representación Rectangular Tridimensional

El espacio tridimensional<sup>1</sup> está definido con tres rectas mutuamente perpendiculares. Estas rectas forman los ejes del sistema de coordenadas rectangulares (ejes X, Y, Z). (Ver figura No.1)

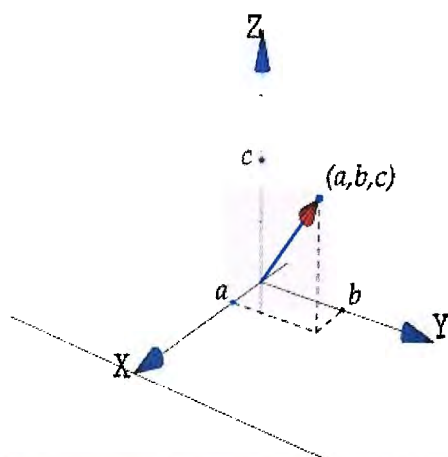


Figura No.1 representación de un punto C en los ejes tridimensionales

Esta referencia coordenada, puede utilizarse adecuadamente para especificar coordenadas de positivo, con la pantalla den el plano  $z=0$ . Entonces los valores mayores situados en el eje  $z$  positivo, se interpretan como si estuvieran más alejados del ordenador.

Otra disposición posible de los ejes coordenados es el sistema por la izquierda. Con el origen coordinado de un sistema por la izquierda colocado en la esquina inferior de la pantalla, el primer cuadrante del plano  $xy$  se sitúa directamente en las coordenadas de la pantalla y los valores positivos se  $Z$  indican posiciones ubicadas detrás de la misma. (ver la figura No.2)

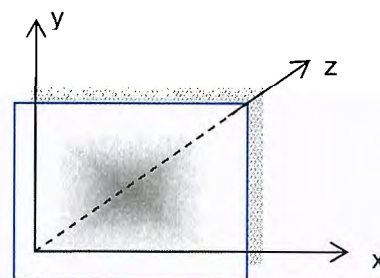


Figura No.2 ejes coordenados por la izquierda superimpuestos en la superficie de una pantalla de despliegue.

También pueden ser útiles otras referencias coordenadas al describir escenas tridimensionales. Para algunos objetos, quizá convenga definir posiciones

<sup>1</sup> El espacio tridimensional, se puede ver como un espacio de puntos y no como un espacio vectorial, tiene la ventaja de que no es necesario designar un punto especial como origen. Marco Paluszny, Hartmut Prautzsch Métodos de Bézier y B-splines. Capítulo 1 Nociones básicas y Espacios. Página No. 3-6

coordenadas mediante el uso de un sistema esférico, cilíndrico y otro que aproveche cualquier simetría que ocurra en forma natural.

Sin embargo, para paquetes de gráficas que permiten solo representaciones en coordenadas mundiales cartesianas, primero cualquier descripción coordenada no cartesiana debe ser convertida por el usuario en las coordenadas correspondientes.

A diferencia de las descripciones bidimensionales, un objeto tridimensional puede observarse desde cualquier posición. Podemos pensar que esto es análogo a fotografiar un objeto, donde el fotógrafo camina en torno al objeto y elige una posición y orientación para la cámara.

## **LÍNEAS EN EL ESPACIO 3D<sup>2</sup>**

Dados dos puntos 3D  $P_1$  y  $P_2$ , podemos definir la línea que pasa a través de dichos puntos parametricamente como:

$$(ec1) \quad \mathbf{P}(t) = (1-t)\mathbf{P}_1 + t\mathbf{P}_2$$

Donde el parámetro  $t$  oscila entre el rango de todos los números reales. El segmento de línea que conecta  $P_1$  y  $P_2$  corresponde a los valores de  $t$  entre 0 y 1.

Un marco es una línea que tiene un solo punto  $P_0$  y que se extiende al infinito definiendo una dirección  $\mathbf{V}$ . Los marcos son expresados comúnmente por ecuaciones paramétricas:

$$(ec2) \quad \mathbf{P}(t) = \mathbf{P}_0 + t\mathbf{V}$$

---

<sup>2</sup> Hearn, Donald, Gráficas Por Computadora. Editorial Prentice Hall. Capítulo 10 Representaciones Tridimensionales, páginas 197-202

Donde  $t$  puede variar entre valores mayores o iguales a cero. Esta ecuación es usada también para la representación de líneas; Nótese que ésta es muy equivalente a la ec1.

## DISTANCIA ENTRE UN PUNTO Y UNA LÍNEA

La distancia  $d$  se forma desde un punto  $Q$  a una línea definida por el punto de inicio  $P_0$ , en donde la dirección  $V$  puede ser encontrada calculando la magnitud del componente de  $Q-P_0$  que es perpendicular a la línea, como se muestra en la figura No.3

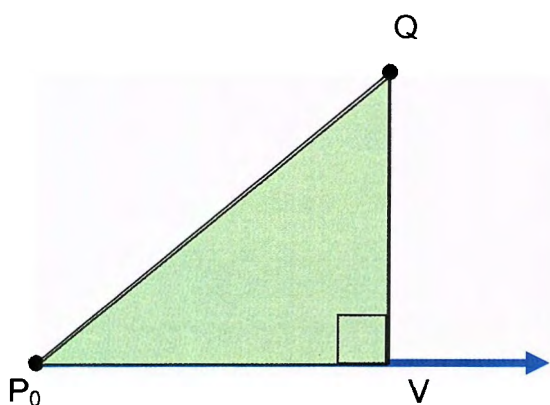


Figura No.3: calculando distancia desde un punto a una línea

Utilizando el teorema de Pitágoras, el cuadrado de la distancia entre  $Q$  y la línea, puede ser obtenido por la resta del cuadrado de la proyección  $Q-P_0$  y la dirección  $V$  desde el cuadrado de  $Q-P_0$ .

Así:

$$d^2 = (Q-P_0)^2 - [\text{proy.}_V(Q-P_0)]^2$$

$$d^2 = (Q-P_0)^2 - \left[ \frac{(Q-P_0) \cdot V}{V^2} \right]^2$$

$$(ec3) \quad d^2 = (Q-P_0)^2 - \left[ \frac{(Q-P_0) \cdot V}{V^2} \right]^2$$

Fácilmente, despejando  $d$  como raíz cuadrada, encontramos la distancia requerida:

$$d = \sqrt{(Q-P_0)^2 - \left[ \frac{(Q-P_0) \cdot V}{V^2} \right]^2}$$

## DISTANCIA ENTRE DOS LÍNEAS<sup>3</sup>

En dos dimensiones, dos líneas son paralelas o se intersectan en un punto, pero en 3D, esto no es verdad. Cuando dos líneas no son paralelas y tampoco se intersectan son llamadas *Líneas Targiversas*.

Suponiendo que nuestras dos líneas (ver figura No.4) definidas por las ecuaciones paramétricas:

$$P(s) = P_0 + sV_p$$

(ec4)  $Q(t) = Q_0 + tV_Q$

Don de S y t varían en el rango de todos los números reales. Entonces el cuadrado de la distancia entre un punto en la línea P(s) y un punto en la línea Q(t) puede escribirse como:

(ec4)  $F(s,t) = ||P(s) - Q(t)||^2$

Expandiendo los cuadrados y sustituyendo la definición de la función P(s) y Q(t) se tiene:

$$F(s,t) = P(s)^2 + Q(t)^2 - 2P(s).Q(t)$$

$$F(s,t) = (P_0 + sV_p)^2 + (Q_0 + tV_Q)^2 - 2P_0.Q_0 - 2sV_p.Q_0 - 2tV_Q.P_0 - 2stV_p.V_Q$$

(ec5)  $F(s,t) = P_0^2 + s^2V_p^2 + 2sP_0.V_p + Q_0^2 + t^2V_Q^2 + 2tQ_0.V_Q - 2P_0.Q_0 - 2sV_p.Q_0 - 2tV_Q.P_0 - 2stV_p.V_Q$

El valor mínimo definido por la función F, puede encontrarse aplicando derivadas parciales con respecto a S y t igual a cero. Así:

(ec6)  $\frac{\partial F}{\partial s} = 2sV_p^2 + 2P_0.V_p - 2V_p.Q_0 - 2tV_p.V_Q = 0$

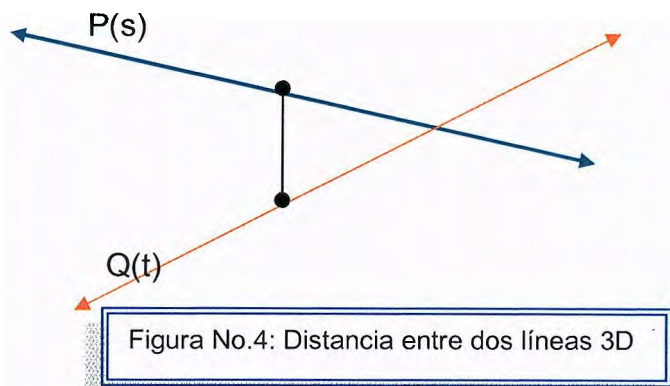


Figura No.4: Distancia entre dos líneas 3D

<sup>3</sup> Huw Jones, Computer Graphics. Through Key Mathematics. Editorial Springer Capítulo 7 Vectors: descriptions of spatial Relationship. Páginas 208-213

$$(ec7) \quad \frac{\partial F}{\partial t} = 2tV_Q^2 + 2Q_0.V_Q - 2V_Q.P_0 - 2sV_P.V_Q = 0$$

Luego eliminando el factor de 2, podemos escribir estas ecuaciones en forma de matrices:

$$(ec8) \quad \begin{bmatrix} V_P^2 & -V_P.V_Q \\ -V_P.V_Q & V_Q^2 \end{bmatrix} \begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} (Q_0 - P_0).V_P \\ (P_0 - Q_0).V_Q \end{bmatrix}$$

Resolviendo esta ecuación para S y t se tiene:

$$\begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} V_P^2 & -V_P.V_Q \\ -V_P.V_Q & V_Q^2 \end{bmatrix}^{-1} \begin{bmatrix} (Q_0 - P_0).V_P \\ (P_0 - Q_0).V_Q \end{bmatrix}$$

$$(ec9) \quad \begin{bmatrix} s \\ t \end{bmatrix} = \frac{1}{V_P^2 V_Q^2 - (V_P.V_Q)^2} \begin{bmatrix} V_Q^2 & V_P.V_Q \\ V_P.V_Q & V_P^2 \end{bmatrix} \begin{bmatrix} (Q_0 - P_0).V_P \\ (P_0 - Q_0).V_Q \end{bmatrix}$$

Estableciendo estos valores obtenidos de S y t, son sustituidos en la función F, dando como resultado el mínimo cuadrado de la distancia entre las dos líneas, la distancia neta obviamente se determina evaluando la raíz cuadrada.

Si la dirección del vector  $V_P$  y  $V_Q$  tiene la unidad como longitud, entonces la ecuación 9, se simplifica teniendo en cuenta que:  $V_P^2=1$  y  $V_Q^2=1$

Si  $V_P^2 V_Q^2 - (V_P.V_Q)^2$  es cero, entonces las líneas son paralelas, y la distancia entre las dos líneas es igual a la distancia entre cualquier punto en una de las líneas y la otra. (ver figura No.5). Particularmente la ec3, puede usarse para determinar la distancia desde el punto  $P_0$  a la línea  $Q(t)$  o la distancia desde el punto  $Q_0$  a la línea  $P(s)$ .

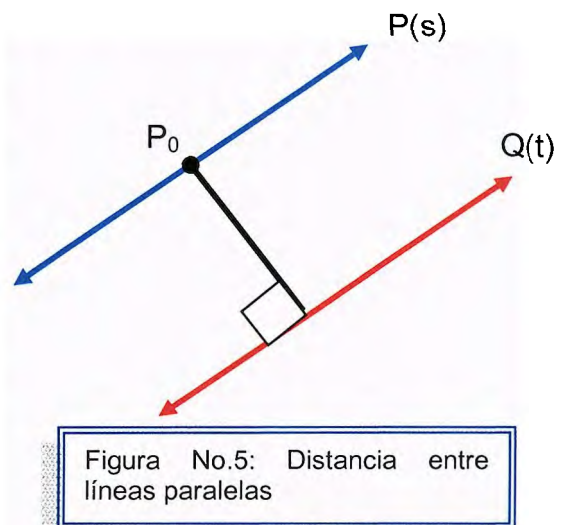


Figura No.5: Distancia entre líneas paralelas



## PLANOS EN EL ESPACIO 3D<sup>4</sup>

Dado un punto 3D  $P_0$  y un vector normal  $N$ , el plano pasando a través del punto  $P_0$  y perpendicular a la dirección  $N$ , puede ser definido como el grupo de puntos  $P$  semejantes a  $N \cdot (P - P_0) = 0$ .

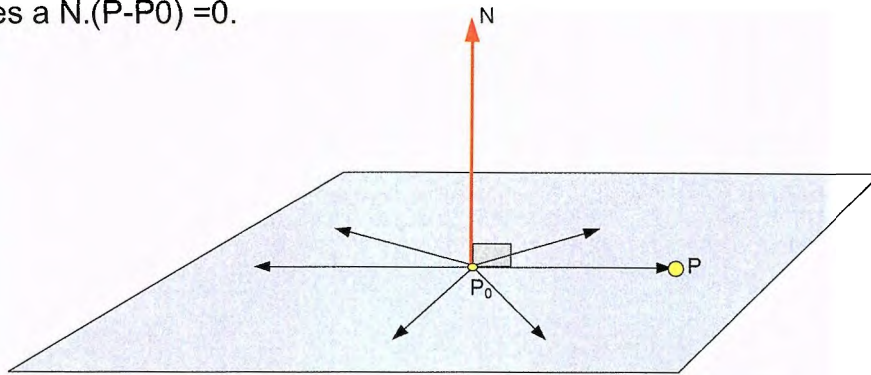


Figura No.6: Un plano está definido por los puntos  $P$  cuya diferencia con el punto  $P_0$ , es que al tenderse al plano, éstos son perpendiculares a la dirección del Vector Normal  $N$ .

La ecuación de un plano es comúnmente escrita como:

$$(ec10) \quad Ax + By + Cz + D = 0$$

Donde  $A, B$  y  $C$  son componentes de  $X$ -,  $Y$ - y  $Z$  es la componente del vector normal  $N$ , y  $D = -N \cdot P_0$  (ver figura No.7)

El vector normal  $N$ , esta también normalizado con la unidad de longitud, en este caso la ecuación de la distancia se define como:

$$(ec11) \quad d = N \cdot Q + D$$

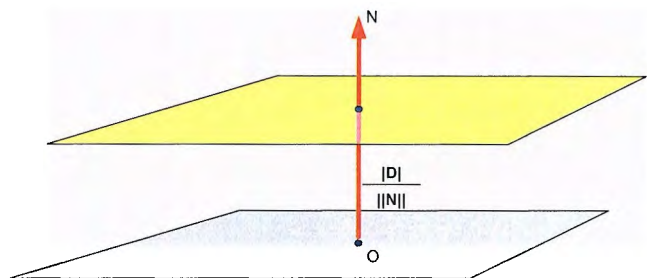


Figura No.7: El valor de  $D$  en la ec10 es proporcional a la distancia perpendicular desde el origen al plano.

<sup>4</sup> Lengyel Eric. Mathematics for 3D Game Programming and Computer Graphics. Editorial Charles Rivera Media. Capítulo 4. Ingeniería Geométrica. Páginas 83-88.

Dando de ésta manera el signo de la distancia desde el plano a un punto arbitrario de Q.

Si  $d=0$ , entonces el punto Q esta situado en el plano.

Si  $d>0$ , decimos que el punto Q esta situado en el lado positivo del plano.

Si  $d<0$ , decimos que el punto Q esta situado en lado negativo del plano.

## INTERSECCIÓN DE UNA LÍNEA Y UN PLANO<sup>5</sup>

Buscar el punto donde una línea intersecta un plano es un cálculo muy común en la ingeniería 3D.

Se tiene  $\mathbf{P}(t) = \mathbf{Q} + t\mathbf{V}$  representando una línea conteniendo el punto Q y corriendo paralelamente hacia la dirección V. Para un plano definido por la dirección normal N y el signo de la distancia D desde el origen, podemos encontrar el punto donde la línea intersecta el plano, resolviendo la siguiente ecuación:

$$(ec12) \quad \mathbf{N} \cdot \mathbf{P}(t) + D = 0$$

Para t, sustituyendo  $\mathbf{Q} + t\mathbf{V}$  para  $\mathbf{P}(t)$  tendríamos:

$$(ec13) \quad \mathbf{N} \cdot \mathbf{Q} + (\mathbf{N} \cdot \mathbf{V})t + D = 0$$

Y luego despejando t, llegamos a:

$$(ec14) \quad t = \frac{-(\mathbf{N} \cdot \mathbf{Q} + D)}{\mathbf{N} \cdot \mathbf{V}}$$

Ya teniendo los valores de t, regresamos a la ecuación  $\mathbf{P}(t) = \mathbf{Q} + t\mathbf{V}$  para identificar el punto de intersección.

Si  $\mathbf{N} \cdot \mathbf{V}$  es cero, entonces la línea es paralela al plano (el plano normal N es perpendicular a la línea de dirección V). En éste caso la línea se tiende al plano mismo si  $\mathbf{N} \cdot \mathbf{Q} + D = 0$ ; de lo contrario, no existe la intersección.

---

<sup>5</sup>D. Foley, James. Computer Graphics: Principles and Practice. Editorial: Addison Wesley. Capítulo 4. Viewing in 3D páginas 230-242

## INTERSECCIÓN DE TRES PLANOS<sup>6</sup>

Las regiones en el espacio están también definidas por una lista de planos que forman la frontera de un poliedro convexo.

Los bordes y vértices pertenecientes a éste poliedro, pueden ser encontrados por medio de una serie de cálculos que determinan los puntos donde los tres planos se intersectan.

Se tiene  $L1 = \langle N1, D1 \rangle$ ,  $L2 = \langle N2, D2 \rangle$  y  $L3 = \langle N3, D3 \rangle$  son tres planos cualquiera.

Podemos encontrar un punto Q que pertenezca a uno de los tres planos, resolviendo el siguiente sistema:

$$L1.Q = 0$$

$$L2.Q = 0$$

$$L3.Q = 0$$

Estas ecuaciones pueden ser escritas en forma matricial:

$$(ec15) \quad MQ = \begin{pmatrix} -D1 \\ -D2 \\ -D3 \end{pmatrix}$$

Donde la matriz M está dada por:

$$(ec16) \quad M = \begin{pmatrix} (N1)x & (N1)y & (N1)z \\ (N2)x & (N2)y & (N2)z \\ (N3)x & (N3)y & (N3)z \end{pmatrix}$$

Asumiendo que la matriz M esta invertida, resolviendo para el punto Q, produce un único punto donde los planos se intersectan:

$$(ec17) \quad Q = M^{-1} \begin{pmatrix} -D1 \\ -D2 \\ -D3 \end{pmatrix}$$

---

<sup>6</sup> D. Foley, James. Computer Graphics: Principles and Practice. Editorial: Addison Wesley. Capítulo 4. Viewing in 3D páginas 253-258

Si  $M$  es una matriz con un determinante igual a cero, entonces los tres planos no se intersectan en un punto. Esto pasa cuando los tres vectores normales están

sobre el mismo plano, como en el caso de la figura No.8



Figura No.8: tres planos no necesariamente intersectados en un punto

Cuando dos planos no paralelos  $L1 = \langle N1, D1 \rangle$  y  $L2 = \langle N2, D2 \rangle$ , en la intersección ellos forman una línea. Como se muestra en la figura No.9, la dirección  $V$ , en la línea de intersección, es perpendicular a las normales de ambos planos, y puede ser expresada como:  $V = N1 \times N2$ .

Para formar una descripción completa de la línea, podemos también proporcionar un punto cualquiera dentro de la línea. Esto puede lograrse con la construcción de un tercer plano  $L3 = V, 0$ , pasando a través del origen y donde la dirección de la normal es  $V$ . Como se muestra en la figura No.10, podemos resolver para el punto donde los tres planos se intersectan, garantizando así la existencia de una tercera ecuación. Usando la ecuación ec17, podemos definir un punto  $Q$ , que se establece sobre la línea de intersección:

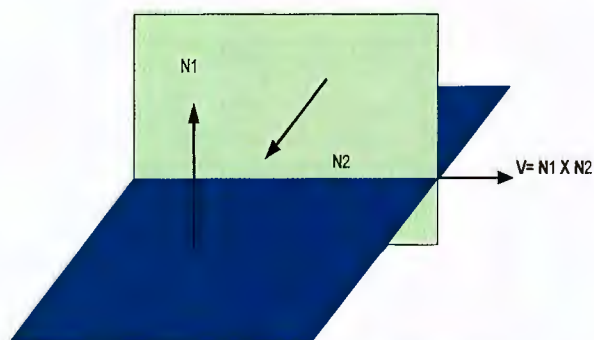


Figura No.9: Dos planos no paralelos intersectados en una línea, con una dirección perpendicular a las normales de ambos planos

$$(ec18) \quad Q = \begin{pmatrix} (N1)_x & (N1)_y & (N1)_z \\ (N2)_x & (N2)_y & (N2)_z \\ V_x & V_y & V_z \end{pmatrix}^{-1} \begin{pmatrix} -D1 \\ -D2 \\ 0 \end{pmatrix}$$

La línea donde los dos planos  $L1$  y  $L2$  se intersectan, estará definida por:

$$(ec19) \quad P(t) = Q + tV.$$

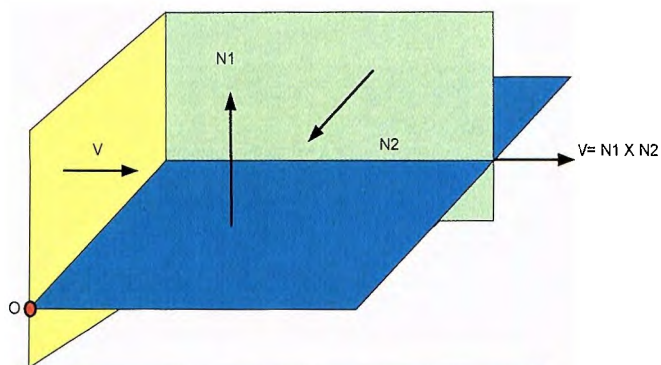


Figura No.10: Adicionando un tercer plano, que pasa a través del origen y donde la normal es igual a la dirección de la línea en que los dos primeros planos se intersectan.

### 3.3 Transformaciones de coordenadas<sup>7</sup>.

Los métodos para trasladar, escalar y hacer girar objetos en 3D, se amplían de métodos bidimensionales, incluyendo consideraciones para la coordenada Z.

Una traslación se logra especificando un vector de traslación tridimensional y la escalación se efectúa mediante la especificación de tres vectores de escalación. Las extensiones de las rotaciones pueden realizarse ahora en torno a los ejes en cualquier orientación espacial. Como sucede en el caso bidimensional, las ecuaciones de transformación geométrica pueden expresarse en términos de matrices de transformación. Por lo tanto, cualquier secuencia de transformaciones se representa como una sola matriz, formada mediante el eslabonamiento de las matrices para las transformaciones individuales de la secuencia.

#### Transformaciones Generales<sup>8</sup>:

Una matriz invertida nxn puede generalmente se considera como una transformación de un sistema de coordenadas a otro. Las columnas, dan la imagen para cada eje principal del sistema original, los cuales son trazados en el nuevo sistema de coordenadas. Por conveniencia, se supone que M es una matriz invertida o transpuesta de 3 x 3. Por ejemplo, cuando M opera en los vectores  $\langle 1,0,0 \rangle$ ,  $\langle 0,1,0 \rangle$ , y  $\langle 0,0,1 \rangle$ , se tiene que:

$$(ec19) \quad \begin{pmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{32} \\ M_{31} & M_{32} & M_{33} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} M_{11} \\ M_{21} \\ M_{31} \end{pmatrix}$$

$$\begin{pmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{32} \\ M_{31} & M_{32} & M_{33} \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} M_{12} \\ M_{22} \\ M_{32} \end{pmatrix}$$

$$\begin{pmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{32} \\ M_{31} & M_{32} & M_{33} \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} M_{13} \\ M_{23} \\ M_{33} \end{pmatrix}$$

<sup>7</sup> Buss Samuel R. 3D Computer Graphics: A Mathematical Introduction with OpenGL. Editorial. Cambridge University Press. Capítulo 2 Transformations and Viewing. Páginas 34 – 50.

<sup>8</sup> Schneider Philip, Eberly David H Capítulo 4. Matrices, Vector Algebra and Transformations. Páginas 126-145

Similarmente, las columnas de  $M^{-1}$  general la imagen principal de cada eje en el nuevo sistema que es trazado en el sistema original. De esta manera, dando cualquier linealidad arbitraria independiente de los vectores  $U$ ,  $V$  y  $W$  podemos construir una matriz de transformación que trace éstos vectores en los nuevos  $\langle 1,0,0 \rangle$ ,  $\langle 0,1,0 \rangle$ , y  $\langle 0,0,1 \rangle$ . Así

$$\begin{aligned}
 & \begin{pmatrix} U_x & V_x & W_x \\ U_y & V_y & W_y \\ U_z & V_z & W_z \end{pmatrix} \begin{pmatrix} U_x \\ U_y \\ U_z \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \\
 \text{(ec20)} \quad & \begin{pmatrix} U_x & V_x & W_x \\ U_y & V_y & W_y \\ U_z & V_z & W_z \end{pmatrix} \begin{pmatrix} V_x \\ V_y \\ V_z \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \\
 & \begin{pmatrix} U_x & V_x & W_x \\ U_y & V_y & W_y \\ U_z & V_z & W_z \end{pmatrix} \begin{pmatrix} W_x \\ W_y \\ W_z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}
 \end{aligned}$$

Múltiples transformaciones pueden ser concatenadas y representadas por una simple multiplicación de matrices. Supongamos que deseamos transformar un primer objeto usando una matriz  $\mathbf{M}$  y un segundo usando una matriz  $\mathbf{G}$ . si como lo mencionamos en el primer capítulo, la multiplicación de matrices es asociativa,  $\mathbf{G}(\mathbf{M}\mathbf{P}) = (\mathbf{G}\mathbf{M})\mathbf{P}$  para cualquier vector  $\mathbf{P}$ , entonces al simple producto de  $\mathbf{G}\mathbf{M}$  se denomina *transformación de objetos*. Esta aseveración, nos permite aplicar un sin número de transformaciones a los vértices de cualquier objeto, sin incurrir en ningún cálculo complejo adicional.



### 3.4 Representaciones matriciales

#### ESCALACIÓN<sup>9</sup>:

La operación matricial para realizar la escalación en tres dimensiones relativa al origen coordenado es:

$$(ec21) \quad \begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Donde los parámetros de escalación  $S_x$ ,  $S_y$  y  $S_z$  se les asigna cualquier valor positivo. Esta operación matricial escala un punto en  $(x,y,z)$  a la posición  $(x',y',z')$  con las ecuaciones de escalación:

$$(ec22) \quad x' = x.S_x, \quad y' = y.S_y, \quad z' = z.S_z$$

Cuando las transformaciones de ec21 se aplica para definir puntos en un objeto, éste se escala y se desplaza en relación con el origen coordenado. Si los parámetros de transformación no son iguales, las dimensiones relativas del objeto también se cambian. Una escalación uniforme ( $S_x = S_y = S_z$ ) preserva la forma original de un objeto, ver figura No.11:

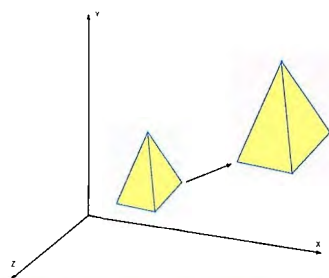


Figura No.11: Duplicación del tamaño de un Objeto con la transformación de ec21 también desplazado el objeto más allá del origen

La escalación con respecto a cualquier posición fija  $(x_f, y_f, z_f)$ , puede obtenerse efectuando la siguiente secuencia de transformaciones:

1. Trasládese el punto fijo al origen.
2. Escálese con la ecuación ec21
3. Vuélvase a trasladar el punto fijo a su posición original.

Esta secuencia de transformaciones se

<sup>9</sup> Legyel Eric. Mathematics for 3D Game Programming and Computer Graphics. Editorial Charles Rivera Media, Inc. Capítulo 3 Transforms, páginas 58-64



demuestra en la figura No.12.

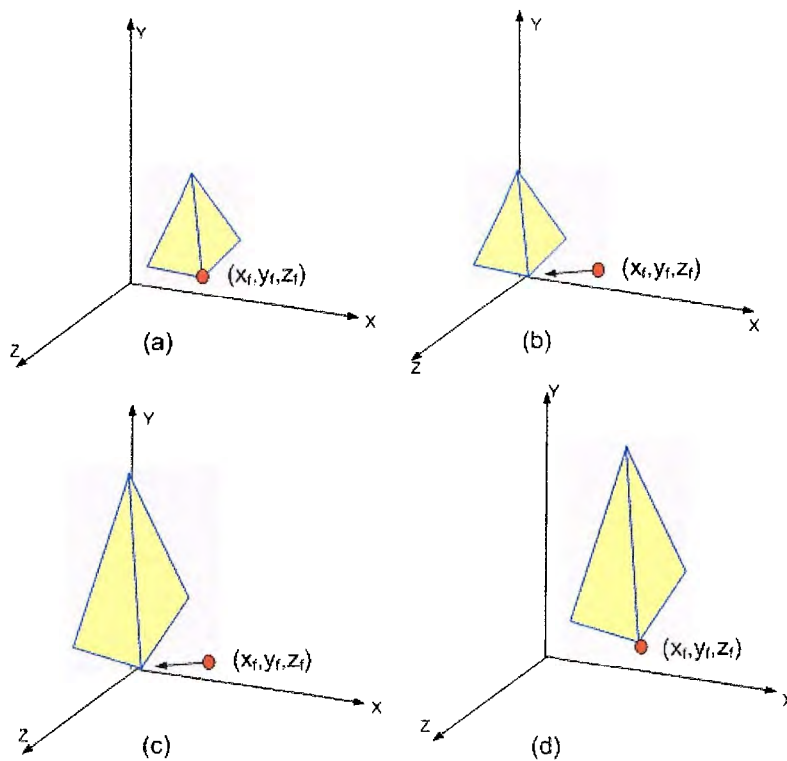


Figura No.12: La escalación de un objeto relativa a un punto fijo seleccionado es equivalente a la secuencia de transformaciones que se muestran.

La matriz de transformación de la escalación arbitraria de punto fijo se forma con el eslabonamiento de éstas transformaciones individuales:

$$(ec23) \quad T(-x_f, -y_f, -z_f) \cdot S(S_x, S_y, S_z) \cdot T(x_f, y_f, z_f) = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ (1-S_x)x_f & (1-S_y)y_f & (1-S_z)z_f & 1 \end{pmatrix}$$

Donde T representa la matriz de traslación y S, la matriz de escalación.

Se forma la matriz de escalación inversa para la ecuación ec21 o bien para la ec23 sustituyendo los parámetros de escalación  $S_x$ ,  $S_y$  y  $S_z$  con sus recíprocos. La matriz inversa genera una transformación de escalación opuesta, de modo que el eslabonamiento de la matriz de escalación y su recíproco produce la matriz identidad.

## TRASLACIÓN<sup>10</sup>:

La traslación, reposiciona un objeto desplazándolo a las nuevas coordenadas. Un punto es trasladado de la posición  $(x,y,z)$  a la posición  $(x',y',z')$  (ver figura No.13) con la siguiente operación matricial.

$$(ec24) \quad \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{pmatrix}$$

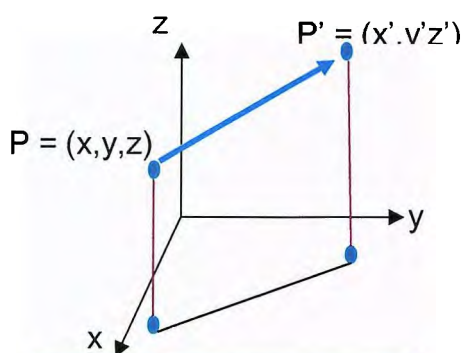


Figura No.13: Traslación de un punto

Los parámetros  $T_x$ ,  $T_y$ ,  $T_z$ , que especifican distancias de traslación para las coordenadas, reciben la asignación de cualquier valor real. La representación matricial de la ecuación ec24 es equivalente a las tres ecuaciones:

$$(ec25) \quad x' = x + T_x, y' = y + T_y, z' = z + T_z$$

Un objeto se traslada en tres dimensiones transformado con cada punto definidor del objeto.

La traslación de un objeto representada como un conjunto de superficies poligonales se efectúa trasladando los valores coordenados para cada vértice de cada superficie. El conjunto de posiciones coordenadas trasladadas de los vértices define entonces la nueva posición del objeto (ver la figura No.14).

Se obtiene la inversa de la matriz de traslación de la ecuación ec24, rechazando las distancias de traslación  $T_x, T_y, T_z$ . Esto produce una traslación en la dirección contraria y el producto de una matriz de traslación y su recíproco produce la matriz identidad.

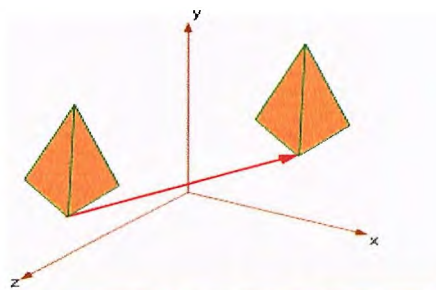


Figura No.14: traslación de un objeto con el vector de traslación T.

<sup>10</sup> Eberly David H. 3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics Editorial Morgan Kaufmann. Capítulo 2 Geometrical Methods, páginas 8-9

## ROTACIÓN<sup>11</sup>

Para especificar una transformación de rotación de un objeto, se debe designar un eje de rotación (en torno al cual se hará girar el objeto) y la cantidad de rotación angular. En aplicaciones bidimensionales, el eje de rotación siempre es perpendicular al plano xy. En tres dimensiones, un eje de rotación puede tener cualquier orientación espacial. Los ejes de rotación más fáciles de manejar son aquellos que son paralelos a los ejes coordenados. Así mismo, podemos valernos

de las rotaciones en torno a los tres ejes coordenados con el fin de producir una rotación en torno a cualquier eje de rotación especificado en forma arbitraria.

Adoptamos el convencionalismo de que las rotaciones en sentido contrario al de las agujas del reloj en torno a un eje coordenado se producen con ángulos de rotación positivos si se observa la mitad positiva del eje en dirección del origen coordenado (ver figura No. 15). Las ecuaciones de rotación del eje z bidimensional se desarrollan fácilmente a tres dimensiones:

$$(ec26) \quad \begin{aligned} x' &= x \cos \Theta - y \sin \Theta \\ y' &= x \sin \Theta + y \cos \Theta \\ z' &= z \end{aligned}$$

El parámetro  $\Theta$  especifica el ángulo de rotación. En forma coordenada homogénea, las ecuaciones de rotación del eje z tridimensional se expresan como:

$$(ec27) \quad \begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} \cos \Theta & \sin \Theta & 0 & 0 \\ -\sin \Theta & \cos \Theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura No.15: Las direcciones de rotación positivas en torno a los ejes coordenados son en sentido contrario al del reloj, como se observa a lo largo de la posición positiva de cada eje en dirección del origen.

<sup>11</sup> Jones Huw, Computer Graphics Through Key Mathematics, Capítulo 10 Drawing and Rendering  
Páginas 319-326

La figura No.16 ilustra la rotación de un objeto en torno al eje z.

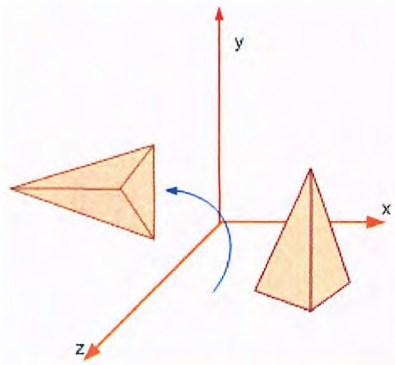


Figura No.16: Rotación de un objeto en torno al eje z

Las ecuaciones de transformación para rotaciones en torno a los otros dos ejes coordenados pueden obtenerse con una permutación cíclica de los parámetros coordenados de las ecuaciones ec26. Sustituimos x por y, y por z y z por x. Utilizando esta permutación en las ecuaciones ec26, se obtienen las ecuaciones para una rotación del eje x:

$$(ec28) \quad \begin{aligned} y' &= y \cdot \cos \Theta - z \cdot \sin \Theta \\ z' &= y \cdot \sin \Theta + z \cdot \cos \Theta \\ x' &= x \end{aligned}$$

Lo que puede escribirse en la forma de coordenadas homogéneas:

$$(ec29) \quad \begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \Theta & \sin \Theta & 0 \\ 0 & -\sin \Theta & \cos \Theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Las coordenadas cíclicamente permutantes de ec28, dan las ecuaciones de transformación para una rotación del eje y. La rotación en torno al eje y, se demuestra en la figura No.17.

$$(ec30) \quad \begin{aligned} z' &= z \cos \Theta - x \sin \Theta \\ x' &= z \sin \Theta + x \cos \Theta \\ y' &= y \end{aligned}$$

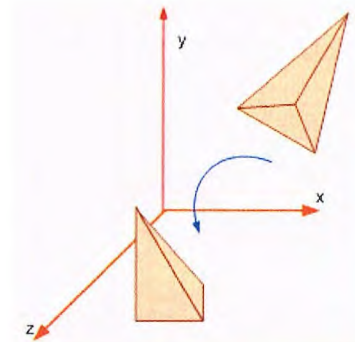


Figura No.17: Rotación de un eje alrededor del eje x

La representación Matricial para una rotación del eje y es:

$$(ec31) \quad \begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{pmatrix} \cos\Theta & 0 & -\text{sen}\Theta & 0 \\ 0 & 1 & 0 & 0 \\ \text{sen}\Theta & 0 & \cos\Theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

En la figura No.18 se muestra un ejemplo de la rotación del eje y.

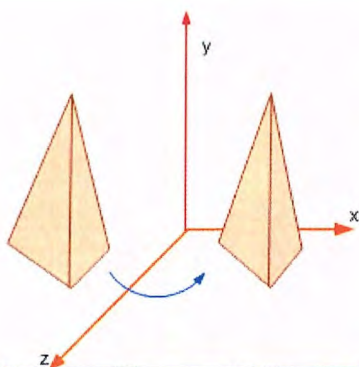


Figura No.18: Rotación de un objeto en torno al eje y

Una matriz de rotación inversa se forma sustituyendo el ángulo de rotación  $\Theta$  por  $-\Theta$ . Los valores negativos del ángulo de rotación generan rotaciones a una dirección igual a la del reloj, de modo que la matriz de identidad se produzca cuando alguna matriz de rotación se multiplique por su recíproco, ya que solo la función seno se ve afectada por el cambio de signo del ángulo de rotación.

La matriz inversa también puede obtenerse intercambiando filas y columnas. Es decir, podemos calcular la inversa de cualquier matriz de rotación  $R$  mediante la evaluación de su transpuesta ( $R^{-1} = R^T$ ). Este método de obtención de una matriz inversa se cumple también para cualquier matriz de rotación compuesta.

## **Rotación Alrededor de un eje Arbitrario<sup>12</sup>:**

Se puede hacer que los objetos giren en torno a cualquier eje seleccionado en forma arbitraria, aplicando una matriz de transformación compuesta cuyas componentes efectúan una secuencia de traslaciones y rotaciones en torno a los ejes coordenados. Esta matriz compuesta se forma con combinaciones de matriz de traslación y las transformaciones ec27, ec29 y ec31.

La secuencia correcta de transformaciones puede determinarse mediante la transformación de posiciones coordenadas del objeto de manera que el eje de rotación seleccionado se desplace sobre uno de los ejes coordenados. Después se hace que el objeto gire alrededor de dicho eje a través del ángulo de rotación especificado. La última etapa consiste en aplicar las transformaciones inversas que devuelvan el eje de rotación a su posición original.

En el caso especial donde el eje de rotación seleccionado es paralelo a uno de los ejes coordenados, la rotación deseada del objeto se logra con el conjunto equivalente de tres transformaciones:

1. trasládese el objeto de manera que el eje de rotación coincida con el coordenado paralelo.
2. Efectúese la rotación especificada
3. Trasládese el objeto de modo que el eje de rotación se devuelva a su posición original.

Estas etapas se ilustran en la figura No.19.

---

<sup>12</sup> Legyel Eric. Mathematics for 3D Game Programming and Computer Graphics. Editorial Charles Rivera Media, Inc. Capítulo 3 Transforms, páginas 60-62

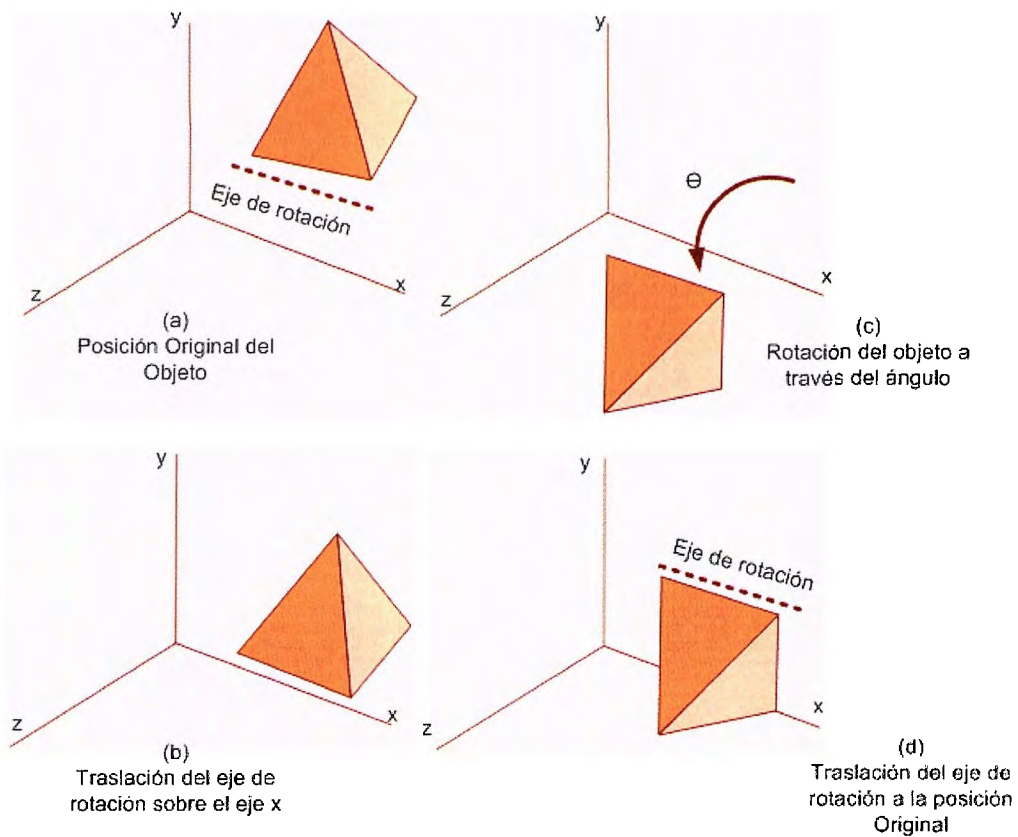


Figura No.19. Secuencia de transformaciones para hacer girar un objeto en torno al eje x.



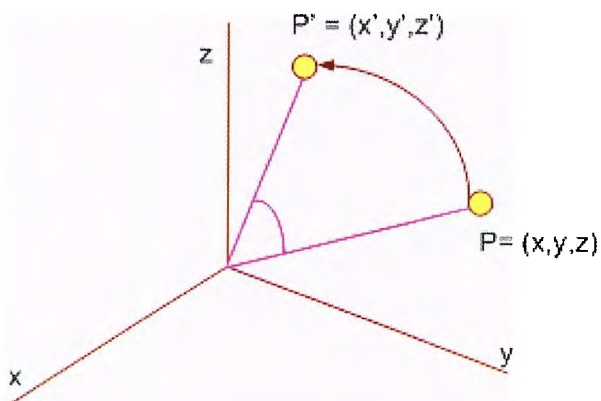
## ROTACIÓN PLANA ALREDEDOR DEL EJE X

Par calcular la expresión de rotación alrededor del eje X, intercambiamos las variables:

$$(ec32) \quad \begin{aligned} x' &= x \\ y' &= y \cos \Theta - z \sin \Theta \\ z' &= y \sin \Theta + z \cos \Theta \end{aligned}$$

En forma matricial:

$$(ec33) \quad \begin{pmatrix} X' \\ Y' \\ Z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \Theta & \sin \Theta & 0 \\ 0 & -\sin \Theta & \cos \Theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = R_x$$



$$(ec34) \quad \mathbf{P}' = \mathbf{P} \cdot \mathbf{R}_x$$

Figura No.20. Rotación alrededor del eje x

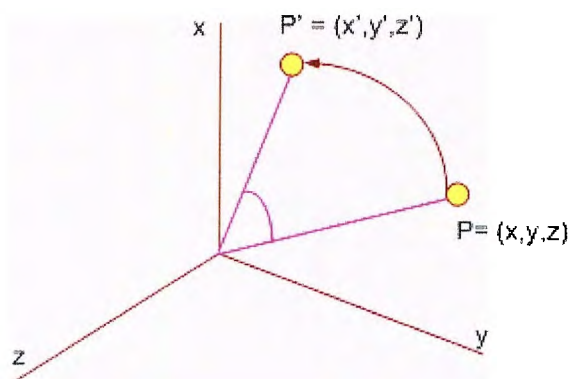


## ROTACIÓN PLANA ALREDEDOR DEL EJE Y

$$(ec35) \quad \begin{aligned} x' &= x \cos \Theta + z \sin \Theta \\ y' &= y \\ z' &= -x \sin \Theta + z \cos \Theta \end{aligned}$$

En forma matricial:

$$(ec36) \quad \begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \Theta & 0 & -\sin \Theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin \Theta & 0 & \cos \Theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = R_y$$



$$(ec37) \quad \mathbf{P}' = \mathbf{P} \cdot \mathbf{R}_y$$

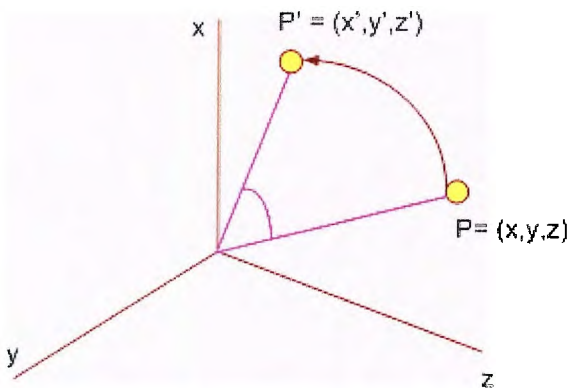
Figura No.21. Rotación alrededor del eje y

## ROTACIÓN PLANA ALREDEDOR DEL EJE Z

$$(ec38) \quad \begin{aligned} x' &= x \cos \Theta - y \sin \Theta \\ y' &= x \sin \Theta + y \cos \Theta \\ z' &= z \end{aligned}$$

En forma matricial

$$(ec39) \quad \begin{pmatrix} X' \\ Y' \\ Z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos \Theta & \sin \Theta & 0 & 0 \\ -\sin \Theta & \cos \Theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = R_z$$



$$(ec40) \quad \mathbf{P}' = \mathbf{P} \cdot \mathbf{R}_y$$

Figura No.22. Rotación alrededor del eje z

## Matrices de transformación<sup>13</sup>

Ahora utilizaremos las operaciones con vectores para determinar la forma en que una rotación alrededor de un eje arbitrario puede expresarse en términos de las matrices de transformación básicas. Dadas las especificaciones para el eje y el ángulo de rotación, podemos lograr la rotación que se requiere en cinco etapas:

1. Trasládese el objeto de manera que el eje de rotación atraviese el origen coordenado.
2. Hágase girar el objeto de modo que el eje de rotación coincida con uno de los ejes coordenados.
3. realícese la rotación específica
4. Aplíquense rotaciones inversas para devolver el eje de rotación a su orientación original.
5. Aplíquese la traslación inversa para devolver el eje de rotación a su posición original.

Podemos transformar el eje de rotación en alguno de los tres ejes coordenados.

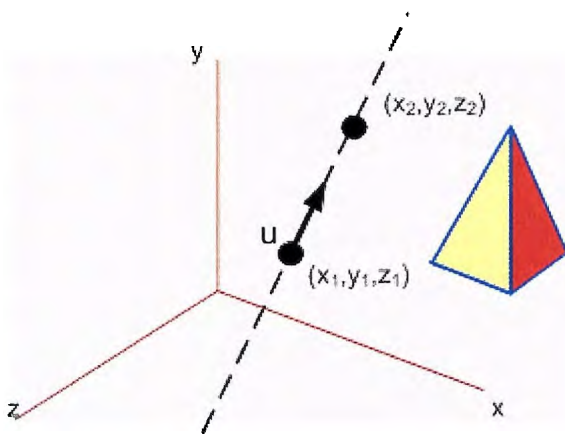


Figura No.23: Un eje de rotación (línea punteada) para un objeto se define por medio de los puntos P1, y P2'. El vector unitario  $u$  se establece a partir de las coordenadas de los dos puntos

Para comenzar se supone que el eje de rotación está definido por dos puntos, como en la figura No.23. Estos dos puntos se utilizan para definir el vector.

$$(ec41) \quad \mathbf{V} = (x_2 - x_1, y_2 - y_1, z_2 - z_1)$$

Que a su vez se emplea para definir el vector unitario  $u$  a lo largo del eje de rotación:

$$(ec42) \quad \mathbf{u} = \frac{\mathbf{V}}{|\mathbf{V}|} = (a, b, c)$$

<sup>13</sup> Hearn Donal, Gráficas por Computadora, Editorial Prentice Hall, Capítul 11 Transformaciones Tridimensionales, páginas 244-247.

Donde las componentes  $a$ ,  $b$  y  $c$  del vector  $u$  son los cosenos direccionales del vector  $V$ :

$$(ec43) \quad a = \frac{x}{|V|} \quad b = \frac{y}{|V|} \quad c = \frac{z}{|V|}$$

La traslación del objeto de manera que el eje de rotación pase por el origen coordenado se lleva a cabo con la matriz de traslación:

$$(ec44) \quad T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ -x1 & -y1 & -z1 & 1 \end{pmatrix}$$

Esto coloca el punto  $P1$  en el origen (ver la figura No.24).

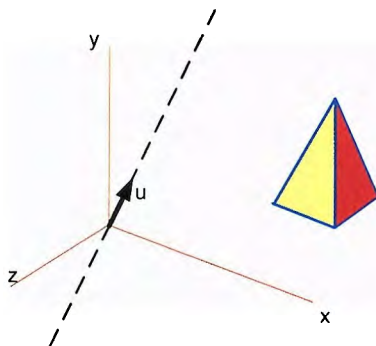


Figura No.24: Traslación del eje de rotación hacia el origen coordenado

Luego necesitaremos las transformaciones que colocarán al eje de rotación sobre el eje  $z$ . Podemos realizar esta transformación en dos etapas. Primero, gírese en torno del eje  $x$  de modo que el vector  $u$  esté en el plano  $xz$ . Ahora gírese alrededor del eje  $y$  para traer a  $u$  al eje  $z$ . Estas dos rotaciones se ilustran en la figura No.25.

Se establece la matriz de transformación para la rotación en torno al eje  $x$  determinando los valores del

seno y del coseno del ángulo de rotación que se necesitan para colocar  $u$  en el plano  $yz$  y del eje positivo (ver figura No.26). Si se designa la proyección de  $u$  en el plano  $yz$  como el vector  $u' = (0,b,c)$ , entonces el coseno del ángulo de rotación  $\alpha$  puede determinar a partir del producto escalar de  $u'$  y el vector unitario  $u_z$ , a lo largo del eje  $z$ .

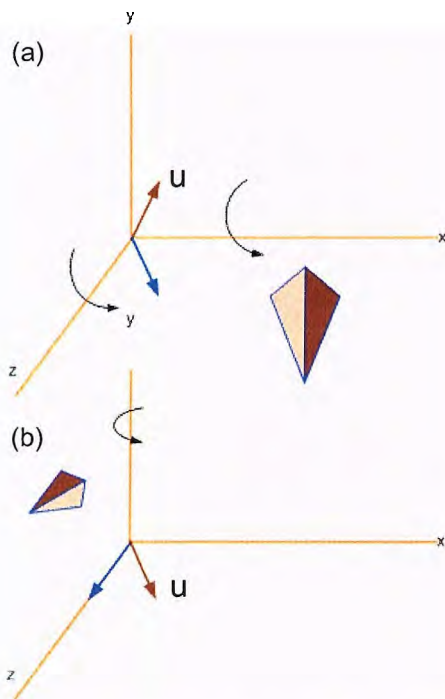


Figura No.25: (a)El vector unitario  $u$  se hace girar en torno al eje  $x$  para traerlo al plano  $xz$ . (b) después se hace girar en torno al eje  $y$  para alinearlo con eje  $z$

$$(ec45) \quad \cos \alpha = \frac{u' \cdot u_z}{|u'| |u_z|} = \frac{c}{d}$$

Donde  $d$  es la magnitud de  $u'$ :

$$(ec46) \quad d = (b^2 + c^2)^{1/2}$$

El seno de  $\alpha$  puede determinarse en forma semejante a partir del producto vectorial de  $u'$  y  $u_z$ . De la definición de producto vectorial, se puede escribir:

$$(ec47) \quad u' \times u_z = u_x |u'| |u_z| \sin \alpha$$

Y utilizando la expresión cartesiana del producto vectorial se tiene:

$$(ec48) \quad u' \times u_z = u_x \cdot b$$

Por lo tanto la función seno puede evaluarse como:

$$(ec49) \quad \sin \alpha = b/d$$

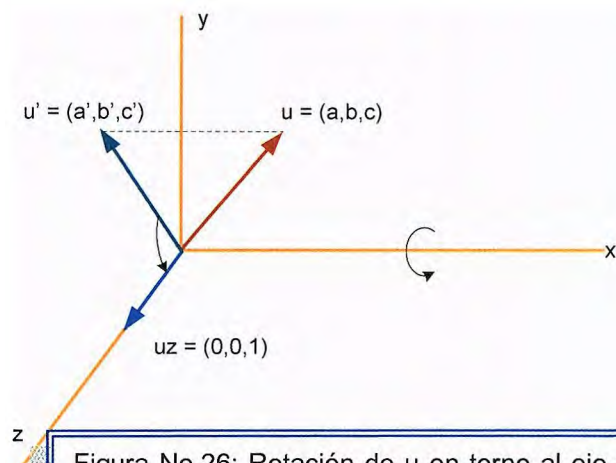


Figura No.26: Rotación de  $u$  en torno al eje  $x$  en el plano  $xz$  que se logra haciendo girar  $u$  (proyección de  $u$ , en el plano  $yz$ ) a través del

Ahora que se han determinado los valores de  $\cos \alpha$  y  $\sin \alpha$  en términos de las componentes del vector  $u$ , podemos evaluar la matriz de rotación en torno al eje  $x$  como:

$$(ec50) \quad R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d & b/d & 0 \\ 0 & -b/d & c/d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Esta matriz hace girar al vector unitario  $u$  del eje  $x$  en el plano  $xz$ .

A continuación determinaremos la forma de la matriz de transformación que hará oscilar el vector unitario en el plano  $xz$  en torno al eje  $y$  sobre el eje  $z$  positivo. La orientación del vector unitario en el plano  $xz$  se muestra en la figura No.27.

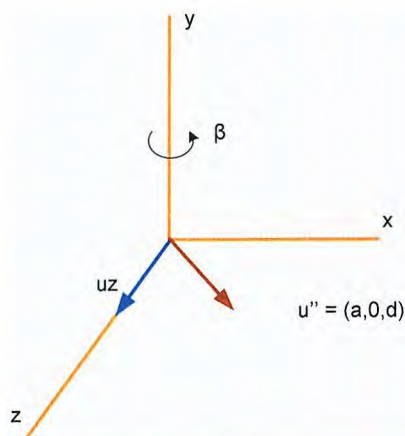


Figura No.27: Rotación del vector unitario  $u''$  (vector  $u$  después de la rotación en el plano  $xz$ ) en torno al eje  $y$ . El ángulo de rotación positivo alinea a  $u''$  con el vector  $u_z$

Este vector rotulado  $u''$  tiene el valor  $a$  para su componente  $x$ . Su componente  $z$  es  $d$  (*magnitud de  $u$* ), ya que el vector  $u'$  se ha hecho girar sobre el eje  $z$ . La componente  $y$  de  $u''$  es 0, ya que ahora se encuentra en el plano  $xz$ . Una vez determinamos las funciones trigonométricas del ángulo de rotación  $\beta$  a partir de expresiones de los productos vectoriales entre los vectores unitarios  $u''$  y  $u_z$ . Según la definición de producto escalar, se tiene:

$$(ec51) \quad \cos \beta = \frac{u'' \cdot u_z}{|u''| |u_z|} = d$$

Cuando  $|u_z| = |u''| = 1$ . Utilizando las dos ecuaciones del producto escalar, se puede escribir:

$$(ec52) \quad u'' \times u_z = u_y |u''| |u_z| \text{ sen } \beta$$

$$y \quad (ec53) \quad u'' \times u_z = u_y \cdot (-a)$$

Al igualar las ecuaciones ec52 y ec53, se obtiene:

$$(ec54) \quad \text{sen } \beta = -a$$

Y la matriz de transformación para la rotación en torno al eje y es:

$$(ec55) \quad R_y(\beta) = \begin{pmatrix} d & 0 & a & 0 \\ 0 & 1 & 0 & 0 \\ -a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Con las matrices de transformación, el eje de rotación se alinea con el eje z positivo. El ángulo de rotación especificado  $\Theta$ , puede aplicarse después como una rotación en torno a z:

$$(ec56) \quad R_z(\Theta) = \begin{pmatrix} \cos\Theta & \text{sen}\Theta & 0 & 0 \\ -\text{sen}\Theta & \cos\Theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Para completar la rotación requerida en torno al eje dado, se debe volver a transformar el eje de rotación a su posición original. Esto se realiza aplicando la inversa de las transformaciones. La matriz de transformación para la rotación en torno a un eje arbitrario puede expresarse entonces como la composición de estas siete transformaciones individuales.

$$(ec57) \quad \mathbf{R}(\Theta) = \mathbf{T} \cdot \mathbf{R}_x(\alpha) \cdot \mathbf{R}_y(\beta) \cdot \mathbf{R}_z \cdot \mathbf{R}_y^{-1}(\beta) \cdot \mathbf{R}_x^{-1}(\alpha) \cdot \mathbf{T}^{-1}$$



## Código de C++ para Rotaciones<sup>14</sup>

### Rotación en el eje X

```
void matriz_rotacion_x(MATRIZ *m, double angulo)
{
    double tmpsin, tmpcos;

    tmpsin=sin(angulo);
    tmpcos=cos(angulo);
    *m = matriz_identidad;
    m->v[1][1] = tmpcos;
    m->v[1][2] = tmpsin;
    m->v[2][1] = -tmpsin;
    m->v[2][2] = tmpcos;
}
```

Ejemplo: `matriz_rotacion_x(&Rx, 0.7)`

Esto crea la matriz de rotación alrededor del eje X, donde Rx es una matriz 4x4 y 0.7 es el ángulo de rotación, en radianes

### Rotación en el eje Y

```
void matriz_rotacion_y(MATRIZ *m, double angulo)
{
    double tmpsin, tmpcos;

    tmpsin=sin(angulo);
    tmpcos=cos(angulo);
    *m = matriz_identidad;
    m->v[0][0] = tmpcos;
    m->v[0][2] = -tmpsin;
    m->v[2][0] = tmpsin;
    m->v[2][2] = tmpcos;
}
```

Ejemplo : `matriz_rotacion_y(&Ry, 0.7)`

donde Ry es la matriz de rotación 4x4, y 0.7 es el ángulo de rotación.

<sup>14</sup> Govil-Pai Shalini, Principles of Computer Graphics: Theory and Practice Using OpenGL and Maya, Editorial Springer. Capítulo 5 3D Modeling, páginas 104-107



## Rotación en el eje Z

```
void matriz_rotacion_z(MATRIZ *m, double angulo)
{
    double tmpsin, tmpcos;

    tmpsin=sin(angulo);
    tmpcos=cos(angulo);
    *m = matriz_identidad;
    m->v[0][0] = tmpcos;
    m->v[0][1] = tmpsin;
    m->v[1][0] = -tmpsin;
    m->v[1][1] = tmpcos;
}
```

Ejemplo : matriz\_rotacion\_z(&Rz, 0.7)

donde Rz es la matriz de 4x4, y 0.7 es el ángulo de rotación en torno al eje Z.

### 3.5 Proyecciones en un plano (en pantalla)<sup>15</sup>

Existen dos métodos básicos para proyectar objetos tridimensionales sobre una superficie de visión bidimensional. Todos los puntos del objeto pueden proyectarse sobre la superficie a lo largo de líneas paralelas o bien los puntos pueden proyectarse a lo largo de líneas que convergen hacia una posición denominada **centro de proyección**.

Los dos métodos básicos son llamados **Proyección en Paralelo** y **Proyección Perspectiva**. En ambos casos la intersección de una línea de proyección con la superficie de visión determina las coordenadas del punto proyectado sobre ese plano. Por ahora, se supone que el plano de proyección de visión es  $z = 0$  de un sistema de coordenadas del lado izquierdo, como se muestra en la figura No.28.

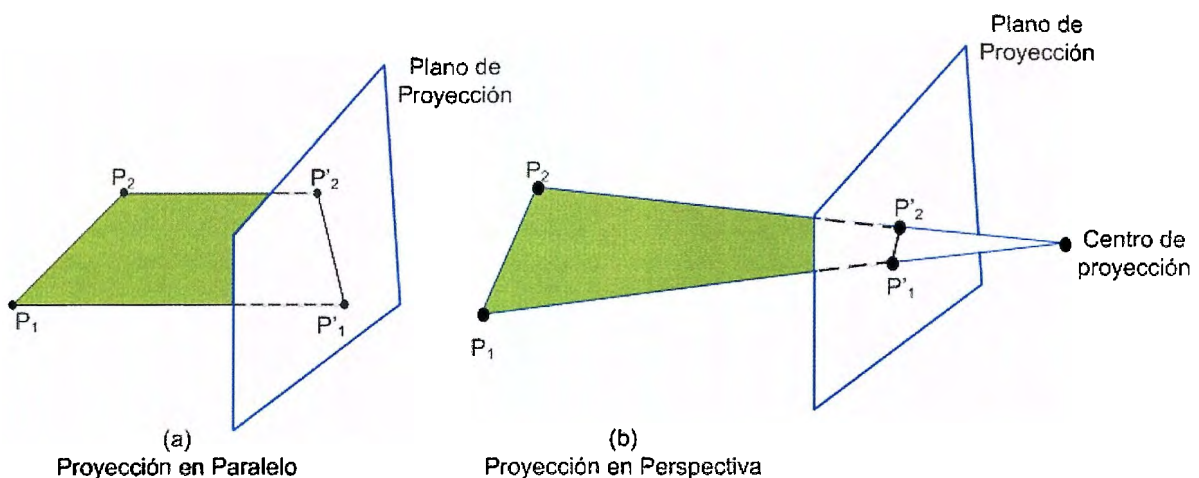


Figura No.28: Dos métodos para proyectar una línea sobre una superficie de visión (plano de proyección)

<sup>15</sup> Zárate Deras Juan Carlos, Alam Zamora Ana Luisa, Planificación de Trayectorias de Objetos Móviles en el Plano y Simulación Tridimensional de las Mismas. Tesis. Páginas 65 -73

### 3.5.1 Proyección Paralela<sup>16</sup>.

Las vistas formadas con proyecciones en paralelo, pueden caracterizarse de acuerdo con el ángulo que la dirección de proyección forma con el plano de proyección.

Existen dos tipos de proyecciones en paralelo:

#### 1. Ortogonal:

Ésta se da cuando la dirección de proyección es perpendicular al plano al cual se proyectará. Se utilizan con mayor frecuencia para producir las vistas del frente, lado y parte superior de un objeto. (Ver la figura No.29).

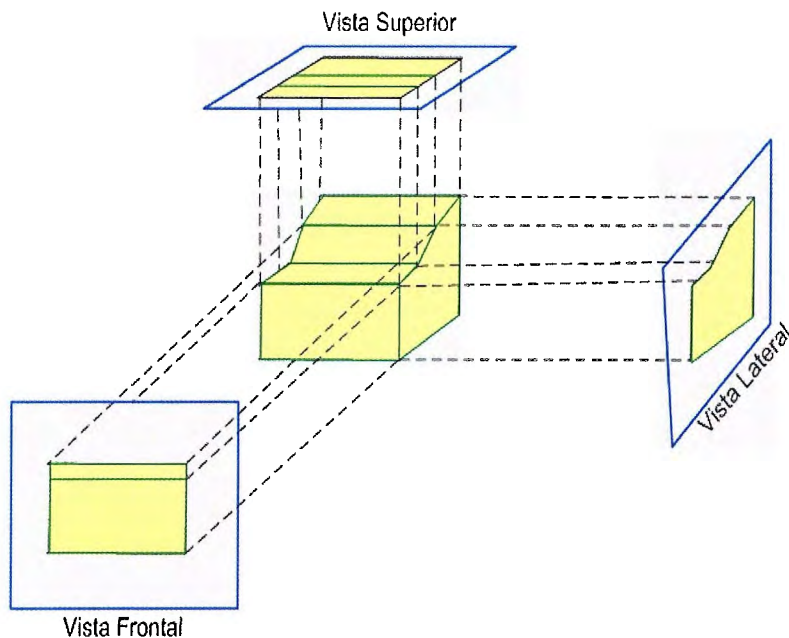


Figura No.29: Tres proyecciones ortogonales de un objeto

Las vistas ortogonales del frente, lado y fondo se denominan “*elevaciones*” y las vistas de la parte superior se conocen como “*plantas*”. Los trazos de ingeniería comúnmente emplean estas proyecciones ortogonales, ya que

<sup>16</sup> Hearn Donald, Gráficas por Computadora. Editorial Prentice Hall. Capítulo 12: Vista Tridimensional, páginas 253-256

las longitudes y los ángulos se representan en forma más exacta y pueden medirse a partir de los trazos.

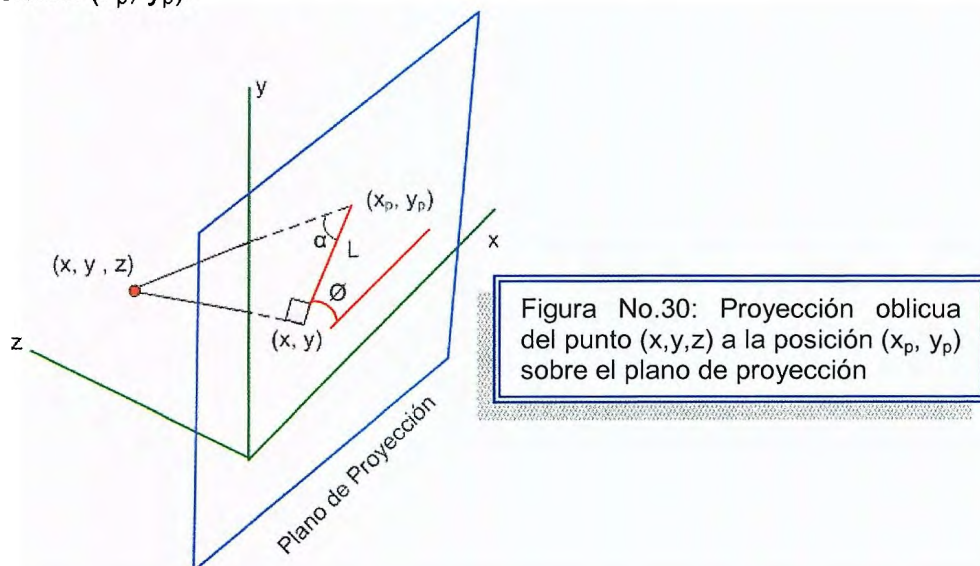
También podemos formar proyecciones ortogonales que muestran más de una cara de un objeto. Tales vistas se denominan proyecciones ortogonales **Axonométricas**, de éstas la que comúnmente es utilizada es la proyección **isométrica**, las cuales se obtienen alineando el plano de proyección de modo que se corte con cada eje coordenado en el cual se defina el objeto a la misma distancia del origen.

Las ecuaciones de transformación para efectuar una proyección paralela ortogonal son directas. Para cualquier punto  $(x,y,z)$ , el punto de proyección  $(x_p, y_p, z_p)$  sobre la superficie se obtiene como:

$$(ec58) \quad x_p = x, \quad y_p = y, \quad z_p = 0$$

## 2. Oblicua

Se obtiene proyectando puntos a lo largo de líneas paralelas que no son perpendiculares al plano de proyección. La figura No.30 muestra una proyección oblicua de un punto  $(x, y, z)$  por una línea de proyección a la posición  $(x_p, y_p)$ .



Las coordenadas de proyección ortogonal en el plano son  $(x, y)$ . La línea de proyección oblicua forma un ángulo  $\alpha$  con la línea sobre el plano de proyección que une  $(x_p, y_p)$  y  $(x, y)$ .

Esta línea de longitud  $L$ , está en un ángulo  $\emptyset$  con la dirección horizontal en el plano de proyección. Podemos expresar las coordenadas de proyección en términos de  $x, y, L$  y  $\emptyset$ :

$$\begin{aligned} x_p &= x + L \cos \emptyset \\ y_p &= y + L \sin \emptyset \end{aligned} \quad (\text{ec59})$$

Una dirección de proyección puede definirse seleccionando valores para los ángulos  $\alpha$  y  $\emptyset$ . Alternativas comunes del ángulo  $\emptyset$  son  $30^\circ$  y  $45^\circ$ , los cuales despliegan una vista combinada del frente, lado y parte superior o inferior de un objeto. La longitud  $L$  es función de la coordenada  $z$  y podemos evaluar este parámetro a partir de las relaciones:

$$\tan \alpha = (z / L) = 1 / L_1 \quad (\text{ec60})$$

Donde  $L_1$  es la longitud de la línea de proyección de  $(x, y)$  a  $(x_p, y_p)$  cuando  $z = 1$ . De ésta ecuación se tiene:

$$\begin{aligned} x_p &= x + z(L_1 \cos \emptyset) \\ y_p &= y + z(L_1 \sin \emptyset) \end{aligned} \quad (\text{ec61})$$

La matriz de transformación para producir cualquier proyección en paralelo puede expresarse como:

$$P_{\text{paralelo}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ L_1 \cos \emptyset & L_1 \sin \emptyset & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (\text{ec62})$$

Una proyección ortogonal se obtiene cuando  $L_1 = 0$ , lo cual ocurre en un ángulo de proyección  $\alpha$  de  $90^\circ$ .

Las proyecciones oblicuas se generan con valores distintos de cero para  $L_1$ . La matriz de proyección ec62 tiene una estructura similar a la de una matriz de corte del eje z. de hecho, el efecto de esta matriz de proyección es el de cortar planos de z constante y proyectarlos sobre el plano de visión. Los valores de las coordenadas x y y dentro de ad plano de z constante se cambian en una cantidad proporcional al valor de z del plano de manera que los ángulos, distancias y líneas paralelas del plano se proyecten con exactitud.

Dos ángulos que se usan comúnmente en las proyecciones oblicuas son aquellas para las cuales  $\tan \alpha = 1$  y  $\tan \alpha = 2$ . En el primer caso  $\alpha = 45^\circ$  y las vistas que se obtienen se denominan **proyecciones caballera**. Todas las líneas perpendiculares al plano de proyección se proyectan sin cambio de longitud.

En el segundo caso, a la vista resultante se le llama **Proyección de Gabinete**. Este ángulo de proyección de aproximadamente  $63.4^\circ$  ocasiona que las líneas perpendiculares a la superficie de visión se proyecten en una mitad de su longitud. Las proyecciones de gabinete parecen más realistas que las de montura debido a esta reducción en la longitud de las perpendiculares.

### 3.5.2 Proyección Perspectiva<sup>17</sup>.

Para obtener una proyección en perspectiva de un objeto tridimensional, se proyectan puntos a lo largo de líneas de proyección que se intersectan en el centro de proyección.

En la figura No.31, el centro de proyección está en el eje z negativo a una distancia  $d$  detrás del plano de proyección. Puede seleccionarse cualquier posición para el centro de proyección, pero la elección de una posición a lo largo del eje z simplifica los cálculos en las ecuaciones de transformación.

Podemos obtener las ecuaciones de transformación de una proyección en perspectiva a partir de las ecuaciones de transformación de una proyección del punto  $P$  al centro de la proyección de la misma figura. La forma paramétrica de ésta línea de proyección es:

$$\begin{aligned}x' &= x - xu \\y' &= y - yu \\z' &= z - (z + d)u\end{aligned}\quad (\text{ec63})$$

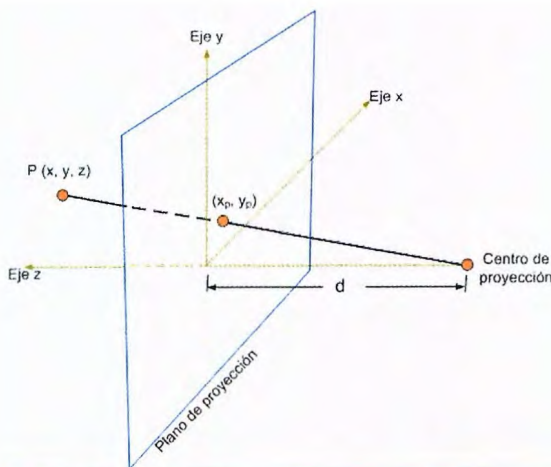


Figura No.31: Proyección perspectiva de un punto  $P$  en las coordenadas  $(x, y, z)$  a la posición  $(x_p, y_p, 0)$  sobre un plano de proyección

El parámetro  $u$  toma valores de 0 a 1 y las coordenadas  $(x', y', z')$  representan cualquier posición situada a lo largo de la línea de proyección. Cuando  $u = 0$ , las ecuaciones ec63 producen el punto  $P$  en las coordenadas  $(x, y, z)$ . En el otro extremo de la línea  $u = 1$  y se tienen las coordenadas del centro de proyección,  $(0, 0, -d)$ . Para obtener las coordenadas en el plano de proyección, se hace  $z' = 0$  y se resuelve para determinar el parámetro  $u$ :

<sup>17</sup> Lengyel Eric. Mathematics for 3D Game Programming and Computer Graphics. Editorial Charles Rivera Media. Capítulo 4. 3D Engine Geometry. Páginas 120-127.

$$(ec64) \quad u = \frac{z}{z + d}$$

El valor del parámetro  $u$  produce la intersección de la línea de proyección con el plano de proyección en  $(x_p, y_p, 0)$ . Al sustituir la ecuación ec64 en las ecuaciones ec63, se obtienen las siguientes ecuaciones de transformaciones de perspectiva:

$$(ec65) \quad \begin{aligned} X_p &= x \left( \frac{d}{z + d} \right) = x \left( \frac{1}{z/d + 1} \right) \\ y_p &= y \left( \frac{d}{z + d} \right) = y \left( \frac{1}{z/d + 1} \right) \\ z_p &= 0 \end{aligned}$$

Mediante una representación en coordenadas homogéneas tridimensionales, podemos escribir la transformación de la perspectiva en forma matricial

$$(ec66) \quad [x_h \ y_h \ z_h \ w] = [x \ y \ z \ 1] \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1/d \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

En ésta representación:

$$(ec67) \quad W = (z/d) + 1$$

Y las coordenadas de proyección en el plano se calculan a partir de las coordenadas homogéneas como:

$$(ec68) \quad [x_h \ y_h \ z_h \ w] = [x_h/w \ y_h/w \ z_h/w \ 1]$$



Cuando un objeto tridimensional se proyecta sobre un plano mediante ecuaciones de transformación de perspectiva, cualquier conjunto de líneas paralelas del objeto que no sean paralelas al plano se proyectan en líneas convergentes. Las líneas paralelas que son paralelas al plano se proyectan como tales. El punto en el cual un conjunto de líneas paralelas proyectadas parece converger se denomina **punto de fuga**. Cada conjunto de líneas paralelas proyectadas tendrá un punto de fuga aparte.

El punto de fuga de cualquier conjunto de líneas que son paralelas a uno de los ejes coordenados mundiales se conoce como **punto de fuga principal**. El número de éstos puntos se controla (uno, dos o tres) con la orientación del plano de proyección y las proyecciones perspectiva se clasifican según esto como **proyecciones de uno, dos o tres puntos**.

El número de los principales puntos de fuga de una proyección se determina por el número de ejes de coordenadas mundiales que cortan el plano de proyección. La figura No.32 ilustra el aspecto de proyecciones en perspectiva de una y dos puntos de un cubo. La figura No.32 (b), el plano de proyección se ha alineado paralelo al plano  $xy$ , de manera que sólo se corte el eje  $z$ . Esta orientación produce una proyección en perspectiva de un punto con un punto de fuga en el eje  $z$ . Para la vista que se muestra en la figura No.32 (c), el plano de proyección corta los ejes  $x$  y  $z$ , pero no el eje  $y$ . La proyección en perspectiva de dos puntos resultante contiene puntos de fuga en los ejes  $x$  y  $z$ .

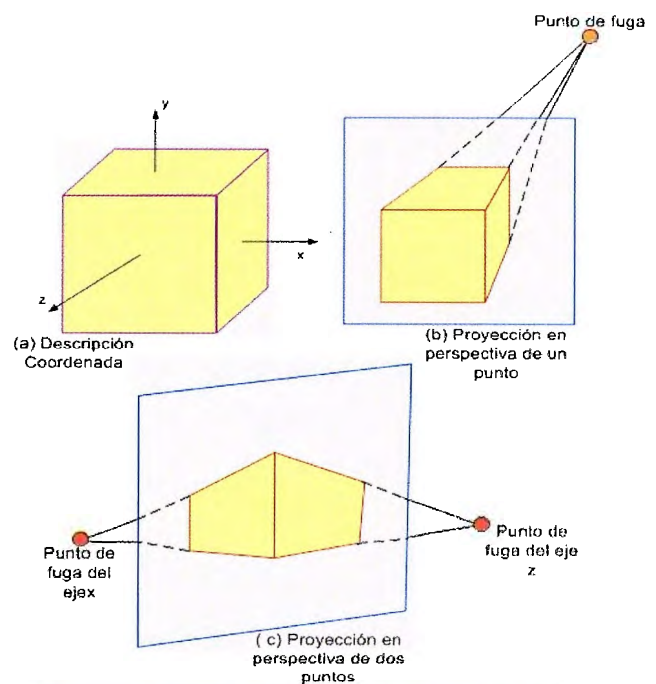


Figura No.32: Vistas en perspectiva de un cubo

### **3.6 Supresión de líneas y superficies ocultas<sup>18</sup>.**

Hasta el momento se conoce como construir y proyectar objetos tridimensionales, en los cuales se ven todas las partes del objeto teniendo así cierta calidad de transparencia. Las figuras que se obtienen de esta forma reciben el nombre de modelos alámbricos<sup>19</sup>.

Los objetos complejos tienen una enorme cantidad de segmentos tanto que pueden parecer segmentos sin sentido, podría ser difícil identificar que líneas pertenecen a la parte frontal de objeto y cuales están situadas detrás.

Para ello evitar este tipo de problemas se puede eliminar aquellas líneas que estarían ocultas por alguna de las partes del objeto, entonces se tendría libertad de construir el modelo completo de un objeto (frente, dorso, interior e exterior) y ser capaz todavía de representarlo tal como se vería en la realidad.

Existen muchos métodos que se pueden utilizar para resolver este problema y se han creado algoritmos para eliminar las partes ocultas de escenas en forma eficaz.

Algunos métodos requieren mas memoria, en algunos intervienen mas tiempo de procesamiento y algunos se aplican a tipos especiales de objetos, el método elegido para una aplicación determinada depende de factores tales como la complejidad de la escena y los tipos de objetos que se desplegaran.

Los algoritmos de líneas y superficies oculta se clasifican según las definiciones de objetos en forma directa o bien con imágenes proyectadas. Estos dos métodos se denominan:

---

<sup>18</sup> Hearn Donald, Gráficas por Computadora. Editorial Prentice Hall, Capítulo 13 Supresión de Superficies y Líneas Ocultas. Páginas 280-294

<sup>19</sup> Modelos alámbricos : sólo presentan los contornos de objetos supuestamente sólidos.

- Objeto-espacio: compara objetos y partes de objetos nos con otros para determinar que superficies y líneas, como un todo, deben rotularse como visibles.
- Imagen espacio: la visibilidad se decide punto por punto en cada posición de píxel sobre el plano de proyección.

Dentro de métodos que se utilizan más utilizados para eliminar superficies y líneas ocultan están:

- Supresión de la cara anterior.
- Método del buffer con profundidad.
- Método de la línea de rastreo.
- Método de ordenamiento con profundidad.
- Método del árbol octal.
- Superficies curvas.

### **Supresión de la cara anterior.**

Un método simple de objeto-espacio para identificar las caras anteriores de objetos se basa en la ecuación de un plano :

$$Ax + By + Cz + D = 0$$

Cualquier punto  $(x', y', z')$  especificado en un sistema de coordenadas del lado derecho esta en el interior de este plan si cumple la desigualdad:

$$Ax' + By' + Cz' + D < 0$$

Si el punto es la posición de visión, cualquier plano para el cual la desigualdad se cumple debe ser una cara anterior. Quiere decir que es aquella que no podemos observar desde la posición de visión.

Un ejemplo mas simple de cara anterior es teniendo un vector normal a un plano que describe la ecuación anterior, este vector tiene las componentes cartesianas (A, B, C): En un sistema de visualización del lado derecho con la dirección d visión a lo largo del eje  $z_v$  negativo, el vector normal tiene la componente C paralela a la dirección de visión.

Si  $C < 0$ , el vector normal apunta lejos de la posición de visión y el plano debe ser una cara anterior.

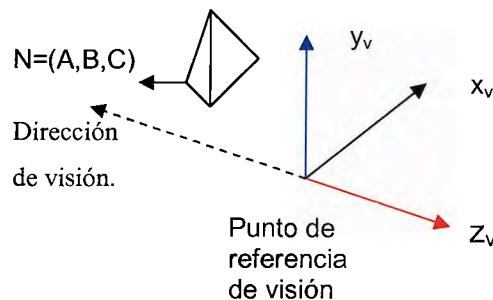


Figura No. 33: Plano don parámetro  $C < 0$  en sistema coordenado de visión del lado derecho que se identifica como una cara anterior cuando la dirección de visión es a lo largo del eje  $z_v$  negativo.

También se puede aplicar métodos similares en paquetes que utilizan un sistema de visión izquierdo, los parámetros A, B, C, y D pueden calcularse a partir de coordenadas de vértice especificadas en un sentido igual al reloj la desigualdad anterior permanece entonces como una prueba valida de puntos interiores, las caras anteriores tienen vectores normales que apuntan lejos de la posición de visión y se identifican por  $C > 0$ .

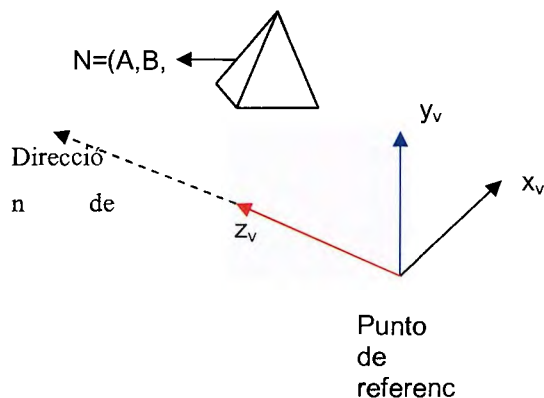


Figura No.34: En un sistema de visión del lado izquierdo con la dirección de visualización a lo largo del eje  $z_v$  positivo, una cara anterior es aquella con el parámetro plano  $C > 0$ .

## Método del buffer con profundidad.

Es un método de imagen-espacio que se utiliza comúnmente para eliminar superficies ocultas. Básicamente este algoritmo verifica la visibilidad de superficies en punto a la vez.

En cada posición del píxel  $(x, y)$  sobre el plano de visión muestra tres superficies en varias profundidades con respecto a  $(x, y)$  en un sistema de visión izquierdo.

La superficie  $S_1$  tiene el menor valor de  $z$  de manera que se salva su valor de intensidad en  $(x, y)$ , se requieren dos áreas diferentes para implementar este método. Un buffer con profundidad se utiliza para almacenar valores de  $z$  para cada posición  $(x, y)$  y a medida que se comparan las superficies y el buffer de renovación almacena los valores de intensidad de cada posición.

Puede implantarse adecuadamente en coordenadas normalizadas, con valores de profundidad que varían de 0 a 1.

Si se trabajara con un polígono la profundidad de los puntos situados sobre la superficie de este se calculan a partir de la ecuación del plano.

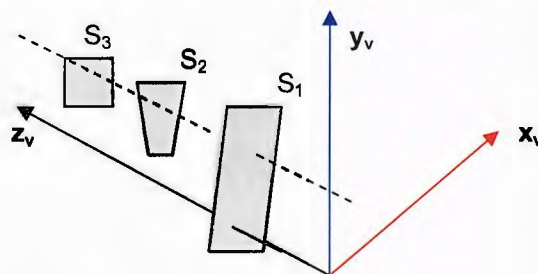


Figura No.35: En la posición  $(x, y)$ , la superficie  $S_1$  tiene el menor valor de profundidad y de esta manera es visible en esa posición.

Inicialmente, todas las posiciones

del buffer de renovación se inicializan en la intensidad del fondo. Cada superficie enlistada en las tablas se procesa después, una a la vez, calculando la profundidad o valor de  $z$ , en cada posición  $(x, y)$ .

Podemos resumir las etapas de un algoritmo de buffer con profundidad así:

1. Inicialícese el buffer con profundidad y el de renovación de manera que para todas las posiciones coordenadas  $(x, y)$ .

2. Para cada posición de cada superficie, compárense los valores de la profundidad con los valore previamente almacenados en el buffer para determinar la visibilidad.

- a) a) Calcúlense valores de  $z$  para cada posición  $(x, y)$  de la superficie.
- b) b) Si  $z < (x, y)$  entonces hágase en la coordenada  $(x, y) = z$  y  $(x, y) = i$ , donde  $i$  es el valor de la intensidad en la superficie en la posición  $(x, y)$ .

En el ultimo paso, si  $z$  no es menor que el valor del buffer con profundidad de esa posición, el punto no es visible.

### **Método de la línea de rastreo.**

Método de imagen-espacio para eliminar superficies ocultas es una extensión del algoritmo de línea de rastreo para llenar interiores de polígonos. En vez de llenar solo una superficie, ya que ahora se trabaja con múltiples superficies.

Se procesan cada línea de rastreo, todas las superficies poligonales que cortan esa línea se examinan a fin de determinar cuales son visibles.

En cada posición situada a lo largo de una línea de rastreo, se hacen cálculos de profundidad en cada superficie para determinar cual es la mas próxima al plano de visión.

Cuando se ha determinado la superficie visible el valor de intensidad de esa posición se mete en el buffer de renovación. Ya que la representación de las superficies poligonales de una escena tridimensional puede formarse para incluir una tabla de aristas y una de una de polígonos. La tabla de aristas contiene extremos coordenados de cada línea de la escena, la pendiente inversa de cada línea y apuntadores en la tabla de polígonos para identificar las superficies limitadas por cada línea.

La tabla de polígonos contiene coeficientes de la ecuación del plano para cada superficie, información de intensidad de las superficies y posibles apuntadores en la tabla de aristas.

Para facilitar la búsqueda de superficies que atraviesan una línea de rastreo dada, se puede preparar una lista activa de aristas a partir de la información contenida en la tabla de aristas. Esta contendrá solamente aristas que atraviesen la línea de rastreo regular, ordenada en el sentido de  $x$  creciente y una señal para cada superficie que se hace on o off para indicar si una posición a lo largo de una línea de rastreo esta dentro o fuera de la superficie.

Las líneas de rastreo se procesan de izquierda a derecha cuando esta mas a la izquierda de la superficie , la señal se activa; y cuando esta mas a la derecha se desactiva.

La figura No. 36, Ilustra el método de la línea de rastreo para localizar porciones visibles de superficies situadas en una línea de rastreo.

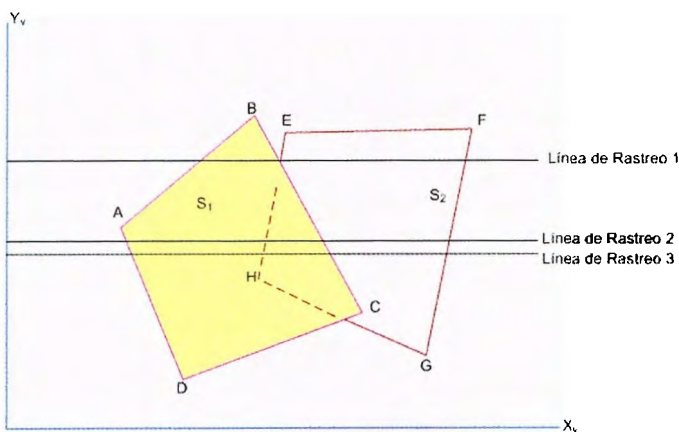


Figura No.36: Línea de rastreo que atraviesan la proyección de las superficies  $S_1$  en el plano de visión. Las líneas punteadas indican la frontera de superficies ocultas.

La lista activa de la línea de rastreo 1 contiene información de la tabla de aristas AB, BC, HE y EF. Para las posiciones en esta línea que se encuentran entre AB y BC, solo la señal de la superficie  $S_1$  esta activada. Por tanto no se necesitan cálculos de profundidad. Análogamente entre HE Y EF, solo la señal de la superficie  $S_2$  esta activada.

Debido a que ninguna otra posición contenida en la línea de rastreo 1 corta superficies, los valores de intensidad de las otra áreas se hacen igual a la intensidad de fondo.



Las listas activas de las líneas E de rastreo 2 y 3 de la figura No.36. Contienen las aristas DA, HE, BC, FG. En la línea de rastreo 2 de la arista DA a la arista HE, solo la señal de la superficie  $S_1$  esta activada. Pero entre HE y BC, ambas superficies están activadas. En este intervalo , los cálculos deben hacerse con los coeficientes del plano de las dos superficies. En el ejemplo anterior la profundidad de la superficie  $S_1$  se supone menor que la  $S_2$ , de modo que las intensidades de la superficie  $S_1$  se cargan al buffer de renovación hasta que se encuentra la frontera BC. Después se desactiva la señal de la superficie de  $S_1$  y las intensidades de la superficie  $S_2$  se almacenan hasta que se pasa la arista FG. Debido a que la línea de rastreo 2 tiene la misma lista activa de aristas que la de la arista 1 , no es necesario volver a hacer cálculos entre HE y BC .por tanto deben estar en la misma orientación como se determino en la línea de rastreo 1.

Puede procesarse cualquier numero de superficies poligonales en superposición con este método, En algunos casos es posible que las líneas oscurezcan una a la otra en forma alternativa, cuando se usan métodos de coherencia se necesita tener cuidado de llevar un registro de que sección de la superficie esta visible en cada línea de rastreo.

### **Método de ordenamiento con profundidad.**

En este método se puede utilizar operaciones imagen-espacio y de objeto- en un espacio un algoritmo de superficie oculta ya que el método de ordenamiento con profundidad es una combinación de estos métodos, y cumple con las siguientes funciones:

1. La superficie se ordenan en sentido de profundidad decreciente.
2. La superficie se convierte con rastreo den orden, comenzando con las superficies de máxima profundidad.



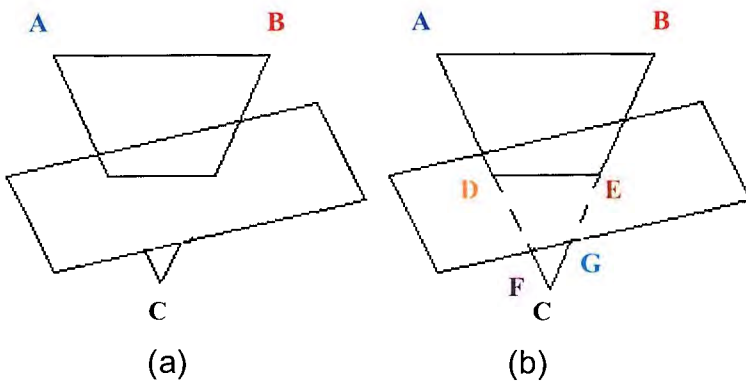


Figura No.37: División de una superficie en múltiples superficies para evitar problemas alternativos de visibilidad entre los planos.

Las operaciones de ordenamiento se efectúan en objeto-espacio y la conversión con rastreo de las superficies poligonales se realizan en imagen espacio.

Este método también se conoce como algoritmo del

pintor. Mediante una técnica análoga, primero se ordenan las superficies según su distancia desde el plano de visión. Los valores de intensidad de la superficie más alejada se introducen después en el buffer de renovación. Tomando cada superficie sucesiva en orden de profundidad creciente, se pintan las intensidades de la superficie en el buffer de estructura sobre las intensidades de las superficies que se han procesado.

El trazo de superficies poligonales sobre el buffer de estructura según la profundidad se lleva a cabo en varias etapas. Primero, la superficies se ordenan de acuerdo con el mayor valor de  $z$  en cada superficie. Segundo la superficie con máxima profundidad ( $S$ ) se compara después con las otras superficies de la lista para determinar si hay alguna superposición de profundidad que se ha proyectado en el plano  $xz$ . Este proceso se repite después con la siguiente superficie de la lista.

Mientras no ocurran superposiciones, cada superficie se procesa por orden de profundidad hasta que todas se hayan convertido con rastreo.

Si se detecta una superposición de profundidad en cualquier punto de la lista, se deben hacer otras comparaciones para determinar si una superficie debe reordenarse.

Las pruebas se enlistan en orden de dificultad creciente y son:

1. Los rectángulos limitantes en el plano  $xy$  de las dos superficies no se superponen.
2. La superficie  $S$  esta en la parte exterior de la superficie de superposición, relativa al plano de visión.
3. La superficie de superposición esta en la parte interior de la superficie  $S$ , relativa al plano de visión.
4. Las proyecciones de las dos superficies sobre el plano de visión no se superponen.

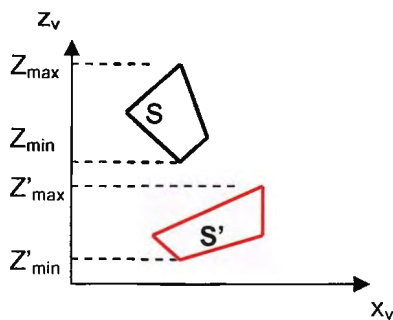


Figura No.38: Dos superficies sin superposición de profundidad

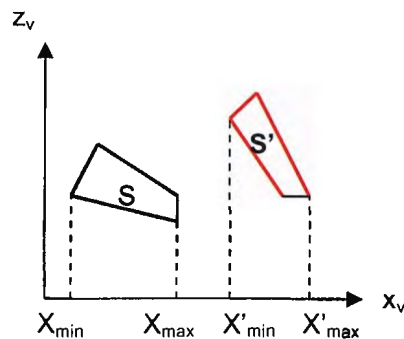


Figura No.39: Dos superficies con superposición de profundidad pero sin superposición en la dirección  $x$

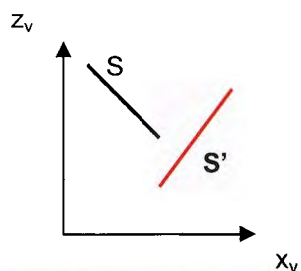


Figura No.40: La superficie S está completamente fuera de la superficie de superposición S' relativa al plano de visión.

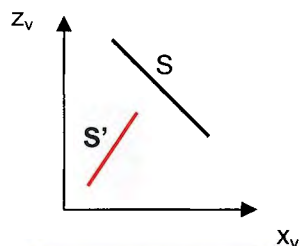


Figura No.41: La superficie de superposición S' está dentro de la superficie S, relativa al plano de visión.

Si alguna de las pruebas se cumple no necesita reordenamiento de esa superficie y puede convertirse en rastreo, y se sabe que dicha superficie no está detrás de S.

La prueba uno se realiza en dos partes : primero se verifica si hay sobre posición en la dirección x, luego en la dirección y. Si alguna de estas direcciones no muestra superposición, los dos planos no pueden oscurecerse entre si

Si las cuatro pruebas llegan a fallar con una superficie de superposición S' determinada, se intercambian las superficies S y S' en la lista ordenada .

En este algoritmo se puede presentar que entre a un ciclo infinito si dos o mas superficies se oscurecen alternativamente entre si, en tales situaciones el algoritmo reorganizara continuamente la posiciones de las superficies.

Para evitar estos ciclos se puede señalar alguna superficie que se haya reordenado en una posición de profundidad mas lejana de manera que no se pueda volver a mover. Si se cambia la superficie por segunda vez, esta se divide en dos partes en la línea de intersección de los dos planos que se someten a comparación. La superficie original es sustituida por las dos nuevas superficies y se continúa el proceso como antes.

## Método del árbol octal.

Se utiliza una representación de árbol octal para el volumen de visualización, la eliminación de las superficies de visión en orden del frontal anterior.

En la figura No.42 la cara frontal de una región de espacio se forma con los octantes 0, 1, 2 y 3 : las superficies situadas en el frente de estos octantes son visibles para el observador. Cualquiera superficie dirigida al fondo de octantes frontales o bien en los traseros 4, 5, 6 y 7 pueden ser ocultados por las superficies frontales.

Las superficies anteriores o traseras se eliminan, en la dirección de visualización, mediante el procesamiento de elementos de datos contenidos en los nodos del árbol octal en el orden 0, 1, 2, 3, 4, 5, 6, 7. Esto origina un recorrido transversal de primera profundidad del árbol, de manera que los nodos se representan en los octantes 0, 1, 2 y 3 de la región en su totalidad sean visitados antes que los nodos de los cuatro sub-octantes traseros. El recorrido transversal del árbol octal continúa en el orden de cada subdivisión del octante.

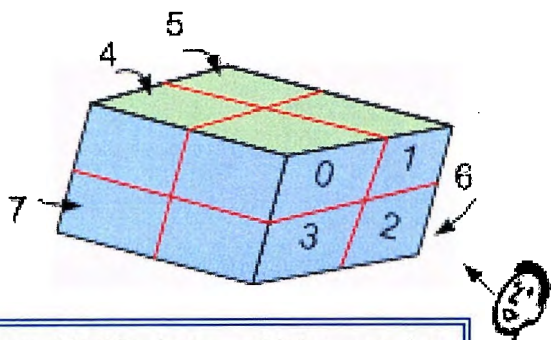


Figura No.42: Los objetos en los octantes 0, 1, 2 y 3 oscurecen los objetos contenidos en los octantes traseros 4, 5, 6, 7 cuando la dirección de visión es como se muestra.

Cuando se encuentra un valor de color en un nodo de un árbol, el área del píxel contenido en el buffer de estructura que corresponde a este recibe ese valor del color solo si no se han almacenado anteriormente valores en esta área.

En la mayoría de los casos, tanto el octante frontal como el anterior deben considerarse en la determinación de los

valores de color correctos de un cuadrante

## **Superficies curvas.**

Un método eficaz par la eliminación de las superficies ocultas de objetos con superficies curvas es la representación de árbol octal. Una vez establecidos los árboles octales a partir de la definición de entrada de los objetos, todas las eliminaciones efectúan los mismos procedimientos de procesamiento, no se da consideraciones especiales a las superficies curvas.

Otro método para manejar partes ocultas de una escena consiste en aproximar cada objeto como un conjunto de superficies poligonales planas. En la lista de superficies, después se sustituye cada superficie curva por la representación poligonal y se utiliza cualquiera de los métodos vistos anteriormente.

También se puede manipular directamente las ecuaciones de curva que definen las fronteras de superficies. Para muchos objetos de interés como esferas, elipses, cilindros y conos, utilizando la ecuación cuadrática para describirlas.

Las ecuaciones cuadráticas pueden expresarse de forma paramétrica y a menudo se utilizan técnicas de aproximación numérica para localizar intersecciones de las superficies curvas con una línea de rastreo.

### 3.7 Representación de Circunferencias Tridimensionales<sup>20</sup>

Aunque la esfera es un ejemplo de superficie curva, ésta puede generarse por medio de otras figuras geométricas, es decir que se puede comenzar el trazo con cualquier polígono regular cuyas facetas se puedan dividir inicialmente en triángulos. Para esto se utiliza la subdivisión recursiva, una poderosa técnica para generar aproximaciones a curvas y superficies a cualquier nivel de exactitud (Aunque OpenGL no apoya la construcción de una esfera, las librerías GLU y GLUT si las incluyen, construidas mediante superficies cuadráticas y aproximaciones poligonales, respectivamente.)

Tomemos como opción alternativa la figura del tetraedro, el cual está compuesto de cuatro triángulos equiláteros, determinado por cuatro vértices. Se comienza aquí con los siguientes cuatro vértices:

$$(0, 0, 1), (0, 2(\sqrt{2})/3, -1/3), (-\sqrt{6}/3, -(\sqrt{2})/3, -1/3), ((\sqrt{6})/3, -\sqrt{2}/3, -1/3)$$

Para que el concepto de trazo de circunferencias 3D quede más claro, generemos una circunferencia a partir de código C++, con ayuda de las herramientas gráficas de OpenGL, lo cual llevamos a cabo de la siguiente manera:

Teniendo los cuatro puntos en un círculo unitario, centrado en el origen. Se obtiene la primera aproximación dibujando un marco alambrado para el tetraedro, utilizando el tipo punto definido anteriormente, que marcan los cuatro vértices globales:

```
point v[4]={0.0, 0.0, 1.0}, {0.0, 0.942809, -0.33333},{-0.816497, -0.471405, -0.333333}, {0.816497, -0.471405, 0.333333};
```

<sup>20</sup> Lengyel Eric. Mathematics for 3D Game Programming and Computer Graphics. Editorial Charles Rivera Media. Capítulo 15 Curves and Surfaces. Páginas 453-472

Se pueden dibujar triángulos mediante la función:

```
void triangle( point a, point b, point c)
{
glBegin(GL_LINE_LOOP);
glVertex3fv(a);
glVertex3fv(b);
glVertex3fv(c);
glEnd();
}
```

El tetraedro se puede dibujar mediante:

```
void tetrahedron(void)
{
triangle(v[0], v[1], v[2]);
triangle(v[3], v[2], v[1]);
triangle(v[0], v[3], v[1]);
triangle(v[0], v[2], v[3]);
}
```

El orden de los vértices obedece a la regla de mano derecha, por lo cual se puede convertir para dibujar polígonos fácilmente. Si se añade el código típico de inicialización, el programa generaría una imagen como en la figura No.43; un poliedro regular simple, pero una pobre aproximación a una esfera.

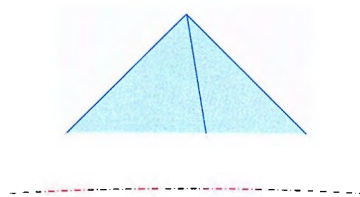


Figura No.43: Salida de programa; poliedro regular simple.

Se puede obtener una aproximación más cercana a una esfera subdividiendo cada faceta del tetraedro en triángulos más pequeños. Ésta subdivisión, asegura que las nuevas facetas sean planas. Existen al menos tres formas, como se muestra en la figura No.44:



## 3.7 Representación de Circunferencias Tridimensionales<sup>20</sup>

Aunque la esfera es un ejemplo de superficie curva, ésta puede generarse por medio de otras figuras geométricas, es decir que se puede comenzar el trazo con cualquier polígono regular cuyas facetas se puedan dividir inicialmente en triángulos. Para esto se utiliza la subdivisión recursiva, una poderosa técnica para generar aproximaciones a curvas y superficies a cualquier nivel de exactitud (Aunque OpenGL no apoya la construcción de una esfera, las librerías GLU y GLUT si las incluyen, construidas mediante superficies cuadráticas y aproximaciones poligonales, respectivamente.)

Tomemos como opción alternativa la figura del tetraedro, el cual está compuesto de cuatro triángulos equiláteros, determinado por cuatro vértices. Se comienza aquí con los siguientes cuatro vértices:

$$(0, 0, 1), (0, 2(\sqrt{2})/3, -1/3), (-\sqrt{6}/3, -(\sqrt{2})/3, -1/3), ((\sqrt{6})/3, -\sqrt{2}/3, -1/3)$$

Para que el concepto de trazo de circunferencias 3D quede más claro, generemos una circunferencia a partir de código C++, con ayuda de las herramientas gráficas de OpenGL, lo cual llevamos a cabo de la siguiente manera:

Teniendo los cuatro puntos en un círculo unitario, centrado en el origen. Se obtiene la primera aproximación dibujando un marco alambrado para el tetraedro, utilizando el tipo punto definido anteriormente, que marcan los cuatro vértices globales:

```
point v[4]={0.0, 0.0, 1.0}, {0.0, 0.942809, -0.333333},{-0.816497, -0.471405, -0.333333}, {0.816497, -0.471405, 0.333333};
```

---

<sup>20</sup> Lengyel Eric. Mathematics for 3D Game Programming and Computer Graphics. Editorial Charles Rivera Media. Capítulo 15 Curves and Surfaces. Páginas 453-472



Se pueden dibujar triángulos mediante la función:

```
void triangle( point a, point b, point c)
{
glBegin(GL_LINE_LOOP);
glVertex3fv(a);
glVertex3fv(b);
glVertex3fv(c);
glEnd();
}
```

El tetraedro se puede dibujar mediante:

```
void tetrahedron(void)
{
triangle(v[0], v[1], v[2]);
triangle(v[3], v[2], v[1]);
triangle(v[0], v[3], v[1]);
triangle(v[0], v[2], v[3]);
}
```

El orden de los vértices obedece a la regla de mano derecha, por lo cual se puede convertir para dibujar polígonos fácilmente. Si se añade el código típico de inicialización, el programa generaría una imagen como en la figura No.43; un poliedro regular simple, pero una pobre aproximación a una esfera.

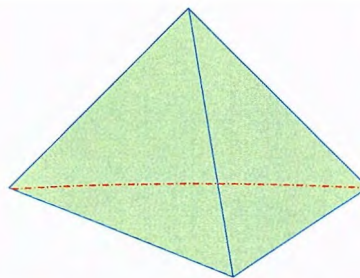
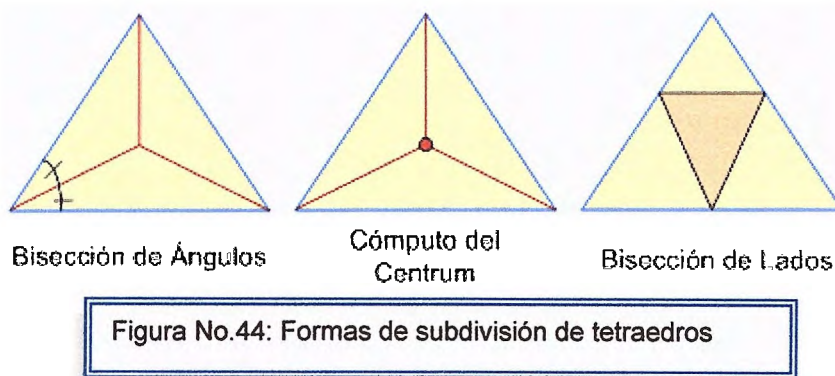


Figura No.43: Salida de programa; poliedro regular simple.

Se puede obtener una aproximación más cercana a una esfera subdividiendo cada faceta del tetraedro en triángulos más pequeños. Ésta subdivisión, asegura que las nuevas facetas sean planas. Existen al menos tres formas, como se muestra en la figura No.44:



**Bisección de Ángulos:** Se puede bisectar<sup>21</sup> cada ángulo del triángulo, y luego dibujar los tres bisectores, los cuales se encuentran en un punto común, generando de esta forma tres nuevos triángulos.

**Cómputo del Centrum:** Se puede computar el centro de masa (centrum) de los vértices simplemente promediándolos, y luego dibujando líneas de este punto a los tres vértices, generando de nuevo tres triángulos. Sin embargo, estas técnicas no preservan los triángulos equiláteros que constituyen los tetraedros regulares.

**Bisección de Lados:** En su lugar, recordando una posible construcción del Gasket de Sierpinski<sup>22</sup>, se puede conectar los bisectores de los lados del triángulo, formando cuatro triángulos equiláteros. Esta será la técnica utilizada aquí.

Después de subdividir una faceta mediante la bisección de lados, los cuatro nuevos triángulos estarán aún en el mismo plano que el triángulo original. Se puede mover los nuevos vértices creados por la bisección a la esfera unitaria al normalizar cada vértice bisectado, usando una función de normalización sencilla, como:

<sup>21</sup> Significa cortar o dividir en dos partes iguales. Si bisectamos un ángulo de 90 grados, tendremos dos ángulos de 45 grados.

<sup>22</sup> El Triángulo de Sierpinski, es un fractal generado con números aleatorios. Se tienen las coordenadas de los vértices A, B y C de un triángulo equilátero. Se hacen corresponder los números 0, 1 y 2 respectivamente a los vértices A, B y C. Se escoge un punto D coordenadas (x,y) dentro del triángulo. Enseguida se genera al azar un número 0, 1, ó 2 y si es 2 por ejemplo, se halla el punto medio de la línea que une C con D y se define allí un punto.

```

void normal(point p)
{
double sqrt();
float d =0.0;
int i;
for(i=0; i<3; i++) d+=p[i]*p[i];
d=sqrt(d);
if(d>0.0) for(i=0; i<3; i++) p[i]/=d;
}

```

Ahora se puede subdividir un solo triángulo, definido por los vértices numerados a, b, y c, como se muestra en la siguiente figura No.45, mediante el código:

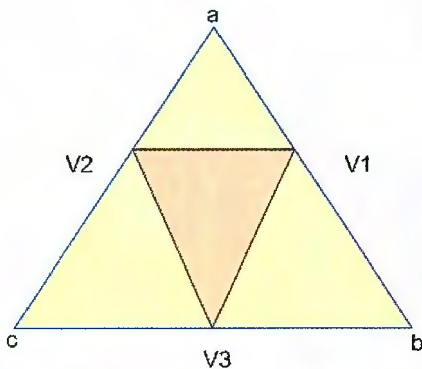


Figura No.45. Subdivisión de un solo triángulo.

```

point v1, v2, v3
int j;
for(j=0; j<3; j++) v1[j]=a[j]+b[j];
normal(v1);
for(j=0; j<3; j++) v2[j]=a[j]+c[j];
normal(v2);
for(j=0; j<3; j++) v3[j]=b[j]+c[j];
normal(v3);
triangle(a, v1, v2);
triangle(c, v2, v3);
triangle(b, v3, v1);
triangle(v1, v3, v2);

```

El resultado se muestra en la figura No.46:

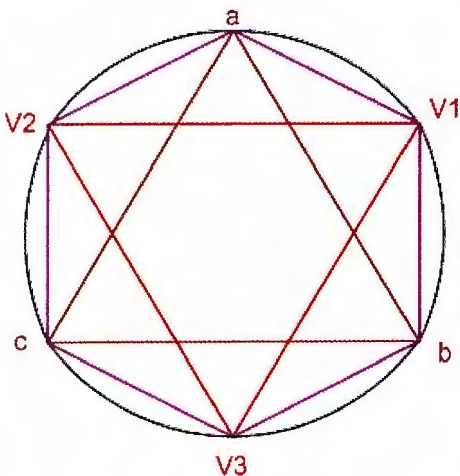


Figura No.46: Salida de programa, muestra más clara de generación de circunferencias a partir de triángulos.

Se puede usar este código en la rutina del tetraedro para generar 16 triángulos en lugar de cuatro, pero sería mejor poder repetir el proceso de subdivisión  $n$  veces para generar sucesivamente aproximaciones más cercanas a la esfera.

Llamando la rutina de subdivisión recursivamente<sup>23</sup>, se puede controlar el número de subdivisiones. Primero, se hace que la rutina del tetraedro dependa de la profundidad de recursión añadiendo el argumento  $m$

```
void tetrahedrum (int m)
{
    divide_triangle(v[0], v[1], v[2], m);
    divide_triangle(v[3], v[2], v[1], m);
    divide_triangle(v[0], v[3], v[1], m);
    divide_triangle(v[0], v[2], v[3], m);
}
```

La función `divide_triangle` se llamará a si misma para subdividir adicionalmente si  $m$  es mayor que cero, pero generará triángulos si  $m$  ha sido reducida a cero. El código es el siguiente:

```
void divide_triangle(point a, point b, point c, int m)
{
    point v1, v2, v3;
    int j;
    if(m>0)
    {
        for(j=0; j<3; j++) v1[j]=a[j]+b[j];
        normal(v1);
        for(j=0; j<3; j++) v2[j]=a[j]+c[j];
        normal(v2);
        for(j=0; j<3; j++) v3[j]=b[j]+c[j];
        normal(v3);
        divide_triangle(a, v1, v2, m-1);
        divide_triangle(c, v2, v3, m-1);
        divide_triangle(b, v3, v1, m-1);
        divide_triangle(v1, v3, v2, m-1);
    }
```

---

<sup>23</sup> Recursión es la forma en la cual se especifica un proceso basado en su propia definición. En el caso de funciones, una función es recursiva cuando se llama a sí misma. Es clásico el ejemplo de cálculo del factorial.

```
}  
else(triangle(a,b,c));  
}
```

De esta manera, al igual que en la figura No.47, se generará una circunferencia más exacta.

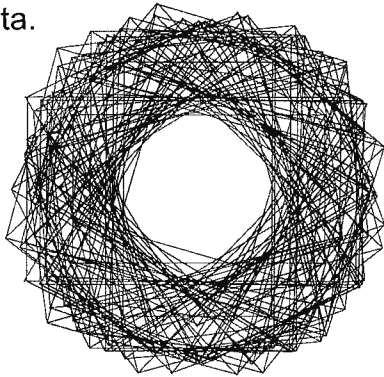


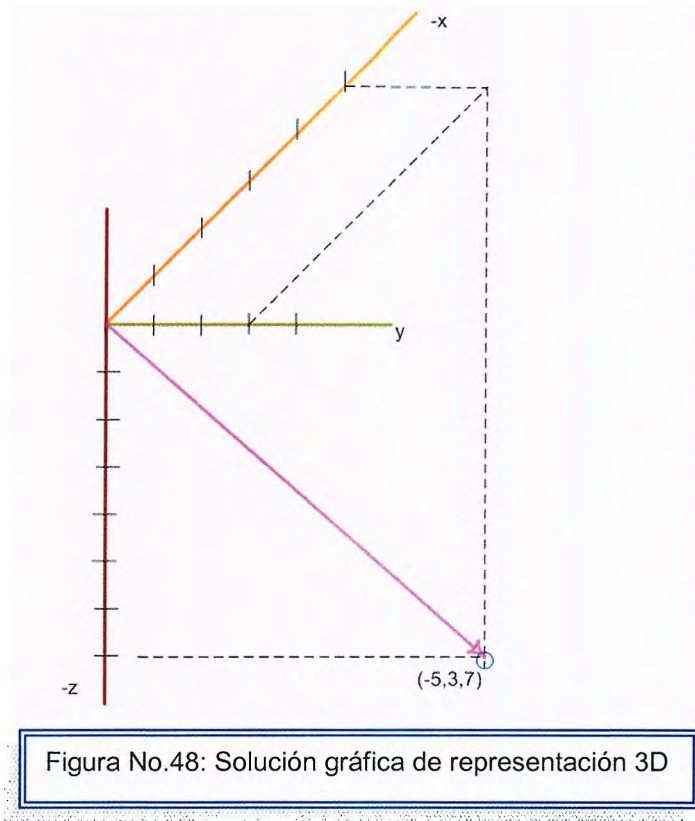
Figura No.47: Salida a partir de m llamadas a función recursiva, muestra una mejor aproximación de circunferencia 3D, generada con triángulos



## EJEMPLOS UNIDAD 3

**Ejemplo 3.1:** Realice la representación 3D del siguiente vector  $[-5, 3, -7]$  en los ejes tridimensionales.

Solución:



**Ejemplo 3.2:** Definir la línea que pasa parametricamente a través de los puntos  $P_1 = (5, 2, 3)$  y  $P_2 = (8, 4, 1)$  donde  $t = 0.09$

Solución:

$$\text{A partir de (ec1): } P(t) = (1-t)P_1 + tP_2$$

$$P(0.09) = (1 - 0.09) \cdot [5 \ 2 \ 3] + 0.09 \cdot [8 \ 4 \ 1]$$

$$P(0.09) = 0.91 \cdot [5 \ 2 \ 3] + [0.72 \ 0.36 \ 0.09]$$

$$P(0.09) = [4.55 \ 1.82 \ 2.73] + [0.72 \ 0.36 \ 0.09]$$

La línea que pasa parametricamente a través de los puntos  $P_1$  y  $P_2$  es:

$$P(0.09) = [5.27 \ 2.18 \ 2.82]$$

La recta pasa muy cerca del punto P1 ya que el parámetro  $t$  de la función es muy cercano a 0.

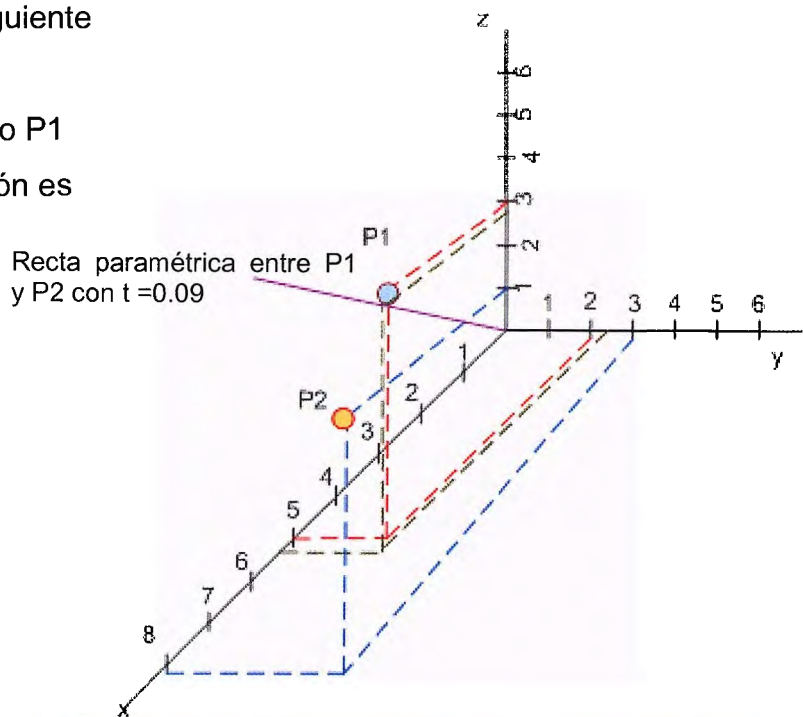


Figura No.49: Solución: Representación gráfica de recta paramétrica entre P1 y P2 con  $t = 0.09$

**Solución:**

$$d^2 = (Q - P_0)^2 - \left[ \frac{(Q - P_0) \cdot V}{V^2} \right]$$

Entonces:

$$V = Q - P_0 = [3 \ 8 \ 1] - [0 \ 0 \ 0] = [3 \ 8 \ 1]$$

$$d^2 = ([3 \ 8 \ 1] - [0 \ 0 \ 0])^2 - \left( \frac{([3 \ 8 \ 1] - [0 \ 0 \ 0]) \cdot [3 \ 8 \ 1]}{[3 \ 8 \ 1]^2} \right)$$

$$d^2 = [3 \ 8 \ 1]^2 - \left( \frac{([3 \ 8 \ 1]) \cdot [3 \ 8 \ 1])}{[36 \ 96 \ 12]} \right)$$

$$d^2 = [36 \ 96 \ 12] - \left( \frac{[36 \ 96 \ 12]}{[36 \ 96 \ 12]} \right) \rightarrow 1$$

$$d^2 = [35 \ 95 \ 13]$$

**Ejemplo 3.4:** Realizar el proceso de escalación uniforme para el punto [3 10 21].

Solución:

La fórmula para aplicar traslación de forma matricial la muestra la ec21:

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{pmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Para el punto [3 10 21] la escalación uniforme se define siguiendo los pasos de Escalación:

1. Trasládese el punto fijo al origen.

Es decir:  $X = 0 \quad Y = 0 \quad Z = 0$

2. realizar una escalación

Recordemos que  $Sx$   $Sy$  y  $Sz$  pueden ser cualquier valor positivo y de deben ser iguales para realizar la escalación uniforme.

$$Sx = Sy = Sz = 51$$

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} 3 & 10 & 21 & 1 \end{bmatrix} \begin{pmatrix} 51 & 0 & 0 & 0 \\ 0 & 51 & 0 & 0 \\ 0 & 0 & 51 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



**R// Al realizar la multiplicación de matrices se tiene:**

$$[x' \ y' \ z' \ 1] = [153 \ 510 \ 1071 \ 1]$$

3. Vuélvase a trasladar el punto fijo a su posición original

$$X=3 \quad Y=10 \quad Z=21$$

**Ejemplo 3.5:** Realizar la traslación del punto  $[-23 \ 14 \ 9]$ , en base a los parámetros de distancia  $T_x = -8 \quad T_y = 55 \quad T_z = 5$ .

Solución:

La ec23 permite realizar la traslación de forma matricial:

$$\begin{bmatrix} x' & y' & z' \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{pmatrix}$$

Entonces para el punto  $[-23 \ 14 \ 9]$  tenemos:

$$\begin{bmatrix} x' & y' & z' \end{bmatrix} = \begin{bmatrix} -23 & 14 & 9 & 1 \end{bmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -8 & 55 & 5 & 1 \end{pmatrix}$$

Realizando la multiplicación matricial se obtiene:

$$[x' \ y' \ z'] = [(-23 + 184) \quad (14+770) \quad (9+45) \quad (1)]$$

**R// El nuevo punto con proceso de traslación es:**

$$[x' \ y' \ z'] = [161 \ 784 \ 54 \ 1]$$

**Ejemplo 3.6:** Realice la rotación en los ejes X, Y y Z para el punto [38 75 2], con un ángulo de rotación de 33.55° (Recuerde operar los datos en grados)

Solución:

Rotación en torno al eje x.

$$x' = x$$

$$y' = y \cdot \cos \Theta - z \cdot \sin \Theta$$

$$z' = y \cdot \sin \Theta + z \cdot \cos \Theta$$

Entonces:

$$x' = 38$$

$$y' = 75 \cos(33.55) - 2 \sin(33.55) = 64.82 - 1.005 = 63.81$$

$$z' = 75 \sin(33.55) + 2 \cos(33.55) = 0.50 + 1.72 = 2.23$$

De forma matricial:

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \Theta & \sin \Theta & 0 \\ 0 & -\sin \Theta & \cos \Theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} 38 & 75 & 2 & 1 \end{bmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos 33.55 & \sin 33.55 & 0 \\ 0 & -\sin 33.55 & \cos 33.55 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} 38 & 75 & 2 & 1 \end{bmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.86 & 0.50 & 0 \\ 0 & -0.50 & 0.86 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

El producto de las matrices es:

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} 38 & 63.3 & 2.72 & 1 \end{bmatrix}$$

Si se observa el valor matricial es casi igual al valor obtenido al de las ecuaciones, ambos métodos son válidos.

Rotación en torno al eje y.

$$y' = y$$

$$x' = z \operatorname{sen} \Theta + x \cos \Theta$$

$$z' = z \cos \Theta - x \operatorname{sen} \Theta$$

Entonces:

$$y' = 75$$

$$x' = 2 \operatorname{sen}(33.55) + 38 \cos(33.55) = 1.005 + 32.844 = 33.85$$

$$z' = 2 \cos(33.55) - 38 \operatorname{sen}(33.55) = 1.728 - 19.11 = -17.38$$

De forma matricial:

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{pmatrix} \cos \Theta & 0 & -\operatorname{sen} \Theta & 0 \\ 0 & 1 & 0 & 0 \\ \operatorname{sen} \Theta & 0 & \cos \Theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} 38 & 75 & 2 & 1 \end{bmatrix} \begin{pmatrix} \cos 33.55 & 0 & -\operatorname{sen} 33.55 & 0 \\ 0 & 1 & 0 & 0 \\ \operatorname{sen} 33.55 & 0 & \cos 33.55 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} 38 & 75 & 2 & 1 \end{bmatrix} \begin{pmatrix} 0.86 & 0 & -0.50 & 0 \\ 0 & 1 & 0 & 0 \\ 0.50 & 0 & 0.86 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

El producto de las matrices es:

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} 34.53 & 75 & -17.85 & 1 \end{bmatrix}$$

Rotación en torno al eje z.

$$z' = z$$

$$x' = x \cos \Theta - y \sin \Theta$$

$$y' = x \sin \Theta + y \cos \Theta$$

Entonces:

$$z' = 2$$

$$x' = 38 \cos (33.55) - 75 \sin (33.55) = -4.88$$

$$y' = 38 \sin (33.55) + 75 \cos (33.55) = 83.94$$

De forma matricial:

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{pmatrix} \cos \Theta & \sin \Theta & 0 & 0 \\ -\sin \Theta & \cos \Theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} 38 & 75 & 2 & 1 \end{bmatrix} \begin{pmatrix} \cos 33.55 & \sin 33.55 & 0 & 0 \\ -\sin 33.55 & \cos 33.55 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} 38 & 75 & 2 & 1 \end{bmatrix} \begin{pmatrix} 0.86 & 0.50 & 0 & 0 \\ -0.50 & 0.86 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

El producto de las matrices es:

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} -4.75 & 83.91 & 2 & 1 \end{bmatrix}$$

**Ejemplo 3.7:** Realizar una rotación plana en torno a los ejes X, Y y Z, para el punto [9 15 21]. Con un ángulo de rotación de  $95^\circ$

Solución:

Rotación plana en torno a X:

$$x' = x$$

$$y' = y \cos \Theta - z \sin \Theta$$

$$z' = y \sin \Theta + z \cos \Theta$$

$$x' = 9$$

$$y' = 15 \cos 95^\circ - 21 \sin 95^\circ = 1.176 - 20.93 = -19.76$$

$$z' = 15 \sin 95^\circ + 21 \cos 95^\circ = 14.95 + 1.64 = 16.60$$

Rotación plana en torno a y:

$$y' = 15$$

$$x' = x \cos \Theta + z \sin \Theta$$

$$z' = -x \sin \Theta + z \cos \Theta$$

$$y' = 15$$

$$x' = 9 \cos 95^\circ + 21 \sin 95^\circ = 0.70 + 20.35 = 21.65$$

$$z' = -9 \sin 95^\circ + 21 \cos 95^\circ = -8.97 + 1.64 = -7.32$$

Rotación Plana en torno a z:

$$z' = z$$

$$x' = x \cos \Theta - y \sin \Theta$$

$$y' = x \sin \Theta + y \cos \Theta$$

$$z' = 21$$

$$x' = 9 \cos 95^\circ - 15 \sin 95^\circ = 0.70 - 14.95 = -14.24$$

$$y' = 9 \sin 95^\circ + 15 \cos 95^\circ = 8.97 + 1.18 = -7.79$$

**Ejemplo 3.8:** Programa ejemplo de Rotación con ayuda de la herramienta  
OPENGL.

```
#include <GL/glut.h>
#include <string.h>
#include "CWin.h"

#pragma comment (lib,"glut32.lib")
#pragma comment (lib,"opengl32.lib")

//inicializa glut
void CWin::InitWin(int argc,char **argv)
{
    glutInit(&argc,argv);
}

//Crea nuestra ventana
void CWin::CreateWin(int iWidth,int iHeight,const char *Title,bool bFullScreen)
{
    glutInitDisplayMode(GLUT_RGB | GLUT_RGBA | GLUT_DOUBLE |
    GLUT_DEPTH);
    if(bFullScreen){
        glutInitWindowPosition(100,100);
        glutInitWindowSize(iWidth,iHeight);
        glutCreateWindow(Title);
        glutFullScreen();
        ChangeSizeWin(iWidth,iHeight);
        strTitle=new char[strlen(Title)+1];
        strcpy(strTitle,Title);
        Width=iWidth;
        Height=iHeight;
    }
    else{
        glutInitWindowPosition(100,100);
        glutInitWindowSize(iWidth,iHeight);
        glutCreateWindow(Title);
        ChangeSizeWin(iWidth,iHeight);
        strTitle=new char[strlen(Title)+1];
        strcpy(strTitle,Title);
        Width=iWidth;
        Height=iHeight;
    }
}

//Devuelve el titulo
```

```

char *CWin::GetCaption()
{
    return strTitle;
}

//Devuelve el tamaño de la ventana
void CWin::GetSizeWin(int &iWidth,int &iHeight)
{
    iWidth =Width;
    iHeight=Height;
}

//Limpia el buffer de profundidad
void CWin::ClearWin()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
}

//Cambia el tamaño de la ventana
void CWin::ChangeSizeWin(int iWidth,int iHeight)
{
    if (iHeight==0){ //Previene una division entre cero
        iHeight=1;
    }
    glViewport(0,0,iWidth,iHeight);    //Ajusta la vista a las dimensiones de la
    ventana
    glMatrixMode(GL_PROJECTION);    //Activa la matriz de proyeccion
    glLoadIdentity();                //Reinicia el sistema de coordenadas
    gluPerspective(28.0f,(GLfloat)iWidth/(GLfloat)iHeight, 1.0f ,1000.0f);
    glMatrixMode(GL_MODELVIEW);     //Activa la matriz del modelador
    glLoadIdentity();                //Reinicia el sistema de coordenadas
}

void Triangle();
void Render();
void Idle(void);

float Ang=0.0f;

CWin Win;
int main(int argc, char **argv)
{

```

```

Win.InitWin(argc,argv);
Win.CreateWin(540,380,"Ejemplo3: Rotacion.",false);
glutDisplayFunc(Render);
glutIdleFunc(Idle);
glutMainLoop();
return 0;
}

```

```

void Idle(void)
{
    Ang++;
    Render();
}

```

```

void Triangle()
{
    glTranslatef(0.0f,0.0f,-6.0f);
    glRotatef(Ang,0.0f,1.0f,0.0f);
    glBegin(GL_TRIANGLES);
        glColor3f(0.38f,0.55f,0.73f);
        glVertex3f( 0.0f, 1.0f, 0.0f);
        glColor3f(0.10f,0.11f,0.15f);
        glVertex3f(-1.0f,-1.0f, 0.0f);
        glColor3f(0.98f,0.98f,0.98f);
        glVertex3f( 1.0f,-1.0f, 0.0f);
    glEnd();
}

```

```

void Render()
{
    Win.ClearWin();
    Triangle();
    glutSwapBuffers();
}

```



Salida del programa:

Este código presenta un triángulo, rotando alrededor del eje y, así

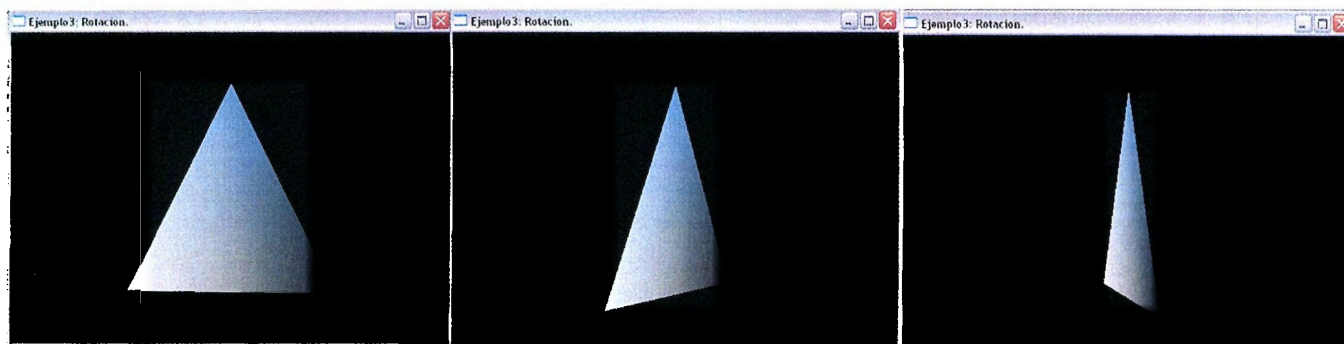


Figura No. 50: Rotación de Figura en torno al eje y con OpenGL

La función que permite cambiar el eje de rotación es `GLROTATEF`, en donde si cambiamos el segundo parámetro `glRotatef(Ang,1.0f,0.0f,0.0f)`; la función rotará la figura en el eje x así:

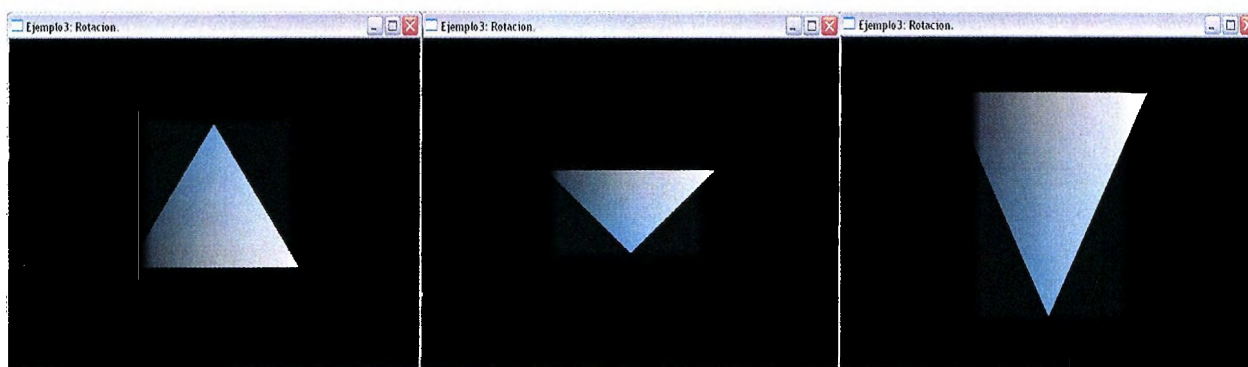


Figura No.51: Rotación de figura en torno al eje x con OpenGL

# 4

## **COLOR, ILUMINACIÓN, SOMBREADO Y TEXTURA**

---

## 1.4 Modelo de color<sup>1</sup>

Antes de conocer los que es el modelo de color se debe saber como el ojo humano ve estas ondas y distingue unas de otras, en el fondo del ojo existen millones de células especializadas en detectar las longitudes de onda procedentes de nuestro entorno estas células, principalmente los conos y bastoncillos<sup>2</sup>, recogen las diferentes partes del espectro de luz<sup>3</sup> solar y las transforman en impulsos eléctricos, que son enviados luego al cerebro a través de los nervios ópticos, siendo éste el encargado de crear la sensación del color.

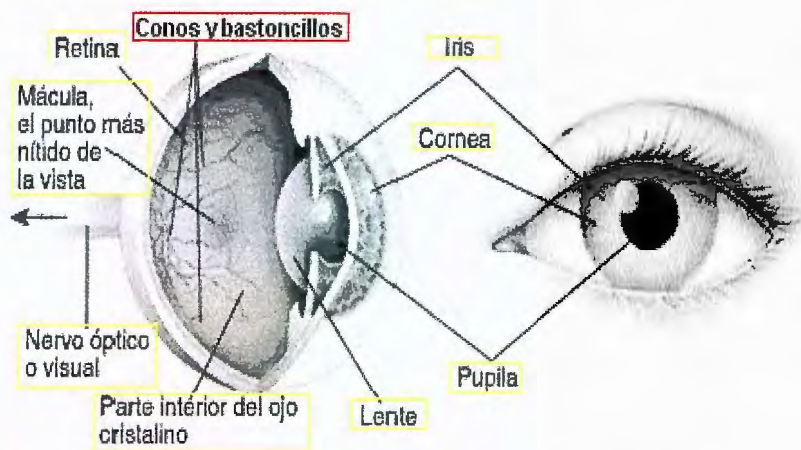


Figura No. 1 Ojo humano, ubicación de conos y bastoncillos

Los conos se concentran en una región cerca del centro de la retina llamada fovea. La cantidad de conos es de 6 millones y algunos de ellos tienen una terminación nerviosa que va al cerebro es así como se puede medir con exactitud la longitud de onda de un color determinado, pero no el concepto producido es totalmente subjetivo dependiendo de la persona en si.

<sup>1</sup> Modelos de color: es un método para explicar las propiedades o el comportamiento del color dentro de algún contexto determinado. Hearn Donal. Gráficas por Computadora. Editorial Prentice Hall Capítulo 14 páginas 318.

<sup>2</sup> Conos y bastoncillos: dos tipos de células sensibles a la luz, se encuentran debajo de las fibras nerviosas, por lo tanto la luz debe atravesarlas primero.

<sup>3</sup> Espectro de luz: es cuando la luz del sol al pasar a través de un prisma, y se divide en varios colores conformando un espectro.

Los conos son los responsables de la visión del color y se cree que hay tres tipos de conos, sensibles a los colores rojo, verde y azul, respectivamente.

La luz de sol al pasar a través de un prisma se divide en varios colores conformando un espectro.

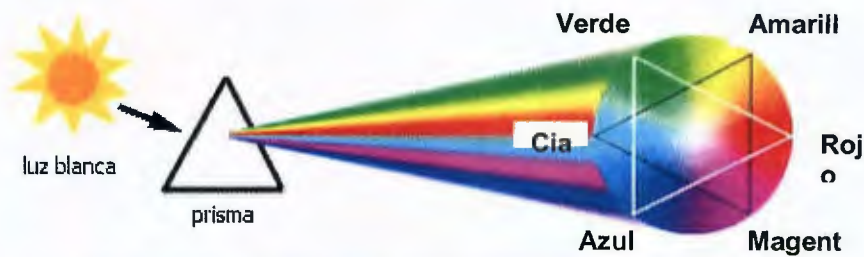


Figura No. 2 Ejemplo de formación de un espectro de Luz.

Así es como observa que la luz natural está formada por seis colores, cuando incide sobre un elemento absorbe algunos de esos colores y refleja otros. Con esta observación dio lugar al siguiente principio: todos los cuerpos opacos al ser



Figura No.3: Espectro de luz visible ante el ojo humano. Estos colores son básicamente Azul violáceo, Azul celeste, Verde, Amarillo, Rojo anaranjado y Rojo púrpura.

iluminados reflejan todos o parte de los componentes de la luz que reciben.

La luz es una banda de frecuencia

electromagnética, algunas otras bandas se denominan onda de radio, ondas infrarrojas, microondas y rayos X. Formando un espectro continuo de radiaciones, que comprende desde longitudes de onda muy pequeñas, (rayos cósmicos), hasta longitudes de onda muy grandes, de más de 1 kilómetro.

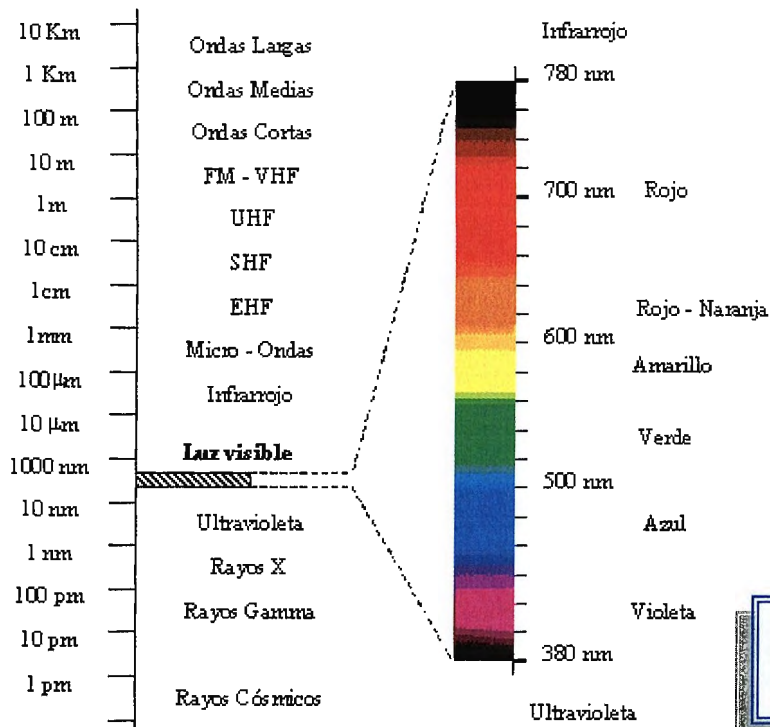


Figura No.4: Espectro electromagn3tico desde un pic3metro hasta un kil3metro.

Por lo tanto cuando vemos una superficie roja, realmente estamos viendo una superficie de un material que contiene un pigmento el cual absorbe todas las ondas electromagn3ticas que contiene la luz blanca con excepci3n de la roja, la cual al ser reflejada, es captada por el ojo humano y decodificada por el cerebro como el color denominado rojo.

Los colores obtenidos directamente naturalmente por descomposici3n de la luz solar o artificialmente mediante focos emisores de luz de una longitud de onda determinada se denominan aditivos<sup>4</sup>.

No es necesaria la uni3n de todas las longitudes del espectro visible para obtener el blanco, ya que si mezclamos solo rojo, verde y azul obtendremos el mismo resultado. Es por esto por lo que estos colores son denominados colores primarios

<sup>4</sup> Colores aditivos: la suma de todos los colores primarios produce el blanco.



porque la suma de estos tres produce el blanco, y todos los colores pueden ser obtenidos a partir de ellos.

¿Que es el color? : es un atributo que percibimos de los objetos cuando hay luz. Básicamente se dividen en dos tipos: primarios<sup>5</sup> y complementarios<sup>6</sup>.

## Los modelos de color

El modo de color es un método que expresa las propiedades y el comportamiento del color dentro de algún contexto determinado a la igual cantidad máxima de datos de color que se pueden almacenar en un determinado formato de archivo gráfico.

Estos modelos se dividen en dos:

1. Modelos Perceptivos: basados en la percepción humana
2. Modelos Reproductivos: se basan en los condicionantes físicos para la reproducción del color.

Dentro de los modelos perceptivos se encuentra:

- Tono, matiz o croma.
- Saturación.
- Brillo.

- Tono (hue), Es la percepción de un tipo de color, normalmente la que se distingue en un arco iris, es decir, es la sensación humana de acuerdo a la cual un área parece similar a otra o cuando existe un



Figura No.5 Diferencia de tono y matiz con igual saturación.

<sup>5</sup> Colores primarios: Son los tres colores principales de la composición de la luz (Rojo, Azul, Verde)

<sup>6</sup> Colores complementarios: Estos son llamados colores pigmento o de impresión, se forman a partir de las combinaciones de los colores luz, estos son: azul, rojo y amarillo en su forma original en donde se llaman cyan, magenta y amarillo; también son llamados los colores puros.

tipo de longitud de onda dominante.

- **Saturación:(saturation)** Se refiere a la cantidad del color o a la "pureza" de este. Describe la limpidez o cuan oscuro se ve el color de la luz. Va de un color "claro" a un color más vivo (azul cielo – azul oscuro). También se puede considerar como la mezcla de un color con el blanco.



Figura No.6 Diferencia de brillo, luminosidad, con tono y saturación constante.

- **Brillo ( brightness)** Es la cantidad de luz o blanco que se le añade o quita a un color.



Figura No. 7: Diferencia de saturación con igual valor y color.

Dentro de los modos reproductivos utilizados en aplicaciones gráficas se encuentran<sup>7</sup>:

### **Modo Bit Map o monocromático**

Está compuesto por un solo bit de color por píxel, requiere la mínima cantidad de memoria de todos los modos de imagen, la cual es monocromática formada exclusivamente por los colores blanco y negro puros, sin tonos intermedios entre ellos.

Para convertir una imagen a modo monocromático se debe transformar antes a modo escala de grises.

<sup>7</sup> D. Foley, James. Computer Graphics: Principles and Practice. Capítulo 13. Achromatic and Colored Light páginas 584-599

## **Modo Escala de Grises**

Están constituidas por 8 bit de información por pixel y usan de 0 a 255 escalas de grises entre blanco y negro simulando gradación de color. Este modo maneja un solo canal (el negro) para trabajar con imágenes monocromáticas

Las imágenes producidas con escáneres en blanco y negro o en escala de grises se visualizan normalmente en el modo escala de grises.

El modo Escala de Grises admite cualquier formato de grabación, y salvo las funciones de aplicación de color, todas las herramientas de los programas gráficos funcionan de la misma manera a como lo hacen con otras imágenes de color.

Si se convierte una imagen modo de color a un modo Escala de Grises y después se guarda y se cierra, sus valores de luminosidad permanecerán intactos, pero la información de color no podrá recuperarse.

## **Modo Color Indexado**

Son imágenes que usan un solo canal (8 bit x pixel) y tienen una tabla de 256 colores. Estas imágenes son útiles a la hora de editar la tabla de color o cuando una paleta de colores limitada se va a exportar, también resulta útil para trabajar con algunos formatos que sólo admiten la paleta de colores del sistema, a la vez reduce una imágenes a color 8 bits para su utilización en aplicaciones multimedia, ya que con ello se consiguen ficheros de menos peso.

El principal inconveniente es que la mayoría de las imágenes del mundo real se componen de más de 256 colores. Además, aunque admite efectos artísticos de color, muchas de las herramientas de los principales programas gráficos no están operativas con una paleta de colores tan limitada.



## **Modo Color RGB**

Las imágenes RGB usan tres colores para reproducir 16.7 millones de colores en la pantalla de un computador, trabaja con tres canales, ofreciendo una imagen tricromática compuesta por los colores primarios de la luz, Rojo(R), Verde(G) y Azul(B), construida con 8 bits/píxel por canal (24 bits en total).

Es un modelo de color aditivo, siendo el estándar de imagen de todo color que se utilice con monitores de video y pantallas de ordenador.

Las imágenes de color RGB se obtienen asignando un valor de intensidad a cada píxel, desde 0 (negro puro) a 255 (blanco puro) para cada uno de los componentes RGB.

## **Modo Color CMYK**

Consisten en 4 colores usados para impresión y separación de colores Cyan (C), Magenta (M), Amarillo(Y) y Negro (K).. Estos son 4 canales de imagen; contienen 32 bit por píxel (8 x 4).

un modelo de color sustractivo, en el que la suma de todos los colores primarios produce teóricamente el negro, que proporciona imágenes a todo color).

El proceso de convertir una imagen RGB al formato CMYK crea una separación de color. En general, es mejor convertir una imagen al modo CMYK después de haberse modificado ya que usa la mayor cantidad de memoria de todos los tipos de imagen, es generalmente lenta para edición de imagen.

## **Modo Color Lab**

Consiste en tres canales, cada uno de los cuales contiene hasta 256 tonalidades diferentes: un canal L de Luminosidad y dos canales cromáticos, A (que oscila entre verde y rojo) y B (que oscila entre azul y amarillo). El componente de

luminosidad L va de 0 (negro) a 100 (blanco). Los componentes A (eje rojo-verde) y B (eje azul-amarillo) van de +120 a -120.

Este modo es independiente del dispositivo, es decir, crea colores coherentes con independencia de los dispositivos concretos, como monitores, impresoras u ordenadores utilizados para crear, o reproducir, la imagen.

Este modo permite cambiar la luminosidad de una imagen sin alterar los valores de tono y saturación del color, siendo adecuado para transferir imágenes de unos sistemas a otros, pues los valores cromáticos se mantienen independientes del dispositivo de salida de la imagen.

Es utilizado para trabajar en imágenes donde se modifica la iluminación y los valores del color de una imagen independientemente..

### **Modo Duotono<sup>8</sup>**

El modo de Duotono se utiliza para incrementar el rango de grises en las imágenes, a las que se le pueden añadir tintas planas (3 para cada imagen, más el negro), con el fin de colorear distintas gamas de grises.. Es usado para monotonos - duotonos – tritonos o tetratonos, pero aún no es policromía. Son esencialmente imágenes en escala de grises de un solo canal (8 bit por pixel).

Con este método podemos obtener fotos en blanco y negro viradas al color que queramos. Suele ser empleado en impresión, donde se usan dos o más planchas para añadir riqueza y profundidad tonal a una imagen de escala de grises.

El problema que presenta este modo es que en los duotonos, tritonos y cuadratonos sólo hay un canal, por lo que no es posible tratar cada tinta de forma distinta según las zonas de la imagen.

### **Modo Multicanal**

---

<sup>8</sup> Imágenes en escala de grises que se imprimen con dos o más tintas, generalmente se utilizan para generar una tonalidad de foto antigua.

Posee múltiples canales de 256 niveles de grises, descomponiendo la imagen en tantos canales alfa como canales de color tuviera el original (una imagen RGB quedará descompuesta en 3 canales y una CMYK en 4 canales).

En este modo, cada tinta es un canal que a la hora de imprimir se superpondrá en el orden que determinemos sobre los otros. Por ello, es posible tratar cada zona de forma particularizada.

Se utiliza en determinadas situaciones de impresión en escala de grises. También, para ensamblar canales individuales de diversas imágenes antes de convertir la nueva imagen a un modo de color, pues los canales de color de tinta plana se conservan si se convierte una imagen a modo multicanal.

Al convertir una imagen en color a multicanal, la nueva información de escala de grises se basa en los valores de color de los píxeles de cada canal. Si la imagen estaba en modo CMYK, el modo multicanal crea canales de tinta plana cian, magenta, amarilla y negra. Si estaba en modo RGB, se crean canales de tinta plana cian, magenta y amarilla.

**Tabla resumen de modos de Color<sup>9</sup>.**

MODO COLOR	Características	Comentario
<b>Mapa de bits</b>	<ul style="list-style-type: none"> <li>· Imágenes monocromáticas: píxeles sólo 100% negro, ó 100% blanco.</li> <li>· Un solo canal de color</li> </ul>	<p>No disponible: capas, filtros, comando ajustar.</p> <p>La imagen tiene que estar en modo escala de grises antes de convertirla a Mapa de bits.</p>
<b>Escala de Grises</b>	<ul style="list-style-type: none"> <li>· Imágenes monocromáticas de tono continuo: Los píxeles son negros, blancos, ó hasta 254 matices de gris.</li> <li>· Admite cualquier formato de</li> </ul>	<p>Si se convierte una imagen modo de color a un modo Escala de grises y después se guarda y se cierra, sus valores de luminosidad permanecerán intactos, pero la información de color no podrá recuperarse.</p>

<sup>9</sup> Buss Samuel R. 3D Computer Graphics: A Mathematical Introduction with OpenGL. Capítulo 6 Transformations and Viewing. Páginas 149-153.

	grabación	
	<ul style="list-style-type: none"> <li>· Sólo posee un canal de color (negro)</li> </ul>	
<b>Color Indexado</b>	<ul style="list-style-type: none"> <li>· Tiene un canal y una tabla de color que contiene un máximo de 256 colores (color de 8 bits).</li> <li>· Sólo posee un canal de color (indexado)</li> </ul>	<p>En este modo la gama de colores de la imagen se adapta a una paleta con un número restringido de ellos. Útil por ejemplo para salvar en algunos formatos de imagen que sólo admiten la paleta de colores del sistema (que se reduce a 256 tonos diferentes)</p> <p>A veces resulta útil reducir las imágenes a color 8 bits para su utilización en aplicaciones multimedia. También se puede convertir para crear efectos artísticos de color</p>
<b>Color RGB</b>	<ul style="list-style-type: none"> <li>· Imágenes a todo color</li> <li>· Admite cualquier formato de grabación</li> <li>· Tiene tres canales (rojo, verde y azul)y-además los canales alfa.</li> </ul>	<p>Algunas aplicaciones de video y multimedia pueden importar una imagen RGB</p> <p>En este modo podríamos utilizar 16 bits por canal y obtener imágenes de 48 bits, pero con los equipos informáticos más comunes no es posible operar con ellas adecuadamente.</p>
<b>Color CMYK</b>	<ul style="list-style-type: none"> <li>· Imágenes a todo color</li> <li>· Admite cualquier formato de grabación Tiene cuatro canales de color (Cyan, Magenta, amarillo y Negro)</li> </ul>	<p>Convertir a este modo cuando se envía a una impresora de color especial, o a separar los colores para la filmación o imprenta (fotolitos).</p>
<b>Color Lab</b>	<ul style="list-style-type: none"> <li>· Con este modo podemos cambiar la luminosidad de una imagen, sin alterar los valores de tono y saturación del color.</li> <li>· Tiene tres canales: Luminosidad: canal A (verde-rojo); canal B (azul-amarillo)</li> </ul>	<p>Es un modo adecuado de transferir imágenes de unos sistemas a otros, pues los valores cromáticos se mantienen independientes del dispositivo de salida de la imagen.</p> <p>Sólo las impresoras PostScript de nivel 2 puede reproducir esta imágenes. Para impresiones normales, se recomienda pasar las imágenes a RGB, o CMYK</p> <p>. En ocasiones se es conveniente este modo para exportar archivos a otros sistemas operativos.</p>



<b>Duotono</b>	<ul style="list-style-type: none"> <li>· Imágenes en escala de grises a las que se le pueden añadir tintas planas (3 para cada imagen, más el negro), con el fin de colorear distintas gamas de grises</li> <li>· Sólo poseen un canal de color (Duotono, Trítono o Cuatritono, dependiendo del número de tintas)</li> </ul>	<p>Con este método se puede obtener fotos en blanco y negro viradas al color que queramos.</p> <p>Método empleado en impresión en el cual se utilizan dos o más planchas para añadir riqueza y profundidad tonal a una imagen de escala de grises.</p>
<b>Multicanal</b>	<ul style="list-style-type: none"> <li>· Compuesta por múltiples canales de 256 niveles de grises</li> <li>· Al seleccionar este modo, la imagen que se tenga en la ventana activa se descompone en tantos canales alfa como canales de color tuviera en origen. (Así, una imagen RGB quedará descompuesta en 3 canales y una CMYK en 4 canales)</li> </ul>	<p>Se utiliza en determinadas situaciones de impresión en escala de grises.</p> <p>También para ensamblar canales individuales de diversas imágenes antes de convertir la nueva imagen a un modo de color; los canales de color de tinta plana se conservan si se convierte un imagen a modo multicanal. si se convierte una imagen a modo color rgb a modo multicanal, los canales rgb se convertirán en magenta, cian y amarillo.</p>

### 4.1.1 Modelo RGB<sup>10</sup>.

Los ojos perciben el color a través de la simulación de tres pigmentos visuales en los conos de la retina. Estos tienen una sensibilidad pico en longitudes de onda de cerca de 630nm (rojo), 530nm (verde), 450nm (azul).

Esta teoría de la visión de tres estímulos es la base para desplegar salida de color en un monitor ya que los ordenadores trabajan con tres colores básicos, a partir de los cuales construyen todos los demás, mediante un proceso de mezcla por unidades de pantalla, estos son el rojo, el azul y el verde, y el sistema así definido se conoce como sistema RGB (Red, Green, Blue).

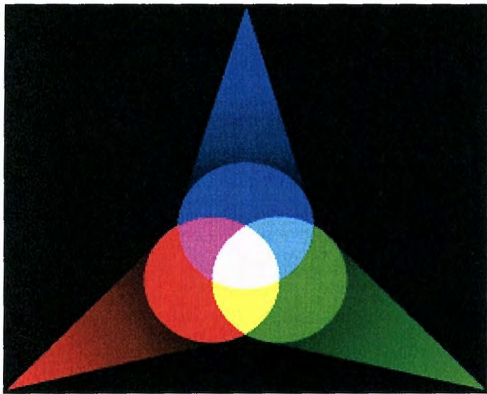


Figura No. 8 Sistema RGB, formando colores rojo, verde, azul.

Por lo que los mismos valores RGB pueden mostrar colores notablemente diferentes en varios dispositivos que empleen este modelo de color. Aunque utilicen un mismo modelo de color, sus espacios de color pueden variar considerablemente.

El modo RGB asigna un valor de intensidad a cada píxel que oscile entre 0 (negro) y 255 (blanco) para cada uno de los componentes, de una imagen en color. Por ejemplo, un color rojo brillante podría tener un valor R de 246, un valor G de 20 y un valor B de 50. El rojo más brillante que se puede conseguir es el R: 255, G: 0, B: 0. Cuando los valores de los tres componentes son idénticos, se obtiene un matiz de gris. Si el valor de todos los componentes es de 255, el

<sup>10</sup> Modelo RGB Hearn Donal. Gráficas por Computadora. Editorial Prentice Hall Capítulo 14 Modelos de Sombreado y Color. Páginas 322-323

7resultado será blanco puro y será negro puro si todos los componentes tienen un valor 0.

Así, por mezcla directa de los colores primarios se obtiene los colores secundarios, cian, magenta y amarillo, y por mezcla directa de estos los colores terciarios.

Al continuar mezclando colores vecinos, se tendrá nuevos colores, consiguiendo una representación de estos muy importantes en diseño, denominada círculo cromático, representativa de la descomposición en colores de la luz solar, que nos va a ayudar a clasificar éstos y a obtener sus combinaciones idóneas.

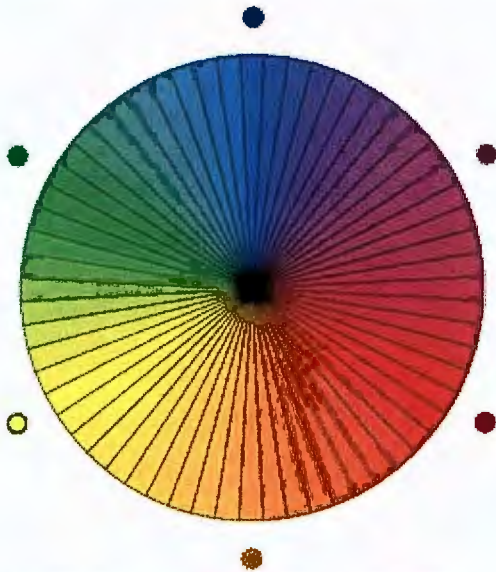


Figura No. 9: Círculo Cromático.

El círculo cromático permite establecer diferentes clasificaciones de los colores, entre las que destacan:

- Cálidos y fríos.
- Claros y oscuros.
- Apagados o sucios y colores pastel.
- Neutros.

Cuando solo se dispone de algunas alternativas de color en un sistema RGB

las opciones pueden ampliarse utilizando métodos de medio tono<sup>11</sup>.

Para obtener una gama inmensa de colores a utilizar.

<sup>11</sup> Métodos de medio tono: Es cuando un dispositivo de salidas graficas es capaz de desplegar solo dos niveles de intensidad por píxel y se usa para suministrar las variaciones de intensidad en la escena. Hearn Donal. Gráficas por Computadora. Editorial Prentice Hall Capítulo 14 páginas 308.

## **4.2 Iluminación.**

### **4.2.1 Modelos básicos de iluminación<sup>12</sup>.**

Para entender en que consisten los modelos de iluminación en si se debe conocer que es una escena, como esta compuesta, como la afecta la luz y los problemas que genera. La iluminación es una de las disciplinas más difíciles de la infografía<sup>13</sup>, pues en el mundo real la luz tiene un comportamiento complejo que no resulta fácil imitar en un ordenador. La principal dificultad deriva del hecho que la luz no sólo procede de un determinado punto y llega a un objeto en una dirección, iluminándolo desde un cierto ángulo, sino que además rebota, iluminando otros puntos que, en principio, parecería que no deberían verse afectados por ella.

Cuando se habla de iluminación en Informática Gráfica, se refiere al proceso que permite averiguar la intensidad luminosa (color) de cada uno de los puntos de una superficie y que se basa en la posición, orientación y características de la superficie y en las características de las fuentes de luz que las iluminan.

Para obtener esta intensidad luminosa se utilizan los denominados modelos de iluminación. El modelado de los colores y efectos de iluminación que se observa en los objetos es un proceso complicado que implica tanto principios físicos como psicológicos. A menudo, los modelos de iluminación derivan de las leyes físicas que describen las intensidades de luz de las superficies.

Por lo general existen 4 clases importantes de luces estas son:

---

<sup>12</sup> Eberly David H. 3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics Editorial Morgan Kaufmann. Capítulo 13 Special Effects páginas 430.

<sup>13</sup> Infografía :es la creación de imágenes que tratan de imitar el mundo tridimensional mediante el cálculo del comportamiento de la luz, los volúmenes, la atmósfera, las sombras, las texturas, la cámara y el movimiento.



## Luz ambiente:

Las luces se diseñan y ubican para proveer iluminación uniforme en una escena. Esta se logra mediante fuentes grandes con difusores cuyo propósito es esparcir la luz en todas las direcciones. la luz no sólo procede de un determinado punto y llega a un objeto en una dirección, iluminándolo desde un cierto ángulo, sino que además rebota. Se puede crear una simulación precisa de tal la misma, modelando todas las fuentes distribuidas, y luego integrando la iluminación de estas fuentes en cada punto de una superficie reflectora. Hacer tal modelo y generar la escena sería un tarea excelente para un sistema gráfico, especialmente si se desea ejecución en tiempo real. De manera alternativa, se puede ver el efecto deseado de las fuentes: lograr un nivel de luz uniforme en la escena. La luz ambiente se caracteriza por una intensidad (**la**), que es idéntica en cada punto de la escena.

La fuente de ambiente tiene tres componentes:

$$(ec1) \quad \mathbf{la} = \begin{pmatrix} la_r \\ la_g \\ la_b \end{pmatrix} \quad \text{Aunque cada punto en la escena recibe la misma iluminación ambiente de } \mathbf{la}, \text{ cada superficie la refleja de manera diferente.}$$

$$(ec2) \quad la(P) = la$$

## Fuente de punto<sup>14</sup>:

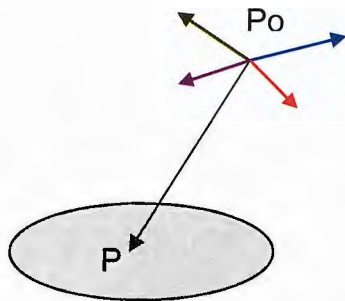
Una fuente de luz o punto ideal emite luz de manera igual en todas las direcciones. Se caracteriza una fuente ideal ubicada en un punto  $p0$  por un vector color de tres componentes:

$$(ec3) \quad \mathbf{l}(P0) = \begin{pmatrix} l_r(P0) \\ l_g(P0) \\ l_b(P0) \end{pmatrix}$$

---

<sup>14</sup> Pocock Lynn, Rosebush Judson The Computer Animator's Technical Handbook. Capítulo 3 The Science of Moving Picture. Páginas 64

La intensidad de iluminación recibida de una fuente de punto es proporcional al inverso del cuadrado de la distancia entre la fuente y la superficie, por lo tanto, en un punto  $p$  (como se ve en la figura No.11) ésta será dada por:



$$(ec4) \quad I_p(P, P_o) = \frac{1}{|P - P_o|^2} I(P_o)$$

El uso de fuentes de punto en la mayoría de las aplicaciones se determina más por su facilidad de uso que por su similitud a la realidad física. Escenas generadas solo con fuentes de punto tienden a tener un alto contraste; los objetos se ven brillantes y oscuros.

Figura No.11: La intensidad de la luz recibida de la fuente de punto en representación vectorial.

### Spot o foco<sup>15</sup>:

Las típicas luces de los teatros o espectáculos, están dirigidas en una dirección concreta se controla la mayor o menor apertura del cono de luz.

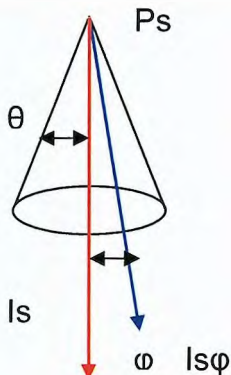


Figura No. 12: Spot realista, se caracterizan por una distribución de luz dentro del cono. Si  $\theta = 180$ , el spot se vuelve una fuente de punto.

Spot se caracterizan por un rango delgado de ángulos por los cuales se emite luz, se puede construir un spot sencillo de una fuente de punto limitando los ángulos de donde la luz de la fuente se puede ver, utiliza un cono cuyo ápice está en  $ps$ , apuntando en la dirección  $Is$ , y cuyo ancho está determinado por el ángulo  $\theta$ .

El calculo según la figura No. 12 Requiere

<sup>15</sup>Shalini Govil-Pai Principles of Computer Graphics: Theory and Practice Using OpenGL and Maya. Capítulo 6: Rendering, Shading and Lighting. Páginas 135

tres multiplicaciones y dos sumas:

$$(ec5) \quad I_s = (P, P_o) = \frac{\cos^e \varphi}{a + b |P - P_o| + c |P - P_o|^2} I(P_o)$$

## Fuentes de Luz Distantes<sup>16</sup>

Llamadas así porque, aunque las situemos a muy poca distancia de la escena los rayos que emiten son paralelos, como prácticamente lo son los del Sol cuando llegan a la Tierra. La mayoría de los cálculos de sombreado requieren la dirección de un punto sobre la superficie a la fuente de luz.

Según se mueve a lo largo de la superficie, todos los objetos cercanos entre si tienen el mismo ángulo.

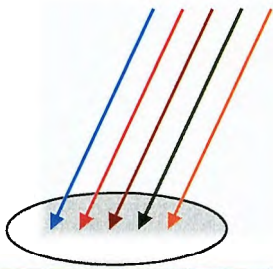


Figura No. 13 ilustra que efectivamente se reemplaza una fuente de luz de punto con una fuente que ilumina objetos con rayos de luz paralelos.

Los cálculos para fuentes distantes son similares a los cálculos para proyecciones paralelas; se reemplaza la ubicación de la fuente por una dirección de la fuente de luz. Por lo tanto, en coordenadas homogéneas, una fuente de luz de punto en  $p_o$  se representará internamente como una matriz columna de cuatro dimensiones.

Donde la luz distante se representa mediante:

$$(ec6) \quad P_o = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

<sup>16</sup> Shalini Govil-Pai Principles of Computer Graphics: Theory and Practice Using OpenGL and Maya. Capítulo 6: Rendering, Shading and Lighting. Páginas 136

La intensidad completa exclusivamente for efectos de iluminación es la siguiente:

$$(ec7) \quad I_s = (P, P_o) = \frac{1}{I_a(p)} ( I(P, P_o) + \cos^e \phi I_s(P, P_o) ) + I_d(p) +$$

### Procesos fundamentales de fuentes de luz<sup>17</sup>.

La luz puede dejar una superficie mediante dos procesos fundamentales:

1. Emisión propia.
2. Reflexión.

#### 1. Emisión propia:

La intensidad de la luz ambiente  $L_a$  es la misma sobre cada punto de la superficie. Parte de esta es absorbida y parte es reflejada. La cantidad reflejada está dada por el coeficiente de reflexión de ambiente  $k_a$ ,  $R_a = k_a$ . Como sólo se refleja una fracción positiva de luz, se debe tener

$$0 \leq K_a \leq 1 \quad \text{donde:} \quad I_a = K_a L_a$$

$L_a$  puede ser cualquiera de las fuentes de luz individuales, o puede el término ambiente global.

#### 2. Reflexión:

Ocurre solo en la dirección especular y depende de la posición relativa de la fuente de luz y el punto de vista. Existen diferentes tipos de reflexión estos son:

---

<sup>17</sup> David F Rogers, Rae A Earnshaw. Computer Graphics Techniques: Theory and Practice. Capítulo 2. Color in Computer Graphics. Editorial Springer. Páginas 50-69

- Reflexión difusa.

Un reflector difuso perfecto esparce la luz que refleja de manera igual en todas las direcciones, viéndose igual para todos los observadores. Sin embargo, la cantidad de luz reflejada depende del material, dado que parte de la luz es absorbida, y de la posición de la fuente de luz relativa a la superficie.

Los rayos de luz que pegan en la superficie con ángulos levemente diferentes se reflejarían en ángulos marcadamente diferentes. Superficies difusas perfectas son tan rugosas que no hay un ángulo preferido de reflexión.

Tales superficies son a veces conocidas como **superficies Lampercianas**, pudiéndose modelar matemáticamente por la ley de **Lambert**.

Ley de Lamber dice: “La componente difusa de la luz reflejada por una superficie es proporcional al coseno del ángulo de incidencia, la cantidad de luz reflejada depende del material, dado que parte de la luz es absorbida, y de la posición de la fuente de luz relativa a la superficie”.

Se puede caracterizar reflexiones difusas matemáticamente. La ley de Lambert dice que:

$$(ec8) \quad R_d \propto \cos \theta$$

Donde  $\theta$  es el ángulo entre la normal  $\mathbf{n}$  en el punto de interés y la dirección de la fuente de luz  $\mathbf{l}$ . Si  $\mathbf{l}$  y  $\mathbf{n}$  son ambos vectores unidad, entonces

$$(ec9) \quad \cos \theta = \mathbf{l} \cdot \mathbf{n}$$

Si se agrega un coeficiente de reflexión  $k_d$  que representa la fracción de luz difusa entrante que es reflejada, se tiene el siguiente término de reflexión

$$(ec10) \quad I_d = k_d (\mathbf{l} \cdot \mathbf{n}) L_d \quad 0 \leq k_d \leq 1$$

- Reflexión especular.

Si se emplea solo reflexión ambiente y difusa, las imágenes serán sombreadas y aparecerán tridimensionales, pero todas las superficies se verán sin vida. Lo que hace falta son la reflexión de secciones más brillantes en los objetos. Esto ocasiona un color diferente del color del ambiente reflejado y luz difusa. Una esfera roja, bajo luz blanca, tendrá un resplandecer blanco que es la reflexión de parte de la luz de la fuente en la dirección del observador.

Una superficie difusa es rugosa, una superficie especular es suave. Mientras mas lisa se la superficie, mas se parece a un espejo.

Según la superficie se hace mas lisa, la luz reflejada se concentra en un rango mas pequeño de ángulos, centrado alrededor del ángulo de un reflector perfecto para modelar superficies especulares realistas puede ser complejo, ya que el patrón por el cual se esparce no es simétrico, dependiendo de el largo de onda de la luz incidente y cambia con el ángulo de reflexión.

Existe un modelo aproximado donde se considera la superficie como rugosa para el término difuso u lisa para el término especular. La cantidad de luz que el observador ve depende del ángulo  $\phi$  entre  $\mathbf{r}$ , la dirección de un reflector perfecto, y  $\mathbf{v}$ , la dirección del observador.

(ec11) 
$$I_s = K_s L_s \cos \phi \text{ donde } 0 \leq K_s \leq 1$$

El coeficiente  $K_s$ ,  $0 \leq K_s \leq 1$  es la fracción reflejada de la luz especular entrante.

- Modelo de Reflexión de Phong

Usa cuatro vectores para calcular el color para un punto arbitrario  $p$  sobre la superficie, el vector  $n$  es la normal en  $p$ ,  $v$  tiene dirección de  $p$  al observador, además,  $l$  tiene dirección de una línea de  $p$  a un punto arbitrario sobre la superficie para una fuente de luz distribuida, o una fuente de luz,  $r$  tiene la dirección de un rayo perfectamente reflejado de  $l$ . La dirección de  $r$  seta determinada por  $n$  e  $l$ .

Este modelo apoya los tres tipos de interacciones material-luz: ambiente, difusa y especular. Si se tiene un conjunto de fuentes puntos, con componentes independientes para cada uno de los tres colores primarios para cada uno de los tres tipos de interacciones material-luz; entonces, se puede describir la matriz de

iluminación para una fuente de luz  $i$  para cada punto  $p$  sobre una superficie, mediante:

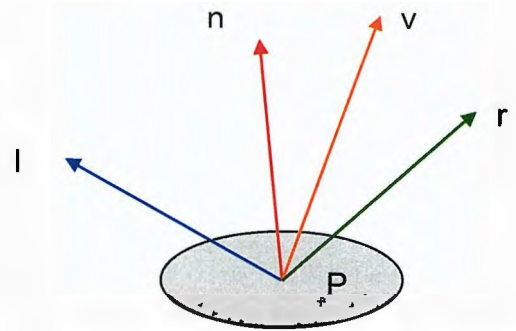


Figura No.14: El modelo usa cuatro vectores para calcular el color para un punto arbitrario  $p$  sobre la superficie.

$$(ec12) \quad L_i = \begin{pmatrix} L_{ira} & L_{iga} & L_{iba} \\ L_{ird} & L_{iga} & L_{ibd} \\ L_{irs} & L_{igs} & L_{ibs} \end{pmatrix}$$

La primera fila contiene las intensidades ambiente para rojo, verde y azul para la fuente  $i$ , la segunda contiene los términos difusos, la tercera fila contiene los términos especulares.

Para cada punto, se puede realizar una matriz de términos de reflexión:

$$(ec13) \quad R_i = \begin{pmatrix} R_{ira} & R_{iga} & R_{iba} \\ R_{ird} & R_{iga} & R_{ibd} \\ R_{irs} & R_{igs} & R_{ibd} \end{pmatrix}$$

Se obtiene la intensidad total sumando todas las contribuciones de todas las fuentes y del ambiente global. Por lo tanto, el término rojo sería:

$$(ec14) \quad I_r = \sum_i (I_{ira} + I_{ird} + I_{irs}) + I_{ar}$$

Donde  $I_{ar}$  es el componente rojo de la luz ambiente global, es así como se debe cuidar el efecto de las luces fijándose si se esta en un interior o un exterior, condiciones climatológicas que se pretende simular o si es de día o de noche y consecuentemente seleccionar en cada caso el tipo, atenuación, color o intensidad de la iluminación idóneos para ese entorno.

Al mismo tiempo se debe asegurar que el efecto de las luces de un modelo genere una sensación de realismo y para ello son imprescindibles las sombras, los reflejos, los destellos, brillos, nieblas o la atenuación con la distancia.

Todos estos elementos permiten crear la atmósfera o sensación que necesite el diseño de un escenario.

La iluminación puede ser de dos tipos<sup>18</sup>:

3. Modelos de iluminación local.
4. Modelos de iluminación global.

---

<sup>18</sup> Glassner Andrew S Principles of Digital Image Synthesis. Editorial MK. Capítulo 6 The Human Visual System and Color. Páginas 59-67



## Modelos de iluminación local

Son métodos simplificados que se basan en las propiedades ópticas de las superficies, las condiciones de la luz ambiente o de fondo, y las características de las fuentes de luz. Las propiedades ópticas de las superficies se especifican mediante parámetros que permiten controlar la cantidad de absorción y reflexión de la luz incidente.

Las fuentes de luz, se consideran como fuentes de luz puntuales que divergen en forma radial desde la posición de la fuente. Este tipo de modelos no tiene en cuenta el inter reflexiones de luz entre los objetos de la escena. En este tipo de algoritmos la iluminación global se modela mediante un término de iluminación ambiente que se considera constante para todos los puntos de los objetos.

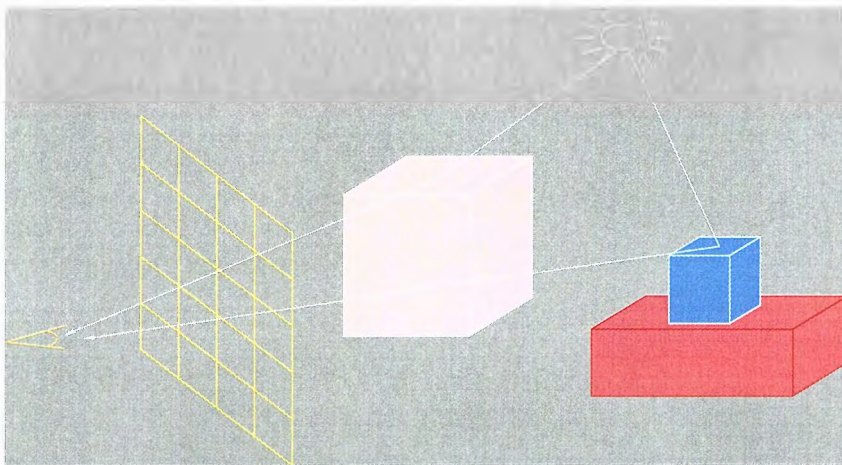


Figura No. 15 Modelo de iluminación local.

## Modelos de iluminación global

Calculan la iluminación o color en punto dependiendo de la luz directamente emitida por las fuentes y dependiendo también de la luz que alcanza el punto después de la reflexión y transmisión a través de otras superficies. Tradicionalmente se han utilizado dos tipos de algoritmos para generar imágenes teniendo en cuenta la contribución de la iluminación global por ejemplo Ray-Tracing y Radiosidad.

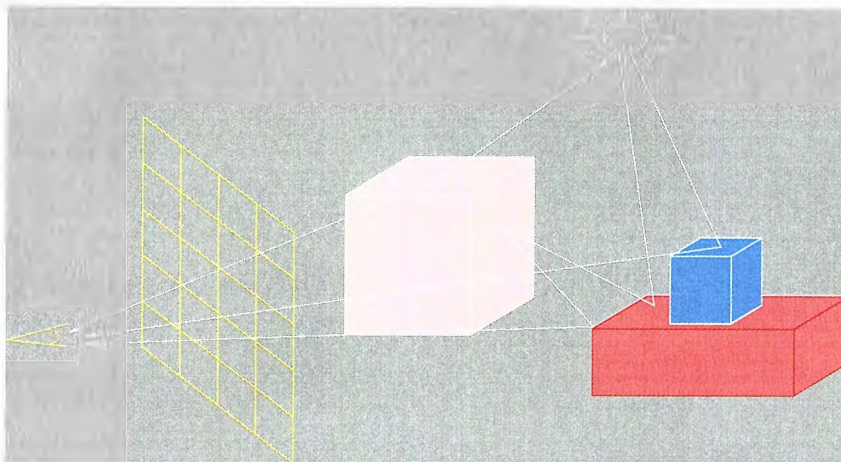


Figura No. 16 Modelo de iluminación global.

### Ray- Tracing o trazado de rayos.

En muchas formas, ray tracing es una extensión al enfoque de rendering con un modelo de iluminación local. Está basado en la observación previa que, de los rayos de luz saliendo de una fuente, los únicos que contribuyen a la imagen son aquellos que entran el lente de la cámara sintética y pasan por el centro de proyección.

Este es de los modelos de iluminación global , que intercala la determinación de superficies visibles con el cálculo de sombras, reflexiones y refracciones. El raytracing es un motor de render utilizado para obtener imágenes en 3D generadas por ordenador.

El trazado de rayos se establece un sistema de coordenadas con las posiciones de píxel designadas en el plano xy, a partir del centro de la proyección, se determina entonces la trayectoria de un rayo que pasa a través del centro de cada posición de píxel en la pantalla. Los efectos de iluminación que se acumulan a lo largo de la trayectoria de este rayo se asignan luego al píxel intenta modelar como vería el mundo el observador, siguiendo cada rayo de luz que emite cada fuente de luz y ver si va a parar al ojo del observador.

Así que lo que se hace es lo contrario: se lanzan rayos desde el observador hacia el mundo y en cierto momento calcula la luz que incide en ese rayo.

Si el objeto fuera de un material semitransparente, entonces además de considerar las reflexiones, habría que considerar la iluminación que atraviesa el objeto en la dirección del rayo de refracción.

Un caso particular de raytracing es el raycasting. El raycasting es idéntico al raytracing pero con la única diferencia que no modela los reflejos, o sea, que modela las escenas sin reflejos, esto lo hace más sencillo que el raytracing.

Los rayos que pegan en una superficie, inicialmente suponiendo que todos los objetos son opacos, requieren calcular un sombreado para el punto de intersección. En el trazo de rayos, en lugar de aplicar inmediatamente el modelo de reflexión, primero se revisa si el punto de intersección entre el rayo irradiado y la superficie está iluminado.

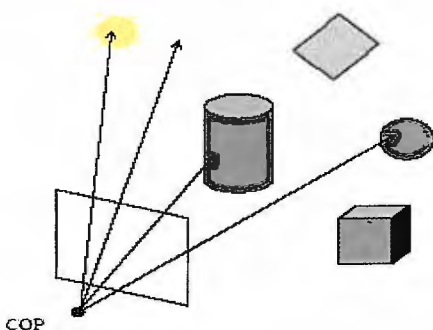
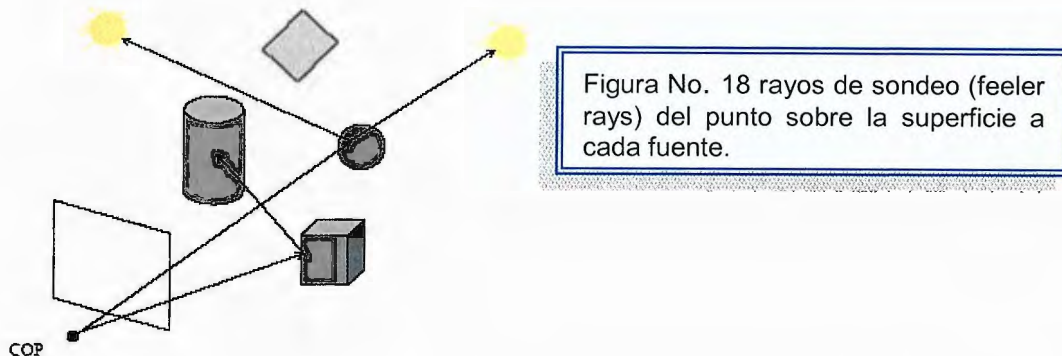


Figura No. 17 **Rayos Irradiados** se consideran solo aquellos que comienzan en el centro de proyección,

En el trazo de rayos, en lugar de aplicar inmediatamente el modelo de reflexión, primero se revisa si el punto de intersección entre el rayo irradiado y la superficie está iluminado. Se calcula los **rayos de sombra (shadow rays)** o **rayos de sondeo**



Si un rayo de sombra intercepta una superficie antes de llegar a la fuente, la luz está bloqueada de llegar al punto bajo consideración y este punto está en sombra, por lo menos de esa fuente. No se tienen que hacer cálculos de intersección para fuentes que estén bloqueadas de un punto sobre una superficie. Si todas las superficies son opacas y no se considera la luz esparcida entre superficies, se tiene una imagen que tiene sombras agregadas a lo ya hecho sin ray tracing.

Ray tracing es particularmente bueno para manejar superficies que sean reflectantes y transmitivas. Usando el modelo básico, se sigue una rayo irradiado a una superficie con la propiedad que, si el rayo de una fuente pega en un punto, entonces la luz de la fuente es parcialmente absorbida, y parte de esta luz contribuye al término de reflexión difusa. El resto de la luz entrante se divide entre un rayo transmitido y un rayo reflectado se requiere llevar a cabo tres tareas:



1. Se procesa la contribución de una fuente de luz en el punto, usando el modelo de reflexión estándar.

2. Se debe irradiar un rayo en la dirección de una reflexión perfecta.

3. Se debe irradiar un rayo en la dirección del rayo transmitido.

Estos dos rayos son tratados como el rayo irradiado original, o sea, se interceptan, si es posible, con otras superficies, terminan en una fuente, o se van al infinito. En cada superficie que estos rayos interceptan, se pueden generar rayos adicionales mediante reflexión o transmisión de la luz.

## **Radiosidad**

El otro algoritmo utilizado en iluminación global es el de radiosidad que separa el cálculo de la iluminación de la determinación de superficies visibles, es ideal para escenas consistiendo solamente de superficies difusas perfectas

Este tipo de algoritmo modela todas las interacciones del entorno con las fuentes de luz independientemente de la vista, y posteriormente, se calcula una o más imágenes desde los puntos de vista deseados utilizando métodos convencionales de obtención de superficies visibles y de sombreado.

El método de radiosidad básico parte la escena en pequeños polígonos planos, o patches, cada una de las cuales se asume es perfectamente difusa y se sombrea con una sombra constante. Lo necesario es encontrar estas sombras.

Existen dos pasos para el método:

1. Se consideran los patches en pares para determinar los factores de forma (form factors) que describen como la energía de luz que deja un patch afecta la otra. Una vez determinados los factores de forma, la ecuación de rendering, que comienza como una ecuación integral, se puede reducir a un conjunto de ecuaciones lineales para radiosidades, esencialmente la reflectividad de las facetas.

2. Una vez resueltas estas ecuaciones, se puede sombrear la escena usando sombreado constante. Aunque la cantidad de cálculos requeridos para computar factores de forma es enorme, es un problema  $O(n^2)$  para  $n$  patches, una vez se determina las radiosidades de cada patch, estos son independientes de la ubicación del observador, dada la suposición que todas las superficies son perfectamente difusas. Por lo tanto se puede sombrear la imagen rápidamente como cualquier escena que use el modelo de iluminación local.

La radiosidad tiene algunos principios que se deben tomar en cuenta estos son:

1. Luz magnitud escalar (Cierta cantidad de energía en cada punto)
2. Todos los objetos fuentes potenciales de luz
3. Los objetos se dividen en trozos o patches
4. Iluminación escena independiente punto de vista

Fuentes de luz como objeto más de la escena (formas variadas)

Las principales características de este método son:

- La versión clásica supone que todos los objetos son difusores perfectos, es decir, que envían su energía reflejada en todas las direcciones por igual. Resulta muy costoso de representar en trazado de rayos, ya que el número de rayos a lanzar en cada intersección aumenta enormemente.
- Otra característica importante es que todos los objetos se deben especificar por superficies descompuestas en trozos o *patches*. Estos trozos no tienen porqué ser necesariamente poligonales pero su tamaño debe ser lo bastante pequeño como para que el cálculo de la radiosidad sea una buena aproximación a la realidad. El método de iluminación por radiosidad intentará calcular la densidad de energía que va a emitir cada uno de estos trozos ( $B_i$ ). Normalmente esta energía se separa en tres componentes RGB para poder asignar un color a cada trozo.

- Una vez calculada la radiosidad de cada trozo, existen diferentes posibilidades para efectuar la visualización final. En este sentido, se debe recordar que el método de radiosidad es solamente un método de iluminación, y no presupone (como hace el trazado de rayos, que calcula el color de pixels) cual es el mecanismo para efectuar la representación gráfica del resultado. Una opción para realizar la visualización es efectuar un trazado de rayos para averiguar la posición de los objetos en la ventana y utilizar el valor de radiosidad para colorearlos. Otra posibilidad es descomponer los trozos en polígonos y utilizar los métodos de tiempo real que vimos en el Tema 3, asignando colores a los vértices según la radiosidad.
- Al considerar solamente las componentes difusas (uniformes) de las funciones de reflectancia de los materiales de la escena, entonces la luz recibida y transmitida por los objetos no depende de la posición del observador en la escena (al contrario de lo que sucede con las distribuciones de tipo especular). Por tanto, toda la iluminación de la escena es independiente de dónde se coloque el observador. Esto va a suponer una gran ventaja ya que, aunque el proceso de calcular la iluminación por radiosidad es muy costoso, sólo será necesario realizarlo una vez, siempre que no cambie la posición de los objetos, sus características materiales o las fuentes de luz de la escena.
- Por otra parte, el hecho de considerar distribuciones de reflexión totalmente uniformes constituye una importante limitación de este método. Al no considerar la componente especular no se podrá calcular la imagen reflejada de un objeto sobre otro, ni siquiera de forma aproximada. El método de radiosidad en su forma pura resulta ser, por tanto, el reverso del trazado de rayos (cuyas versiones más simples aproximan la distribución reflejada mediante uno o pocos rayos con comportamiento especular). Mientras que con el trazado de rayos no resulta factible reproducir los

efectos de la distribución real de energía en un entorno difusor (por esa razón las escenas representadas con un trazador de rayos sencillo carecen de profundidad y buenos efectos de iluminación), con la radiosidad se reproducen bien los efectos de interacción de color y sombras entre objetos, pero no la formación de imágenes reflejadas.

- Una de las ventajas que aporta el método de radiosidad es que las fuentes de luz se definen utilizando trozos extensos de superficie y se consideran a todos los efectos como un objeto más de la escena. Por tanto, pueden modelarse con la forma que se desee y resultan visibles como cualquier otro objeto. Se debe tener en cuenta que en el trazado de rayos las fuentes son puntuales y deben definirse de forma separada a los objetos de la escena. En la radiosidad los objetos que actúan como fuentes de luz solamente se diferencian en que emiten energía luminosa por sí mismos, es decir, que su componente de emisión será diferente de cero. De este modo, fuentes extensas como los tubos fluorescentes pueden simularse de manera adecuada.

La radiosidad se define como la cantidad de energía que se transmite por unidad de tiempo y de superficie.

(ec15)

$$B = E / t.s$$

La ecuación global de la radiosidad se analiza de la siguiente forma:

Una escena formada por objetos cuya superficie ha sido descompuesta en trozos y se quiere calcular la radiosidad que emite cada uno de esos trozos como resultado del intercambio de energía radiante que se ha dado entre ellos hasta llegar a un estado de equilibrio.



Como no se toma en cuenta la naturaleza ondulatoria de la luz, no considera la posibilidad de interferencias, y la energía que procede de diferentes sitios y llega a un mismo punto se suma directamente

$$(ec16) \quad B_i = E_i + \rho_i \cdot \sum_j F_{ij} B_j$$

Donde:

$E_i$ : energía debida a su propia emisión.

$\rho_i$ : coeficiente de reflexión difusa.

$F_{ji}$ : es el factor de forma o proporción de la energía emitida por un elemento  $j$  llega al elemento  $i$

## 4.2.2 Algoritmos de Iluminación Global<sup>19</sup>

### Algoritmos de trazado de rayos.

El trazado de rayos clásico cumple dos funciones diferentes. Por una parte incluye el cálculo geométrico de la imagen, la proyección de la escena tridimensional sobre la ventana de visualización, resolviendo el problema que en el procesamiento de gráficos en tiempo real se abordaba mediante la proyección de los vértices de los polígonos. En este sentido, el trazado de rayos puede considerarse como un método de visualización diferenciado.

La segunda función del trazado de rayos consiste en efectuar la evaluación de la intensidad luminosa (color) transportada por cada rayo en función de su recorrido por la escena. En este sentido el trazado de rayos implica la existencia de un modelo de iluminación global en el que, aparecen progresivamente otros rayos en un proceso de evaluación recursiva.

Algoritmo del proceso de trazado de rayos.

1. Generar un determinado número de rayos, para cada pixel de la imagen.
2. Para cada uno de estos rayos: Evaluar el color del rayo, calculando las intersecciones con los objetos y generando nuevos rayos.
3. Calcular el color del píxel, para determinar la intensidad que el rayo transportará cuando llegue al punto de observación procedente de la escena se sigue un proceso recursivo, resultando un algoritmo como el siguiente:

---

<sup>19</sup> Lengyel Eric. Mathematics for 3D Game Programming and Computer Graphics. Editorial Charles Rivera Media. Capítulo 6. Illuminations. Páginas 163-167.

### Algoritmo para evaluar (rayo)

1. hallar intersección (más cercana)
2. evaluar variación de la intensidad durante la transmisión por el medio
3. generar rayos hijos para cada hijo
4. evaluar (rayo.hijo)

En un modelo sencillo los rayos hijos creados a partir de la intersección con una superficie serán solamente un rayo reflejado y uno transmitido.

El resultado de este proceso recursivo es la creación de un *árbol de rayos*, en el que cada uno depende de las intensidades asignadas a sus hijos.

Una forma de asignar color a los rayos sería fijarse solamente en los rayos terminales:

- Cuando el rayo acabe en una fuente de luz se calcula la intensidad producida por ésta.
- Si no ocurre así, se puede asignar una intensidad de fondo IFONDO.

Otra opción mucho más eficiente para la asignación de color consiste en añadir un término de intensidad a los rayos intermedios que no depende de sus rayos hijos. Para calcular esta contribución se lanza un tercer rayo desde el punto de intersección hasta la fuente de luz (rayo de sombra). Si el rayo de sombra no intercepta a ningún otro objeto antes de llegar a la fuente de luz, entonces ésta es visible desde el punto de intersección y se puede calcular la intensidad aportada al rayo utilizando un modelo del sombreado

De este modo tendrá el algoritmo:

### Algoritmo de interacción de superficie para evaluar (rayo)

1. si no hemos alcanzado la profundidad máxima del árbol crear rayos hijos  $I_r$ ,  $I_t$  y evaluarlos.
2. calcular intensidad propia superficie lanzar rayo de sombra a cada fuente de luz.

3. si (rayo sombra no intercepta objeto) aplicar modelo iluminación par calcular I propia modelo sombreado: componente ambiente + componente difusa + componente especular.
4. devolver como color del rayo la intensidad propia combinada con  $I_r$  ,  $I_t$

### **Algoritmo de Radiosidad.<sup>20</sup>**

Para resolver el sistema de ecuaciones de la radiosidad se emplea los métodos clásicos de resolución de sistemas de ecuaciones lineales, hallando directamente los valores de las incógnitas  $B_i$ . Este sería el método directo de resolución de la radiosidad. Este método requiere una gran capacidad de almacenamiento para guardar todos los valores de la matriz (una escena puede tener miles de trozos, y por tanto se requeriría calcular y almacenar millones de factores de forma), y los cálculos también resultan muy costosos.

Es por ello por lo que se han desarrollado otros métodos de resolución conocidos como métodos progresivos. Estos métodos no consideran la ecuación de radiosidad como la representación del estado final de equilibrio energético, sino que tratan de ver cómo se desarrolla a lo largo del tiempo la interacción de energía entre los trozos, reproduciendo de esta forma el fenómeno físico.

### **Algoritmo del método de radiosidad progresiva**

1. Definir las siguientes variables:

$B_i$  acumula la radiosidad total de cada trozo.

$\Delta_i$  representa la cantidad de energía que queda por emitir desde cada trozo

$i$  hacia los demás.

El algoritmo inicializa estas variables suponiendo que la energía está al principio concentrada en las fuentes de luz.

---

<sup>20</sup> Huw Jones, Computer Graphics. Through Key Mathematics. Editorial Springer Capítulo 10 Drawing and Rendering: How to Create Picture. Páginas 321-323

2. Luego entra en un bucle, en el que se escoge aquel trozo que tiene más energía por enviar, y esta energía se distribuye a los demás trozos.
3. El proceso termina cuando la cantidad de energía que queda por enviar es más pequeña que un cierto umbral, que será el error máximo que estamos cometiendo.

## **4.3. Sombras<sup>21</sup>**

### **4.3.1. Introducción a sombras.**

El concepto intuitivo de una sombra no es más que la ausencia de luz producida generalmente por la superposición de un cuerpo opaco entre una fuente de luz y otro cuerpo.

Este efecto luminoso va a añadir un gran realismo en la escena así como una mayor sensación tridimensional, aunque sobrecarga los ya complejos cálculos de coordenadas de la luz.

La forma más sencilla de generar una sombra es superponer una figura sobre otra y proyectar el primero sobre el segundo. Esta proyección en OpenGL tiene sus restricciones, ya que las proyecciones se realizan sobre volúmenes, no sobre el plano que conforma el suelo.

## **4.4. Sombras proyectadas**

La forma más sencilla para dibujar una sombra en OpenGL, es mediante sombras proyectadas. La idea consiste en calcular la proyección del modelo sobre el plano sobre el que se quiera calcular la sombra. Una vez calculada esta matriz, se le aplica la matriz del modelo obteniendo el modelo proyectado en el plano. Una característica que añade OpenGL a las sombras es la capacidad de tener color.

Hay que considerar la dirección y distancia de la fuente de luz. La dirección de la fuente determina la forma de la sombra y su tamaño. La siguiente función toma tres puntos que descansan en el plano en el que queremos que aparezca la sombra, la posición de la fuente de luz y, finalmente, un puntero a una matriz de transformación que construirá esta función. Lo que hace esta función es deducir los coeficientes de la ecuación del plano en el que aparecerá la sombra, y usarla con la posición de la luz para construir una matriz de transformación. Si

---

<sup>21</sup> Huw Jones, Computer Graphics. Through Key Mathematics. Editorial Springer Capítulo 10 Drawing and Rendering: How to Create Picture. Páginas 319-320

multiplicamos esta matriz por la del modelo actual, todo el dibujo será proyectado sobre este plano

```
//Crea una matriz de perspectiva de sombras a partir de la ecuación //
// del plano y la posición de la luz. El valor devuelto se almacena en
// destMat[][]
void MakeShadowMatrix(GLfloat points[3][3], GLfloat lightPos[4],
GLfloat destMat[4][4])
{
    GLfloat planeCoeff[4];
    GLfloat dot;
    // busca los coeficientes de la ecuación del plano
    // Busca los tres primeros coeficientes de la misma manera en
    // que buscamos una normal
    calcNormal(points,planeCoeff);
    planeCoeff[3] = - (
    (planeCoeff[0]*points[2][0]) +
    (planeCoeff[1]*points[2][1]) +
    (planeCoeff[2]*points[2][2]));
    // Producto mixto del plano y la posición de la luz
    dot = planeCoeff[0] * lightPos[0] +
    planeCoeff[1] * lightPos[1] +
    planeCoeff[2] * lightPos[2] +
    planeCoeff[3] * lightPos[3];
    // Ahora hace la proyección
    // Primera columna
    destMat[0][0] = dot - lightPos[0] * planeCoeff[0];
    destMat[1][0] = 0.0f - lightPos[0] * planeCoeff[1];
    destMat[2][0] = 0.0f - lightPos[0] * planeCoeff[2];
    destMat[3][0] = 0.0f - lightPos[0] * planeCoeff[3];
    // Segunda columna
    destMat[0][1] = 0.0f - lightPos[1] * planeCoeff[0];
    destMat[1][1] = dot - lightPos[1] * planeCoeff[1];
    destMat[2][1] = 0.0f - lightPos[1] * planeCoeff[2];
    destMat[3][1] = 0.0f - lightPos[1] * planeCoeff[3];
    // Tercera Columna
    destMat[0][2] = 0.0f - lightPos[2] * planeCoeff[0];
    destMat[1][2] = 0.0f - lightPos[2] * planeCoeff[1];
    destMat[2][2] = dot - lightPos[2] * planeCoeff[2];
    destMat[3][2] = 0.0f - lightPos[2] * planeCoeff[3];
    // Cuarta Columna
    destMat[0][3] = 0.0f - lightPos[3] * planeCoeff[0];
    destMat[1][3] = 0.0f - lightPos[3] * planeCoeff[1];
    destMat[2][3] = 0.0f - lightPos[3] * planeCoeff[2];
    destMat[3][3] = dot - lightPos[3] * planeCoeff[3];
}
```

### 4.3.2. Modelo de Sombreado Poligonal<sup>22</sup>.

La ventaja de usar modelos poligonales para los objetos, para polígonos planos, es que se puede reducir bastante el procesamiento requerido para el sombreado. La mayoría de los sistemas gráficos, incluyendo OpenGL, explota las posibles eficiencias para polígonos planos, descomponiendo superficies curvas en muchos polígonos planos pequeños, cada uno teniendo un vector normal bien definido. Se debe considerar tres formas para el sombreado de polígonos:

- Sombreado plano o constante.
- Sombreado interpolativo o Gouraud.
- Sombreado Phong.

#### 4.3.2.1. Sombreado Plano (Flat)<sup>23</sup>

Consiste en tres vectores,  $\mathbf{l}$ ,  $\mathbf{n}$  y  $\mathbf{v}$ , pueden variar según se mueve entre puntos sobre una superficie.

- Para un polígono plano,  $\mathbf{n}$  es constante.
- Si se asume un observador distante (la bandera `GL_DISTANT_VIEWER` en OpenGL),  $\mathbf{v}$  es constante sobre el polígono.
- Si la fuente de luz es distante,  $\mathbf{l}$  es constante.

Un observador distante se puede interpretar como una fuente o un observador en el infinito; en particular, si los polígonos son pequeños, las distancias relativas para el infinito no tienen que ser muy grandes. Los ajustes necesarios, como cambiar la ubicación de la fuente u observador hacia la dirección de la fuente u observador, de forma correspondiente. Si los tres vectores son constantes,

---

<sup>22</sup> Lengyel Eric, Mathematics for 3D Game Programming and Computer Graphics. Editorial: Charles Rivera Media. Capítulo 6. Illuminations, páginas 166-167

<sup>23</sup> D. Foley, James. Computer Graphics: Principles and Practice. Editorial: Addison Wesley. Capítulo 16. Illuminations and Shading páginas 745-753



entonces el cálculo de sombreado se lleva a cabo una sola vez para cada polígono, y se asignará la misma sombra a cada punto en el polígono. Esta técnica se conoce como sombreado plano o constante.

En OpenGL, se especifica el sombreado plano mediante el siguiente comando:

```
glShadeModel(GL_FLAT);
```

Si se usa sombreado plano, OpenGL usará las **normales** asociadas con el primer vértice de un polígono haciéndolo sencillo para el cálculo de sombreado. Para primitivas como un strip de triángulo, OpenGL usa la normal del tercer vértice para el primer triángulo, la normal del cuarto vértice para el segundo, y así sucesivamente. Reglas similares se aplican para otras primitivas, como cuadriláteros.

El sombreado plano mostrará diferencias de sombreado entre los polígonos, si las fuentes de luz y el punto de observación están cerca del polígono, por lo tanto los vectores  $\mathbf{v}$  y  $\mathbf{l}$  serán diferentes para cada polígono. Sin embargo, si los polígonos se diseñaron para modelar una superficie suave, sombreado plano no será el mas apropiado, ya que se verán aunque sea diferencias de sombreado pequeñas entre polígonos adyacentes, como se ve en la siguiente figura No.19

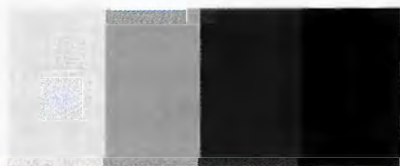


Figura No.19: Superficie de polígonos adyacentes con sombreado

El sistema visual humano tiene una sensibilidad remarcada para pequeñas diferencias de intensidad de color, causada por la propiedad de inhibición lateral. Si

se ve una secuencia incremental de

intensidades, se percibe los incrementos de brillantez reforzados en un lado del intervalo de intensidad y decremento de brillantez reforzado del otro lado, como se ve en la figura No.20.

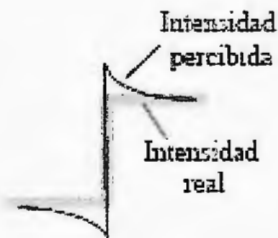


Figura No.20: Fenómeno de Bandas Mach

En la figura No.21 se ven bandas, llamadas bandas Mach, a lo largo de los bordes. Este fenómeno es una consecuencia de como se conectan los conos al nervio óptico en el ojo, y no hay nada que se pueda hacer al respecto, fuera de buscar técnicas de sombreado mas suaves que no produzcan grandes diferencias en sombreado en los bordes de los polígonos.



Figura No.21: Aplicación a una imagen 3D de sombreado plano.

#### 4.3.2.2. Sombreado Interpolativo y Gouraud.

El método de sombreado Interpolativo y Gouraud es asignado a un modelo de sombreado para que se suavice. Para utilizar este método en OpenGL se utilizara:

```
glShadeModel(GL_SMOOTH);
```

OpenGL interpolará los colores para las primitivas, como las líneas. Si se permite sombreado y luz suaves, y se asigna a cada vértice la normal del polígono siendo sombreado, se calcularía la luz en cada vértice, determinando el color del vértice, usando las propiedades materiales y los vectores  $\mathbf{v}$  y  $\mathbf{l}$  se computan para cada vértice. Si la fuente de luz es distante, y el observador es distante o no hay

reflexiones especulares, entonces el sombreado interpolativo sombreara un polígono con un color constante.

Como múltiples polígonos se juntan en vértices interiores, cada uno teniendo su propia normal, la normal en los vértices es discontinua. Aunque esta situación pudiera complicar las matemáticas, Gouraud se dio cuenta que la normal en el vértice se puede definir de manera que se obtenga sombras mas suaves mediante interpolación. Si se tiene un vértice interior donde cuatro polígonos se juntan, cada

uno con su normal, como se ve en la figura No.22.

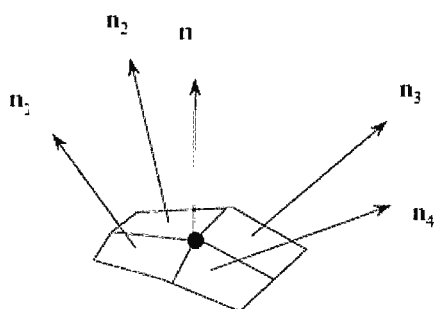


Figura No.22: Polígonos con sus normales y la intersección de estas (n).

En el sombreado Gouraud, se define la normal en un vértice como el promedio normalizado de las normales de los polígonos que comparten el vértice.

Tomando como ejemplo la figura No.22, la normal del vértice está dada por:

$$(ec17) \quad n = \frac{n_1 + n_2 + n_3 + n_4}{|n_1 + n_2 + n_3 + n_4|}$$

En OpenGL, el sombreado Gouraud es sencillo. Existen varios problemas al utilizar este método; el modelo al que se le aplica el sombreado puede que no sea coherente, podría emitir más luz de la que recibe. En coordenadas de vistas, las distancias "x" y "y" no son lineales respecto a "z".



Figura No.23: Aplicación a una imagen 3D de sombreado Gouraud.



### 4.3.2.3 Sombreado Pong

Incluso el sombreado Gouraud no puede prevenir la aparición de bandas Mach. En el método Pong se propone, en lugar de interpolar intensidades de los vértices, como es aplicado en el sombreado Gouraud, se interpole normales a lo largo del polígono. Para un polígono que comparte lados y vértices con otros polígonos, como se ve en la siguiente figura No.24.

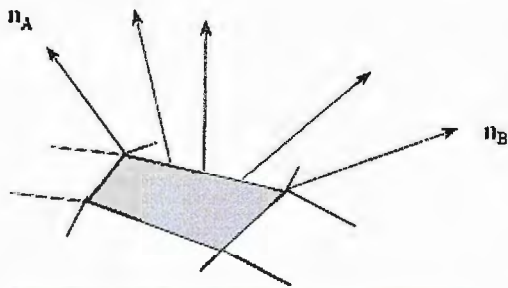


Figura No.24: Las diferentes normales de un polígono.

Se puede calcular las normales en los vértices interpolando sobre las normales de los polígonos que comparten el vértice. Luego, se puede usar interpolación bilineal, para interpolar las normales sobre el polígono, como se ve en la siguiente figura No.25.

Se puede usar las normales interpoladas en los vértices  $A$  y  $B$  para interpolar a lo largo de lado entre ellas:

$$(ec18) \quad n(\alpha) = (1 - \alpha)n_A + \alpha n_B$$

Se puede hacer una interpolación similar en todos los lados. La normal en cualquier punto interior se puede obtener de los puntos en los lados mediante

$$(ec19) \quad n(\alpha, \beta) = (1 - \beta)n_C + \beta n_D$$

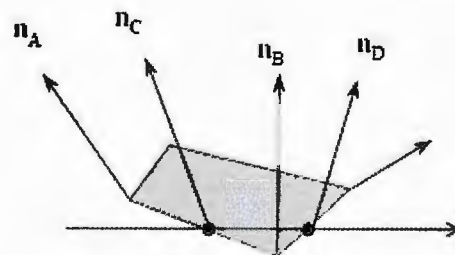


Figura No.25: Interpolación de normales sobre un polígono.

Una vez obtenidas las normales en cada punto, se puede hacer cálculos de sombreado independientes. Normalmente, este proceso se combinará con la

rasterización del polígono, para que la línea entre  $C$  y  $D$  proyecte a una línea de rastreo en el frame buffer.

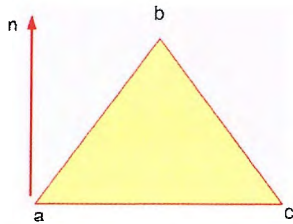
Sombreado Phong producirá imágenes mas suaves que con el sombreado Gouraud, El problema del el sombreado, ocupa bastante recursos de un sistema computacional, porque interpola vectores en lugar de escalares y calcula la intensidad de cada punto. Existen varias implementaciones en hardware para el sombreado Gouraud, pero no así mismo para sombreado Phong.



Figura No.26 Aplicación a una imagen 3D de sombreado Phong.

#### 4.3.2.4. Sombreado el modelo de la esfera<sup>24</sup>

Para sombrear las esferas aproximadas con el modelo de sombreado de OpenGL, se deben asignar normales. Una sencilla selección, es el sombreado plano para



cada triángulo, usando los tres vértices para determinar una normal, y luego asignar esta normal al primer vértice, como se ve en la siguiente figura No.27.

Figura No.27: representación de vector normal, para modelado de esferas

Siguiendo el enfoque del ejemplo, se usa el producto cruz, y luego se normaliza el resultado.

$$(ec20) \quad n = \frac{ab \times ac}{|ab \times ac|}$$

$$(ec21) \quad ab \times ac = \begin{pmatrix} x & y & z \\ b[0] - a[0] & b[1] - a[1] & b[2] - a[2] \\ c[0] - a[0] & c[1] - a[1] & c[2] - a[2] \end{pmatrix}$$

Una función de producto cruz es

```
cross(point a, point b, point c, point d)
{
d[0]=(b[1]-a[1])*(c[2]-a[2])-(b[2]-a[2])*(c[1]-a[1]);
d[1]=(b[2]-a[2])*(c[0]-a[0])-(b[0]-a[0])*(c[2]-a[2]);
d[2]=(b[0]-a[0])*(c[1]-a[1])-(b[1]-a[1])*(c[0]-a[0]);
normal(d);
}
```

<sup>24</sup> Buss Samuel R. 3D Computer Graphics: A Mathematical Introduction with OpenGL. Editorial: Cambridge University Press. Capítulo 5. Texture Mapping .Páginas 127 – 138.

Asumiendo que las fuentes de luz han sido definidas y habilitadas, se puede cambiar la rutina del triángulo para producir las esferas sombreadas:

```
void triangle( point a, point b, point c)
{
    point n;
    cross(a,b,c,n);
    glBegin(GL_LINE_LOOP);
    glNormal3fv(n);
    glVertex3fv(a);
    glVertex3fv(b);
    glVertex3fv(c);
    glEnd();
}
```

En el resultado del sombreado plano, aunque se incrementen el número de subdivisiones para que el interior de la esfera parezca suave, aún se notan los bordes de los polígonos alrededor de la esfera. Este tipo de borde se conoce como silueta (silhouette edge).

Fácilmente se puede aplicar sombreado interpolativo al modelo de esfera, ya que se conoce que la normal en cada punto  $p$  sobre la superficie está en la dirección del origen a  $p$ . Se puede asignar la normal verdadera a cada vértice, y OpenGL interpolará las sombras en estos vértices a lo largo de cada triángulo. Por lo tanto, se puede cambiar la función *triangle*, como el siguiente ejemplo.

```
void triangle( point a, point b, point c)
{
    point n;
    int j;
    glBegin(GL_POLYGON);
    for(j=0; j<3; j++) n[j]=a[j];
    normal(n);
    glVertex3fv(a);
    for(j=0; j<3; j++) n[j]=b[j];
```

```
normal(n);
glNormal3fv(n);
glVertex3fv(b);
for(j=0; j<3; j++) n[j]=c[j];
normal(n);
glNormal3fv(n);
glVertex3fv(c);
glEnd();
}
```

Aunque usando las normales verdaderas produce imágenes más realistas que sombreado plano, el ejemplo no es general, ya que se ha usado normales que se conocen analíticamente. Tampoco se ha provisto una imagen con sombreado Gouraud verdadera. Si se supone que se quiere una imagen con sombreado Gouraud de la esfera aproximada; en cada vértice se necesita conocer las normales de todos los polígonos incidentes en el vértice.

#### 4.4. Detalles de Superficie (texturas)<sup>25</sup>

Las técnicas de texturación permiten representar variaciones en los atributos de la superficie de los objetos como el color, la transparencia, etc. Utilizando estas técnicas se mejora notablemente la calidad de las imágenes generadas por ordenador, que de otra forma presentan un aspecto demasiado pulido y suave.

Las texturas pueden insertarse sobre distintos tipos de primitivas como polígonos o superficies curvas, y éstas pueden repetirse para cubrir toda la superficie del objeto. Por otro lado, pueden tener distintas dimensiones aunque lo más habitual es utilizar imágenes 2D definidas sobre una matriz o patrón de textura. Finalmente, la textura puede aplicarse de distintas formas: puede utilizarse para colorear una primitiva, para simular los reflejos del entorno, entre otros.

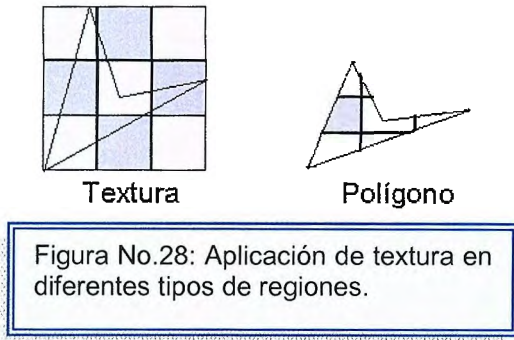
Las texturas son matrices de datos relacionados con el color, la luminosidad o la transparencia.

---

<sup>25</sup> Eberly David H. 3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics Editorial Morgan Kaufmann. Capítulo 3 The Graphics Pipeline, páginas 105-109



Cada uno de sus elementos individuales se denomina *texel*. Una de las ventajas de la aplicación de texturas, es que puede realizarse sobre regiones no rectangulares, como se muestra en la figura No.28



#### 4.4.1. Tipos de texturas.<sup>26</sup>

Toda textura es una imagen de alguna clase, por lo que puede dividirse en dos tipos:

- Textura 1D. Es una imagen con anchura pero sin altura, o viceversa; tienen un único píxel de ancho o de alto.
- Textura 2D. Es una imagen con más de un píxel de alto o ancho y se abre generalmente con un fichero .BMP. Se usan comúnmente para reemplazar complejas geometrías de superficie en edificios, árboles y demás.

Todas estas texturas se componen de valores de color RGB y pueden incluir valores alfa (de transparencia). Naturalmente, debemos definir una imagen para textura antes de que podamos dibujar polígonos texturados en OpenGL. Las texturas siguen las mismas reglas de almacenamiento que los mapas de bits.

<sup>26</sup> Hearn Donald, Gráficas por Computadora. Editorial Prentice Hall, Capítulo 14 Modelos de Sombreado y Color. Páginas 311-316

### 4.4.1.1. Textura 1D.

La librería de gráficos 3D, OpenGL proporciona una función para definir texturas 1D, `glTexImage1D`, que acepta ocho argumentos:

```
void glTexImage1D(GLenum objetivo, GLint nivel, GLint componentes,
GLsizei ancho, GLint borde, GLenum formato, GLenum tipo, const GLvoid
*pixels);
```

- El argumento objetivo especifica qué textura hay que definir; este argumento debe ser `GL_TEXTURE_1D`.
- El argumento nivel indica el nivel de detalle, y normalmente es 0.
- El argumento componentes, especifica el número de valores de color usados en cada píxel. Para las texturas RGB y RGBA se usan 3 y 4, respectivamente.
- El argumento borde controla el número de píxels de bordes que OpenGL debe esperar y puede valer 0, 1 ó 2.
- El argumento ancho especifica la anchura de la textura original (sin borde) y debe ser una potencia de 2.
- El argumento formato indica el tipo de valores de color esperados, `GL_COLOR_INDEX`, `GL_LUMINANCE`, `GL_RGBA`.

Un ejemplo de una textura 1D<sup>27</sup>:

```
Void LoadAllTextures(void)
{
    static unsigned char royg biv_image[8][3] =
    {
        { 0x3f, 0x00, 0x3f },    /* Violeta rojo (para 8 colores...) */
        { 0x7f, 0x00, 0x7f },    /* Violeta */
        { 0xbf, 0x00, 0xbf },    /* Indigo */
    }
```

<sup>27</sup> Govil-Pai Shalini, Principles of Computer Graphics: Theory and Practice Using OpenGL and Maya, Editorial Springer. Capítulo 6 Rendering, Shading and Lighths , páginas 154-162

```

{ 0x00, 0x00, 0xff }, /* Azul */
{ 0x00, 0xff, 0x00 }, /* Verde */
{ 0xff, 0xff, 0x00 }, /* Amarillo */
{ 0xff, 0x7f, 0x00 }, /* Naranja */
{ 0xff, 0x00, 0x00 } /* Rojo */
};

glNewList(RainbowTexture = glGenLists(1), GL_COMPILE);
glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
glTexImage1D(GL_TEXTURE_1D, 0, 3, 8, 0, GL_RGB,
GL_UNSIGNED_BYTE, tesis_image);
glEndList();
}

```

El código de ejemplo crea una vista de visualización que contiene la textura y los filtros deseados de magnificación y minimización, GL\_LINEAR. EL filtro de minimización se usa cuando el polígono dibujado es más pequeño que la textura. El filtro de magnificación se usa cuando el polígono es más grande que la textura.

Cuando se utiliza el filtro GL\_LINEAR, OpenGL interpola linealmente los valores de color de la textura antes de dibujar. Otros filtros que se pueden utilizar con GL\_TEXTURE\_MIN\_FILTER son:

Filtro	Descripción
GL_NEAREST	Filtraje de vecino más próximo.
GL_LINEAR	Interpolación lineal.
GL_NEAREST_MIPMAP_LINEAR	Filtraje de vecino más cecano con textura multimapa.
GL_LINEAR_MIPMAP_LINEAR	Interpolación lineal de textura multimapa interpolada.
GL_LINEAR_MIPMAP_NEAREST	Interpolación lineal de textura multimapa.

### 4.4.1.2. Textura 2D.

Para definir una textura 2D en OpenGL, llamamos a `glTexImage2D`, que toma un argumento de altura aparte de los que usa `glTexImage1D`:

```
void glTexImage2D(GLenum objetivo, GLint nivel, GLint componentes,
GLsizei ancho, GLsizei alto, GLint borde, GLenum formato, GLenum tipo,
const GLvoid *pixels);
```

- El argumento objetivo (`GLenum`), debe ser `GL_TEXTURE_2D`.
- El argumento nivel (`GLint`), normalmente es cero a menos que se usen texturas multimapas.
- El argumento componentes (`GLint`), el número de componentes de color: de uno a cuatro.
- El argumento ancho (`GLsizei`), es el ancho de la textura. Debe ser una potencia de dos.
- El argumento alto (`GLsizei`), es el alto de la textura. Debe ser una potencia de dos.
- El argumento borde (`GLint`), especifica la anchura del borde. Debe ser cero, uno ó dos.
- El argumento formato (`GLenum`), identifica el formato de la información de píxel. Los formatos válidos son:

- `GL_COLOR_INDEX`: Los valores de píxel son índices de color.
- `GL_RED`: Los valores de píxel son intensidades de rojo.
- `GL_GREEN`: Los valores de píxel son intensidades de verde.
- `GL_BLUE`: Los valores de píxel son intensidades de azul.
- `GL_ALPHA`: Los valores de píxel son intensidades alfa.
- `GL_RGB`: Los valores de píxel son colores RGB.
- `GL_RGBA`: Los valores de píxel son colores RGBA.
- `GL_LUMINANCE`: Los valores de píxel son colores en escala de grises.

- GL\_ALPHA\_LUMINANCE: Los valores de píxel son colores alfa y en escala de grises.
- El argumento tipo (GLenum), se declara el tipo de dato de cada valor de píxel.

```

Void LoadAllTextures(void)
{
    BITMAPINFO *info;           /* Información del mapa de
bits */
    void *bits;                /* Bits por píxel del mapa */
    GLubyte *rgb;              /* Pixels RGB del mapa */

    /*
    * Intenta abrir el mapa de bits y convertirlo a RGB...
    */
    bits = LoadDIBitmap("sky.bmp", &info);
    if (bits == NULL)
        return;
    rgb = ConvertRGB(info, bits);
    if (rgb == NULL)
    {
        free(info);
        free(bits);

        return;
    };
    glGenLists(1), GL_COMPILE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_NEAREST);
    /*
    * Define la textura 2D.
    */
    glPixelStorei(GL_UNPACK_ALIGNMENT, 4); /* Fuerza la alineación a 4
bytes*/
    glPixelStorei(GL_UNPACK_ROW_LENGTH, 0);
    glPixelStorei(GL_UNPACK_SKIP_ROWS, 0);
    glPixelStorei(GL_UNPACK_SKIP_PIXELS, 0);

```



```

    glTexImage2D(GL_TEXTURE_2D, 0, 3, info->bmiHeader.biWidth, info-
    >bmiHeader.biHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, rgb);
    glEndList();

/*
 * Libera el mapa de bits y la imagen RGB, y devuelve un 0 (sin errores).
 */

    free(rgb);
    free(info);
    free(bits);
}

```

#### 4.4.2. Dibujo de polígonos con texturas<sup>28</sup>.

Una vez definida una textura, se debe activar para los polígonos, a los que se le aplicara.

```

glDisable(GL_TEXTURE_2D);
glEnable(GL_TEXTURE_1D);
glTexEnv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);

```

La llamada a glEnable activa las texturas 1D. La función glTexEnv selecciona el modo de texturas superposición, que significa que las imágenes se superponen directamente a los polígonos. Esta función utiliza otros modos, los cuales son:

Modo	Descripción
GL_MODULATE	Los píxel de la textura filtran los píxeles de color existentes en la pantalla
GL_DECAL	Los píxeles de la textura reemplazan los píxeles de color existentes en la pantalla
GL_BLEND	Los píxeles de la textura filtran los píxeles de color existentes y se combinan con un color constante

<sup>28</sup> Dave Astle, Kevin Hawkins, Beginning OpenGL Game Programming. Editorial: Thomson Course Technology. Capítulo 7. Texture Mapping. Páginas 169-177.

El modo de textura `GL_MODULATE` multiplica el color de la textura por el largo de la pantalla. Para texturas de una componente, éstos se traducen en un filtro de brillo que variará el brillo de la pantalla basándose en la textura. Para texturas de tres componentes (RGB), podemos generar efectos de filtro de lentes coloreadas. `GL_BLEND` permite mezclar un color constante en la escena basándonos en la textura. Una vez que hemos definido el modo de textura que hay que usar, podemos proceder a dibujar en nuestros polígonos. En el código a continuación se muestra cómo dibujar un arcoiris con textura:

```
glEnable(GL_TEXTURE_1D);
glCallList(RainbowTexture);
glBegin(GL_QUAD_STRIP);
  for (th = 0.0; th <= M_PI; th += (0.03125 * M_PI))
  {
    /*Borde superior del arcoiris*/
    x = cos(th) * 50.0;
    y = sin(th) * 50.0;
    z = -50.0;
    glTexCoord1f(0.0);
    glVertex3f(x, y, z);

    /*Borde inferior del arcoiris*/
    x = cos(th) * 55.0;
    y = sin(th) * 55.0;
    z = -50.0;
    glTexCoord1f(1.0);
    glVertex3f(x, y, z);
  };
glEnd();
```

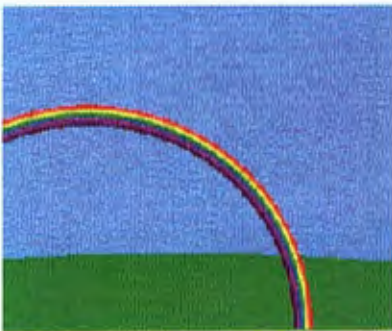


Figura No.28: Imagen con aplicación de textura.

Según la figura No.28, para colocar la textura en el arcoiris, se llama a la función `glTexCoord`. Para texturas 1D, podemos llamar a una de las funciones `glTexCoord1f`, `glTexCoord1d`, `glTexCoord1s` o `glTexCoord1i`. Un valor de 0.0 representa el píxel más a la izquierda de la imagen, y un valor de 1.0 representa el píxel más a la derecha.

Los valores fuera de este rango se manejan de manera diferente, dependiendo del valor del parámetro `GL_TEXTURE_WRAP_S`, si tiene seleccionado `GL_CLAMP` (por defecto), las coordenadas de textura se restringen al rango de 0.0 a 1.0, incluidos. Cuando un polígono pierde la textura, se dibuja usando el color a lo largo del borde de la textura. Las coordenadas de la textura se refieren tradicionalmente a (s,t) en lugar de x e y. Si se utiliza `GL_REPEAT`, la textura se **tesela** sobre el polígono. Las coordenadas de textura se emplean con módulo 1.0, la textura se repite a intervalos regulares.

### 4.4.3. Texturas Multimapa<sup>29</sup>.

Las secuencias animadas necesitan diferentes niveles de detalle dependiendo de la distancia del observador. Por ejemplo, cuando se camina por una habitación virtual, se podría requerir una vista de alta resolución de una foto de una imagen cercana, y sólo un bosquejo desde lejos.

La librería OpenGL permite el soporte de texturas con múltiples imágenes, lo que se llama texturas **multimapa**. El multimapa selecciona la textura más cercana a la resolución de pantalla de un polígono. Las texturas multimapa se definen proporcionando un parámetro de nivel específico para cada mapa. Para la aplicación de este tipo de textura, se explica a continuación.

---

<sup>29</sup> Lengyel Eric. Mathematic for 3D Game Programming and Computer Graphics. Editorial: Charles Rivera Media. Capítulo 6 Illuminations. Páginas 140-141



```

static unsigned char imagen_rnavai0[16][3];
static unsigned char imagen_rnavai1[8][3];
static unsigned char imagen_rnavai2[4][3];
static unsigned char imagen_rnavai3[2][3];
static unsigned char imagen_rnavai4[1][3];
glNewList(TexturaArcoiris=glGenLists(1), GL_COMPILE);
    glTexParameteri(GL_TEXTURE1D,          GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
    glTexParameteri(GL_TEXTURE1D,          GL_TEXTURE_MIN_FILTER,
GL_NEAREST_MIPMAP_LINEAR);
    glTexImage1D(GL_TEXTURE1D, 0, 3, 16, 0, GL_RGB,
GL_UNSIGNED_BYTE, imagen_rnavai0);
    glTexImage1D(GL_TEXTURE1D, 1, 3, 8, 0, GL_RGB,
GL_UNSIGNED_BYTE, imagen_rnavai1);
    glTexImage1D(GL_TEXTURE1D, 2, 3, 4, 0, GL_RGB,
GL_UNSIGNED_BYTE, imagen_rnavai2);
    glTexImage1D(GL_TEXTURE1D, 3, 3, 2, 0, GL_RGB,
GL_UNSIGNED_BYTE, imagen_rnavai3);
    glTexImage1D(GL_TEXTURE1D, 4, 3, 1, 0, GL_RGB,
GL_UNSIGNED_BYTE, imagen_rnavai4);
glEndList();

```

El nivel de cada imagen está especificado en el primer parámetro de `glTexImage1D()`. El nivel 0 es el primario, textura de mayor resolución.

El nivel 1 tiene la mitad de tamaño que la textura primaria, y continua disminuyendo. Cuando se dibujen polígonos con textura multimapa, se utilizaran diferentes filtros de minimización, estos se describen en la siguiente tabla:

Filtro	Descripción
GL_NEAREST_MIPMAP_NEAREST	Utiliza la imagen más cercana a la resolución de la pantalla. Usaremos el filtro GL_NEAREST cuando usemos este mapa
GL_NEAREST_MIPMAP_LINEAR	Utiliza la imagen más cercana a la resolución de la pantalla. Usaremos el filtro GL_LINEAR cuando usemos este mapa
GL_LINEAR_MIPMAP_NEAREST	Interpola linealmente entre las dos imágenes más cercanas a la resolución de la pantalla.

	Usaremos el filtro GL_NEAREST cuando utilizemos este mapa.
GL_LINEAR_MIPMAP_LINEAR	Interpola linealmente entre las dos imágenes más cercanas a la resolución de la pantalla. Usaremos el filtro GL_LINEAR cuando utilizemos este mapa.

Los filtros GL\_LINEAR\_MIPMAP\_NEAREST y GL\_LINEAR\_MIPMAP\_LINEAR pueden resultar muy pesados en tiempos de ejecución. GL\_NEAREST\_MIPMAP\_NEAREST es muy parecido a GL\_NEAREST en ejecución, pero generalmente produce resultados mucho mejores. Las imágenes multimapa se eligen comparando el tamaño de los polígonos a medida que se dibujan en la pantalla.

La librería de utilidades OpenGL proporciona dos funciones que generan automáticamente imágenes multimapa basadas en una única textura de alta resolución. En el siguiente código, las funciones gluBuild1DMipmaps y gluBuild2DMipmaps

```

/*Textura 1D*/
glTexParameteri(GL_TEXTURE_1D,          GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
glTexParameteri(GL_TEXTURE_1D,          GL_TEXTURE_MIN_FILTER,
GL_NEAREST_MIPMAP_LINEAR);
gluBuild1DMipmaps(GL_TEXTURE_1D,      3,      8,      0,      GL_RGB,
GL_UNSIGNED_BYTE, imagen_rnavai);

/*Textura 2D*/
glTexParameteri(GL_TEXTURE_2D,          GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,          GL_TEXTURE_MIN_FILTER,
GL_NEAREST_MIPMAP_LINEAR);
gluBuild1DMipmaps(GL_TEXTURE_2D,info->bmiCabecera.biAncho,      info->
bmiCabecera.biAlto, 0, GL_RGB, GL_UNSIGNED_BYTE, rgb);

```

Dado que las funciones `gluBuild1DMipmaps` y `gluBuild2DMipmaps` crean imágenes de una imagen, puede que la apariencia de algunas texturas no sea muy precisa.

## 4.5. Transparencias (alpha Blending)<sup>30</sup>

Este tipo de función usa el valor Alfa (valor de material difuso) del código RGBA, y permite combinar el color del fragmento que se procesa con el del píxel que ya está en el buffer. Imagina por ejemplo dibujar una ventana transparente de color azul claro enfrente de una caja roja. El Alpha blending permite simular la transparencia de la ventana, de manera que la caja vista a través del cristal aparezca con un tono magenta.

Cuando se usan valores alpha, un nuevo concepto llamado 'mezcla alpha' o 'alpha blending' se debe explicar. El alpha blending es el proceso de mezclar colores según los valores alpha de la fuente (SRC) y del destino (DST). El objeto fuente es el creado más tarde en la cadena gráfica, ya sea porque pertenece a un grupo con valor `GLinsert` mayor o porque está creado más tarde. El color final de cada píxel se define como:

$$(ec22) \quad C = C_{src} * F_{src} + C_{dst} * F_{dst}$$

Donde C es el color original y F se refiere a la función de mezcla (blending function) usada. Por ejemplo cuando se utiliza el código:

```
glBlendFunc GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA
```

La función usada será:

$$(ec23) \quad C = C_{src} * \alpha_{src} + C_{dst} * (1 - \alpha_{src})$$

---

<sup>30</sup> Zerbst Stefan 3D Game Engine Programming. Editorial: Thomson Course Technology. Capítulo 5: Materials, Textures and Transparency. Páginas 226-230.

En este caso el color del objeto de adelante (SRC) se multiplica por su propio valor alpha, y el color del objeto de destino se multiplica por (1 menos el alpha del objeto fuente). Así que veremos más del objeto trasero cuando el alpha del objeto fuente es menor. Puede pensarse el alpha como el valor de transparencia, siendo 1 totalmente opaco y 0 totalmente transparente.

## EJEMPLOS UNIDAD 4:

**Ejemplo 4.1:** Determinar la intensidad de iluminación  $I_p$  del punto  $P = (-5, 8, 0)$  y  $P_o = (18, 9, 7)$

Solución:

$$I_p(P, P_o) = \left( \frac{1}{|P - P_o|} \right) (P_o)$$

$$I_p(P, P_o) = \left( \frac{1}{|[-5 \ 8 \ 0] - [18 \ 9 \ 7]|} \right) \cdot [18 \ 9 \ 7]$$

$$I_p(P, P_o) = \left( \frac{1}{|[-23 \ -1 \ -7]|} \right) [18 \ 9 \ 7]$$

Nota: Recordemos que la división de matrices se establece como:

Sean las Matrices A y B. La división de  $A/B = A \cdot B^{-1}$

Entonces:

Determinemos a  $P_o^{-1}$

$$P_o^{-1} = [-1/23 \quad -1 \quad -1/7].$$

**Ejemplo 4.2:** Este ejemplo muestra un programa desarrollado con ayuda de OpenGL, que genera un polígono triangular, el programa explicará la utilización de colores e iluminación.

```
#include <GL/glut.h>
#include <gl/glut.h>
#include <string.h>
#include "CWin.h"

#pragma comment (lib,"glut32.lib")
#pragma comment (lib,"opengl32.lib")

//inicializa glut
void CWin::InitWin(int argc,char **argv)
{
    glutInit(&argc,argv);
}

//Crea nuestra ventana
void CWin::CreateWin(int iWidth,int iHeight,const char *Title,bool bFullScreen)
{
    glutInitDisplayMode(GLUT_RGB | GLUT_RGBA | GLUT_DOUBLE |
    GLUT_DEPTH);
    if(bFullScreen){
        glutInitWindowPosition(100,100);
        glutInitWindowSize(iWidth,iHeight);
        glutCreateWindow(Title);
        glutFullScreen();
        ChangeSizeWin(iWidth,iHeight);
        strTitle=new char[strlen(Title)+1];
        strcpy(strTitle,Title);
        Width=iWidth;
        Height=iHeight;
    }
    else{
        glutInitWindowPosition(100,100);
        glutInitWindowSize(iWidth,iHeight);
        glutCreateWindow(Title);
        ChangeSizeWin(iWidth,iHeight);
        strTitle=new char[strlen(Title)+1];
        strcpy(strTitle,Title);
        Width=iWidth;
        Height=iHeight; }}
}
```



```

char *CWin::GetCaption()//Devuelve el titulo
{
    return strTitle;
}

//Devuelve el tamaño de la ventana
void CWin::GetSizeWin(int &iWidth,int &iHeight)
{
    iWidth =Width;
    iHeight=Height;
}

//Limpia el buffer de profundidad
void CWin::ClearWin()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
}

//Cambia el tamaño de la ventana
void CWin::ChangeSizeWin(int iWidth,int iHeight)
{
    if (iHeight==0){ //Previene una division entre cero
        iHeight=1;
    }
    glViewport(0,0,iWidth,iHeight);    //Ajusta la vista a las dimensiones de la
    ventana
    glMatrixMode(GL_PROJECTION);    //Activa la matriz de proyeccion
    glLoadIdentity();    //Reinicia el sistema de coordenadas
    gluPerspective(45.0f,(GLfloat)iWidth/(GLfloat)iHeight, 1.0f ,1000.0f);
    glMatrixMode(GL_MODELVIEW);    //Activa la matriz del modelador
    glLoadIdentity();    //Reinicia el sistema de coordenadas
}

void Triangle();

void Render();

CWin Win;

/*CUERPO DE PROGRAMA */

```

```

int main(int argc, char **argv)
{

Win.InitWin(argc,argv);
Win.CreateWin(640,480,"Ejemplo2: Agregando color.",false);
glutDisplayFunc(Render);
glutMainLoop();
return 0;
}

/*declaración de función triangle y render*/
void Triangle()
{
    glTranslatef(0.0f,0.0f,-6.0f);
    glBegin(GL_TRIANGLES);
        glColor3f(0.38f,0.38f,0.38f);
            glVertex3f( 0.5f, 0.5f, 0.5f);
            glColor3f(0.90f,0.90f,0.90f);
            glVertex3f(-1.0f,-1.0f, 0.0f);
            glColor3f(0.75f,0.75f,0.75f);
            glVertex3f( 1.0f,-1.0f, 0.0f);
        glEnd();
}

void Render()
{
    Win.ClearWin();
    Triangle();
    glutSwapBuffers();
}

```

#### Salida de Programa:

El programa genera un triángulo con una tenue iluminación enfocada desde la parte inferior de la figura.

Estos efectos de color e iluminación, pueden ser cambiados con las funciones `glVertex3f`, en la función de creación del polígono. Así:



Figura No.29: Salida de programa



```

void Triangle()
{
    glTranslatef(0.0f,0.0f,-6.0f);
    glBegin(GL_TRIANGLES);
        glColor3f(0.38f,0.38f,0.38f);
        glVertex3f( 0.5f, 0.5f, 0.5f);
        glColor3f(0.90f,0.90f,0.90f);
        glVertex3f(-1.0f,-1.0f, 0.0f);
        glColor3f(0.98f,0.98f,0.88f);
        glVertex3f( 1.0f,-1.0f, 0.0f);
    glEnd();
}

```

El triángulo cambia de la siguiente manera (ver figura No.30):

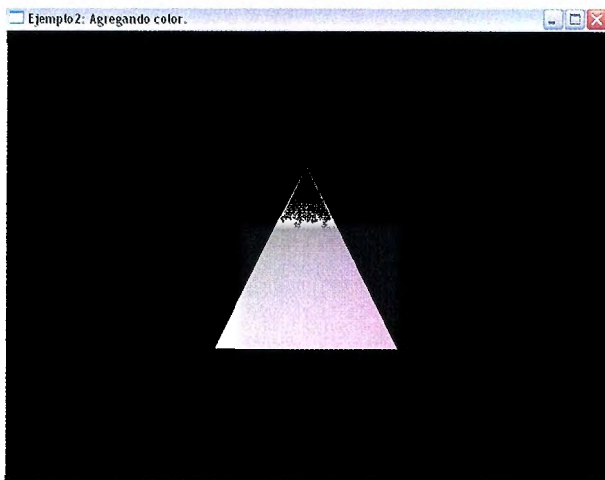


Figura No.30: Cambio de iluminación y color en la figura con la función glColor3f

La orientación de la iluminación puede ser cambiada en ésta línea de código: glColor3f(0.30f,0.30f,0.40f); Mostrando la imagen de la siguiente forma: (Ver figura No.31)

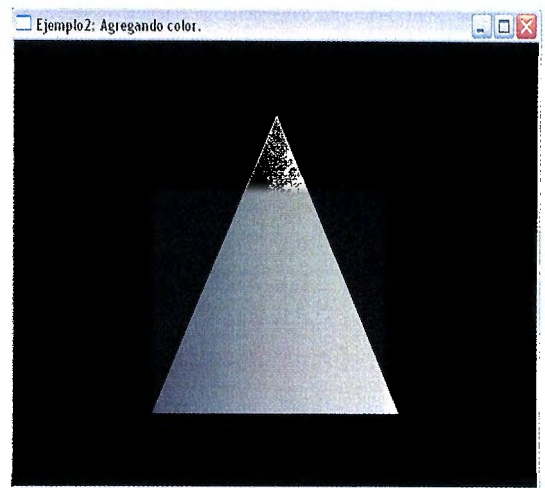


Figura No.31: La figura ha sido escalada y se ha realizado el cambio de la orientación en la iluminación

En ésta última imagen el triángulo ha sido ampliado con:

gluPerspective(28.0f,(GLfloat)iWidth/(GLfloat)iHeight, 1.0f ,1000.0f); Donde el primer parámetro permite la escalación de las imágenes de forma directa.

**Ejemplo 4.3:** Este ejemplo presenta un cubo formado con un tipo de textura.

```
#include <GL/glut.h>
#include <string.h>
#include "CWin.h"
#include "CImage.h"

#include <iostream.h>
#include <gl/glaux.h>
#include <fstream.h>

#pragma comment (lib,"glut32.lib")
#pragma comment (lib,"glaux.lib")
#pragma comment (lib,"opengl32.lib")

#define ID_TEXTURE 0

void Quad();
void Render();
void Idle(void);
void InitGL();

float Ang=0.0f;
unsigned int Texture[1];
CImage Image;
CWin Win;

// DECLARACIONES PRINCIPALES
//inicializa glut
void CWin::InitWin(int argc,char **argv)
{
    glutInit(&argc,argv);
}

//Crea nuestra ventana
void CWin::CreateWin(int iWidth,int iHeight,const char *Title,bool bFullScreen)
{
    glutInitDisplayMode(GLUT_RGB | GLUT_RGBA | GLUT_DOUBLE |
    GLUT_DEPTH);
    if(bFullScreen){
        glutInitWindowPosition(100,100);
        glutInitWindowSize(iWidth,iHeight);
        glutCreateWindow(Title);
        glutFullScreen();
        ChangeSizeWin(iWidth,iHeight);
        strTitle=new char[strlen(Title)+1];
```

```

strcpy(strTitle,Title);
Width=iWidth;
Height=iHeight;
}
else{
glutInitWindowPosition(100,100);
glutInitWindowSize(iWidth,iHeight);
glutCreateWindow(Title);
ChangeSizeWin(iWidth,iHeight);
strTitle=new char[strlen(Title)+1];
strcpy(strTitle,Title);
Width=iWidth;
Height=iHeight;
}}

//Devuelve el titulo
char *CWin::GetCaption()
{
return strTitle;
}

//Devuelve el tamaño de la ventana
void CWin::GetSizeWin(int &iWidth,int &iHeight)
{
iWidth =Width;
iHeight=Height;
}

//Limpia el buffer de profundidad
void CWin::ClearWin()
{
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
}

//Cambia el tamaño de la ventana
void CWin::ChangeSizeWin(int iWidth,int iHeight)
{
if (iHeight==0){ //Previene una division entre cero
iHeight=1;
}
glViewport(0,0,iWidth,iHeight); //Ajusta la vista a las dimensiones de la
ventana
glMatrixMode(GL_PROJECTION); //Activa la matriz de proyeccion
glLoadIdentity(); //Reinicia el sistema de coordenadas
gluPerspective(45.0f,(GLfloat)iWidth/(GLfloat)iHeight, 1.0f ,1000.0f);

```



```

    glMatrixMode(GL_MODELVIEW);      //Activa la matriz del modelador
    glLoadIdentity();                //Reinicia el sistema de coordenadas
}

int main(int argc, char **argv)
{
    Win.InitWin(argc,argv);
    Win.CreateWin(640,480,"Ejemplo 3: Mapeado de textura.",false);
    InitGL();
    glutDisplayFunc(Render);
    glutIdleFunc(Idle);
    glutMainLoop();
    return 0;
}

void Idle(void)
{
    Ang++;
    Render();
    glutPostRedisplay();
}

void InitGL()
{
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glClearDepth(1.0f);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_TEXTURE_2D);
    glDepthFunc(GL_LEQUAL);
    if(!Image.CreateTexture(Texture,ID_TEXTURE,"texture.bmp")){
        cerr << "Error al cargar la textura" << endl;
        exit(1);
    }
}

void Quad()
{
    glTranslatef(0.0f,0.0f,-6.0f);
    glRotatef(Ang,0.0f,1.0f,0.0f);
    glBindTexture(GL_TEXTURE_2D,Texture[ID_TEXTURE]);
    glBegin(GL_QUADS);
        glTexCoord2f(0.0, 0.0); glVertex3f(-0.5, -0.5, 0.5);
        glTexCoord2f(1.0, 0.0); glVertex3f( 0.5, -0.5, 0.5);
        glTexCoord2f(1.0, 1.0); glVertex3f( 0.5, 0.5, 0.5);
        glTexCoord2f(0.0, 1.0); glVertex3f(-0.5, 0.5, 0.5);
}

```

```

glTexCoord2f(1.0, 0.0); glVertex3f(-0.5, -0.5, -0.5);
glTexCoord2f(1.0, 1.0); glVertex3f(-0.5, 0.5, -0.5);
glTexCoord2f(0.0, 1.0); glVertex3f( 0.5, 0.5, -0.5);
glTexCoord2f(0.0, 0.0); glVertex3f( 0.5, -0.5, -0.5);

glTexCoord2f(0.0, 1.0); glVertex3f(-0.5, 0.5, -0.5);
glTexCoord2f(0.0, 0.0); glVertex3f(-0.5, 0.5, 0.5);
glTexCoord2f(1.0, 0.0); glVertex3f( 0.5, 0.5, 0.5);
glTexCoord2f(1.0, 1.0); glVertex3f( 0.5, 0.5, -0.5);

glTexCoord2f(1.0, 1.0); glVertex3f(-0.5, -0.5, -0.5);
glTexCoord2f(0.0, 1.0); glVertex3f( 0.5, -0.5, -0.5);
glTexCoord2f(0.0, 0.0); glVertex3f( 0.5, -0.5, 0.5);
glTexCoord2f(1.0, 0.0); glVertex3f(-0.5, -0.5, 0.5);

glTexCoord2f(1.0, 0.0); glVertex3f( 0.5, -0.5, -0.5);
glTexCoord2f(1.0, 1.0); glVertex3f( 0.5, 0.5, -0.5);
glTexCoord2f(0.0, 1.0); glVertex3f( 0.5, 0.5, 0.5);
glTexCoord2f(0.0, 0.0); glVertex3f( 0.5, -0.5, 0.5);

glTexCoord2f(0.0, 0.0); glVertex3f(-0.5, -0.5, -0.5);
glTexCoord2f(1.0, 0.0); glVertex3f(-0.5, -0.5, 0.5);
glTexCoord2f(1.0, 1.0); glVertex3f(-0.5, 0.5, 0.5);
glTexCoord2f(0.0, 1.0); glVertex3f(-0.5, 0.5, -0.5);
glEnd();
}
void Render()
{
    Win.ClearWin();
    Quad();
    glutSwapBuffers();
}

```



Figura No.32.: Salida de programa

Salida de Programa:

La salida es un cubo que rota sobre el eje z con una textura tipo piedra. Esta textura se forma con la ayuda de un archivo **texture.bmp** guardado en la carpeta de compilación del programa.

Para cambiar el tipo de textura solo se hace referencia a un nuevo archivo en la parte de **Image.CreateTexture(Texture, ID\_TEXTURE, "texture.bmp")** así:

```
Image.CreateTexture(Texture, ID_TEXTURE, "texture1.bmp")
```

Al realizar éstos cambios en el programa el cubo cambia la textura haciendo



Figura No.33: Cambio de textura de la imagen

referencia al archivo indicado. (ver figura No.33)

Como se observa en ésta última, la imagen ha sido ampliada sin cambiar los ejes principales, esto se logra en la siguiente línea de comando:

```
gluPerspective(35.0f, (GLfloat)iWidth / (GLfloat)iHeight, 1.0f, 1000.0f);
```

Donde el primer parámetro 35.0f indica el cambio de escala de los vértices.



**Ejemplo 4.4:** Este ejemplo presenta el mismo cubo del ejemplo anterior pero con la aplicación de luz y uso del teclado. El programa permite que con la letra "L" se apague y encienda la iluminación de la figura:

```
#include <GL/glut.h>
#include <string.h>
#include "CWin.h"

#include <iostream.h>
#include "CImage.h"
#include "CEffect.h"
#include <gl/glaux.h>
#include <fstream.h>

#pragma comment (lib,"glut32.lib")
#pragma comment (lib,"glaux.lib")
#pragma comment (lib,"opengl32.lib")

#define ID_TEXTURE 0

void Quad();
void Render();
void Idle(void);
void InitGL();
void Key(unsigned char Tecla,int x, int y);

float Ang=0.0f;
unsigned int Texture[1];

CImage Image;
CWin Win;

//inicializa glut
void CWin::InitWin(int argc,char **argv)
{
    glutInit(&argc,argv);
}

//Crea nuestra ventana
void CWin::CreateWin(int iWidth,int iHeight,const char *Title,bool bFullScreen)
{
    glutInitDisplayMode(GLUT_RGB | GLUT_RGBA | GLUT_DOUBLE |
    GLUT_DEPTH);
```

```

if(bFullScreen){
    glutInitWindowPosition(0,0);
    glutInitWindowSize(iWidth,iHeight);
    glutCreateWindow(Title);
    glutFullScreen();
    ChangeSizeWin(iWidth,iHeight);
    strTitle=new char[strlen(Title)+1];
    strcpy(strTitle,Title);
    Width=iWidth;
    Height=iHeight;
}
else{
    glutInitWindowPosition(100,100);
    glutInitWindowSize(iWidth,iHeight);
    glutCreateWindow(Title);
    ChangeSizeWin(iWidth,iHeight);
    strTitle=new char[strlen(Title)+1];
    strcpy(strTitle,Title);
    Width=iWidth;
    Height=iHeight;
}
}

//Devuelve el titulo
char *CWin::GetCaption()
{
    return strTitle;
}

//Devuelve el tamaño de la ventana
void CWin::GetSizeWin(int &iWidth,int &iHeight)
{
    iWidth =Width;
    iHeight=Height;
}

//Limpia el buffer de profundidad
void CWin::ClearWin()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
}

//Cambia el tamaño de la ventana
void CWin::ChangeSizeWin(int iWidth,int iHeight)

```



```

{
    if (iHeight==0){ //Previene una division entre cero
        iHeight=1;
    }
    glViewport(0,0,iWidth,iHeight);    //Ajusta la vista a las dimensiones de la
    ventana
    glMatrixMode(GL_PROJECTION);    //Activa la matriz de proyeccion
    glLoadIdentity();    //Reinicia el sistema de coordenadas
    gluPerspective(28.0f,(GLfloat)iWidth/(GLfloat)iHeight, 1.0f ,1000.0f);
    glMatrixMode(GL_MODELVIEW);    //Activa la matriz del modelador
    glLoadIdentity();    //Reinicia el sistema de coordenadas
}

int main(int argc, char **argv)
{
    Win.InitWin(argc,argv);
    Win.CreateWin(540,380,"Ejemplo 4: Manejo de luces y de teclado.",false);

    InitGL();
    glutKeyboardFunc(Key);
    glutDisplayFunc(Render);
    glutIdleFunc(Idler);
    glutMainLoop();
    return 0;
}

void Idler(void)
{
    Ang++;
    Render();
    glutPostRedisplay();
}

void Key(unsigned char Tecla,int x, int y)
{
    CEffect Effect;
    static bool bLight=true;
    switch(Tecla){
        case 27: //exit (0);
            break;
        case 'I': case 'L':
            bLight=!bLight;
    }
}

```

```

        Effect.EnableLight(bLight);
        break;
    }
}

void InitGL()
{
    CEffect Effect;
    GLfloat Diffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f };
    GLfloat Ambient[] = { 0.5f, 0.5f, 0.5f, 1.0f };
    GLfloat Position[] = { 0.0f, 0.0f, 2.0f, 1.0f };
    GLfloat Specular[] = { 0.0f, 0.0f, 0.0f, 1.0f };
    cout << "Oprime I o L para activar/desactivar las luces" << endl;
    cout << "Oprime <Esc> para salir" << endl;

    if(!Image.CreateTexture(Texture,ID_TEXTURE,"texture1.bmp")){
        cerr << "Error al cargar la textura" << endl;
        // exit(1);
    }
    glEnable(GL_TEXTURE_2D);
    glShadeModel(GL_SMOOTH);
    glClearColor(0.0f, 0.0f, 0.0f, 0.5f);
    glClearDepth(1.0f);
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
    Effect.SetAmbientLight(Ambient);
    Effect.SetDiffuseLight(Diffuse);
    Effect.SetSpecularLight(Specular);
    Effect.SetPositionLight(Position);
    Effect.EnableLight(true);
    Effect.SetNumLight(GL_LIGHT0);
}

void Quad()
{
    glTranslatef(0.0f,0.0f,-6.0f);
    glRotatef(Ang,0.0f,1.0f,0.0f);
    glBindTexture(GL_TEXTURE_2D,Texture[ID_TEXTURE]);

    glBegin(GL_QUADS);
    glNormal3f( 0.0f, 0.0f, 1.0f);
        glTexCoord2f(0.0, 0.0); glVertex3f(-0.75, -0.75, 0.75);
        glTexCoord2f(1.0, 0.0); glVertex3f( 0.75, -0.75, 0.75);
        glTexCoord2f(1.0, 1.0); glVertex3f( 0.75, 0.75, 0.75);
        glTexCoord2f(0.0, 1.0); glVertex3f(-0.75, 0.75, 0.75);
}

```



```

glNormal3f( 0.0f, 0.0f,-1.0f);
    glTexCoord2f(1.0, 0.0); glVertex3f(-0.75, -0.75, -0.75);
    glTexCoord2f(1.0, 1.0); glVertex3f(-0.75, 0.75, -0.75);
    glTexCoord2f(0.0, 1.0); glVertex3f( 0.75, 0.75, -0.75);
    glTexCoord2f(0.0, 0.0); glVertex3f( 0.75, -0.75, -0.75);

glNormal3f( 0.0f, 1.0f, 0.0f);
    glTexCoord2f(0.0, 1.0); glVertex3f(-0.75, 0.75, -0.75);
    glTexCoord2f(0.0, 0.0); glVertex3f(-0.75, 0.75, 0.75);
    glTexCoord2f(1.0, 0.0); glVertex3f( 0.75, 0.75, 0.75);
    glTexCoord2f(1.0, 1.0); glVertex3f( 0.75, 0.75, -0.75);

glNormal3f( 0.0f,-1.0f, 0.0f);
    glTexCoord2f(1.0, 1.0); glVertex3f(-0.75, -0.75, -0.75);
    glTexCoord2f(0.0, 1.0); glVertex3f( 0.75, -0.75, -0.75);
    glTexCoord2f(0.0, 0.0); glVertex3f( 0.75, -0.75, 0.75);
    glTexCoord2f(1.0, 0.0); glVertex3f(-0.75, -0.75, 0.75);

glNormal3f( 1.0f, 0.0f, 0.0f);
    glTexCoord2f(1.0, 0.0); glVertex3f( 0.75, -0.75, -0.75);
    glTexCoord2f(1.0, 1.0); glVertex3f( 0.75, 0.75, -0.75);
    glTexCoord2f(0.0, 1.0); glVertex3f( 0.75, 0.75, 0.75);
    glTexCoord2f(0.0, 0.0); glVertex3f( 0.75, -0.75, 0.75);

glNormal3f(-1.0f, 0.0f, 0.0f);
    glTexCoord2f(0.0, 0.0); glVertex3f(-0.75, -0.75, -0.75);
    glTexCoord2f(1.0, 0.0); glVertex3f(-0.75, -0.75, 0.75);
    glTexCoord2f(1.0, 1.0); glVertex3f(-0.75, 0.75, 0.75);
    glTexCoord2f(0.0, 1.0); glVertex3f(-0.75, 0.75, -0.75);
glEnd();
}

void Render()
{
    Win.ClearWin();
    Quad();
    glutSwapBuffers();
}

```

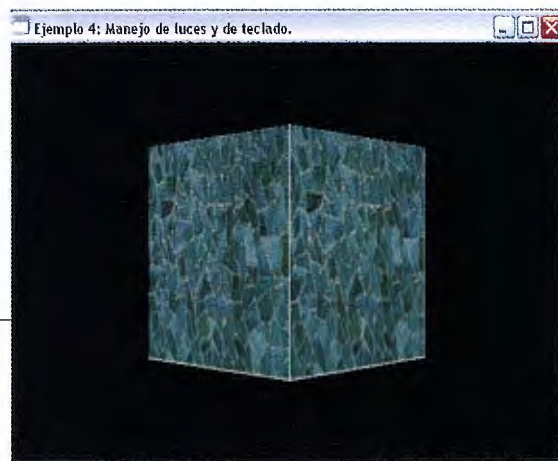
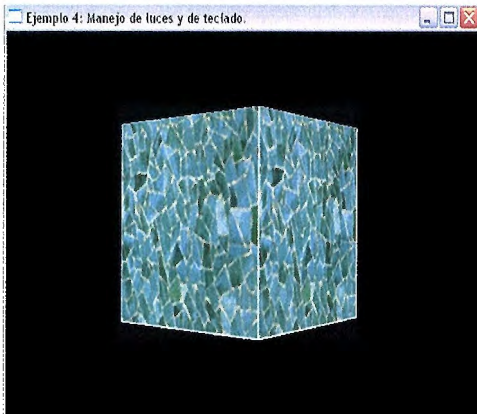


Figura No.34: Salida de Programa, la imagen se presenta sin iluminación en la corrida del programa

Salida del Programa:

Como ya se mencionó en la presentación del ejemplo, el programa permite que cuando se presiona la letra "L" de luz, se le aplique luminosidad a la imagen. (Ver figura No.35).



Este efecto de Iluminación, se logra con:

```
bLight=!bLight;
```

```
Effect.EnableLight(bLight);
```

En el caso de que se desee mantener una imagen iluminada, simplemente ambas líneas de código se colocan fuera del lazo dentro de la función main () o cuerpo principal del programa.

Figura No.35: El cubo se ilumina luego de indicar en el teclado la letra

**Ejemplo 4.5:** En este ejemplo se desea presenta una combinación de texturas, iluminación, uso de teclado y transparencias con OpenGL.

```
#include <GL/glut.h>
#include <string.h>
#include "CWin.h"
#include <iostream.h>
#include "CImage.h"
#include "CEffect.h"

#pragma comment (lib,"glut32.lib")
#pragma comment (lib,"glaux.lib")
#pragma comment (lib,"opengl32.lib")

void Quad();
void Render();
void Idle(void);
void InitGL();
void Key(unsigned char Tecla,int x, int y);
void Scale(float fScale);
```

```

float Ang=0.0f;
float fScale=0.90f;

const int ID_TEXTURE=0;
unsigned int Texture[1];

CImage Image;
CWin Win;

//inicializa glut
void CWin::InitWin(int argc,char **argv)
{
    glutInit(&argc,argv);
}

//Crea nuestra ventana
void CWin::CreateWin(int iWidth,int iHeight,const char *Title,bool bFullScreen)
{
    glutInitDisplayMode(GLUT_RGB | GLUT_RGBA | GLUT_DOUBLE |
    GLUT_DEPTH);
    if(bFullScreen){
        glutInitWindowPosition(100,100);
        glutInitWindowSize(iWidth,iHeight);
        glutCreateWindow(Title);
        glutFullScreen();
        ChangeSizeWin(iWidth,iHeight);
        strTitle=new char[strlen(Title)+1];
        strcpy(strTitle,Title);
        Width=iWidth;
        Height=iHeight;
    }
    else{
        glutInitWindowPosition(100,100);
        glutInitWindowSize(iWidth,iHeight);
        glutCreateWindow(Title);
        ChangeSizeWin(iWidth,iHeight);
        strTitle=new char[strlen(Title)+1];
        strcpy(strTitle,Title);
        Width=iWidth;
        Height=iHeight;
    }
}

//Devuelve el titulo

```



```

char *CWin::GetCaption()
{
    return strTitle;
}

//Devuelve el tamaño de la ventana
void CWin::GetSizeWin(int &iWidth,int &iHeight)
{
    iWidth =Width;
    iHeight=Height;
}

//Limpia el buffer de profundidad
void CWin::ClearWin()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
}

//Cambia el tamaño de la ventana
void CWin::ChangeSizeWin(int iWidth,int iHeight)
{
    if (iHeight==0){ //Previene una division entre cero
        iHeight=1;
    }
    glViewport(0,0,iWidth,iHeight);    //Ajusta la vista a las dimensiones de la
    ventana
    glMatrixMode(GL_PROJECTION);    //Activa la matriz de proyeccion
    glLoadIdentity();    //Reinicia el sistema de coordenadas
    gluPerspective(45.0f,(GLfloat)iWidth/(GLfloat)iHeight, 1.0f ,1000.0f);
    glMatrixMode(GL_MODELVIEW);    //Activa la matriz del modelador
    glLoadIdentity();    //Reinicia el sistema de coordenadas
}

int main(int argc, char **argv)
{
    Win.InitWin(argc,argv);
    Win.CreateWin(640,480,"Ejemplo 5: Manejo de Transparencias.",false);
    InitGL();
    glutKeyboardFunc(Key);
    glutDisplayFunc(Render);
}

```

```

glutIdleFunc(Idler);
glutMainLoop();
return 0;
}

void Idler(void)
{
    Ang++;
    Render();
    glutPostRedisplay();
}

void Key(unsigned char Tecla,int x, int y)
{
    static bool bBlend=true;
    static bool bLight=false;
    CEffect Effect;

    switch(Tecla){
        case 27: //exit (0);
            break;
        case 'b': case 'B':
            bBlend=!bBlend;
            if(bBlend){
                Effect.EnableBlend(true);
                glDisable(GL_DEPTH_TEST);
            }
            else{
                Effect.EnableBlend(false);
                glEnable(GL_DEPTH_TEST);
            }
            break;
        case 'l': case 'L':
            bLight=!bLight;
            Effect.EnableLight(bLight);
            break;
        case '+': fScale+=0.25;
            break;
        case '-': fScale-=0.25;
            break;
    }
}

void InitGL()
{
    CEffect Effect;

```



```

    GLfloat Diffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f };
    GLfloat Ambient[] = { 0.5f, 0.5f, 0.5f, 1.0f };
    GLfloat Position[] = { 0.0f, 0.0f, 2.0f, 1.0f };
    cout << "<Esc> para salir." << endl;
    cout << "<b> Activa/Desactiva la transparencia." << endl;
    cout << "<l> Activa/Desactiva las luces." << endl;
    cout << "<+> Aumenta el tamaño del objeto." << endl;
    cout << "<-> Disminuye el tamaño del objeto." << endl;

    if(!Image.CreateTexture(Texture,ID_TEXTURE,"texture1.bmp")){
        cerr << "Error al cargar la textura" << endl;
//        exit(1);
    }
    glEnable(GL_TEXTURE_2D);
    glShadeModel(GL_SMOOTH);
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glClearDepth(1.0f);
    glColor4f(1.0f,1.0f,1.0f,0.75f);
    Effect.SetBlend(GL_SRC_ALPHA,GL_ONE);
    glDisable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
    Effect.EnableLight(false);
    Effect.SetNumLight(GL_LIGHT0);
    Effect.SetAmbientLight(Ambient);
    Effect.SetDiffuseLight(Diffuse);
    Effect.SetPositionLight(Position);
    Effect.EnableBlend(true);
}

void Quad()
{
    glTranslatef(0.0f,0.0f,-6.0f);
    glRotatef(Ang,0.0f,1.0f,0.0f);
    glScalef(fScale,fScale,fScale);
    glBindTexture(GL_TEXTURE_2D,Texture[ID_TEXTURE]);

    glBegin(GL_QUADS);
    glNormal3f( 0.0f, 0.0f, 1.0f);
        glTexCoord2f(0.0, 0.0); glVertex3f(-0.75, -0.75, 0.75);
        glTexCoord2f(1.0, 0.0); glVertex3f( 0.75, -0.75, 0.75);
        glTexCoord2f(1.0, 1.0); glVertex3f( 0.75, 0.75, 0.75);
        glTexCoord2f(0.0, 1.0); glVertex3f(-0.75, 0.75, 0.75);

    glNormal3f( 0.0f, 0.0f,-1.0f);

```



```

    glTexCoord2f(1.0, 0.0); glVertex3f(-0.75, -0.75, -0.75);
    glTexCoord2f(1.0, 1.0); glVertex3f(-0.75, 0.75, -0.75);
    glTexCoord2f(0.0, 1.0); glVertex3f(0.75, 0.75, -0.75);
    glTexCoord2f(0.0, 0.0); glVertex3f(0.75, -0.75, -0.75);

    glNormal3f(0.0f, 1.0f, 0.0f);
    glTexCoord2f(0.0, 1.0); glVertex3f(-0.75, 0.75, -0.75);
    glTexCoord2f(0.0, 0.0); glVertex3f(-0.75, 0.75, 0.75);
    glTexCoord2f(1.0, 0.0); glVertex3f(0.75, 0.75, 0.75);
    glTexCoord2f(1.0, 1.0); glVertex3f(0.75, 0.75, -0.75);

    glNormal3f(0.0f, -1.0f, 0.0f);
    glTexCoord2f(1.0, 1.0); glVertex3f(-0.75, -0.75, -0.75);
    glTexCoord2f(0.0, 1.0); glVertex3f(0.75, -0.75, -0.75);
    glTexCoord2f(0.0, 0.0); glVertex3f(0.75, -0.75, 0.75);
    glTexCoord2f(1.0, 0.0); glVertex3f(-0.75, -0.75, 0.75);

    glNormal3f(1.0f, 0.0f, 0.0f);
    glTexCoord2f(1.0, 0.0); glVertex3f(0.75, -0.75, -0.75);
    glTexCoord2f(1.0, 1.0); glVertex3f(0.75, 0.75, -0.75);
    glTexCoord2f(0.0, 1.0); glVertex3f(0.75, 0.75, 0.75);
    glTexCoord2f(0.0, 0.0); glVertex3f(0.75, -0.75, 0.75);

    glNormal3f(-1.0f, 0.0f, 0.0f);
    glTexCoord2f(0.0, 0.0); glVertex3f(-0.75, -0.75, -0.75);
    glTexCoord2f(1.0, 0.0); glVertex3f(-0.75, -0.75, 0.75);
    glTexCoord2f(1.0, 1.0); glVertex3f(-0.75, 0.75, 0.75);
    glTexCoord2f(0.0, 1.0); glVertex3f(-0.75, 0.75, -0.75);

    glEnd();
}

void Render()
{
    Win.ClearWin();
    Quad();
    glutSwapBuffers();
}

```

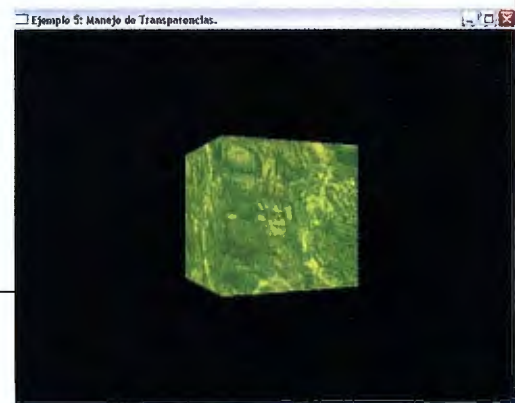


Figura No.36: Salida de programa

## Salida de Programa:

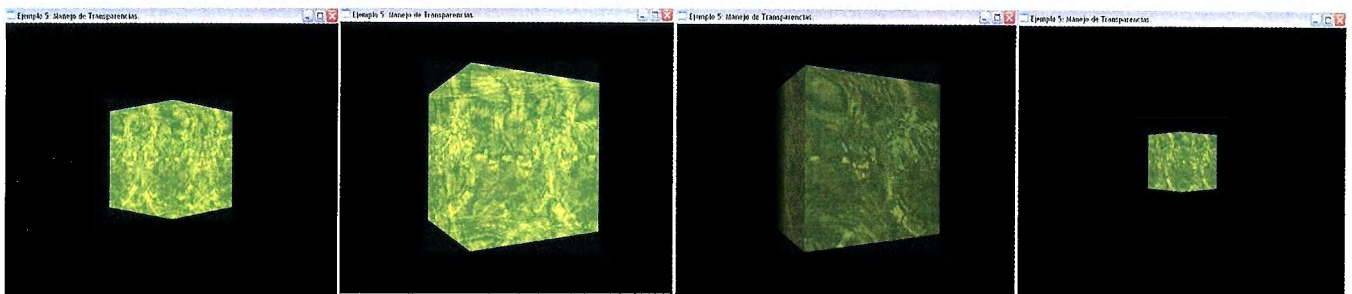
El programa permite una interacción con el teclado de la siguiente forma:

“L” para iluminación

“T” para activar la transparencia

“+” aumenta el tamaño del objeto

“-” disminuye el tamaño.



a) Con la tecla “T” aplicamos transparencia.

b) Con la tecla “+” aumentamos el tamaño del objeto.

c) Con la tecla “L” se apaga o activa la iluminación.

d) Con la tecla “-” disminuye el tamaño de la imagen

Figura No.37: Combinación de instrucciones del teclado en el objeto

5

ANIMACIÓN

---

## 5.1 Introducción.

La animación tridimensional<sup>1</sup> es una disciplina dentro de las artes digitales, que aprovecha las ventajas de la computación para enfrentar el enorme volumen de trabajo que implica el cuadro a cuadro característico de la animación tradicional.

Ésta se basa, en un principio parecido al de cuadros iniciales y finales elaborados previamente para dibujar luego los cuadros intermedios.

Pero en el caso de la animación 3D no se elaboran dibujos sino que se trata de instrucciones que el operador da a la computadora sobre la forma exacta de los objetos de una escena, su posición, el color, textura, grado de brillantez, reflexión, transparencia, refracción y rugosidad de las superficies; así como posición, intensidad, dirección, color y tipo de cada luz que afecta a la escena, lugar y dirección de cámara, tipo de lente, distancia focal, formato entre otros aspectos.

Se puede alterar todos estos datos desde el momento inicial hasta el fin de un intervalo. La forma de los objetos se determina por medio de vistas (frontal, lateral, superior, inferior), que se ven como planos de diseño industrial o arquitectónico.

Con esta información, la computadora calcula perspectiva, brillos, reflejos, sombras, y cambia las posiciones y características gradualmente para cada cuadro de animación de acuerdo a la información inicial y final de cada intervalo.

El operador controla cada característica y puede hacerla variar independientemente de las otras con respecto al tiempo. Finalmente, luego de estos cálculos la computadora elabora una imagen generalmente muy realista de cada cuadro de la escena y va archivando estas imágenes en sucesión, ya sea en

---

<sup>1</sup> Ilusión de movimiento al visionar una sucesión de imágenes fijas generadas por ordenador.

carpetas de cuadros numerados o como un documento de video digital. Este proceso se llama render.

Son necesarios 30 cuadros para un segundo de animación para video<sup>2</sup>. Un segundo de animación para cine necesita 24 cuadros. Una de las ventajas principales de la animación 3D es que mientras un dibujante podría tardarse horas o hasta días en terminar los cuadros de una toma pequeña una computadora puede hacer lo mismo en minutos.

Lo que más tiempo toma es introducir toda la información previa al render.

Sin embargo, la animación tridimensional también tiene sus desventajas, sobre todo porque en este método es difícil lograr soltura en los personajes, libertad de movimientos y expresividad facial y corporal, ya que todos los personajes y objetos son finalmente cuerpos geométricos que tienden a la rigidez matemática.

## 5.2 Conceptos Generales de Animación.<sup>3</sup>

Existen diferentes tipos de Animación, los más comunes son los siguientes:

1. **Fotograma<sup>4</sup> a Fotograma:** También llamado **Animación Off-Line**, es cuando los fotogramas se van generando a partir de una especificación previa y se van almacenando en ficheros (o en un fichero único para toda la animación) o en cualquier otro soporte analógico o digital.

Esta es la única forma de conseguir animaciones finales de gran calidad, dado que el tiempo de cómputo de cada fotograma puede oscilar entre unos pocos segundos y varias horas.

---

<sup>2</sup> Esto quiere decir que cada segundo pasan 30 fotografías frente a nuestros ojos.

<sup>3</sup> Buss Samuel R. 3D Computer Graphics: A Mathematical Introduction with OpenGL. Capítulo 12 Animations and Kinematics, páginas 289-290.

<sup>4</sup> Se denomina fotograma a cada una de las imágenes proyectadas a un plano de 24 cuadros por segundo que producen la ilusión de movimiento. Esto se debe a la incapacidad del cerebro de ver estas imágenes como fotografías separadas. Esta persistencia en la visión hace que el cerebro mezcle estas imágenes dando la sensación de movimiento natural.

- 2. Animación Interactiva:** También llamada **Animación On-Line:** Una situación diferente se produce cuando el proceso que crea la animación va mostrando los fotogramas inmediatamente después de producirlos, permitiendo al usuario responder inmediatamente y teniendo en cuenta estas respuestas en la síntesis de imágenes.

La frecuencia de presentación de las imágenes viene determinada por la velocidad de cómputo, que a su vez depende de la potencia del equipo y la complejidad de la escena.

- 3. Animación en Tiempo Real:** El tiempo de respuesta o Tiempo Real ( $T_r$ ) se define como aquel período de tiempo que transcurre entre la entrada de un dato y la obtención de una salida. En un sistema de simulación visual se representaría el tiempo que transcurre desde que el sujeto efectúa una acción (como mover el ratón) hasta que, después de efectuarse los cálculos correspondientes al modelo matemático del proceso y la visualización, se presentan las imágenes resultantes.

Una de las condiciones que podemos poner para hablar de tiempo real es limitar este parámetro a un valor máximo permitido (no queremos que haya un retraso mayor de un cierto intervalo), o incluso imponer que el tiempo de respuesta sea constante en todo momento.

A partir de estos conceptos, podemos considerar que una animación describe el cambio de una imagen a lo largo del tiempo, con el suficiente número de fotogramas por segundo para dar un efecto de continuidad.

Existen diversas técnicas que intentan conseguir este objetivo. A grandes rasgos podemos dividirlos en dos<sup>5</sup>:

**1. Técnica de Animación Clásica:** genera la secuencia de imágenes por métodos pictóricos, lo que entendemos por una imagen “dibujada”, formada por píxeles cuya coloración se asigna manualmente o semi-automáticamente, por mecanismos sencillos guiados de forma manual (por ejemplo, sistemas de relleno automático). No emplea ningún tipo de síntesis para conseguir efectos de profundidad y perspectiva, sino que es labor de los dibujantes conseguir estas sensaciones por técnicas manuales.

Las imágenes deben generarse una por una, aunque esta tarea suele distribuirse en varios niveles; separando el dibujo de momentos claves en la acción de los personajes, el dibujo de los fondos (que usualmente no cambian de un fotograma a otro) y las tareas de interpolación y coloreado de cada imagen

**2. Técnica de Animación de síntesis por ordenador:** Ésta a su vez, puede basarse en una representación 2D ó 3D de los objetos (aunque el resultado final será siempre, obviamente, bidimensional).

Crea las imágenes por un proceso automático a partir de una representación de los objetos que forman parte de la escena y de su movimiento.

Este modelo de los objetos puede ser bidimensional, con lo cual el resultado se parece más a la animación tradicional, o puede basarse en una representación 3D, con lo que pueden aplicarse métodos realistas de sombreado y simulación física.

---

<sup>5</sup> Eberly David H. 3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics Editorial Morgan Kaufmann. Capítulo 4 Hierarchical Effects Representations páginas 147-148.



## **5.3 TECNICA DE ANIMACIÓN DE SINTESIS POR ORDENADOR<sup>6</sup>**

### **Técnicas de Fotogramas Clave o keyframes:**

Este tipo de técnicas es una adaptación de los métodos de la animación tradicional para describir los cambios temporales en una escena. La idea básica consiste en definir mediante edición manual o mediante alguna función automática cuál es el estado de la escena en ciertos instantes (llamados 'fotogramas clave' o keyframes).

La descripción de la escena en cada fotograma clave debe incluir la posición y orientación de los objetos y fuentes de luz, sus propiedades, y también la posición y características del observador o cámara.

Se adapta así la idea de la animación tradicional en la que el dibujante experto traza en blanco y negro las siluetas de los personajes en los momentos clave, para que posteriormente los interpoladores se encarguen de generar las imágenes intermedias.

En la animación 3D, la interpolación en el tiempo se efectúa a partir de los valores que determinan los fotogramas clave (posiciones, orientaciones, velocidades y propiedades) mediante algún algoritmo automático.

Este proceso de interpolación es primordial, ya que debe producir un resultado coherente, de apariencia natural, sin que aparezcan saltos bruscos o cambios extraños (en algunos casos no es fácil realizar una interpolación correcta, por ejemplo en los movimientos de figuras articuladas como el cuerpo humano).

---

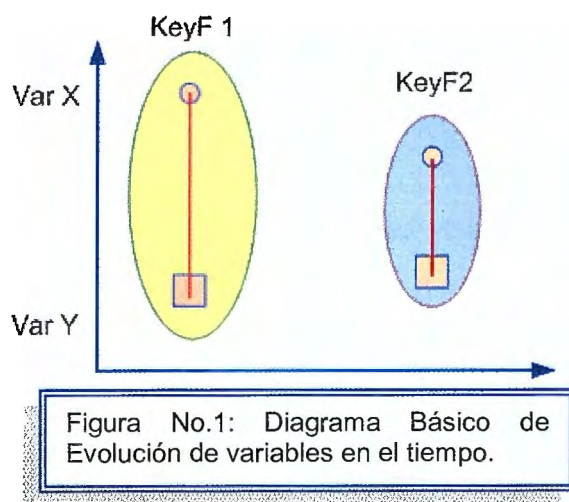
<sup>6</sup> David F Rogers, Rae A Earnshaw. Computer Graphics Techniques: Theory and Practice. Editorial Springer. Capítulo 4. Animations. Páginas 237-253

Para conseguir aproximarse más a los efectos deseados, el diseñador de la animación puede insertar mayor número de fotogramas clave (con menos separación temporal entre ellos).

Además de dar los valores que definen estáticamente cada fotograma clave, el diseñador puede también indicar cuál es el valor de ciertas variables dinámicas, lo que le otorgará un mayor grado de control sobre la interpolación.

La representación más formal de una animación por keyframes sería el llamado **Diagrama de Movimiento o Motion Graph**: una gráfica en la que aparecen los valores de cada una de las variables que definen la escena y su variación con el tiempo.

A partir de las bases que demuestra la figura No.1, se pueden formar otros diagramas que relacionan unas variables con otras. Por ejemplo, la trayectoria de un móvil, puede verse únicamente en función de sus componentes espaciales o a través de la relación de ambas coordenadas espaciales en el tiempo. Ver figura No.2.



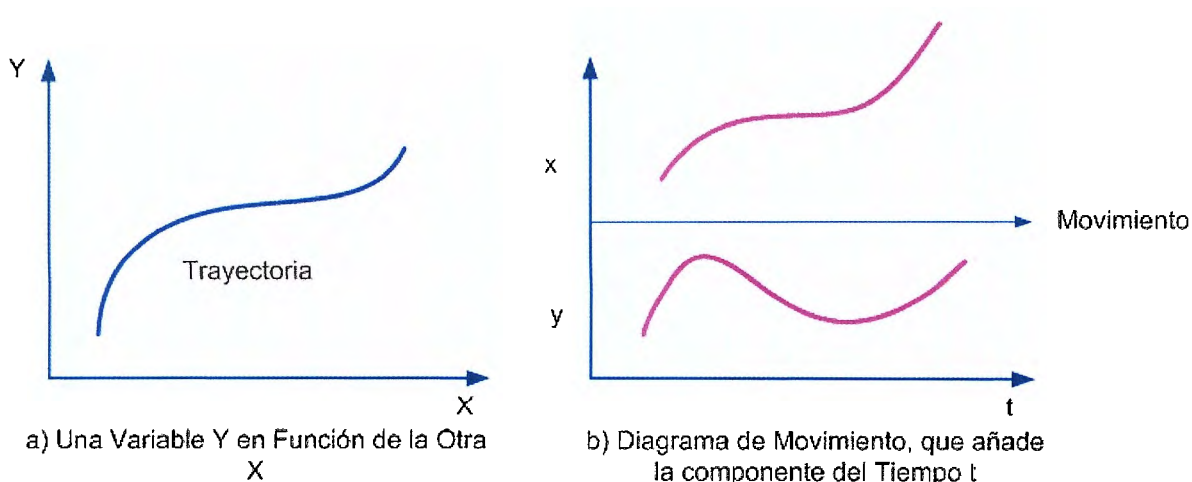


Figura No.2: Representación de la trayectoria + Diagrama de Movimiento.

Podríamos pensar que a partir de la representación espacial de la trayectoria podemos averiguar la velocidad del movimiento, observando la distancia recorrida entre dos keyframes. Pero esto no es cierto, ya que el objeto no tiene porqué moverse a velocidad constante sobre la trayectoria.

La velocidad instantánea real vendrá dada por las pendientes de la función de la posición en el diagrama de movimiento. Así, el vector velocidad instantáneo será:  $\left( \frac{dx}{dt}, \frac{dy}{dt} \right)$   
Y su módulo será la rapidez de traslación sobre la trayectoria.

Para construir los valores intermedios de las variables a partir de los keyframes necesitamos una función de interpolación con buenas propiedades y a la vez poco costoso de evaluar. Se suelen emplear curvas paramétricas, ya que tienen un comportamiento suave y son controlables.

Definamos algunas curvas paramétricas para comprobar cuáles pueden resultar más convenientes:

- **Splines:** La característica de las splines es que la curva resultante no tiene que pasar exactamente por los puntos de control, que en este caso son los valores de los keyframes. Esta característica hace que los keyframes dejen de representar de forma exacta la situación de la escena en ciertos instantes, lo que puede resultar inconveniente.

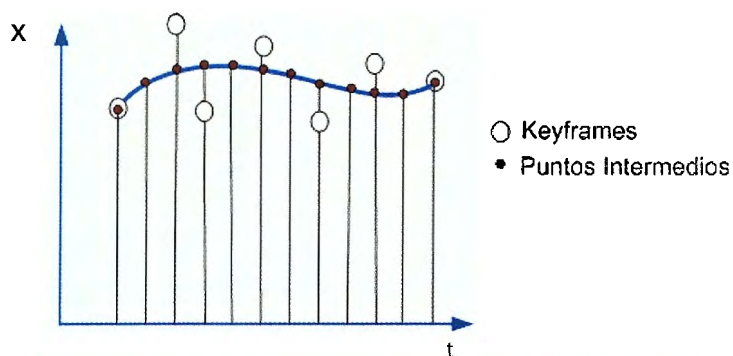


Figura No.3: Representación de Líneas Splines

- **Curvas cúbicas de Bèzier a trozos:** Estas curvas tienen la característica de que cada trozo empieza y termina en los puntos de control, pasando por los valores correspondientes a los keyframes. Además, si añadimos dos puntos de control auxiliares en cada tramo podremos controlar la tangente de la función, y por tanto determinar de forma más exacta el camino para ir de un punto a otro.

Por ejemplo, en el caso de que la variable representada sea una posición, este sistema nos permitiría controlar la velocidad del movimiento en el instante definido por el keyframe. Este tipo de curvas es, por tanto, ampliamente utilizado.

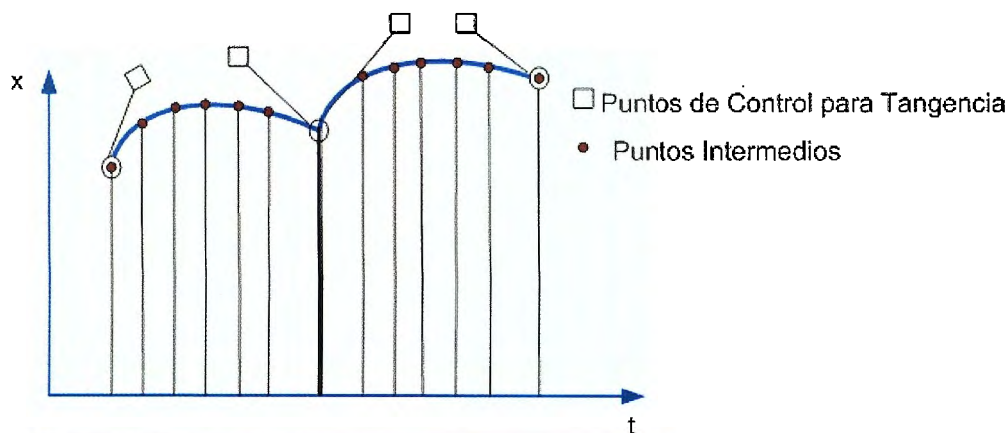


Figura No.4: Representación de Curvas cúbicas de Bézier

Además de estas curvas paramétricas también se puede emplear una simple interpolación lineal entre los valores de los keyframes. Esto permite resolver la interpolación mediante un cálculo muy sencillo, pero se trata de un tipo de ecuación que producirá cambios bruscos en la derivada de la variable y no permite un control detallado si no se añaden suficientes keyframes.

Además del tipo de interpolación, resulta fundamental la elección correcta de los parámetros a controlar. Por ejemplo, una interpolación lineal de la posición puede ser adecuada para una partícula que se mueve con una trayectoria suave (ver figura No.5 (a)). Sin embargo, si deseamos describir la rotación de un objeto alrededor de un punto, resultaría poco adecuado hacerlo con una interpolación lineal de la posición, siendo mejor hacerlo con una interpolación, lineal o no, sobre un ángulo que describe el giro (ver figura No.5 (b)).

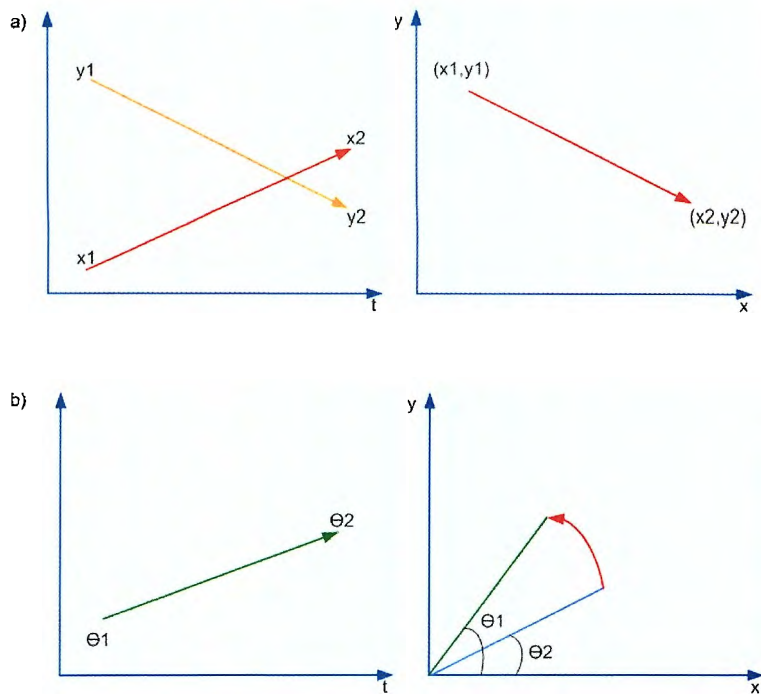


Figura No.5. a) Interpolación Lineal de Posición.  
b) Interpolación Lineal de ángulo

El método de fotogramas clave deja un problema abierto: cómo realizar la asignación de valores a las variables en los keyframes. Podemos resolverlo de distintos modos. Si disponemos de algún método algorítmico (por ejemplo una simulación basada en ecuaciones físicas) para calcular el valor de estas variables en el tiempo, entonces podríamos usarlo para calcular el estado de la escena en ciertos instantes y luego utilizar la interpolación entre keyframes.

Este sistema puede ser útil cuando el algoritmo exacto resulta demasiado costoso para utilizarlo en el cálculo de cada fotograma aislado.

## Manipulación Directa

La manipulación directa es el método más creativo para generar animaciones, asignando de forma manual (normalmente mediante un interfaz gráfico) los valores de las variables en cada keyframe.

Todos los programas de animación 3D proporcionan interfaces de manipulación directa que permiten realizar dos tipos de tareas distintas, que pueden combinarse entre sí para garantizar la coherencia del resultado:

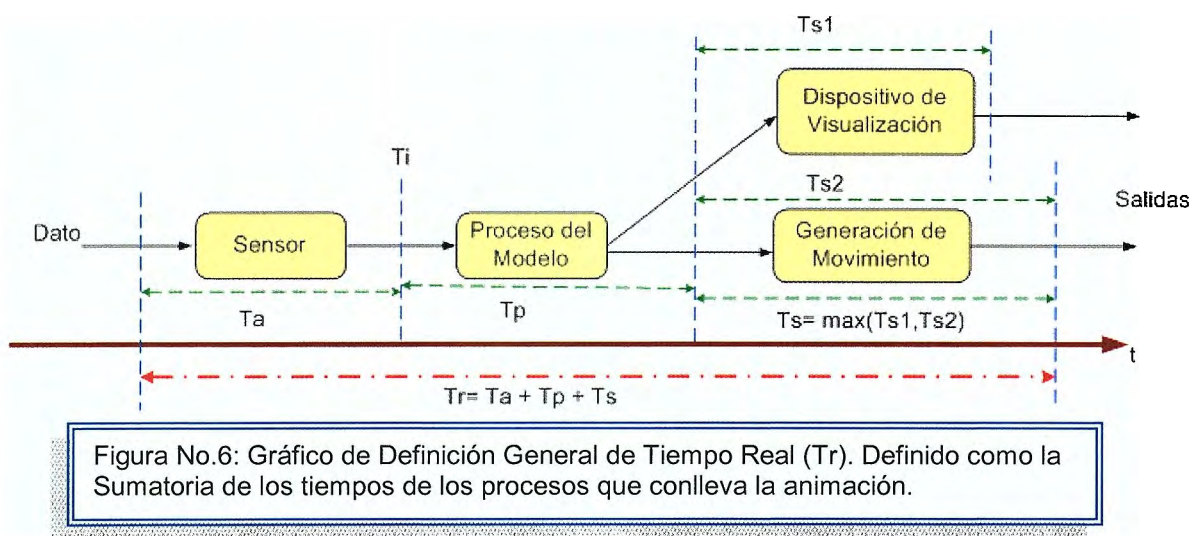
- a) Para determinar los keyframes, el interfaz deberá permitirnos dar la secuencia de valores para una variable dada. Para ello podremos manipular cada objeto y sus propiedades hasta colocarlo en el estado deseado y luego asignar este estado a un instante de tiempo (keyframe).
- b) Dada una asignación de valores a los diferentes keyframes, podremos representar la situación de la escena en cualquier instante, para comprobar si el movimiento de los distintos objetos se aproxima al efecto que queremos conseguir. Es decir, dispondremos de un modo de pre-visualización conjunta de la escena, al menos para instantes concretos de tiempo.

A pesar de su flexibilidad y potencia expresiva, este método manual plantea el problema de la dificultad para reproducir con naturalidad comportamientos de sistemas complejos donde intervienen actores (personas, animales u objetos animados) o sistemas que obedecen a leyes físicas.



## 5.4 ANIMACIÓN EN TIEMPO REAL<sup>7</sup>

Al hablar de Animación por computadora, entendemos que la principal característica de ésta es: **Simulación en Tiempo Real**, en donde la aplicación informática debe mostrar las imágenes sintéticas a medida que se van produciendo, de forma que reflejen los cambios producidos por las acciones del usuario sobre el programa, a esto también se le conoce como **concepto de interactividad**. Estos cambios suelen responder a la representación de un fenómeno real cuya evolución temporal se intenta replicar.



El concepto de Tiempo Real, se puede aplicar a distintas ramas de la Informática y proviene de la ingeniería de control, y en general se refiere a la existencia de determinadas restricciones sobre el comportamiento temporal de nuestro sistema. Dependiendo del tipo de aplicación, esas limitaciones serán de una clase u otra. Por tanto, puede aplicarse de forma más o menos estricta según lo duras que sean las restricciones impuestas.

<sup>7</sup> Glassner Andrew S Principles of Digital Image Synthesis. Editorial MK. Capítulo 4 Rendering. Páginas 1057-1058

Desde el punto de vista gráfico, esas restricciones, generan el principal desafío, que es optimizar el coste de los cálculos necesarios para realizar una visualización. Para conseguir dibujar varios fotogramas por segundo debe utilizarse casi obligatoriamente el método de procesamiento gráfico de proyección y transformación visto en las unidades 2 y 3 de ésta monografía.

Como veremos, normalmente las técnicas utilizadas para conseguir la disminución del coste se basan en controlar el número de objetos a visualizar, el número de polígonos que los forman y el coste de rellenar estos polígonos.

## **FACTORES DE COSTE EN VISUALIZACIÓN<sup>8</sup>**

Cada una de las fases en las que se divide la pipeline<sup>9</sup> o proceso de visualización tiene un coste asociado. Podemos hablar de tres tipos básicos de costes:

1. **Coste de las operaciones por vértice (proyección, iluminación, audio):** el coste será más o menos proporcional al número de vértices que se han enviado desde la base de datos a la escena, el número de luces y el modelo de iluminación. En principio el número de vértices es constante, pero podemos representar a los objetos con diferentes niveles de detalle (con diferente número de vértices y diferentes propiedades visuales) y escoger en cada momento el más conveniente.
2. **Coste de las operaciones por pixel (rellenado → comparación z-buffer, interpolación del color, texturas, transparencia):** es proporcional al número de pixels a rellenar y depende también de las características del objeto que afecten a las operaciones por pixel, como el modelo de iluminación (plano o Gouraud), si tiene textura, transparencia, entre otras.

---

<sup>8</sup> Brooker Darren. Essential Cg Lighting Techniques. Editorial Focal Press, Capítulo 13. In Production. Páginas 237-240

<sup>9</sup> Se denomina Píppelin: al proceso computacional que lleva desde la descripción de los objetos en la escena hasta la imagen final creadas en sistemas en tiempo real, mediante una serie de fases consecutivas y conectadas entre sí (la salida de datos de una fase constituye la entrada de la siguiente).

3. **Coste de las transferencias de datos entre diferentes partes de la pipeline, o de los accesos a memoria:** En muchos casos este coste no puede despreciarse, especialmente cuando la transferencia de datos se produce a través de un canal compartido por otros recursos del sistema (por ejemplo el bus de datos), o cuando el sistema debe hacer transferencias de páginas de memoria (por ejemplo, de texturas) es decir algoritmos básicos.

En gráficos animados en tiempo real, el coste final de todas las operaciones depende fuertemente de su implementación, y fundamentalmente del hardware encargado de ejecutarlas (razón por la cual, un sistema gráfico capaz de generar 60 Hz. con escenas complicadas puede valer varios millones de dólares).

Las técnicas de optimización del coste se basan en el control de las características de la escena relacionadas con el número de vértices y las operaciones por pixel, teniendo en cuenta además la arquitectura del sistema gráfico y sus limitaciones en la transferencia de datos.

Para conseguir esta disminución del coste de la representación gráfica se utilizan librerías de funciones que implementan técnicas especiales de visualización, y también existe la posibilidad, dadas las características de los procesos que efectúan, de utilizar hardware especial para acelerar los cálculos.

Las características que se pueden destacar en este tipo de sistemas son:

- Trabajan con primitivas poligonales. Los objetos 3D siempre serán representados como superficies para proceder a su visualización y, a su vez, las funciones de las librerías trabajaran internamente con objetos poligonales que se descompondrán en triángulos.

- Para permitir mayor grado de realismo se incorpora el pegado de texturas de color, normalmente bidimensionales.
- Se intenta que el mayor número posible de cálculos se resuelva mediante operaciones lineales, reducibles a sumas y multiplicaciones, y fácilmente sean implementadas por hardware.
- Se incluye algún tipo de método para eliminar superficies ocultas. Las dos alternativas más comunes son:
  - BSP-Trees para ordenar los polígonos por distancias y aplicar el algoritmo del pintor comenzando la visualización por los más lejanos.
  - Utilizar una memoria especial llamada Z-buffer que para cada pixel guarda la distancia a la que está el punto correspondiente del objeto tridimensional cercano de los dibujados en esa dirección hasta el momento.
- En estos métodos se utiliza, además de la memoria de pantalla (frame buffer) convencional, memoria adicional para permitir ciertas operaciones superponer diferentes tipos de información sobre cada pixel. El número de bits de información que se puede almacenar para cada píxel (número de bitplanes) constituye una característica fundamental.

Las técnicas de optimización del coste se basan en el control de las características de la escena relacionadas con el número de vértices y las operaciones por pixel, teniendo en cuenta además la arquitectura del sistema gráfico y sus limitaciones en la transferencia de datos.

Por tanto, no es tan importante controlar cómo es la descripción original de la escena, sino más bien cómo son los datos de la escena que se envían al sistema de procesamiento gráfico.

Algunas técnicas de optimización del coste son las siguientes:

## **1. Selección de Objetos según la Posición del Observador en Una Partición Espacial**

La escena puede contener muchos objetos en zonas alejadas o aisladas del entorno visible para el observador. Un ejemplo típico aparece en visualización de interiores arquitectónicos; hay muchas habitaciones, pero el observador solamente puede ver en cada momento aquellos objetos que están dentro de la sala donde él se encuentra.

Mediante cualquiera de los métodos de partición espacial estudiados podemos organizar los objetos en zonas y enviar al sistema de visualización solamente aquellos que están en las zonas accesibles. Esta forma de selección supone tener información a priori sobre la distribución de los objetos y las relaciones de visibilidad entre las diferentes zonas.

## **2. Comprobación de Visibilidad**

En general un objeto puede no ser visible desde la posición del observador por dos motivos:

- a) Bien porque se encuentre fuera del campo de visión, caso que es fácil comprobar sin un gran coste computacional;
- b) O bien porque, aún estando dentro del campo de visión, aparece totalmente obstruido por otro objeto.

En este segundo caso la comprobación de la condición de visibilidad supone un mayor coste, pues habría que proyectar la silueta del objeto que puede ser obstruido hasta el punto de observación y ver si es completamente tapada o no por otros posibles objetos.

En cualquiera de estos dos casos podríamos llegar a determinar que un objeto no va a ser visible, dejando de enviar sus datos a la pipeline gráfica. Si el sistema de visualización recibiera los datos de estos objetos 'invisibles' no llegaría nunca a rellenar los pixels, pero sí necesitaría proyectar sus vértices para saber si caen fuera o dentro de la ventana (como en la fase de recorte), y en el caso de un objeto obstruido (que sí cae dentro del campo de visión) también debería hacer la comparación de z-buffer con cada pixel cubierto para poder determinar su no-visibility. Por tanto, al dejar de enviar sus datos estamos evitando todas esas operaciones

### **3. Selección del Nivel de Detalle de los Objetos**

Cuando el coste total de la escena es excesivo, podemos intentar simplificarla, enviando al sistema de visualización representaciones menos detalladas de los objetos. Normalmente estas diferentes representaciones han sido generadas previamente. Lo lógico será representar con más detalle (mayor número de polígonos, mejor iluminación, texturas de más calidad) aquellos objetos que en cada instante se consideren más importantes para la percepción de la escena.

Como es claro, el criterio más empleado para elegir el nivel de detalle de cada objeto es la *distancia a la que se encuentra del punto de observación*, aunque también se pueden emplear otros como la velocidad transversal del objeto o su tamaño relativo, que pueden dar idea de su importancia dentro de la imagen.

## **Estructuras de Datos Jerárquicas para Visualización en Tiempo Real**

Algunas librerías gráficas para tiempo real definen estructuras de datos jerárquicas para organizar y optimizar la base de datos con el fin de disminuir el coste de visualización.

En general todas estas librerías contemplan mecanismos de selección de objetos, implementándolas gracias a funciones automáticas asociadas a los nodos de la estructura jerárquica<sup>10</sup>.

El algoritmo de visualización recorre esta estructura de forma recursiva a partir del nodo raíz y selecciona cuáles son los datos enviados al sistema de procesamiento gráfico. La razón para permitir que dos o más nodos puedan compartir hijos es ahorrar espacio de almacenamiento.

Si queremos dibujar varias copias del mismo objeto podemos colocarlo como hijo de varios nodos, con lo cual el algoritmo de recorrido llegará hasta él varias veces, por caminos diferentes y se dibujarán varias copias.

Los nodos de la jerarquía de la escena contienen información de diversos tipos, y ejercen diferentes funciones.

### **NODOS VISUALES**

Estos nodos contienen los datos que van a enviarse al sistema de procesamiento gráfico para dibujar los objetos.

A su vez pueden dividirse en dos tipos:

---

<sup>10</sup> Las estructuras de datos jerárquicas empleadas en visualización son grafos acíclicos (similares a un árbol pero permitiendo que dos nodos compartan hijos).



- a) **Nodos de geometría.** Contienen datos sobre la forma geométrica del objeto, utilizando primitivas poligonales y sus correspondientes vértices. Cada vértice suele estar caracterizado por la posición y otros parámetros opcionales como el valor del vector normal, el color o las coordenadas de textura.

Cuando el algoritmo de recorrido llega a un nodo de geometría, normalmente los nodos terminales del grafo, sus datos son enviados a la pipeline, y los polígonos correspondientes son dibujados.

Los nodos de geometría no suelen tener hijos, y a veces la librería gráfica lo prohíbe expresamente.

- b) **Nodos de propiedades.** Describen cómo debe alterarse alguna propiedad u opción visual del contexto gráfico cuando el algoritmo de recorrido llegue a ese nodo. Se utilizan, por tanto, para especificar materiales, texturas, el modelo de iluminación y otras variables del contexto gráfico que determinan la apariencia final de los objetos.

Por ejemplo, cuando el algoritmo de recorrido de la jerarquía llega a un nodo de propiedades que especifica el material, se cambiará el material actual en el contexto gráfico.

Lo habitual es que cuando el algoritmo haya recorrido toda la rama que cae bajo este nodo y siga el recorrido en otro lugar del árbol, se restaure en el contexto gráfico el material que hubiera antes. Pero esto no es siempre así. Por ello habrá que tener en cuenta para cada librería gráfica y para cada tipo de nodo si éstos tienen delimitado su alcance a sus hijos o su efecto es acumulativo (al volver de la recursión no se restaura el estado anterior).

En este último caso, más difícil de controlar, hay que tener en cuenta el orden de recorrido del grafo, que es usualmente primero en profundidad (depth-first).

## NODOS DE OPERACIONES

Estos nodos no contienen ningún dato sobre los objetos. Su misión es controlar el algoritmo de recorrido, es decir, seleccionar cuáles de sus hijos van a seguir recorriéndose. Son estos nodos los que van a permitir implementar las operaciones de optimización del coste.

Entre los nodos de operaciones tenemos:

- a) **Nodos de grupo.** Todos sus hijos se recorren. Tiene solamente la misión de agrupar varias ramas (objetos) para organizar la base de datos permitiendo definir operaciones o propiedades sobre estas agrupaciones cuando el grupo cuelga bajo los nodos apropiados.
- b) **Nodos de selección por visibilidad y nodos de partición.** El objetivo de estos nodos es obligar al algoritmo de recorrido a hacer una comprobación de visibilidad. En el caso de los nodos de selección simple, si esa comprobación falla (la parte de la escena no es visible) no se recorrerá ninguno de los hijos del nodo, y si es visible se recorrerán todos.

En el caso de los nodos de partición, éstos agrupan a sus hijos por zonas espaciales, y solamente se recorrerán aquellos que pertenezcan a zonas visibles. Solamente la comprobación de visibilidad dentro del campo de visión suele ser practicable en tiempo real. Para que esta comprobación pueda realizarse, el nodo de visibilidad o partición debe describir el volumen de la zona de la escena cuya visibilidad debe comprobarse.

Usualmente estos volúmenes envolventes (bounding box) son esféricos o en forma de caja rectangular.

Los nodos de visibilidad pueden anidarse, formando una jerarquía de volúmenes envolventes que permite refinar la comprobación con las partes de los objetos.

- c) **Nodos de nivel de detalle (LOD).** Estos nodos se utilizan para implementar la selección de detalle. Un nodo LOD forzará al algoritmo a efectuar una comprobación que escogerá uno solo de sus hijos para ser recorrido.

Se supone que cada hijo representa el mismo objeto pero con un diferente nivel de detalle. Lo más normal es emplear la distancia al observador.

Para efectuar la comprobación de distancia estos nodos necesitan almacenar cierta información adicional, como lo es:

- **Centro:** el punto desde el cual se mide la distancia del objeto al observador.
- **Rangos de distancia para cada hijo:** es decir, entre qué distancias es visible cada uno de los niveles de detalle. Normalmente se da una lista ordenada de valores de distancia, para los cuales se cambia de un hijo a otro (ver figura No.7).

El problema que presenta la técnica de selección de detalle es que, al variar la distancia, el observador podrá ver una transición brusca entre dos niveles de detalle.

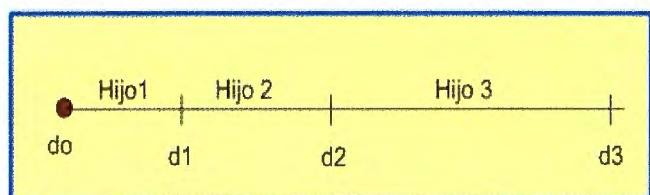


Figura No.7: Representación de un nodo LOD.

Para solucionarlo se pueden emplear dos técnicas:

1. **Fundido o fading** que realiza una mezcla progresiva de las dos imágenes correspondiente a los dos niveles de detalle vecinos, obteniéndose buenos resultados pero incrementando el coste de la visualización (momentáneamente se dibujan dos de los hijos en lugar de uno)
  2. **Morphing** en este caso no se trata de modificar una imagen sino la forma del objeto en tres dimensiones. Este cambio continuo puede utilizarse en animación para representar las deformaciones o crear efector visuales. Otra forma más técnica, consiste en suavizar las transiciones entre diferentes representaciones de un mismo objeto cuando éstas tienen diferente nivel de detalle.
- d) **Nodos de control por programa.** Cuando se desea realizar un cambio en la escena por otro criterio distinto de los anteriores (por ejemplo, porque la animación cambia el estado visual de un objeto) podemos utilizar un nodo de este tipo. El nodo tiene un campo que indica el índice del hijo que deberá ser recorrido, y este valor puede cambiarse por programa.

## 5.5 CONTROL DE FRECUENCIA DE REFRESCO

Otra característica de los sistemas gráficos de animación en tiempo real es la restricción de la frecuencia de refresco, que debe intentar mantenerse en unos límites adecuados.

Así, podemos ver el proceso de generación de imágenes como un ejemplo de problema de control, donde queremos obtener una salida con una cierta característica deseada, en nuestro caso el número de imágenes por segundo.

Las posibilidades de control aparecen desde el momento en que podemos variar, para la misma escena, los datos enviados a la pipeline gráfica, y por tanto ajustar el tiempo que se tardará en procesarlos. Usualmente esta modificación se efectúa haciendo variar el nivel de detalle de los objetos. (Ver figura No.8)

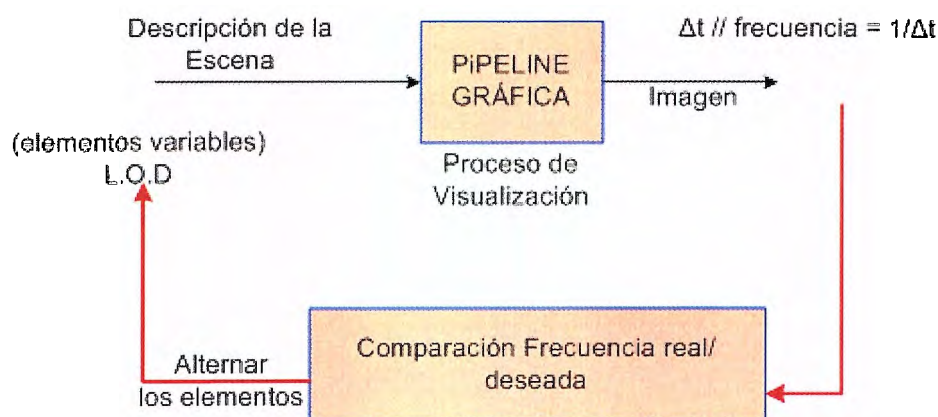


Figura No.8: Control de Frecuencia de Refresco

Como en todo proceso de control podemos enfocar el problema según dos métodos:

1. Control en lazo cerrado clásico aplicado al control de gráficos: Método Adaptativo del control de la frecuencia.
2. Control en lazo abierto: Método predictivo.

## 1. Método Adpatativo:

En lazo cerrado se compara el valor producido por el sistema con el valor que deseamos obtener, y se intenta hacer las modificaciones adecuadas sobre la entrada del módulo controlado.

Se define un **factor de estrés (f.e)** como un número real positivo:

$$(ec1) \quad f.e = \frac{f. \text{ Deseada}}{f. \text{ Real}} \quad \begin{cases} f.e > 1 \text{ frecuencia real menor que la deseada (sobrecarga)} \\ f.e < 1 \text{ frecuencia real mayor que la deseada} \end{cases}$$

Para obtener la frecuencia de refresco deseada, el factor de estrés debería ser exactamente uno, pero esto es una situación ideal.

Si sólo deseamos poner un valor mínimo a la frecuencia, basta con requerir que el estrés sea menor o igual que uno. En el bucle de retroalimentación debemos utilizar el valor del estrés para cambiar el nivel de detalle de los objetos.

Las variables de distancia aplicados por el algoritmo de recorrido serán:

$$(ec2) \quad d_i' = \frac{d_i}{f.e}$$

Cuando haya sobrecarga del sistema gráfico, el factor de estrés será mayor que uno, y las distancias efectivas resultarán más pequeñas (ver figura No.9), con lo que probablemente disminuirá el nivel de detalle de los objetos en la escena, ya que para una misma distancia se escogerán intervalos con índice igual o más alto, y por tanto con menor detalle.

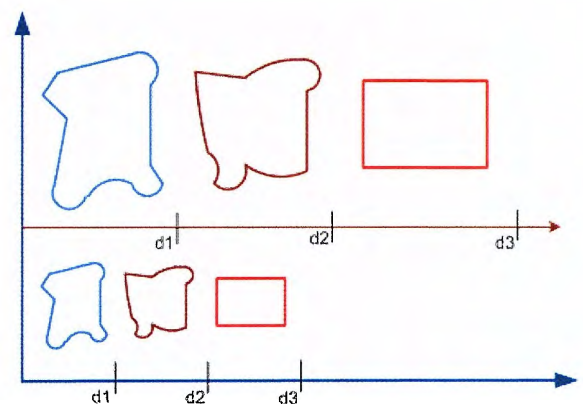


Figura No. 9: Niveles de detalle de un objeto según la variación de los rangos de distancia.

A la inversa, si el sistema tiene recursos sobrados y el estrés es menor que uno, aumentarán los rangos de distancia y por tanto, el detalle.

Como vemos, se trata de una técnica relativamente sencilla y fácil de implementar. Su principal ventaja es que no conlleva ningún tipo de dependencia respecto a las características específicas de cada equipo gráfico.

Sin embargo, no resulta fácil obtener un control exacto de la frecuencia de refresco, debiendo permitir un equilibrio entre el retraso de la adaptación y las oscilaciones de la frecuencia, cuya razón explicamos a continuación, y que es común a otros sistemas de control.

Para medir la frecuencia real de refresco debe contarse el tiempo transcurrido desde que se envían los datos gráficos correspondientes a un fotograma hasta que el fotograma es completamente dibujado.

Si solamente utilizamos el tiempo transcurrido en el último fotograma, la frecuencia real sufrirá oscilaciones bruscas, ya que el observador va moviéndose, y hay objetos que aparecen, desaparecen o cambian de detalle. El sistema será incapaz de adaptarse a una frecuencia que varía tan rápidamente y sufrirá fácilmente oscilaciones, puesto que se alterará el nivel de detalle, y eso producirá un cambio inmediato en la frecuencia, que puede provocar a continuación el cambio opuesto.

Por otra parte, podemos calcular la frecuencia real haciendo la media de tiempos entre los últimos intervalos de procesamiento. De esa manera evitaremos oscilaciones bruscas, pero tardaremos un cierto tiempo en responder a cambios estables en la frecuencia. Por ejemplo, si el observador entra en una zona de la escena donde aparecen bruscamente muchos objetos, aumentará súbitamente el tiempo requerido para la visualización, pero hasta que no hayan transcurrido algunos fotogramas, ese cambio no hará variar significativamente la media, y por tanto no se adaptará la frecuencia de refresco. Por consiguiente, hay que buscar



un valor de compromiso para el número de fotogramas utilizados en el cálculo de la frecuencia.

## 2. Método Predictivo

Se trata de un control en lazo abierto, es decir obtener la salida deseada sin utilizar su valor real.

La manera de conseguir directamente una frecuencia de refresco determinada es alimentar el procesamiento gráfico con datos cuyo coste de computación es conocido y se corresponde con la frecuencia de salida deseada.

Para poder calcular el coste que representa cada nivel de detalle de los objetos y seleccionar los más apropiados, se define:

$$\forall \text{ Objeto } i \left\{ \begin{array}{l} C_{ij} \text{ Coste de la visualización para cada nivel de detalle } j \text{ de } i \\ b_{ij} \text{ Beneficio Visual} \end{array} \right.$$

El beneficio visual se corresponde con la 'importancia' del objeto en la visualización y puede evaluarse a partir de su distancia, área aparente y otros criterios.

Es más difícil de obtener el coste de visualización, ya que no es tarea sencilla predecir lo que le va a costar a la máquina visualizar un cierto objeto. Para simplificar, este coste se suele calcular como una combinación del coste previsible por vértice y del coste previsible por pixel, proporcional al área aparente del objeto.

## 5.6 ALGORITMOS BÁSICOS DE ANIMACIÓN

El algoritmo de animación es un bucle sin fin en el que se dibuja un objeto, se retrasa la ejecución del proceso durante un período de tiempo y a continuación se dibuja de nuevo el objeto con el color de fondo para conseguir el efecto de borrado y se actualiza el valor de las coordenadas de acuerdo a sus incrementos o decrementos.

Se debe de comprobar que los nuevos valores de las coordenadas no sobrepasan los límites de la pantalla. En caso de que así sea, se actualizan para crear el efecto visual de rebote.

Para conseguir un efecto más llamativo, hay un conjunto de datos numéricos cuyos valores iniciales no tienen por qué ser fijos sino que cualquier valor en un determinado rango es válido.

En general las técnicas para algoritmos de animación más básicas son:

1. **Fotogramas Claves o keyframes:** en este caso, la escena se detiene en ciertos instantes. Dichos keyframes se interpolan a partir de posiciones, orientaciones y propiedades con una función de interpolación que puede ser: curvas paramétricas, splines ó curvas Beziér.

Para ello, se define un diagrama de movimiento, que almacena los valores de las variables de la escena y su variación en el tiempo.

Sus características principales son:

- Interpolación + keyframes
- Totalmente Controlable
- No es interactivo

2. **Animación Procedural y Basada en Reglas:** Los métodos procedurales, definen la evolución del conjunto de parámetros por medio de una expresión paramétrica en función del tiempo. En este caso se hace mucho énfasis en la animación física en el que se describen movimientos y otros cambios reproduciendo con cierta fidelidad la realidad física.

Los algoritmos basados en reglas, según su propio nombre, éstos definen un conjunto de reglas que determinan el valor de los parámetros en cada instante de tiempo.

Sus características principales son:

- Sigue las leyes de nuestro macro-mundo o mundo real
- Los cálculos se hacen en tiempo real
- Es interactivo

3. **Lenguajes script o guiones:** En este caso se aplican tanto Animación Procedural como Fotogramas Claves. La unión de ambas técnicas generan un algoritmo más complicado, pero animaciones de mayor calidad.

Sus características principales son:

- Imitación de los fenómenos físicos
- Comportamiento Plausible
- Mucho Realismo (Depende de la Rapidez de Cálculo y restricciones de estabilidad)

En openGl, la clave para que la animación funcione, es que cada frame esté completo, es decir que hay terminado de renderizar. Un pseudocódigo ejemplo puede ser:

```
Abre_ventana (){  
  For (i=0; i< ultimo_frame; i++){  
    Borra_ventana ();  
    Dibuja_frame(i);  
    Espera_hasta_1/24_parte_de_segundo(); /*recuerdese que en el momento en  
    que el ojo humano percibe 24 frames por segundo, el cerebro lo interpreta como  
    un movimiento real*/  
  }  
}
```

Si el borrado de pantalla y el dibujado del frame tardan más de 1/24 de segundo, antes de que se pueda dibujar el siguiente, el borrado ya se habrá producido. Esto puede causar un parpadeo en la pantalla ya que no hay sincronización de tiempos. Para solucionar este problema OpenGL, nos permite usar la clásica técnica del doble buffering; las imágenes renderizadas se van colocando en el primer Buffer y cuando termina el renderizado, se vuelca el segundo buffer, que es el que se dibuja en pantalla. Eso hace que nunca veamos una imagen cortada y por lo tanto el parpadeo de pantalla es eliminado.

Por lo tanto el pseudocódigo se establece de la siguiente forma:

```
Abre_ventana (){  
  For (i=0; i< ultimo_frame; i++){  
    Borra_ventana ();  
    Dibuja_frame(i);  
    swap_buffers();  
  }  
}
```

La cantidad de frames siempre quedará limitada por el refresco del monitor. Aunque OpenGL es capaz de rederizar 100 frames por segundo, si el periférico utilizado solo alcanza para 60 hz, es decir 60 imágenes por segundo, aproximadamente 40 frames renderizados se perderán.

Las librerías auxiliares utilizadas en OpenGL, son las siguientes:

- GLU: primitivas 3D, operaciones con matrices y otros factores matemáticos
- GLUT: tratamiento de ventanas, ratón, teclado y otros periféricos.
- GLAUX: igual que GLUT, pero sólo para Windows.
- GLX: para un sistema X-Windows de Linux.
- AGL, PGL, WGL: como GLX para Apple Macintosh, IBM OS/2 Warp y Windows NT y 95.
- OpenAL, OpenNL, OpenIN: para audio, red o entrada respectivamente.

## EJEMPLOS UNIDAD 5:

**Ejemplo 5.1:** Este ejemplo muestra un programa desarrollado con ayuda de OpenGL, animando un objeto con red 3D, girando además en el plano z.

```
#if defined(_WIN32)
#include <windows.h>
#endif

#include <GL/gl.h>
#include <GL/glut.h>

#include <io.h>
#include <stdio.h>
#include <math.h>

// Prototipos de las funciones para poder llamarlas
void Display();
void Keypressed(unsigned char key, int x, int y);
void Reshape(GLsizei width, GLsizei height);
void Animate();
void DrawAxis();
void DrawIcosahedron();

// ID de la ventana
static int win;

// Rotacion de la ventana
const GLfloat cRX = 30.0;
const GLfloat cRY = -30;
static GLfloat RX = cRX;
static GLfloat RY = cRY;

// Tamaño de la tetera
static float size = 0.30;

// Traslacion
static GLfloat tx = 0.0;
static GLfloat ty = 0.0;
static GLfloat tz = 0.0;

// Rotation
static GLfloat spin = 0.0;
const GLfloat rx = 1.0;
const GLfloat ry = 0.0;
```

```

const GLfloat rz = 0.0;

static int IcosahedronFactor = 0;

int main(int argc, char **argv)
{
    // Inicializa glut
    glutInit(&argc, argv);

    // Especifica el tipo de display y que es double buffer
    // Se usa porque esto es animado
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);

    // define el tamaño
    glutInitWindowSize(500,500);

    // la posicion de la ventana en la pantalla
    glutInitWindowPosition(100, 100);

    // crea la ventana con el id de la misma
    win = glutCreateWindow("Tetera");

    // callbacks
    glutDisplayFunc(Display);
    glutReshapeFunc(Reshape);
    glutKeyboardFunc(Keypressed);

    //clearscreen
    glClearColor(0.0, 0.0, 0.0, 0.0);

    glClearDepth(1.0);
    glEnable(GL_DEPTH_TEST);

    // Proyeccion ortogonal
    glMatrixMode(GL_PROJECTION_MATRIX);
    glLoadIdentity();
    glOrtho(-1.0, 1.0, -1.0, 1.0, -2.0, 2.0);

    glFrontFace(GL_CW);
    glEnable(GL_CULL_FACE);
    glCullFace(GL_BACK);

    glPolygonMode(GL_FRONT, GL_LINE);
    glPolygonMode(GL_BACK, GL_LINE);

    // animaciones

```



```

        glutIdleFunc(Animate);

        // Entra en el ciclo principal
        glutMainLoop();

        return 0;
    }

void Display()
{
    static unsigned short pattern = 0xAAAA;
    // hace un clearscreen
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

    // rota todo
    glMatrixMode(GL_MODELVIEW_MATRIX);
    glLoadIdentity();

    glRotatef(RX, 1.0, 0.0, 0.0);
    glRotatef(RY, 0.0, 1.0, 0.0);

    // dibuja el eje
    DrawAxis();

    DrawIcosahedron();

    // dibuja la tetera
    glPushMatrix();
    glTranslatef(tx, ty, tz);
    glRotatef(spin, 0.0, 0.0, 1.0);
    glScalef(1.0, 2.3, 1.0);
    glColor3f(0.5, 0.5, 0.5);
    glutWireTeapot(size);
    glPopMatrix();

    glutSwapBuffers();
}

void DrawAxis()
{
    const float arrow = 1.0 / 10.0;
    const float base = 1.0 / 30.0;

    // eje x
    glColor3f(1.0, 0.0, 0.0);

```

```

glBegin(GL_LINE_STRIP);
glVertex3f(-1.0, 0.0, 0.0);
glVertex3f(1.0, 0.0, 0.0);
glEnd();
glPushMatrix();
glTranslatef(1.0 - arrow, 0.0, 0.0);
glRotatef(90.0, 0.0, 1.0, 0.0);
glutSolidCone( base, arrow, 20, 1);
glPopMatrix();

```

```

// eje y
glColor3f(0.0, 1.0, 0.0);
glBegin(GL_LINE_STRIP);
glVertex3f(0.0, -1.0, 0.0);
glVertex3f(0.0, 1.0, 0.0);
glEnd();
glPushMatrix();
glTranslatef(0.0, 1.0 - arrow, 0.0);
glRotatef(-90.0, 1.0, 0.0, 0.0);
glutSolidCone( base, arrow, 20, 1);
glPopMatrix();

```

```

// eje z
glColor3f(0.0, 0.0, 1.0);
glBegin(GL_LINE_STRIP);
glVertex3f(0.0, 0.0, -1.0);
glVertex3f(0.0, 0.0, 1.0);
glEnd();
glPushMatrix();
glTranslatef(0.0, 0.0, 1.0 - arrow);
glutSolidCone( base, arrow, 20, 1);
glPopMatrix();

```

```

}

```

```

void drawtriangle(float *v1, float *v2, float *v3)
{
    glBegin(GL_POLYGON);
        glVertex3fv(v1);
        glVertex3fv(v2);
        glVertex3fv(v3);
    glEnd();
}

```

```

void normalize(float v[3]) {

```

```

GLfloat d = sqrt(v[0]*v[0]+v[1]*v[1]+v[2]*v[2]);
if (d == 0.0) {
    throw "zero length vector";
    return;
}
v[0] /= d; v[1] /= d; v[2] /= d;
}

void subdivide(float *v1, float *v2, float *v3, long depth)
{
    GLfloat v12[3], v23[3], v31[3];
    GLint i;

    if (depth == 0) {
        drawtriangle(v1, v2, v3);
        return;
    }
    for (i = 0; i < 3; i++) {
        v12[i] = v1[i] + v2[i];
        v23[i] = v2[i] + v3[i];
        v31[i] = v3[i] + v1[i];
    }
    normalize(v12);
    normalize(v23);
    normalize(v31);
    subdivide(v1, v12, v31, depth-1);
    subdivide(v2, v23, v12, depth-1);
    subdivide(v3, v31, v23, depth-1);
    subdivide(v12, v23, v31, depth-1);
}

void DrawIcosahedron()
{
    #define X .525731112119133606
    #define Z .850650808352039932

    static GLfloat vdata[12][3] = {
        {-X, 0.0, Z}, {X, 0.0, Z}, {-X, 0.0, -Z}, {X, 0.0, -Z},
        {0.0, Z, X}, {0.0, Z, -X}, {0.0, -Z, X}, {0.0, -Z, -X},
        {Z, X, 0.0}, {-Z, X, 0.0}, {Z, -X, 0.0}, {-Z, -X, 0.0}
    };

    static GLint tindices[20][3] = {
        {0,4,1}, {0,9,4}, {9,5,4}, {4,5,8}, {4,8,1},
        {8,10,1}, {8,3,10}, {5,3,8}, {5,2,3}, {2,7,3},
        {7,10,3}, {7,6,10}, {7,11,6}, {11,0,6}, {0,1,6},
        {6,1,10}, {9,0,11}, {9,11,2}, {9,2,5}, {7,2,11} };

```

```

    glColor3f(0.0, 1.0, 1.0);
    glPushMatrix();
    glScalef(0.5, 0.5, 0.5);
    for (int i = 0; i < 20; i++) {
        /* Informacion del color aqui */
        subdivide(&vdata[tindices[i][0]][0],
                 &vdata[tindices[i][1]][0],
                 &vdata[tindices[i][2]][0], IcosahedronFactor);
    }
    glPopMatrix();
}

void Keypressed(unsigned char key, int x, int y)
{
    switch(key)
    {
        case 'a':
            tx = tx - 0.05;
            break;
        case 'd':
            tx = tx + 0.05;
            break;
        case 'r':
            ty = ty + 0.05;
            break;
        case 'f':
            ty = ty - 0.05;
            break;
        case 'w':
            tz = tz - 0.05;
            break;
        case 's':
            tz = tz + 0.05;
            break;
        case 'z':
            tx = 0.0;
            ty = 0.0;
            tz = 0.0;
            RX = cRX;
            RY = cRY;
            break;
        case 'i':
            RX = RX - 1;
            break;
        case 'k':

```



```

        RX = RX + 1;
        break;
    case 'j':
        RY = RY + 1;
        break;
    case 'l':
        RY = RY - 1;
        break;
    case '+':
        IcosahedronFactor = min(IcosahedronFactor + 1,
7);
        break;
    case '-':
        IcosahedronFactor = max(IcosahedronFactor - 1,
0);
        break;
    case 27:
        glutDestroyWindow(win);
        exit(0);
        break;
    }
    printf("%d", key);
    printf("x:%f y:%f z:%f \n", tx, ty, tz);
}

void Reshape(GLsizei width, GLsizei height)
{
    glViewport(width / 2 - 250, height / 2 - 250, 500, 500);
}

void Animate()
{
    spin = spin + 1;
    if(spin > 360)
        spin = spin - 360;
    glutPostRedisplay();
}

```

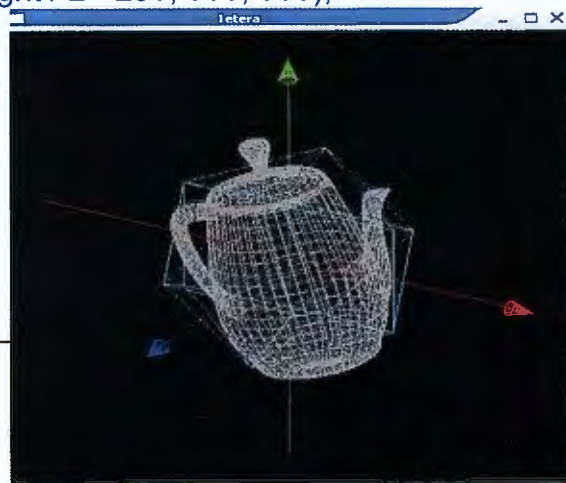


Figura No. 10: Salida de programa:  
Animación de tetera con OpenGL y  
rotación en el eje z

**Ejemplo 5.2:** Animación de cámaras con OpenGL, este ejemplo muestra la animación de entornos de acercamiento y alejamiento de visualización.

```
#include <GL/glut.h>
#include <iostream.h>
#include "CWin.h"
#include "CCamera.h"
#include "CImage.h"

#pragma comment(lib,"opengl32.lib")
#pragma comment(lib,"glut32.lib")
#pragma comment(lib,"glu32.lib")
#pragma comment(lib,"glaux.lib")

#define ID_TEXTURE 0

CWin Win;
CImage Image;
CCamera Camera(2.0f,1.0f,-6.0f,0.0f,1.0f,0.0f);
unsigned int Texture[1];

void Key(unsigned char Tecla,int x,int y);
void Render();
void Terrain();
void Special(int iKey, int x, int y);
void InitOpenGL();

int main(int argc, char **argv)
{
    Win.InitWin(argc,argv);
    Win.CreateWin(640,480,"Ejemplo 5_2: Manejo de camara.",false);
    InitOpenGL();
    glutKeyboardFunc(Key);
    glutDisplayFunc(Render);
    glutSpecialFunc(Special);
    glutMainLoop();
    return 0;
}

void Special(int iKey, int x, int y)
{
    switch(iKey){
        case GLUT_KEY_UP:
            Camera.Move(MOVE_FORWARD,0.30f);
            break;
```

```

        case GLUT_KEY_DOWN:
            Camera.Move(MOVE_BACKWARD,0.30f);
            break;
    }
    glutPostRedisplay();
}
void Key(unsigned char Tecla,int x, int y)
{
    switch(Tecla){
        case 27: // exit (0);
            break;
    }
}

void InitOpenGL()
{
    cout << "Oprime <Esc> para salir" << endl;
    cout << "<Flecha de arriba> se mueve hacia adelante." << endl;
    cout << "<Flecha de abajo> se mueve hacia abajo." << endl;

    if(!Image.CreateTexture(Texture,ID_TEXTURE,"textura.bmp")){
        cerr << "Error al cargar la textura" << endl;
        exit(1);
    }
    glClearColor(0.4f, 0.4f, 1.0f, 0.0f);
    glShadeModel(GL_SMOOTH);
    glEnable(GL_TEXTURE_2D);
    glEnable(GL_DEPTH_TEST);
    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
}

void Terrain()
{
    int x, z = 0;
    Camera.SetLook();
    glBindTexture(GL_TEXTURE_2D,Texture[ID_TEXTURE]);
    for(int i = 0; i < 10; i++){
        for(int j = 0; j < 10; j++){
            glBegin(GL_QUADS);
            glTexCoord2f(0.0, 0.0); glVertex3f(15.0f + x, -0.01f, 15.0f + z);
            glTexCoord2f(1.0, 0.0); glVertex3f(10.0f + x, -0.01f, 15.0f + z);
            glTexCoord2f(1.0, 1.0); glVertex3f(10.0f + x, -0.01f, 10.0f + z);
            glTexCoord2f(0.0, 1.0); glVertex3f(15.0f + x, -0.01f, 10.0f + z);
            glEnd();
            x += -5;
        }
        x = 0;
    }
}

```



```
    z += -5;
}
}

void Render()
{
    Win.ClearWin();
    Terrain();
    glutSwapBuffers();
}
```

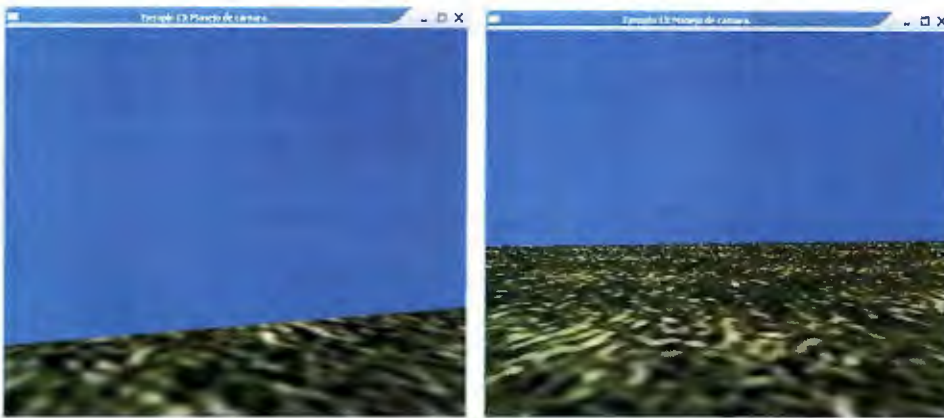


Figura No.11: Salida de Programa: movimiento de cámara con alejamiento y acercamiento.

**Ejemplo 5.3:** Animación de cámaras con OpenGL, siguiendo las especificaciones del ejemplo anterior, éste muestra la visualización con rotación, es decir movimientos de: Alejamiento, acercamiento, vista derecha, izquierda, atrás y adelante.

```
#include <GL/glut.h>
#include <string.h>
#include <assert.h>
#include <stdarg.h>
#include <stdio.h>
#include "CWin.h"
#include <windows.h>
#include <iostream.h>
#include "CCamera.h"
#include "CImage.h"
#pragma comment(lib,"opengl32.lib")
#pragma comment(lib,"glut32.lib")
#pragma comment(lib,"glu32.lib")
#pragma comment(lib,"glaux.lib")

#define ID_TEXTURE 0

CWin Win;
CImage Image;
CCamera Camera(2.0f,1.0f,-6.0f,
               0.0f,1.0f,0.0f,
               0.0f,1.0f,0.0f);
unsigned int Texture[1];

void Key(unsigned char Tecla,int x,int y);
void Render();
void Terrain();
void Special(int iKey, int x, int y);
void InitOpenGL();

//inicializa glut

void CWin::InitWin(int argc,char **argv)
```

```

{
    glutInit(&argc,argv);
}

//Crea nuestra ventana
void CWin::CreateWin(int iWidth,int iHeight,const char *Title,bool bFullScreen)
{
    glutInitDisplayMode(GLUT_RGB | GLUT_RGBA | GLUT_DOUBLE |
    GLUT_DEPTH);
    if(bFullScreen){
        glutInitWindowPosition(100,100);
        glutInitWindowSize(iWidth,iHeight);
        glutCreateWindow(Title);
        glutFullScreen();
        ChangeSizeWin(iWidth,iHeight);
        strTitle=new char[strlen(Title)+1];
        assert(strTitle!=0);
        strcpy(strTitle,Title);
        Width=iWidth;
        Height=iHeight;
    }
    else{
        glutInitWindowPosition(100,100);
        glutInitWindowSize(iWidth,iHeight);
        glutCreateWindow(Title);
        ChangeSizeWin(iWidth,iHeight);
        strTitle=new char[strlen(Title)+1];
        assert(strTitle!=0);
        strcpy(strTitle,Title);
        Width=iWidth;
        Height=iHeight;
    }
}

//Devuelve el titulo
char *CWin::GetCaption()
{
    return strTitle;
}

//Devuelve el tamaño de la ventana
void CWin::GetSizeWin(int &iWidth,int &iHeight)
{
    iWidth =Width;
    iHeight=Height;
}

```



```

}

//Limpia el buffer de profundidad
void CWin::ClearWin()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
}

//Cambia el tamaño de la ventana
void CWin::ChangeSizeWin(int iWidth,int iHeight)
{
    if (iHeight==0){ //Previene una division entre cero
        iHeight=1;
    }
    glViewport(0,0,iWidth,iHeight);    //Ajusta la vista a las dimensiones de la
    ventana
    glMatrixMode(GL_PROJECTION);    //Activa la matriz de proyeccion
    glLoadIdentity();    //Reinicia el sistema de coordenadas
    gluPerspective(45.0f,(GLfloat)iWidth/(GLfloat)iHeight, 1.0f ,1000.0f);
    glMatrixMode(GL_MODELVIEW);    //Activa la matriz del modelador
    glLoadIdentity();    //Reinicia el sistema de coordenadas
}

void CWin::SetCaption(const char *strText)
{
    glutSetWindowTitle(strText);    //Asignamos el titulo a la ventana
}

int main(int argc, char **argv)
{
    Win.InitWin(argc,argv);
    Win.CreateWin(640,480,"Ejemplo 14: Manejo de camara. (Rotacion)",false);
    InitOpenGL();
    glutKeyboardFunc(Key);
    glutDisplayFunc(Render);
    glutSpecialFunc(Special);
    glutMainLoop();
    return 0;
}

```

```

void Special(int iKey, int x, int y)
{
    switch(iKey){
        case GLUT_KEY_UP:
            Camera.Move(MOVE_FORWARD,0.4f);
            break;
        case GLUT_KEY_DOWN:
            Camera.Move(MOVE_BACKWARD,0.4f);
            break;
        case GLUT_KEY_RIGHT:
            Camera.Rotate(-0.025f, 0.0f, 1.0f, 0.0f);
            break;
        case GLUT_KEY_LEFT:
            Camera.Rotate(0.025f, 0.0f, 1.0f, 0.0f);
            break;
    }
    glutPostRedisplay();
}

void Key(unsigned char Tecla,int x, int y)
{
    switch(Tecla){
        case 27: exit (0);
                break;
    }
}

void InitOpenGL()
{
    cout << "Oprime <Esc> para salir" << endl;
    cout << "<Flecha de arriba> se mueve hacia adelante." << endl;
    cout << "<Flecha de abajo> se mueve hacia abajo." << endl;
    cout << "<Flecha hacia la derecha> rota hacia la derecha" << endl;
    cout << "<Flecha hacia la izquierda> rota hacia la izquierda" << endl;

    if(!Image.CreateTexture(Texture,ID_TEXTURE,"textura.bmp")){
        cerr << "Error al cargar la textura" << endl;
        exit(1);
    }
    glClearColor(0.4f, 0.4f, 1.0f, 0.0f);
    glShadeModel(GL_SMOOTH);
    glEnable(GL_TEXTURE_2D);
    glEnable(GL_DEPTH_TEST);
    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
}

```



```

}

void Terrain()
{
    int x, z = 0;
    Camera.SetLook();

    glBindTexture(GL_TEXTURE_2D, Texture[ID_TEXTURE]);
    for(int i = 0; i < 10; i++){
        for(int j = 0; j < 10; j++){
            glBegin(GL_QUADS);
            glTexCoord2f(0.0, 0.0); glVertex3f(15.0f + x, -0.01f, 15.0f + z);
            glTexCoord2f(1.0, 0.0); glVertex3f(10.0f + x, -0.01f, 15.0f + z);
            glTexCoord2f(1.0, 1.0); glVertex3f(10.0f + x, -0.01f, 10.0f + z);
            glTexCoord2f(0.0, 1.0); glVertex3f(15.0f + x, -0.01f, 10.0f + z);
            glEnd();
            x += -5;
        }
        x = 0;
        z += -5;
    }
}

void Render()
{
    Win.ClearWin();
    Terrain();
    glutSwapBuffers();
}

```

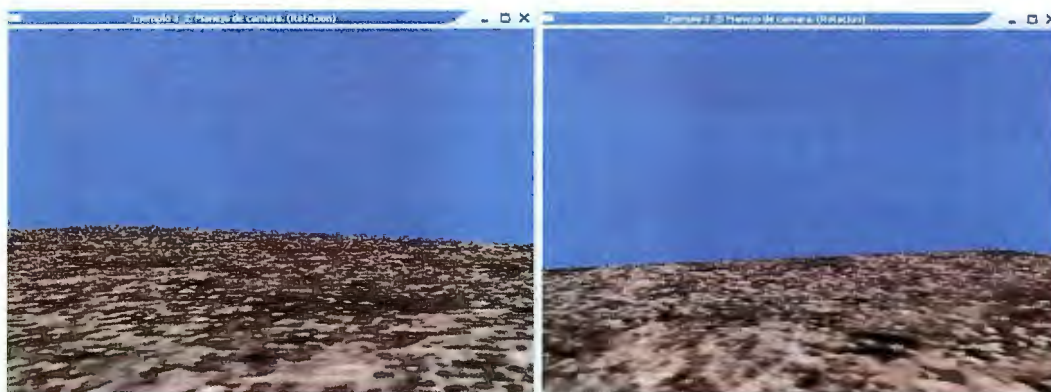


Figura No.12: Salida de Programa: movimiento de cámara con alejamiento, acercamiento, y rotación

### Ejemplo 5.4:

```
#include <vcl.h>
#pragma hdrstop

#include "TPoligonos.h"

TCara::TCara()
{
    IdVerPol = new int[MAXVPC];
    IdVerTex = new int[MAXVPC];

    //Marcado como no introducido...
    for (int i=0; i < MAXVPC; i++)
    {
        IdVerPol[i] = NVISIT;
        IdVerTex[i] = NVISIT;
    }

    NumVerPol = 0;
    NumVerTex = 0;
}

void TCara::AddVerticePol(const int &x)
{
    IdVerPol[NumVerPol] = x;
    NumVerPol++;
}

void TCara::AddVerticeTex(const int &x)
{
    IdVerTex[NumVerTex] = x;
    NumVerTex++;
}

int TCara::ObtVerticePol(const int &x)
{
    return IdVerPol[x];
}

TCara::~TCara()
{
    delete[] IdVerPol;
    delete[] IdVerTex;
}

TMaterial::TMaterial()
```



```

{
    Archivo = "";
    Nombre = "";
    pBitmap = NULL;
}

void TMaterial::PonArchivo(const AnsiString &x)
{
    Archivo = x;
}

AnsiString TMaterial::ObtArchivo()
{
    return Archivo;
}

void TMaterial::PonNombre(const AnsiString &x)
{
    Nombre = x;
}

AnsiString TMaterial::ObtNombre()
{
    return Nombre;
}

void TMaterial::PonRGB(const int &u, const int &v, const int &w)
{
    RR = u;
    GG = v;
    BB = w;
}

void TMaterial::ObtRGB(int &u, int &v, int&w)
{
    u = RR;
    v = GG;
    w = BB;
}

void TMaterial::Activar()
{
    glEnable(GL_TEXTURE_2D);
    glColor3ub(255, 255, 255);
    glBindTexture(GL_TEXTURE_2D, Tx);
}

```

```

void TMaterial::Abrir()
{
    if(Archivo.IsEmpty()) return;

    pBitmap = auxDIBImageLoad(Archivo.c_str());

    glGenTextures(1, &Tx);

    glPixelStorei (GL_UNPACK_ALIGNMENT, 1);

    glBindTexture(GL_TEXTURE_2D, Tx);

    gluBuild2DMipmaps(GL_TEXTURE_2D, 3, pBitmap->sizeX, pBitmap->sizeY,
GL_RGB, GL_UNSIGNED_BYTE, pBitmap->data);

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

    if (pBitmap)
    {
        if (pBitmap->data)
        {
            free(pBitmap->data);
        }
        free(pBitmap);
    }
    // Free the bitmap structure
}

TMaterial::~TMaterial()
{
}

TObjeto::TObjeto()
{
    //Se crea...
    Ver = new TVertice [MAXVER];
    UV = new TUV [MAXVER];
    Car = new TCara * [MAXCAR];
    Mat = new TMaterial * [MAXMAT];

    //Todo a NULL...
    for (int i=0; i < MAXCAR; i++) Car[i] = NULL;
    for (int i=0; i < MAXMAT; i++) Mat[i] = NULL;

    //Nada dentro...
}

```



```

        NumVer = 0;
        NumMat = 0;
        NumCar = 0;
        NumUV = 0;

        Textura = false;
    }

void TObjeto::AddCara(TCara * Cr)
{
    Car[NumCar] = Cr;
    NumCar++;
}

void TObjeto::AddVertice(const TVertice &x)
{
    Ver[NumVer] = x;
    NumVer++;
}

void TObjeto::AddMaterial(TMaterial * Tx)
{
    Mat[NumMat] = Tx;
    NumMat++;
}

void TObjeto::PonNombre(const AnsiString &x)
{
    Nombre = x;
}

AnsiString TObjeto::ObtNombre()
{
    return Nombre;
}

int TObjeto::ObtNumVertices()
{
    return NumVer;
}

int TObjeto::ObtNumCaras()
{
    return NumCar;
}

```

```

int TObjeto::ObtNumMateriales()
{
    return NumMat;
}

int TObjeto::ObtNumCoordenadasUV()
{
    return NumUV;
}

int TObjeto::ObtSector()
{
    return Sector;
}

TMaterial * TObjeto::ObtMaterial(const int &i)
{
    return Mat[i];
}

void TObjeto::PonTextura(const bool &i)
{
    Textura = i;
}

void TObjeto::PonMaterial(const int &i)
{
    Material = i;
}

void TObjeto::Establecer_Vertices(TVertice *P, const int &L)
{
    if (Ver != NULL) delete Ver;
    Ver = P;
    NumVer = L;
}

void TObjeto::Establecer_UV(TUV *P, const int &L)
{
    if (UV != NULL) delete UV;
    UV = P;
    NumUV = L;
}

void __fastcall TObjeto::Compilar()
{
    glNewList(1, GL_COMPILE);

```



```

    if (Textura)
    {
        Mat[Material]->Activar();
    }
    else
    {
        glDisable(GL_TEXTURE_2D);
        glColor3ub(255, 255, 255);
    }

    glBegin(GL_TRIANGLES);

    for(int j = 0; j < NumCar; j++)
    {
        for(int whichVertex = 0; whichVertex < 3; whichVertex++)
        {
            int index = Car[j]->ObtVerticePol(whichVertex);

            glNormal3f(Nor[ index ].x, Nor[ index ].y, Nor[ index ].z);

            if (Textura)
            {
                glTexCoord2f(UV[ index ].u, UV[ index ].v);
            }
            else
            {
                if (Material >= 0)
                {
                    BYTE RR, GG, BB;
                    Mat[Material]->ObtRGB(RR, GG, BB);
                    glColor3ub(RR, GG, BB);
                }
            }

            glVertex3f(Ver[ index ].x, Ver[ index ].y, Ver[ index ].z);
        }
    }

    glEnd();

    glEndList();
}

void __fastcall TObjeto::Draw()
{
    glCallList(1);
}

```

```

}

void __fastcall TObjeto::Agua()
{
    int Fila = 0;
    float Y,Ant,Act;
    float Anc = 0.2f;

    if (Textura)
    {
        Mat[Material]->Activar();
    }
    else
    {
        glDisable(GL_TEXTURE_2D);
        glColor3ub(255, 255, 255);
    }

    Y = 180 / sqrt((NumCar / 2));

    glBegin(GL_TRIANGLES);

    for(int j = 0; j < NumCar; j++)
    {
        //Calcula la fila...
        if ((j % 2) == 0)
        {
            Fila = j / ((sqrt((NumCar / 2)))*2);

            for(int whichVertex = 0; whichVertex < 3; whichVertex++)
            {
                int index = Car[j]->ObtVerticePol(whichVertex);

                glNormal3f(Nor[ index ].x, Nor[ index ].y, Nor[ index ].z);

                if (Textura)
                {
                    glTexCoord2f(UV[ index ].u, UV[ index ].v);
                }
                else
                {
                    if (Material >= 0)
                    {
                        BYTE RR, GG, BB;
                        Mat[Material]->ObtRGB(RR, GG, BB);
                    }
                }
            }
        }
    }
}

```



```

        glColor3ub(RR, GG, BB);
    }
}

    if (Fila == 0)
    {
        Ant = Ver[ index ].y;
    }
    else
    {
        Ant = Ver[ index ].y+ (sin(((Fila-1)*Y)+Ang)*Anc);
    }
    if ((j%2)==0)
    {
        if (whichVertex > 0)
            Act = Ver[ index ].y + (sin((Fila*Y)+Ang)*Anc);
        else
            Act = Ant;
    }
    else
    {
        if (whichVertex == 2)
            Act = Ver[ index ].y + (sin((Fila*Y)+Ang)*Anc);
        else
            Act = Ant;
    }

    glVertex3f(Ver[ index ].x, Act, Ver[ index ].z);
}

}

glEnd();
Ang += 0.1;
}

```

```

void __fastcall TObjeto::CalcularNormales()
{
    TVector vVector1, vVector2, vNormal, vPoly[3];

    TVector *pNormals = new TVector [NumCar];
    TVector *pTempNormals = new TVector [NumCar];
    Nor = new TVector [NumVer];

    for(int i=0; i < NumCar; i++)
    {
        vPoly[0].x = Ver[Car[i]->ObtVerticePol(0)].x;
        vPoly[0].y = Ver[Car[i]->ObtVerticePol(0)].y;
        vPoly[0].z = Ver[Car[i]->ObtVerticePol(0)].z;
    }
}

```



```

        vPoly[1].x = Ver[Car[i]->ObtVerticePol(1)].x;
        vPoly[1].y = Ver[Car[i]->ObtVerticePol(1)].y;
        vPoly[1].z = Ver[Car[i]->ObtVerticePol(1)].z;
        vPoly[2].x = Ver[Car[i]->ObtVerticePol(2)].x;
        vPoly[2].y = Ver[Car[i]->ObtVerticePol(2)].y;
        vPoly[2].z = Ver[Car[i]->ObtVerticePol(2)].z;

        vVector1 = Vector(vPoly[0], vPoly[2]);
        vVector2 = Vector(vPoly[2], vPoly[1]);

        vNormal = Cross(vVector1, vVector2);
        pTempNormals[i] = vNormal;
        vNormal = Normalize(vNormal);

        pNormals[i] = vNormal;
    }

    TVector vSum = {0.0, 0.0, 0.0};
    TVector vZero = vSum;
    int shared=0;

    for (int i = 0; i < NumVer; i++)
    {
        for (int j = 0; j < NumCar; j++)
        {
            if (Car[j]->ObtVerticePol(0) == i ||
                Car[j]->ObtVerticePol(1) == i ||
                Car[j]->ObtVerticePol(2) == i)
            {
                vSum = AddVector(vSum, pTempNormals[j]);
                shared++;
            }
        }

        Nor[i] = DivideVectorByScaler(vSum, float(-shared));

        Nor[i] = Normalize(Nor[i]);

        vSum = vZero;
Reset the sum
        shared = 0;
Reset the shared
    }

    delete [] pTempNormals;
    delete [] pNormals;
}

```

```

bool TObjeto::ObtTextura()
{
    return Textura;
}

void __fastcall TObjeto::AbrirMateriales()
{
    for (int i = 0; i < NumMat; i++)
    {
        if (!Mat[i]->ObtArchivo().IsEmpty()) Mat[i]->Abrir();
    }
}

TObjeto::~TObjeto()
{
    //Borra el contenido...
    for (int i = 0; i < NumMat; i++)
        if (Mat[i] != NULL) delete Mat[i];
    //for (int i = 0; i < NumVer; i++)
    //    if (Ver[i] != NULL) delete Ver[i];
    for (int i = 0; i < NumCar; i++)
        if (Car[i] != NULL) delete Car[i];

    //Borra la tabla...
    delete[] Mat;
    delete[] Ver;
    delete[] Car;
    delete[] UV;
}

```



Figura No.13: Salida de programa: Barco movimiento con opengl.

6

**GRAFICADO EN WEB CON  
SWIFT 3D**

---

## 6.1 Introducción a Swift 3D v4.

A lo largo de los capítulos anteriores se estudió la manera de cómo realizar imágenes en dos y tres dimensiones, algoritmos, color, animación.

Teniendo ya los conocimientos sobre lo que es el graficado, solo se debe dejar volar la imaginación y ponerlos en práctica.

El Software SWIFT 3D; Es una herramienta que permite la capacidad de crear y desarrollar gráficos profesionales 3D, exportación 3D a Flash sobre formatos SWF, y crear animaciones en WEB.

El entorno de Swift, como se ve en la figura No 1, es sencillo de utilizar, contiene una variedad de barras de herramientas y un entorno de trabajo multifuncional, ya que el usuario tiene la capacidad de combinar vistas según lo requirieran los objetos a crear.



Figura No. 1 Interfase de área de trabajo Swift



Los elementos principales de la barra de herramientas son:



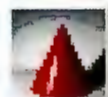
Crear Nuevo documento



Crear Piramide



Abrir un documento



Crear cono



Grabar un documento



Crear cilindro



Crear texto



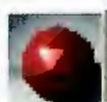
Crear toroide



Crear circulo



Crear plano



Crear superficie esferica



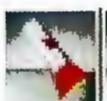
Crear poliedro



Crear cubo



Crear punto de luz



Adición de luces puntuales



Crear punto de luz  
orientado

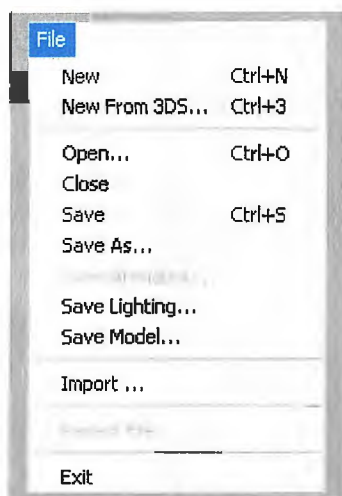


Crear una vista de cámara



Enfoque de la vista de cámara

Parte de los conocimientos básicos es identificar los menús y las opciones que estos ofrecen:



**New:** crear Nuevo proyecto.

**New from 3d:** nueva forma 3d.

**Open:** abrir proyecto.

**Close:** Cerrar aplicación actual.

**Save:** guardar archivo activo.

**Save as:** guardar con otro nombre.

**Save animation:** guardar animación.

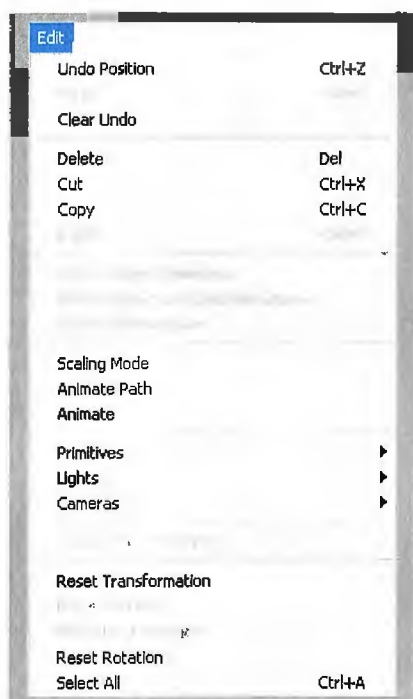
**Save lighting:** guardar enfoque de iluminación

**Save model:** guardar modelo.

**Import:** abre un proyecto \*.eps.

**Recent file:** archivo reciente.

**Exit:** salir del programa.



**Undo position:** deshacer la acción anterior.

**Redo:** rehacer

**Clear undo:** limpiar las acciones realizadas.

**Delete:** Elimina la pieza anterior.

**Cut:** cortar

**Copy:** copiar.

**Paste:** pegar.

**Delete object:** borrar solo el objeto seleccionado.

**Delete object and animation:** borra el objeto seleccionado y la animación de éste.

**Delete all animation:** elimina todas las animaciones pero no los objetos.

**Scaling mode:** cambia de escala.

**Animate path:** esta es opción es arbitraria, ya que pueden existir animaciones como archivos predeterminados.

**Animate:** activa la animación del objeto.

**Primitives:** muestra las diferentes formas basicas.

**Light:** selección de luces.

**Camera:** posiciona un enfoque en un punto determinado

**Convert tex to path:** esta opción permanece siempre inactiva.

**Reset transformation:** regresa a la configuración por default de las opciones de transformación

**Reset position:** regresa las opciones default de la configuración de posicionamiento del objeto.

**Reset pivot location :** resetea pivote de localización.

**Reset rotation:** resetea rotacion.

**Select all:** seleccionar todo.



**Status Bar:** Muestra la barra de estado.

**Property Tools:** Muestra la paleta de Propiedades.

**Trackball Tools:** Muestra el inspector para mover el objeto.

**Gallery Tools:** Muestra la paleta de animaciones y colores.

**Lighting Tools:** Muestra el Inspector de luminosidad.

**Edit Tools:** Muestra la barra de iconos.

**Animation Time Line:** Muestra la barra de tiempo.

**Zoom View Port:** Diferentes porcentajes para el zoom.



**Stop Playing Animation:** Para la animación.

**Play Animation:** Corre la Animación.

**First Frame:** Va al primer frame de la animación.

**Last Frame:** Va al ultimo frame de la animación.

**Previous Frame:** Va al frame anterior.

**Next Frame:** Va al frame siguiente.

**Loop:** hace que la animacion se repita.



**Animation:** Te permite organizar las animaciones y cambiarles el nombre.

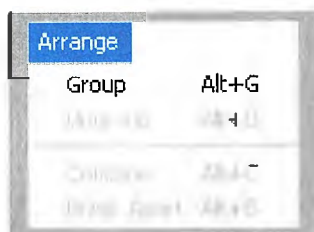
**Materials:** Te permite crear nuevos materiales como organizar el existente, cambiar el nombre y la categoría

**Enviroments :** ambiente.

**Lighting :** luces.

**User prefernces :** usar preferencias.



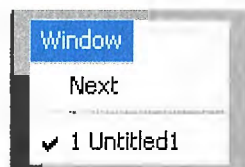


**Group:** Teniendo dos o mas objetos puedes hacer que sean una misma unidad.

**Ungroup:** Teniendo objetos agrupados, te permite desagruparlos.

**Combine:** agrupa varios objetos seleccionados

**Break Apart:** particiona varios objetos seleccionados



**Next:** Cambia de proyecto, saltando de una ventana a la otra.



**Help Context:** La ayuda del programa.

**User guide :** guia de uso en archivo pdf.

**About Swift 3D:** Sobre los creadores y tal.

**Register Swift 3D on Line:** Registrate tu swift original Online

**Electric Rain Web Site:** Pagina principal de Electric Rain.

**Technical Support Site:** Pagina de soporte tecnico del programa

**Swift 3D CustomersS Only Syte:** Pagina de ayuda, solo para los registrados

### 6.1.1 Objetos en Swift.

Cuando se crea un nuevo documento se debe especificar determinados colores y estilos; En Swift3d uno de ellos es el color de fondo aunque normalmente no es muy importante debido a que cuando se importa solo se toma en cuenta el objeto. Otro factor es la luz ambiente, Esta afecta la animación ya que modifica el aspecto de todo el objeto.

La manera de realizarlos es seleccionar la pestaña de Enviroment (entorno) y determinar el color deseado así como lo muestra la figura 2

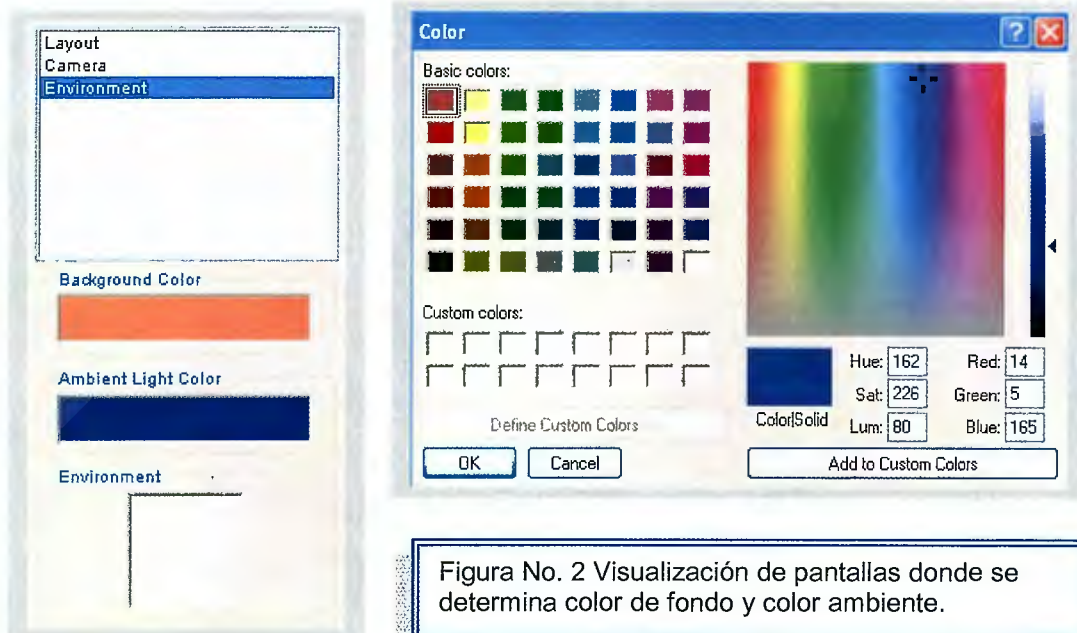
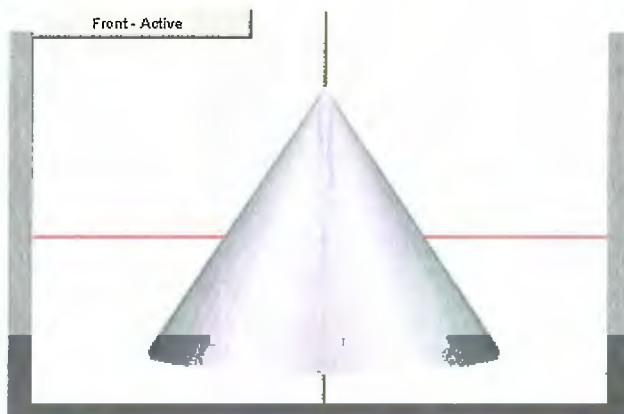


Figura No. 2 Visualización de pantallas donde se determina color de fondo y color ambiente.

Conociendo un poco el entorno de Swift se pueden crear objetos dentro de una escena y crear sus características propias, la manera de hacerlo es seleccionar una de las figuras de la barra de herramientas esta se puede posicionar donde el usuario lo desee.

Por ejemplo un cono



El cual es posible modificar desde el menú propiedades donde ofrece las opciones de escalas, posición, rotación, así como lo refleja la figura No. 6

Figura No.5: Representación gráfica de un cono, como imagen predeterminada

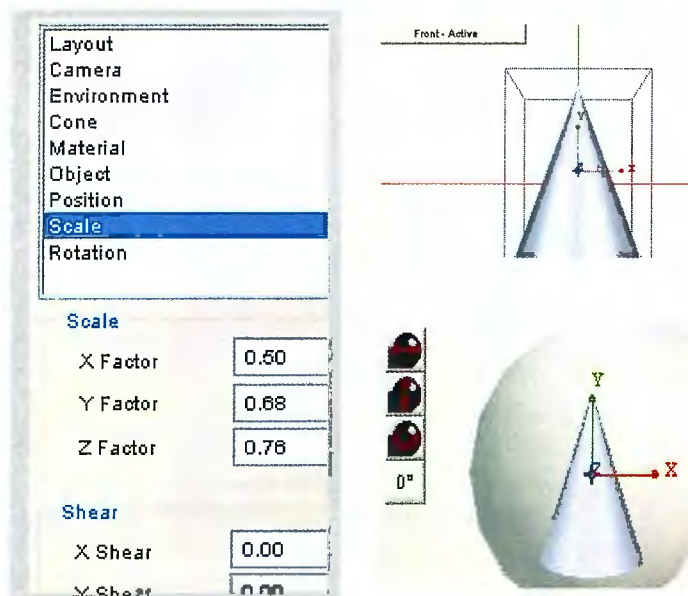


Figura No. 6 Se muestran las propiedades de la figura al igual que la modificación de escala y la esfera de control.

Existen muchas opciones de lo que se puede realizar, como agregar texto, con las opciones de la barra de herramientas, duplicar figuras con el menú de edición utilizando copiado y pegado según lo muestra la figura No. 7 y también modificarlas dentro de la pantalla de propiedades.

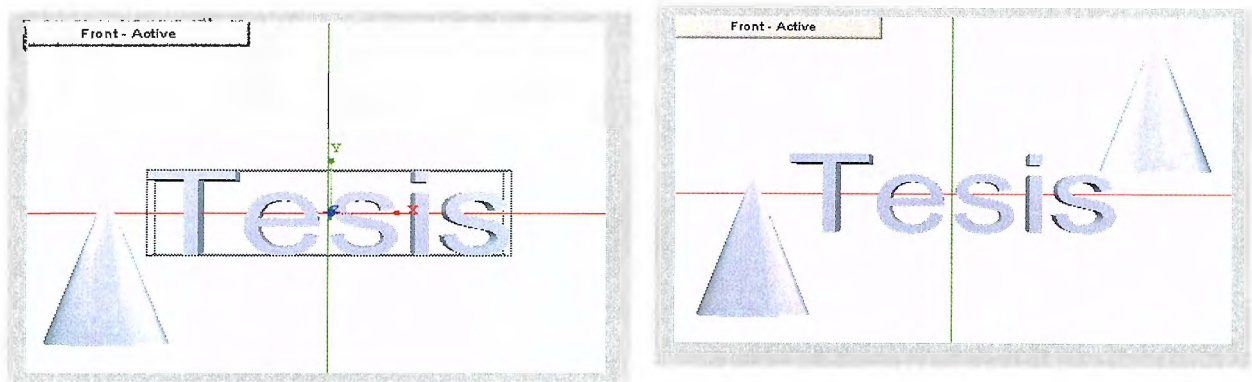
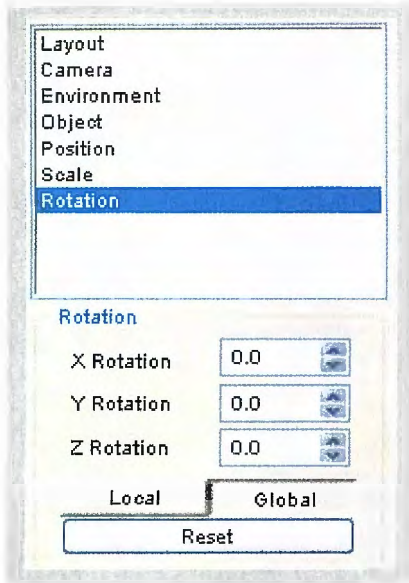


Figura No. 7 Muestra un texto agregado y la duplicación de la figura .

Para modificar el objeto creado se debe agrupar cada uno de los elementos que se les quiera aplicar rotación o movimientos, esto se hace seleccionando un objeto de la escena, luego selecciona el menú Arrange donde se escoge la opción Group, o también la tecla shift y seleccionando los otros objetos.

Para luego poderle dar un efecto de rotación según lo muestra la Figura No. 8



Recuadro que Muestra todos los objetos agrupados

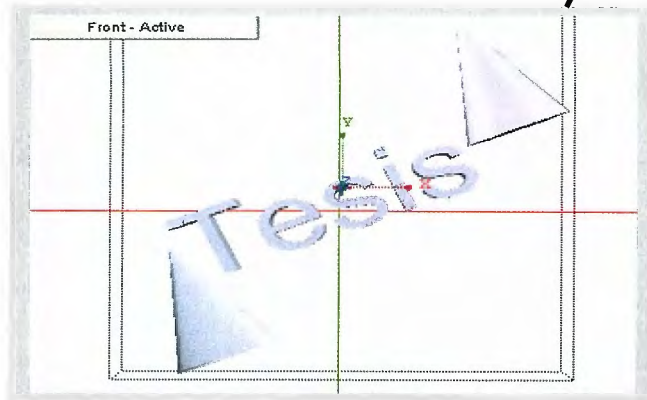


Figura No.8 muestra la agrupación y la rotación de la escena.

Cada vez que se quiera realiza una acción a la escena se deben seleccionar los objetos y agruparlos para afectarlos a todos.

Una de las partes más importantes de toda escena es escoger colores, texturas, transparencias, tonos, color, efectos de animaciones, y de más.

Esto se logra seleccionando en la galería.



Imagen No.9: Librería de Colores, texturas y animación.



Donde:

**Layout:** (propiedades del fondo) elegimos el tamaño del área de trabajo, el zoom de la Cámara y distintos tipos de visualización: Objetos completos o solo líneas y guías.

**Enviroment:** Color del fondo y de la iluminación de ambiente o fondo de trabajo.

**Text:** aca podemos modificar las propiedades del texto como el tipo de letra o la alineación del texto.

**Bevel:** Aca cambiamos las características 3D del objeto donde Style son los distintos tipos de estilos de biseles y Depth es la profundidad de los mismos.

**Sizing:** Aca podemos cambiar Alto (X), Ancho (Y), Profundidad (Z), Espacio entre Caracteres y Espacio entre Líneas de Texto.

**Materials:** Sirve para elegir el color y el brillo del objeto. Está dividido en tres sectores: **All faces**, que es para cambiar las propiedades de las caras; **All Bevels**: que es para cambiar el color de los relieves o biseles y **All Edges** es para la profundidad.

La manera de cambiar las propiedades es la siguiente. En el extremo inferior derecho de la pantalla se encuentra la librería de materiales y colores separados por grupos (ver figura No. 10).



**Show Materials:** es el primer icono de la barra anterior, permite cambiar las texturas, colores y transparencias del objeto.



**Show Animations:** aquí aplicamos diferentes tipos de animaciones como: giros sobre un eje (Common Spin), deformaciones (Deformations), vuelos (flys) y giros en ciclos (path).



**Show Ligthings:** Como su nombre lo indica, esta opción es la herramienta para la orientación de luz o diferentes tipos iluminación sobre los objetos



**Show Environments:** presenta la variedad de colores y texturas

para los entornos o fondos del objeto.



**Show Models:** Contiene objetos prediseñados de la galería de Swift.



**Show Bevels:** Con esta opción aplicamos relieves, bordes según la galería que contiene el programa.

Ahora, debemos hacer clic sobre el material elegido y luego sin soltar el botón llevarlo hasta donde se encuentran las palabras All Faces, All Bevels y All Edges y soltarlos sobre la zona elegida.

**Object:** aca podemos ocultar un objeto o seleccionar si queremos que muestre o no las caras opuestas.

Cada una de las opciones despliega un sub menú de show material y la opción es Reflective.



Figura No.10: Barra de despliegue de texturas

Aquí esta la variedad de texturas para aplicar a los objetos, la manera de hacerlo es escoger la forma y arrastrarla con el Mouse hasta el objeto al que se le quiere aplicar. Dándole la apariencia que el usuario desee.



Figura No. 11 Escena donde se modifico el color y textura de los objetos.



Además de las opciones que Swift tiene establecidas también se puede crear materiales, iluminación, animación y demás, con los que se desee trabajar para ello se selecciona la opción setup del menú la cual despliega las siguiente pantalla

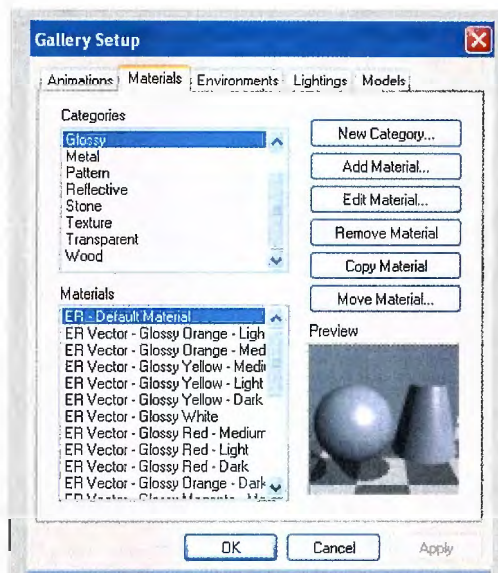
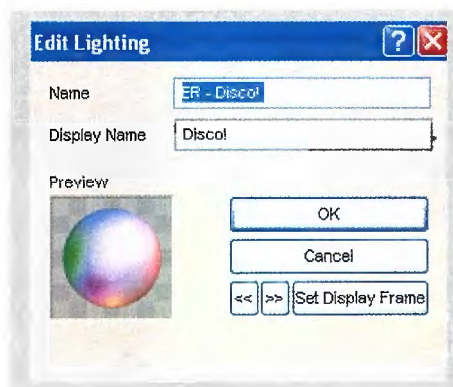


Figura No. 13 pantalla desplegada por menú setup donde presenta las opciones de animación, material, iluminación, medio ambiente, luces y modelado ya existente.

Cuando ya se ha creado y hecho las modificaciones deseadas se pueden guardar con un nombre para el caso después de seleccionar este tipo de iluminación y darle formas a la figura se selecciona la opción edit para guardar el material existente.



Otra de las ventajas con las que cuenta este software, es la importación de imágenes, el programa importa casi cualquier imagen de vectores en EPS (encapsulated postscript).

Todos los programas de dibujo vectorial que importan al formato de EPS pueden usarse como herramientas de diseño preliminares para manipular los diseños posteriores en 3d.

Para ello selecciona del menú file la opción import, donde se selecciona la imagen a importar y luego se escoge la opción import, y así se puede manipular cualquier objeto EPS como si fuese cualquier objeto swift.

### 6.1.2 Luces e iluminación en Swift

Las luces y la dirección de una escena, se logra escogiendo el botón Add Spot Light (agregar punto de iluminación) y rotarlo hacia una dirección específica. (Ver figura No.3)

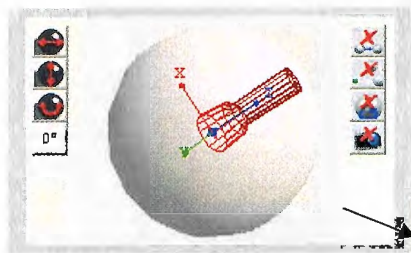


Figura No.3: Add Spot Light. Se puede girar con el mouse hacia donde ira la luz desde la esfera de control.

Existen también los puntos de luz que actúan como una bombilla enfocando directamente un punto, en cambio los Spot Light iluminan zonas concretas y son más utilizados para crear brillo y reflejos en una escena específica u objeto.

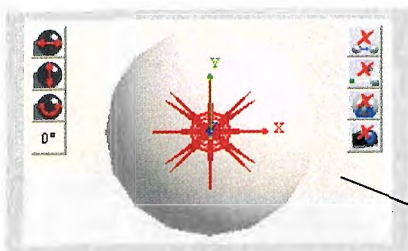
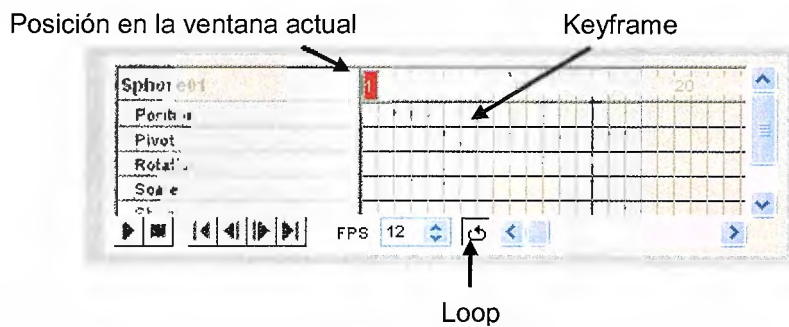


Figura No.4: Esfera de Control de la ubicación del punto de luz

## 6.2 Animaciones.

Las animaciones están basadas en keyframes que no son más que un control del comienzo y fin de la animación.

Comprender el funcionamiento de éstos resulta fundamental si se desea obtener buenos resultados. Básicamente representa un punto en la animación en el que introducimos modificaciones. Entre dos keyframes se realizan cambios de aspecto mediante la función Tweening (como en Flash),



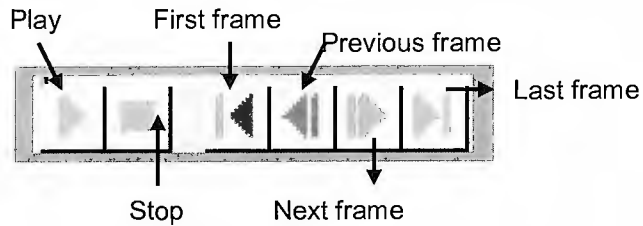
Los controles de animación se describen según las propiedades que puede tener un objeto y se describen así



**Position:** Se refiere al cambio de posición del objeto dentro del espacio que delimita la escena y que podemos medir según su posición en las coordenadas X, Y y Z en relación a la rejilla que define el espacio.

**Rotation:** Cuando un objeto rota entorno un punto o eje.

**Scale:** Cuando el objeto varía de tamaño



### Botones

**Play:** Corre la animación

**Stop:** Parar la animación

**First Frame:** Llevar hasta el primer Frame de la escena.

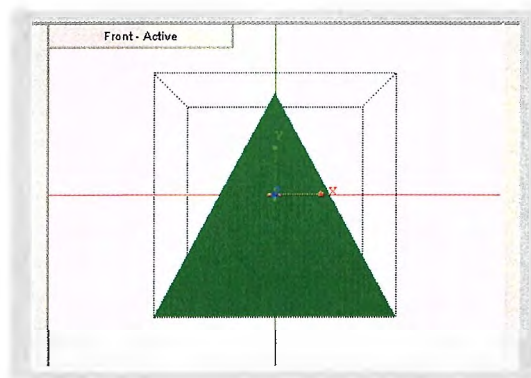
**Previous Frame:** Llevar al frame anterior.

**Next Frame:** Avanza hasta el siguiente frame.

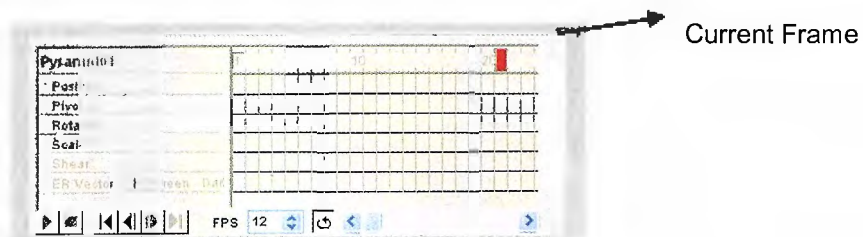
**Last Frame:** Avanza hasta el último frame

El manejo de la animación es sencillo primero se debe tener un objeto en la escena seleccionado colores, materiales, textura, y demás.

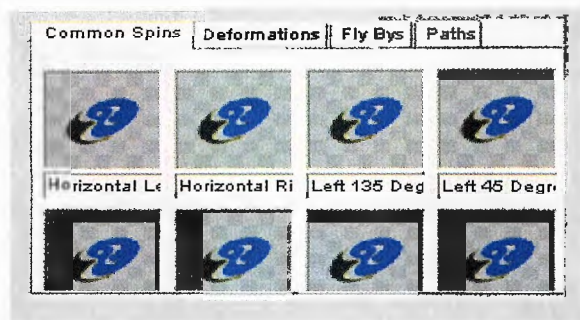
Creando así la siguiente escena:



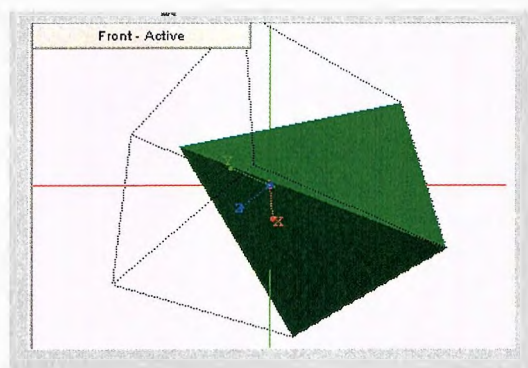
Luego se selecciona y arrastramos el Current Frame Indicator tanto como desee que dure la animación.



Y se elige la opción de animación que se quiere utilizar



Permitiendo así un movimiento en el objeto y la duración en que debe realizarse

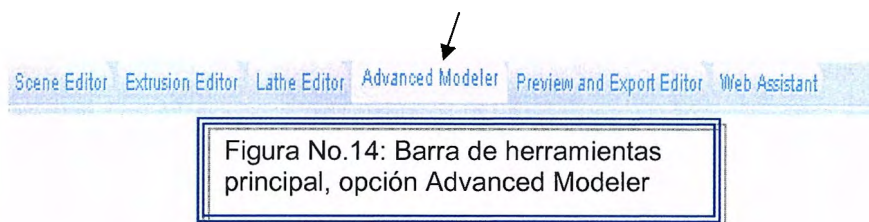


## 6.3 GRÁFICOS 3D

En los primeros capítulos, ésta monografía hace énfasis en los temas de introducción al graficado en tres dimensiones vectores y polígonos, ahora retomamos esos principios para modelar gráficos en 3D, lo cual se basa en dos aspectos:

1. Modelado Vectorial
2. Modelado Poligonal

Ambos casos dependen de una opción básica de la barra de herramientas principal: Advanced Modeler o en español Modelador Avanzado (figura No.14)



Ésta es el área de trabajo que permite un mejor campo visual y comodidad a la hora de desarrollar o modelar los objetos.

Esta herramienta permite modificar todas las partes de cualquier figura básica (vértices, caras, cortes o agrupaciones de éstas mismas).



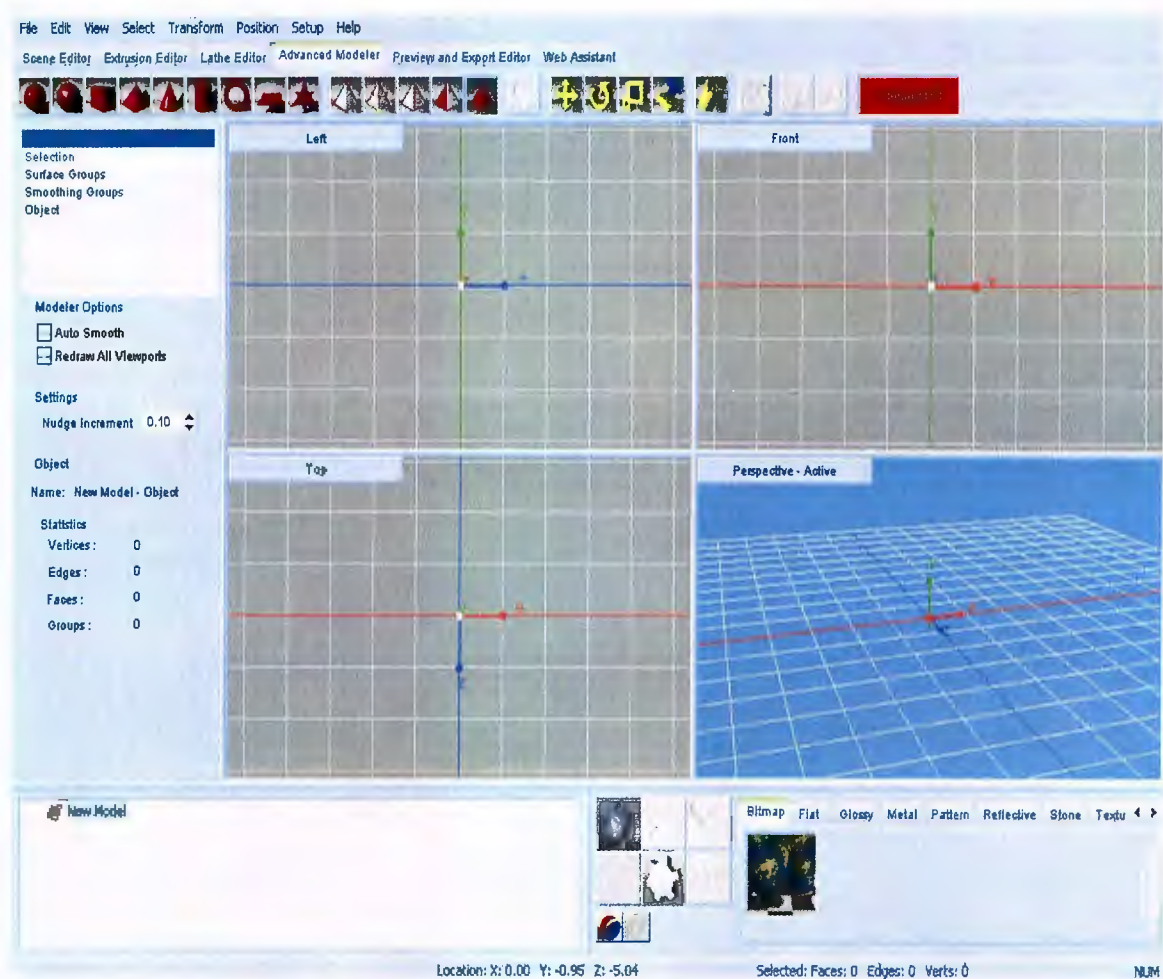


Figura No.15: Representación del área de trabajo en Advanced Modeler

Como se visualiza en la Figura No.15, se cuenta con 4 vistas diferentes de diseño: IZQUIERDA, FRONTAL, SUPERIOR Y PERSPECTIVA.

Es muy importante mencionar que para poder trabajar bajo esta herramienta, es necesario haber creado el objeto en el Scene Editor.



### 6.3.1 MODELADO VECTORIAL Y POLIGONAL

En Swift 3D v4, el ambiente de modelado poligonal permite crear cualquier cosa imaginable, así como afinar los detalles de los modelos ya existentes. Esta opción se encuentra siempre sobre la barra principal bajo el nombre de Editing Mesh. (Figura No.16)



Figura No.16: Modelado Vectorial y Poligonal en una sola opción.

Al activar el botón, éste se torna en fondo color rojo, lo cual indica que el usuario tiene total acceso a la estructura poligonal del objeto,

Y el modelado vectorial, es simplemente la desactivación de esta opción.

## 6.4 IMPLEMENTACIÓN EN LA WEB

Swift nos proporciona una alternativa fácil de utilizar y de muchísima utilidad para la presentación de objetos 3D en la web, trabajando con la barra de herramientas principal en la opción Preview and Export Editor.

Scene Editor   Extrusion Editor   Lathe Editor   Advanced Modeler   Preview and Export Editor   Web Assistant

Figura No.17 Barra de herramientas opción para exportación a la web

La utilización de ésta herramienta requiere que inicialmente el usuario haya creado previamente un objeto con o sin animaciones. Analicemos con un ejemplo sencillo:

Tomemos de la barra de modelo un objeto predeterminado cualquiera, para este caso hemos elegido el objeto ER-TV.



Figura No.18: Elección de un objeto predeterminado de la barra Models.

Luego le aplicamos una animación, tal como lo hemos aprendido con el tema 6.2.



Figura No.19: Animación del objeto elegido

Ahora bien, nuestro objetivo principal es exportar el objeto, y es aquí donde pasamos a la pestaña de Preview and Export Editor, la cual presenta el siguiente entorno (ver figura No.20)

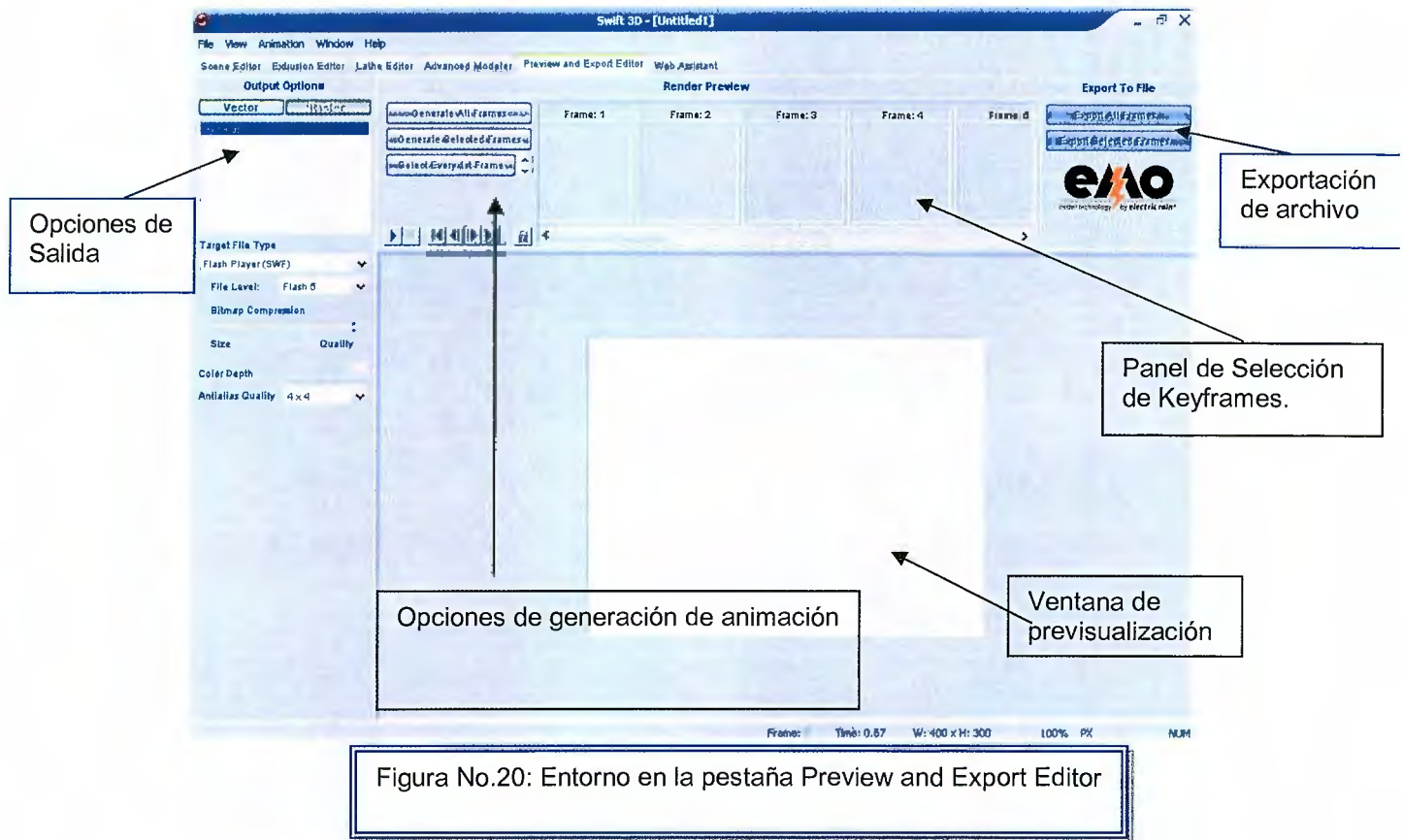


Figura No.20: Entorno en la pestaña Preview and Export Editor

**Opciones de Salida:** Proporciona 2 alternativas de exportación: Vectorial y Rasterizado la diferencia entre ellos es que en el modo Vectorial los objetos no se exportan con la misma calidad y nitidez que en el modo Pasterizado. Ambos cuentan con la opción de elección del formato a exportar en la lista de selección Target File Type:

1. Exportación a Flash Player (SWF)
2. Exportación a Swift 3D Flash Importer (SWFT)
3. Encapsulate Postscrip (EPS)
4. Adobe Ilustrador (AI)
5. Scalable Vector Grapichs (SVG)

Figura No.21:  
Target File  
Types.



Para dar seguimiento a nuestro ejemplo seleccionaremos la opción Vector para un tipo de archivo Flash Player. Ver figura No.22

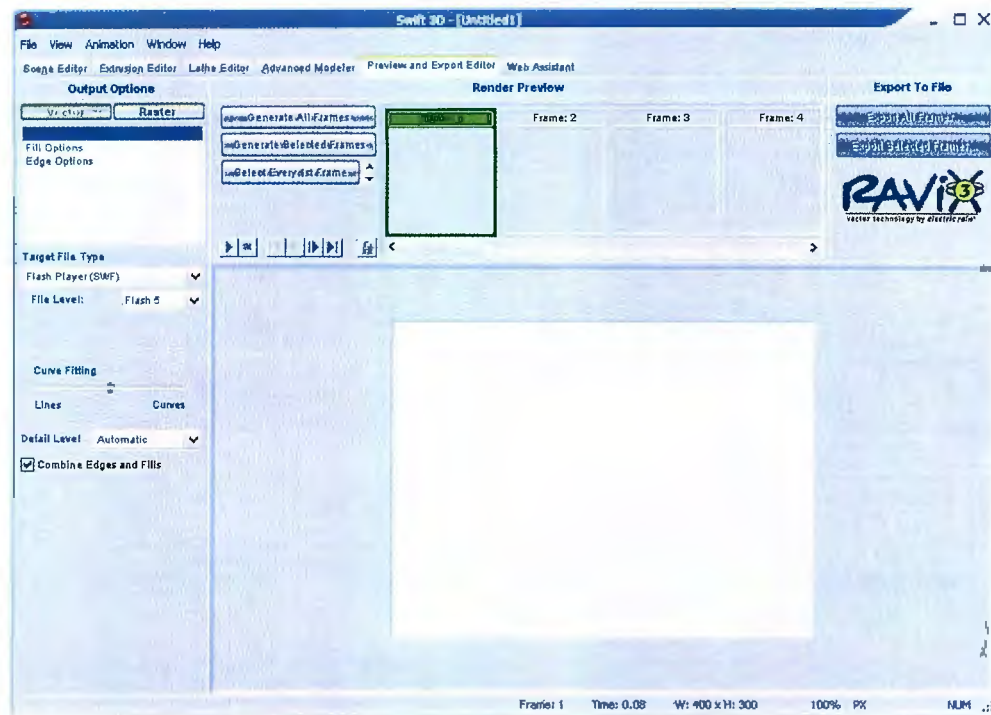


Figura No.22: Ejemplo entorno bajo las opciones Vector para un exportación a Flash Player

**Opciones de Generación de animación:** Aquí se cuentan con 3 opciones:

1. **Generate All Frames:** Genera la vista preliminar de todos los frames creados con el objeto. Recordemos que los frames se crean cuando hemos aplicamos animaciones a los objetos, en caso contrario únicamente se creará un frame.
2. **Generate Selected Frames:** Como su nombre lo indica, tenemos la opción seleccionar los frames o transiciones de la animación que queremos exportar.
3. **Select Every 1st. Frame:** con esta opción únicamente seleccionamos el primer frame de cada objeto creado.



Cada una de estas opciones genera una vista de los frames en la Ventana de Previsualización. Inicialmente solo aparece indicada la cantidad de frames, pero al dar clic sobre Generate all Frames se comenzarán a previsualizar. Así:

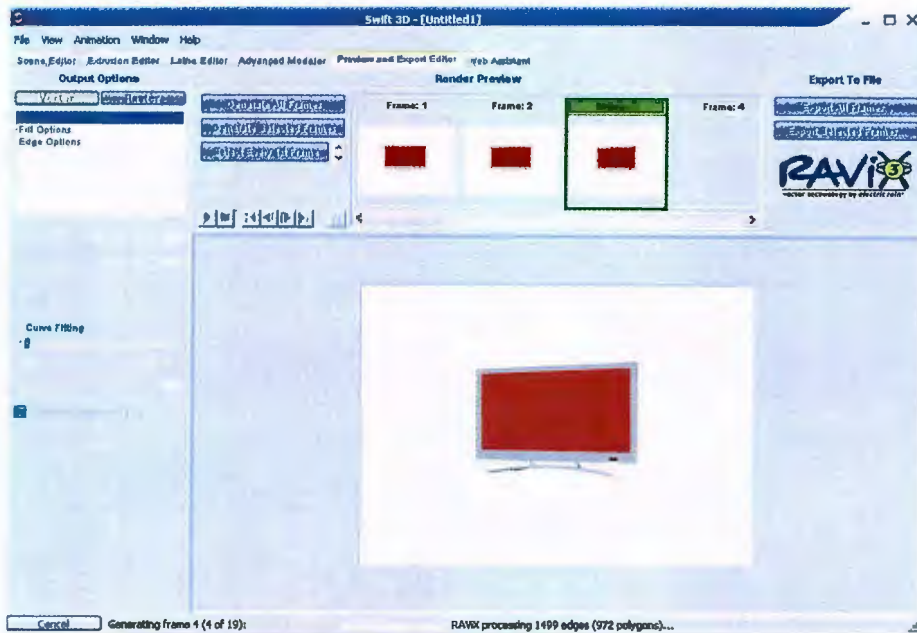


Figura No.23: Generación de Frames

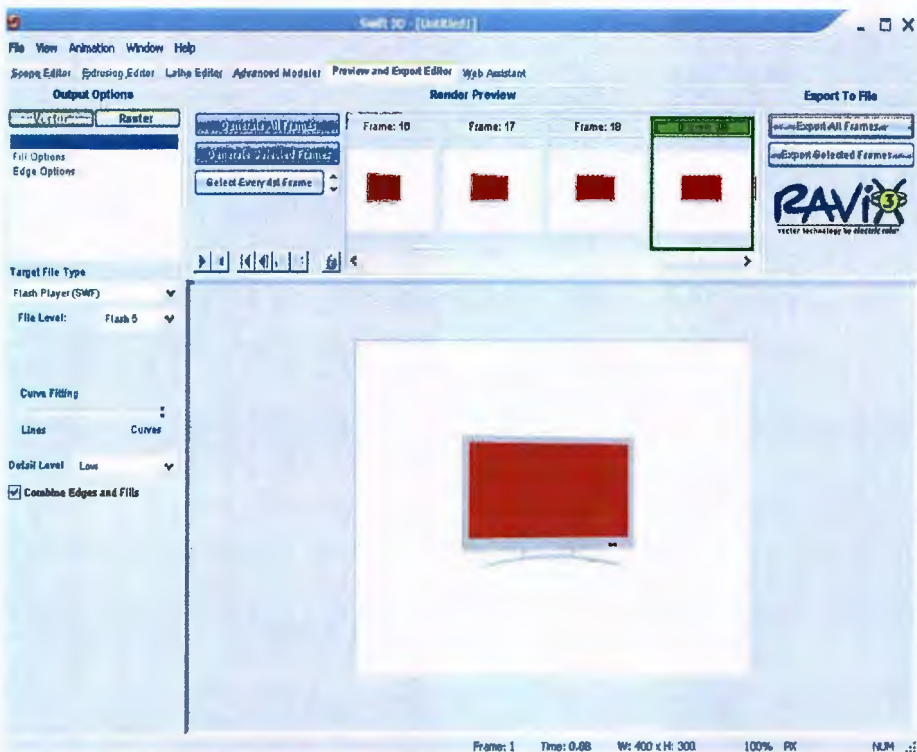


Figura No.24: Finalización de la generación de frames.

**Exportación de Archivos:** Aquí se tienen dos opciones:

1. Exportar Todos los Frames
2. Exportar Frames seleccionados.

Siguiendo con nuestro ejemplo damos clic sobre la opción uno y dejaremos que el software termine el proceso de exportación, de la siguiente manera:

1. Aparece la siguiente ventana, en la cual indicaremos el nombre del archivo y su localidad en el computador.

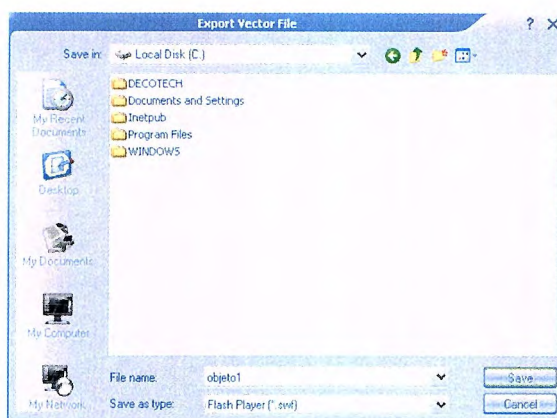


Figura No.25: Guardar el objeto con formato SWF

2. El archivo se guarda automáticamente, y buscas en tu ordenador el archivo nombre\_archivo.swf y das doble clic sobre él para previsualizarlo, y el archivo estará listo para subirlo a cualquier página web.



Figura No.26: Objeto Terminado bajo formato SWF de Flash Player

# EJEMPLOS CON SWIFT 3D

## Ejemplo 6.1. Una Lámpara 3D

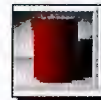
- **La Base de la Figura**

1. Haga clic en la opción de **Environment** de la barra de propiedades.
2. Haga doble clic en la caja de selección de color del fondo **Background Color**
3. Escoja el color Blanco.
4. Haga clic en **Ok**.

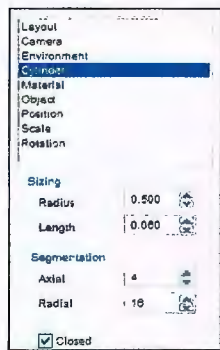


- **La Base de la Lámpara con un Cilindro Básico.**

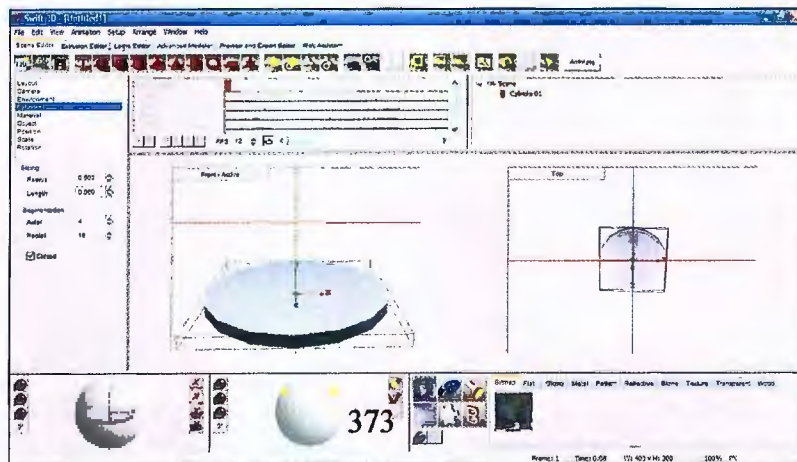
1. Clic en crear cilindro en la barra de herramientas principal.



2. En la barra de propiedades se ajustara las dimensiones de la siguiente manera:



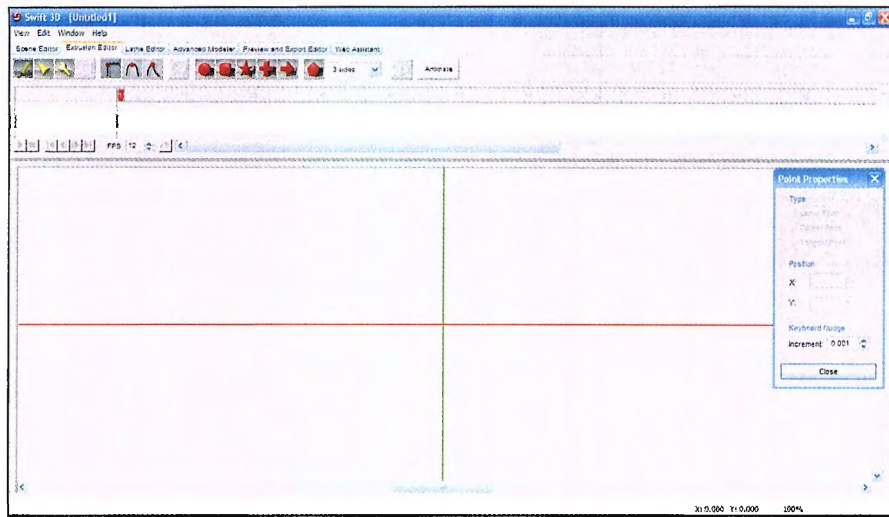
3. Haga clic sobre el cilindro y arrástrelo hacia la parte inferior de la vista como se muestra en la pantalla.



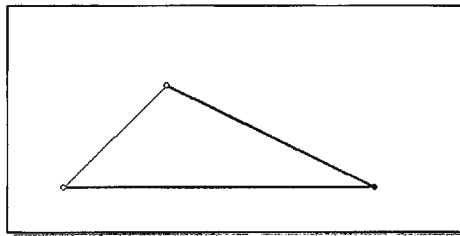


- El Soporte del Brazo de la Lámpara con *Extrusion Editor*

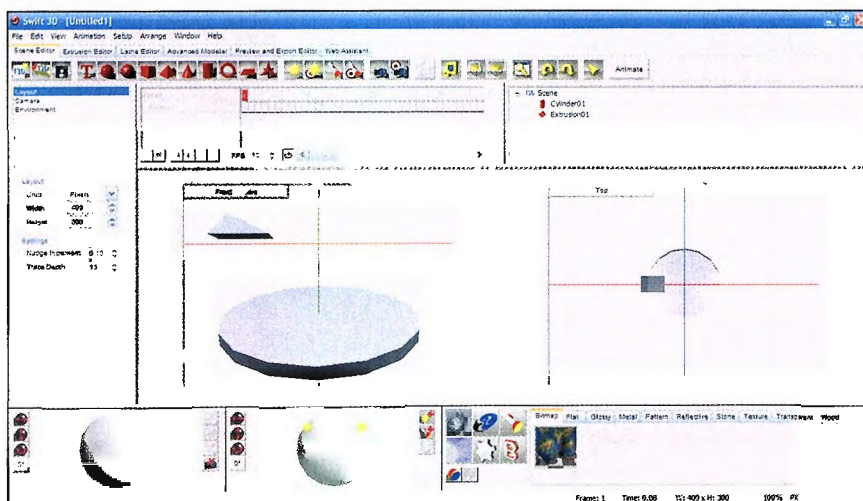
1. Haga clic en la pestaña llamada **Extrusion Editor** ubicada en la parte superior del área de trabajo.



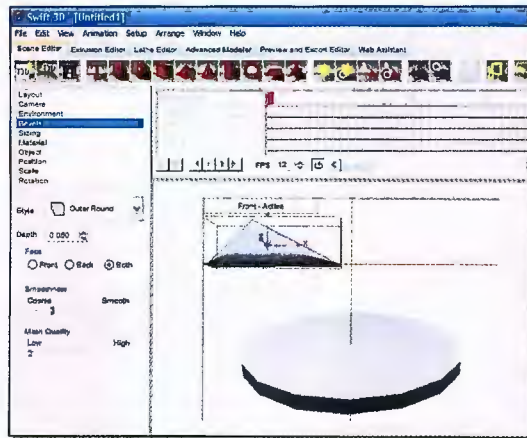
2. Dibuje un triángulo similar a la imagen que se muestra.



3. Después de dibujar el triángulo regrese al editor de escenas **Scene Editor**

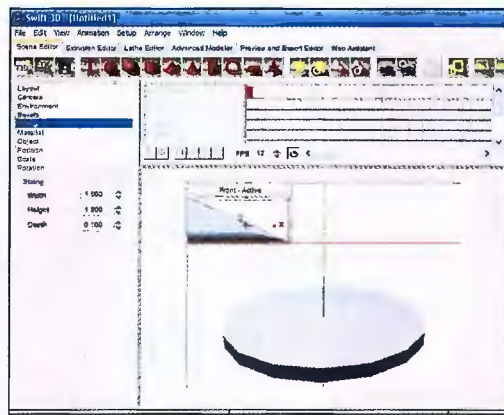


- Vaya a la barra de propiedades con la opción **Bevels** seleccionada, ajuste el estilo (**style**) a la opción **Outer Round**

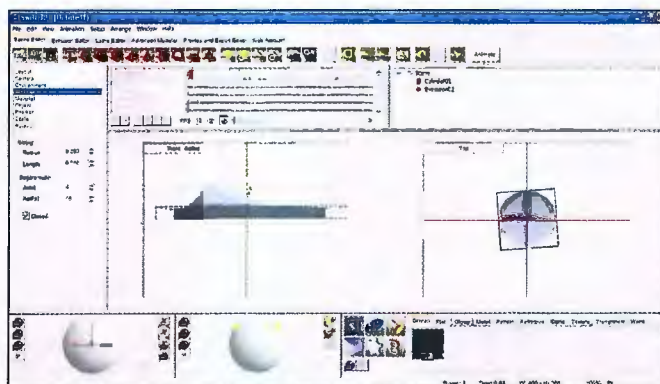


- Seleccione la sección de tamaño en la barra de propiedades **sizing**. Y ajuste las dimensiones de la siguiente manera:

<b>Width</b>	1.500
<b>Height</b>	1.200
<b>Depth</b>	0.100

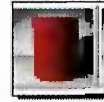


- Haga clic sobre el “Brazo de soporte” recién creado y arrástrelo con el Mouse hasta que esté sobre la base de la lámpara como se muestra en la imagen.

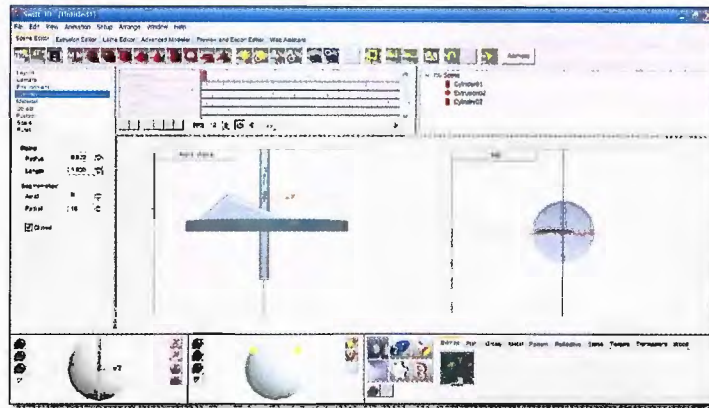


- **El Brazo de la lámpara con Cilindro Básico**

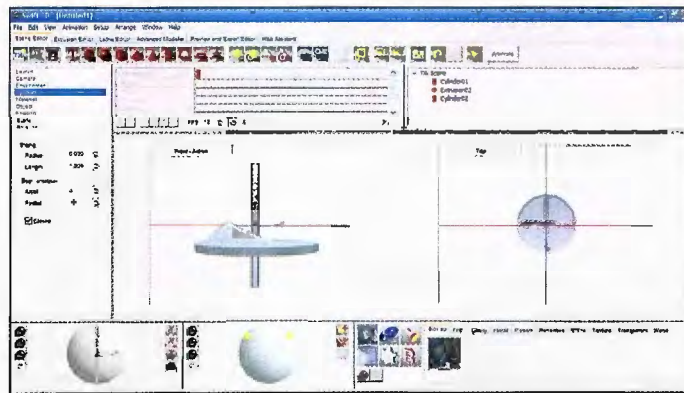
1. Haga clic en el botón de Crear Cilindro Básico



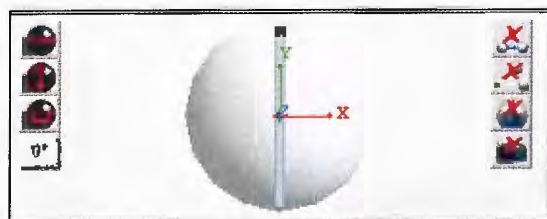
2. Con la opción **Cylinder** de la barra de propiedades seleccionada ajuste la propiedad **Radius** a 0.030.



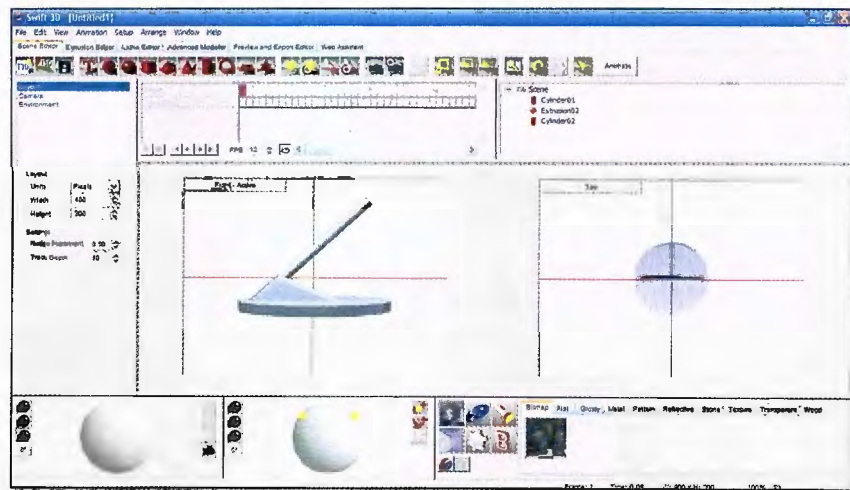
3. Con la vista frontal activada, **Front-active**, haga clic en el botón **Frame all objects**, de la barra de herramientas principal; esta acción ajustará la vista para que todos los objetos sean visualizados.



4. Con el brazo de la lámpara seleccionado, active el botón *lock spin* del *TrackBall* de Rotación como se muestra en la imagen, esta acción limitará la rotación para que el objeto no se mueva hacia el fondo o hacia el frente.

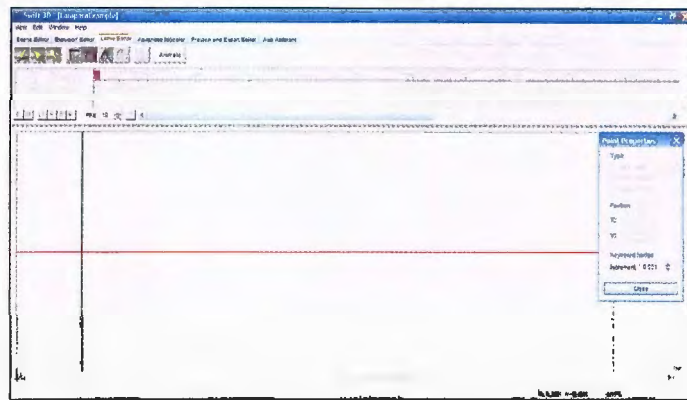
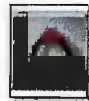


5. Rote y arrastre el brazo de la lámpara hasta que se posicione dentro del soporte de la lámpara como se muestra en la imagen.

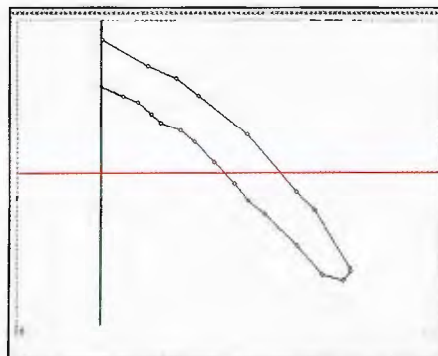


- **Creando el Casco de la Lámpara**

1. Haga clic en la pestaña **Lathe editor** ubicada en la parte superior de la pantalla y posteriormente seleccione la herramienta **Curve Point**.

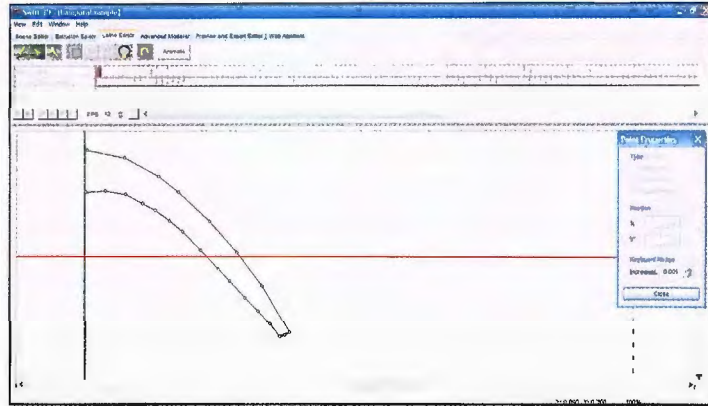


2. Se dibujara una forma curva haciendo clic en diferentes puntos como se muestra en la imagen, no importa si no queda perfecta ya que el siguiente paso es ajustarla. Fijarse que la forma quede sobre o en la línea verde.

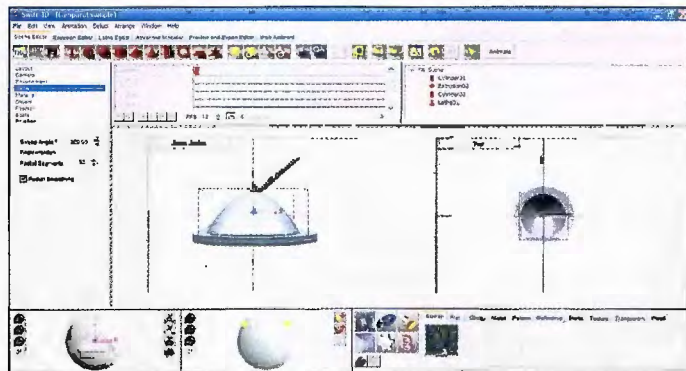




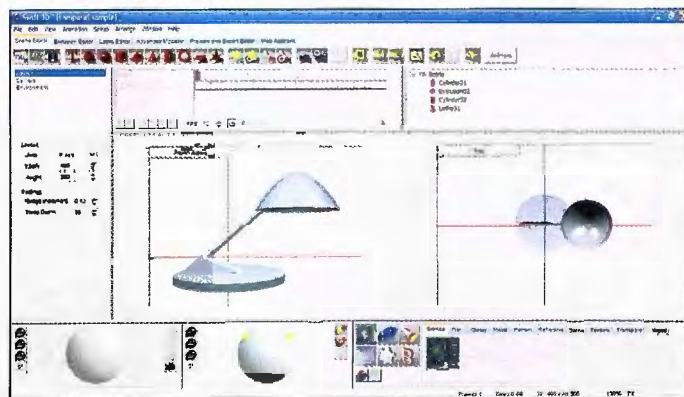
- Ajuste la forma utilizando la herramienta **Shape Tool**, ajustando cada punto hasta obtener la forma de una curva como se muestra en la imagen.



- Regrese al **Scene editor** y haga clic en el botón **Frame all objects** para ajustar la vista y que todos los objetos puedan visualizarse.



- Al regresar al editor de escenas automáticamente se habrá creado una forma con la forma de casco de la lámpara. Arrástrela hasta el brazo de la lámpara para que forme la figura completa como se muestra en la imagen.



- El Foco de la Lámpara con una Esfera.

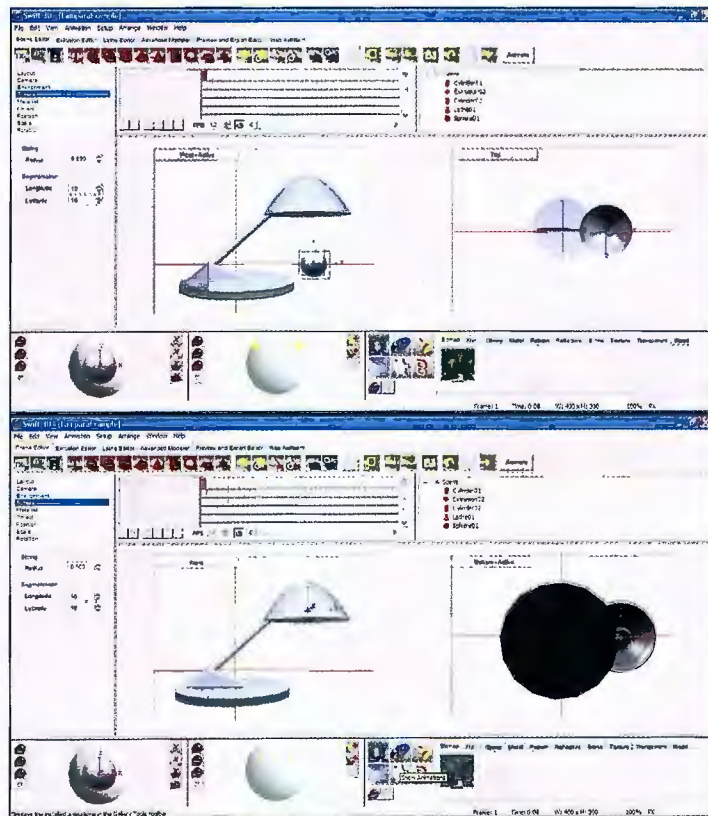
1. Haga clic en el botón **Create Sphere** de la barra de herramientas.



2. Con la nueva esfera seleccionada Active el botón **Scaling Mode** de la barra de herramientas.

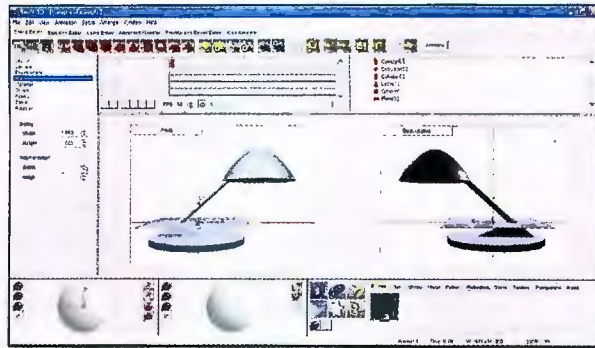


3. Haga clic sostenido sobre la esfera y mueva el cursor hasta obtener un tamaño adecuado, posteriormente coloque la esta esfera en la parte de debajo de el casco a cabeza de la lámpara como se muestra en la imagen.

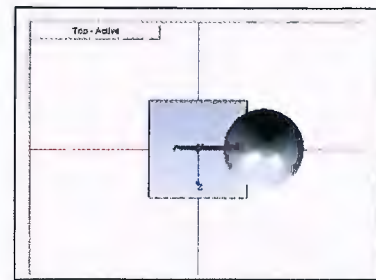
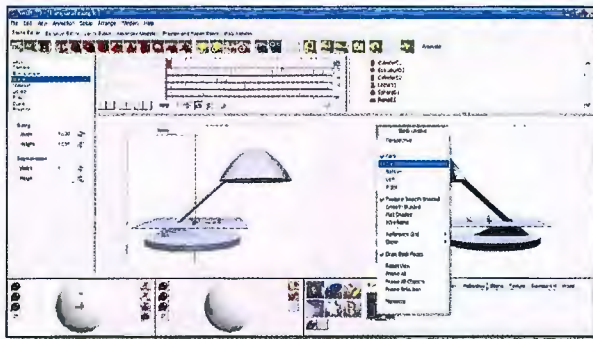


- Una base plana para la Lámpara.

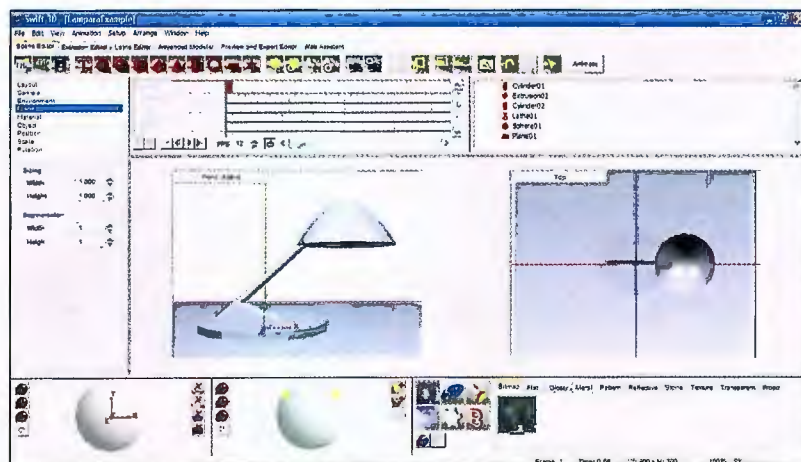
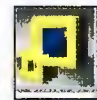
1. Haga clic en **Create Plane Primitive** y arrástrela hasta la base de la lámpara a modo de colocarla como si fuera la mesa.



2. Cambien la vista y actívela a Vista desde arriba **Viewpoint Top**.



3. Active el Botón **Scaling Mode** y ajuste el la figura plana hasta que sea lo suficientemente grande.



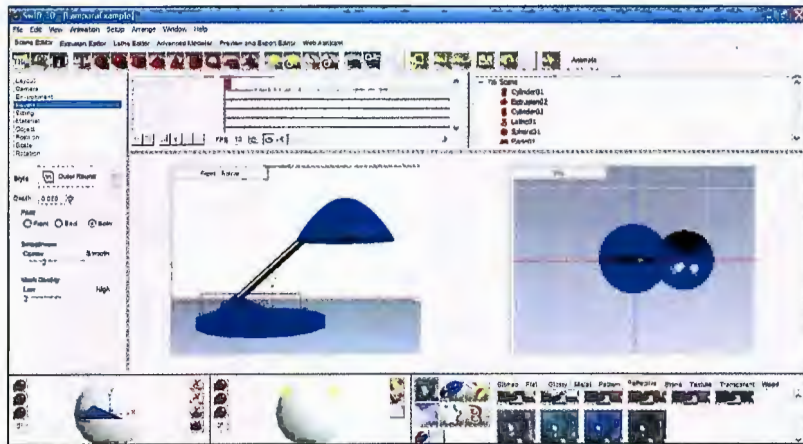
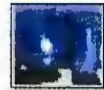


- **Aplicando Materiales A Todos Los Objetos De La Escena**

1. Haga clic en la pestaña de **Reflective Materials** de la galería de materiales.



2. Desde cualquier vista haga clic y arrastre el Material llamado **ER Vector – Reflective Blue** hacia la base de la lámpara, el soporte de la lámpara, al plano y la cabeza o casco de la lámpara. De esta forma se aplicará dicho material a los objetos que desee. Aplique el material **ER Vector – Reflective Black** al brazo de la lámpara.

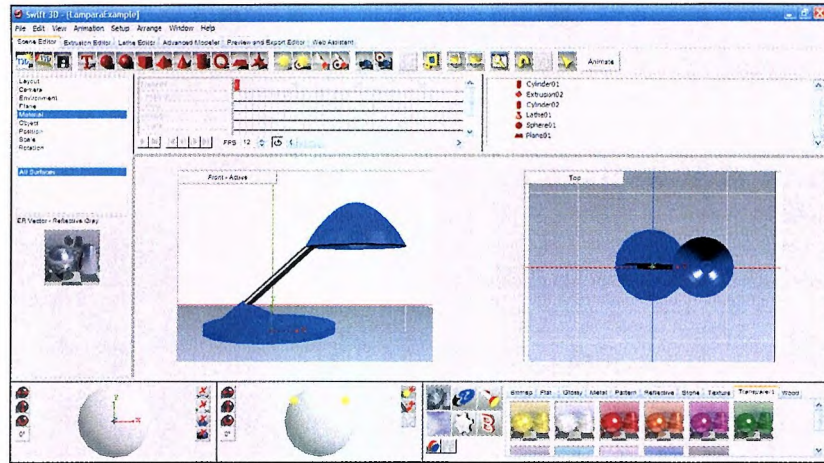


3. Seleccione la pestaña **Transparent** de la paleta de materiales y aplique el material **ER Vector – Transparent White** al foco o bombillo de la lámpara.



- **Ajustando El Material Del Plano.**

1. Seleccione el objeto plano que representa el suelo o la mesa de la lámpara.



2. En la barra de propiedades vaya a la sección **Material** y haga doble clic en **Material Preview window** (ventana de previsualización de material).



3. Haga doble clic en el cuadro para seleccionar el color de **Reflection color**, seleccione el color gris, haga clic en ok, y luego en ok nuevamente. Recuerde entrar y seleccionar el color gris en el área marcada aunque ya aparezca el color gris ya que esto permitirá visualizar reflexiones en el material

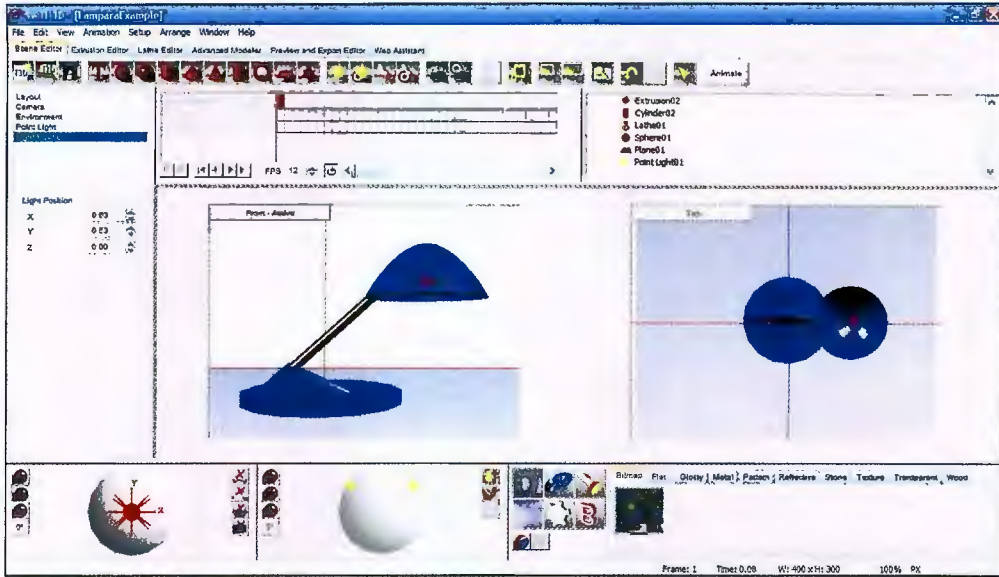


- **Agregando Una Luz Al Foco o Bombillo.**

1. Haga clic en el botón **Create Free Point Light**.



2. Mover el punto de luz creado hacia donde se encuentra el bombillo.

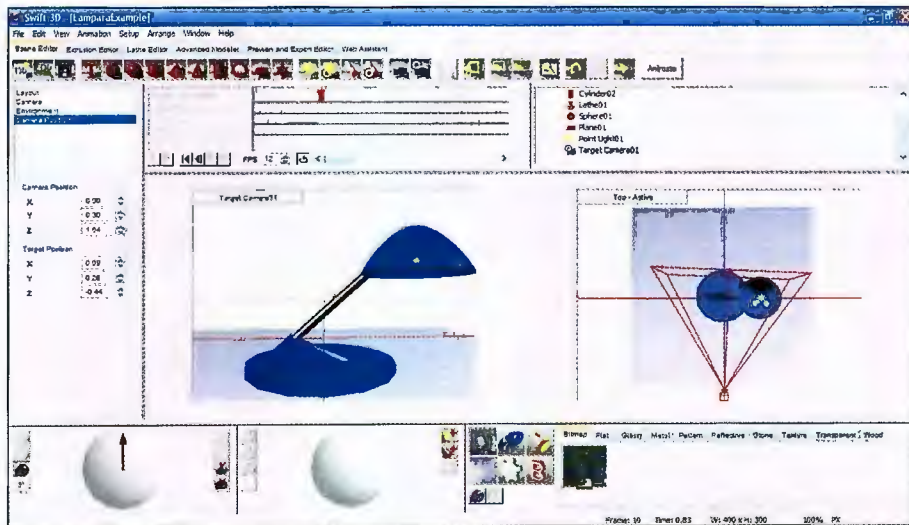


## INSERTANDO Y POSICIONANDO UNA CÁMARA

1. Haga clic en el botón **Create Target Camera** para crear una cámara que puede moverse con el fin de crear una animación moviendo solo la cámara.

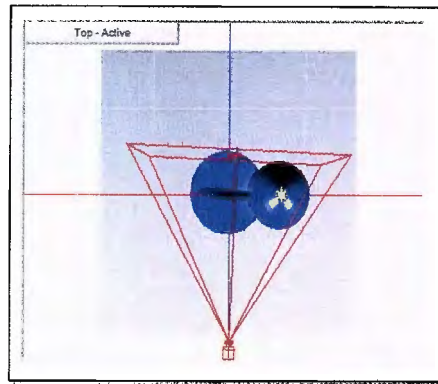
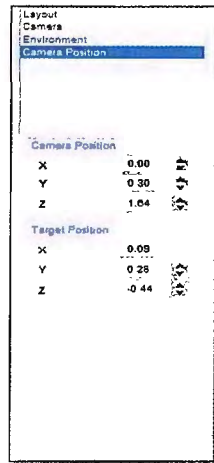


2. Active la vista **Viewpoint Top**.

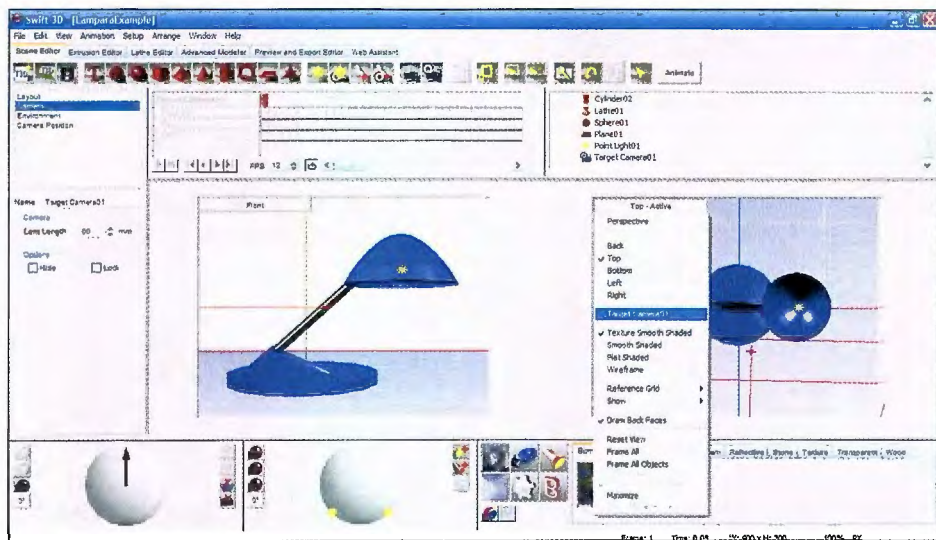




- Haga clic sobre la cámara creada y arrástrela a su gusto hasta tener el ángulo deseado. Si no se logra ver la cámara para dar el ajuste adecuado, utilice las propiedades de la cámara. Con **Target Camera01** seleccionada, busque en las propiedades **Camara Position**, y ubique la viñeta **Camara Position** y modifique el eje **z** hasta que pueda ver la cámara, con esto podrá seleccionar la cámara para darle el ángulo que necesite.



- La cámara se guardara automáticamente con el nombre **Target Camera01**.
- Seleccione la vista **Target Camera01** desde el menú vistas para ver el objeto directamente desde la cámara que se creó.



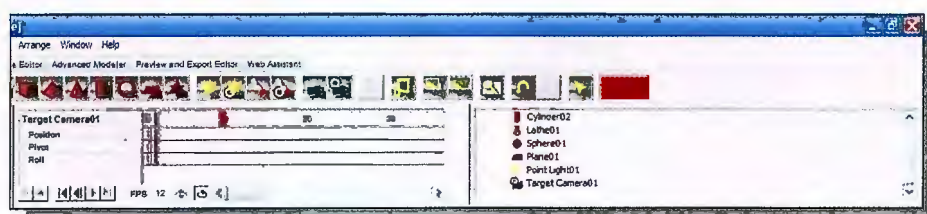


## CREANDO UNA ANIMACIÓN DE CÁMARA.

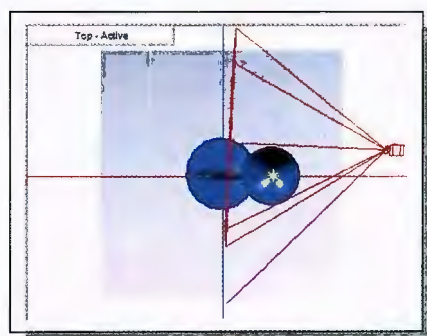
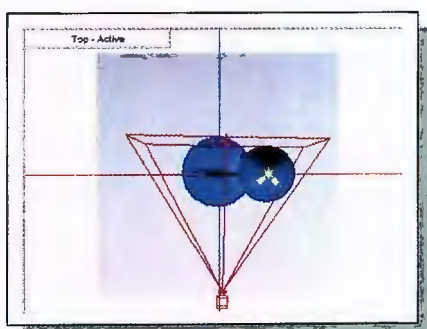
1. Active el botón **Animate** de la barra de herramientas principal.

Animate

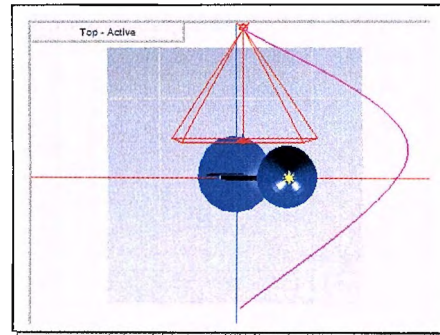
2. Este botón habilitará la línea de tiempo para la animación. Seleccione el Frame Numero 10.



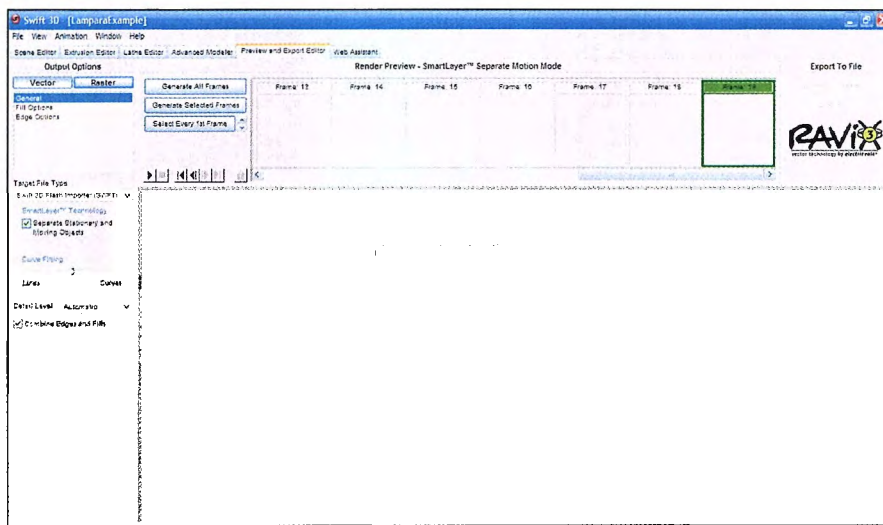
3. Mueva la cámara hacia la derecha y arriba desde la vista **View top**.



- ahora haga clic en el Frame 20 para mover nuevamente la cámara un poco más arriba.



- Renderizando Y Exportación de la Escena con Ravix 3.**
- Asegúrese de tener activa la vista **Target camera01** y Haga clic en la pestaña **Preview and Export Editor**.



- Bajo la categoría **General** de **Output Options** seleccione **Flash Player** (SWF) en la opción **Target File Type**.



- Con la opción **Vector** seleccionada haga clic en la categoría **Fill Options** y establezca los siguientes valores:

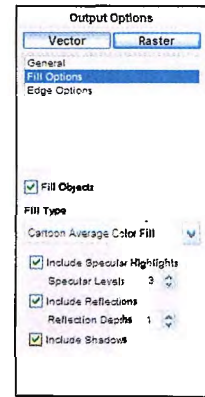
**Fill Object**, debe estar chequeado.

**Fill type**, seleccione **Cartoon Average Color Fill**.

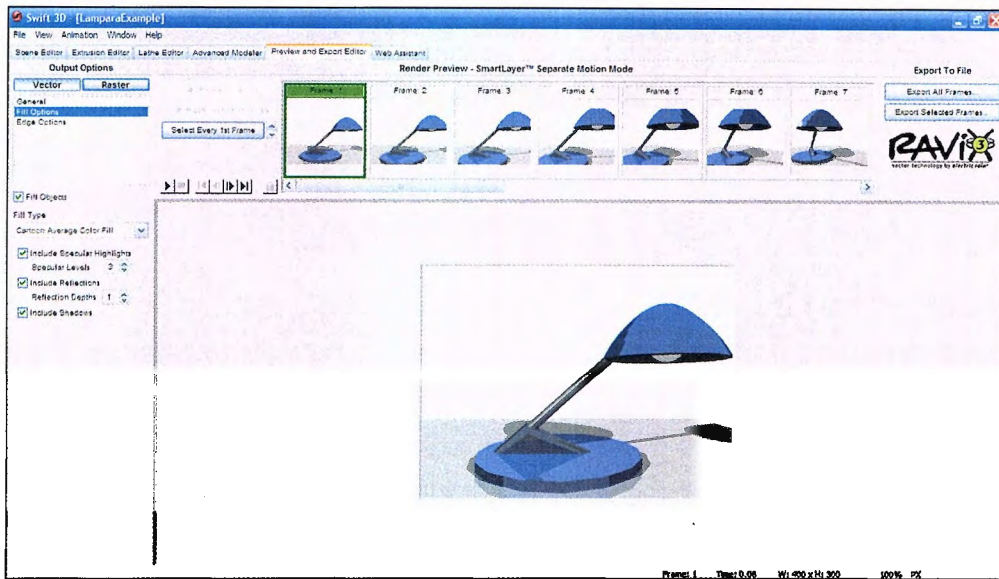
**Include Specular Highlights**, debe estar chequeado y debe tener 3 en el **Specular Levels**

**Include Reflections**, debe estar chequeado y debe tener 1 en **Depths Reflection**.

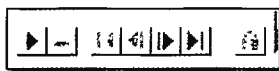
**Include Shadows**, debe estar chequeado.



- Para renderizar la escena con estas opciones hacer clic en **Generate all frames**.



- Después de que los frames hayan sido generados se podrá previsualizar la animación con los controles de película.



- Una vez que estén generados todos los fragmentos puede proceder a exportar la película haciendo clic en el botón **Export All Frames**. Esta acción abrirá una ventana de Guardar como en la que podrá elegir el nombre y el destino del archivo.

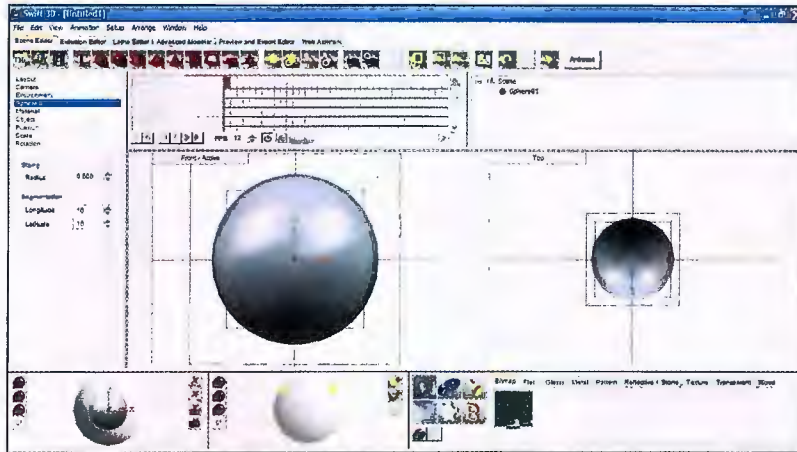




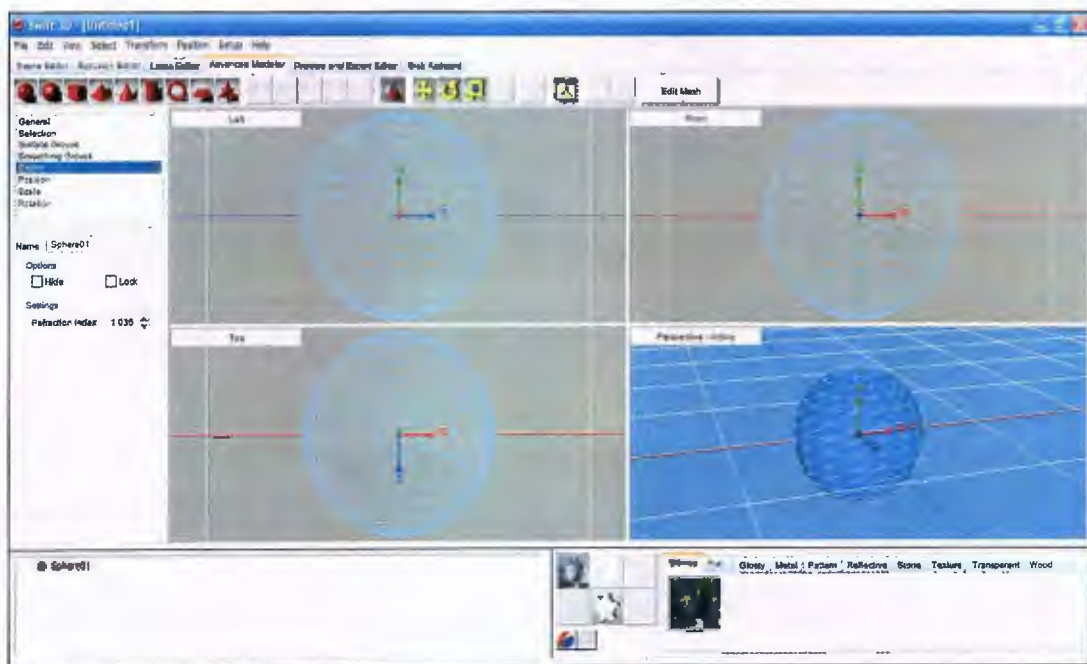
## Ejemplo 6.2. Un Ser Humano

- **Creando la cabeza con una esfera**

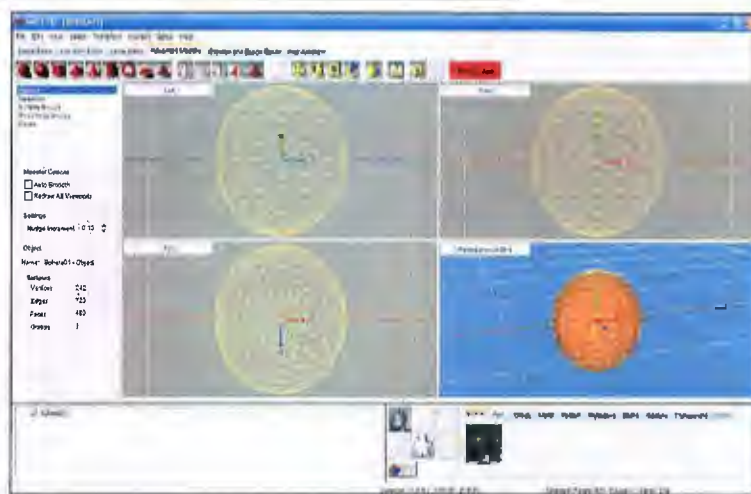
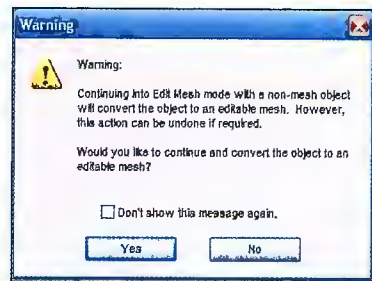
1. Haga clic en **Create Sphere**



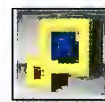
2. Para modificar las dimensiones de la esfera, seleccionaremos la pestaña **Advance Modeler**.

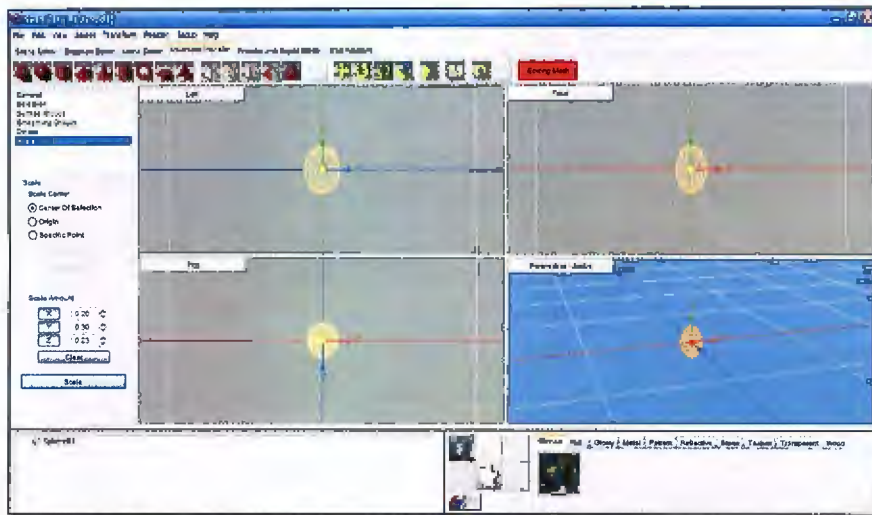


- En el menú desplegable **Edit**, seleccione **Edit Mesh**. Aparecerá un mensaje de advertencia, el cual advertirá si desea continuar con este método, de clic en **yes**.

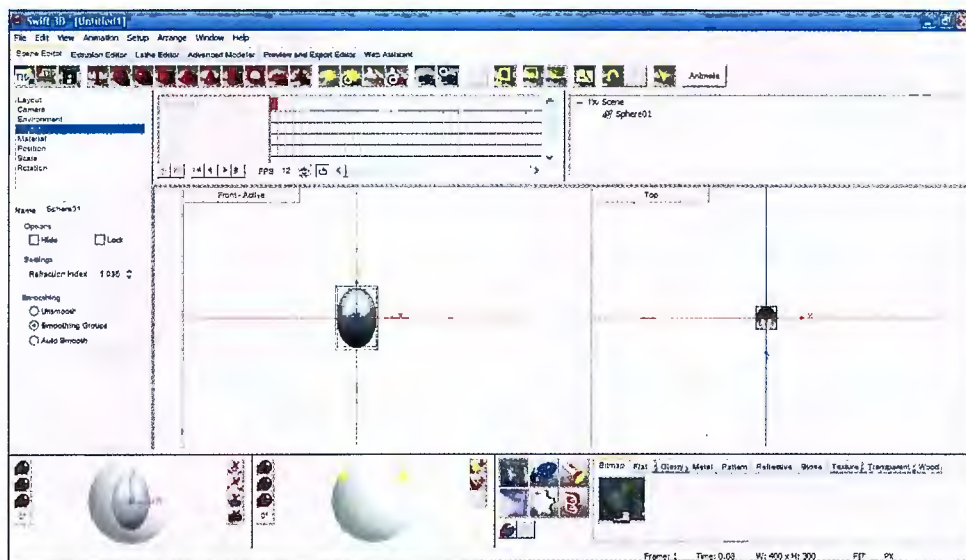


- Seleccione **Scale Uniform Mode (s)**, en los parametros de escala de la esfera **Scale Amount** introduzca los siguientes valores **Factor x: 0.20, Factor y: 0.30 y Factor z:0.23**. luego de introducir los valores de clic en **Scale**, para q tenga efecto los cambios.

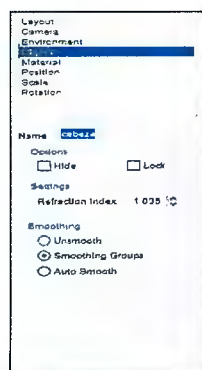




5. Regrese al **Scene Editor**, como se puede ver se ha cambiado la esfera.

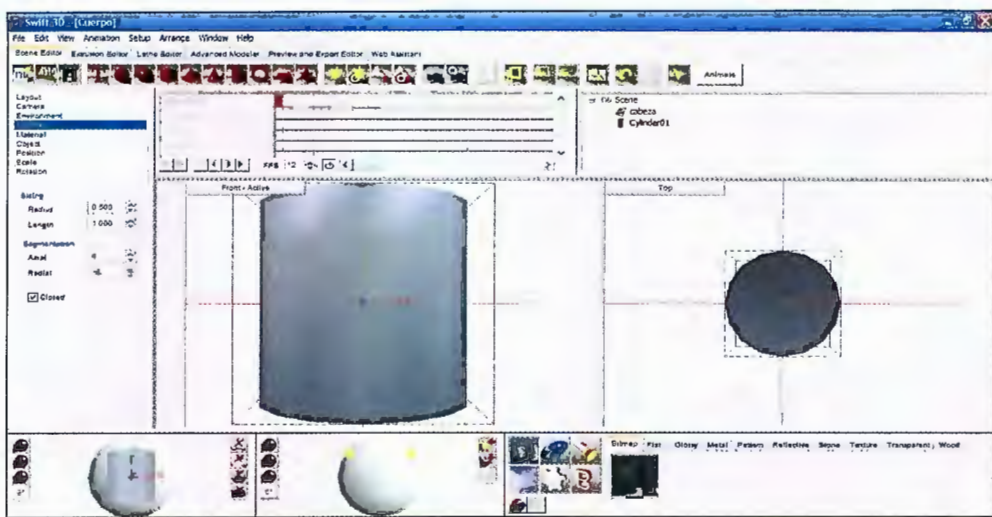


6. A continuación procederemos a cambiar el nombre del objeto, para tener una mayor manipulación de los objetos. Seleccione el objeto, y en **name** escriba cabeza.



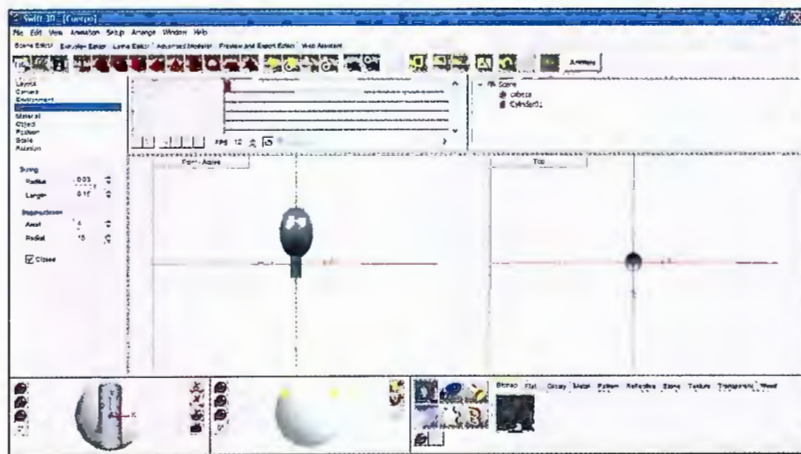
- **Creando el cuello con un cilindro**

1. De clic en **Create Cylinder** para formar el cuello.



- 2.

En las propiedades del cilindro, en **Sizing** cambie los valores de **Radius: 0.030** y **Length: 0.15**. Seleccione la cabeza y arástela hacia arriba, sin que quede espacio vacío entre la cabeza y el cilindro.



3. Cambie el nombre del objeto por **cuello**.

- **Crear el resto del Cuerpo.**

1. Con las técnicas que se vieron en las 2 secciones anteriores se creará la figura con las dimensiones que se presentan.



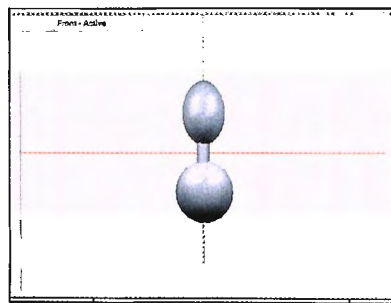
**A) Torax.**



X Factor: 0.27

Y Factor: 0.29

Z Factor: 0.17

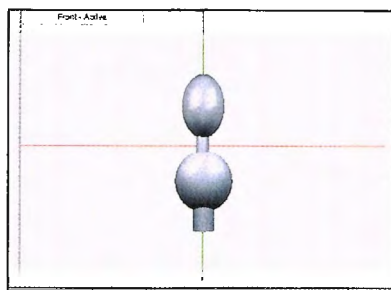


**B) Cintura.**



Radius: 0.050

Length: 0.170



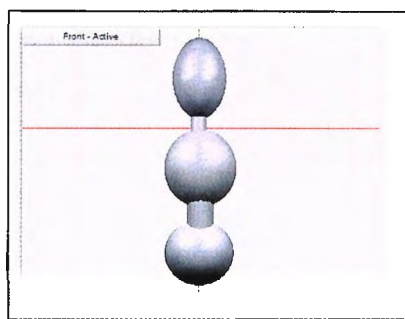
**D) Cadera.**



X Factor: 0.26

Y Factor: 0.24

Z Factor: 0.17

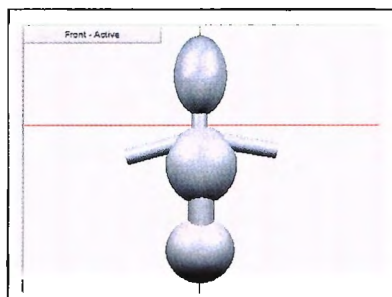


**E) Brazo,** se crearan dos y se nombraran según su posición Brazo mas Izq si es izquierda o Der si es Derecha (Ej.: BrazoIzq).



Radius: 0.030

Length: 0.240

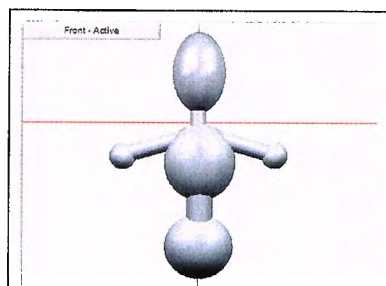


---

**F) Codo,** se crearan dos y se nombraran según su posición Codo más Izq si es izquierda o Der si es Derecha (Ej.: Codol Izq).



X Factor: 0.10  
Y Factor: 0.10  
Z Factor: 0.10

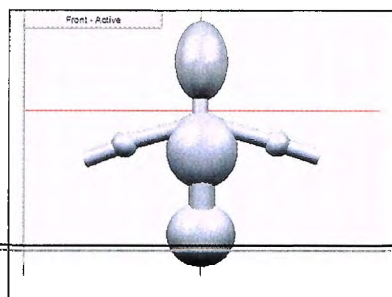


---

**G) Antebrazo,** se crearan dos y se nombraran según su posición AnteBrazo más Izq si es izquierda o Der si es Derecha (Ej.: Antebrazo Izq).



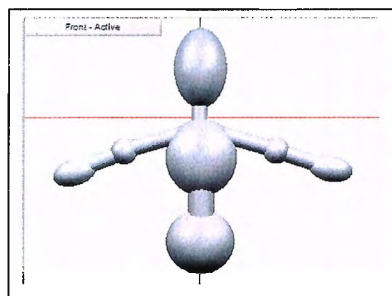
Radius: 0.030  
Length: 0.160



---

**H) Mano,** se crearan dos y se nombraran según su posición Mano más Izq si es izquierda o Der si es Derecha (Ej.: Mano Izq).

X Factor 0.17  
Y Factor 0.10  
Z Factor 0.10

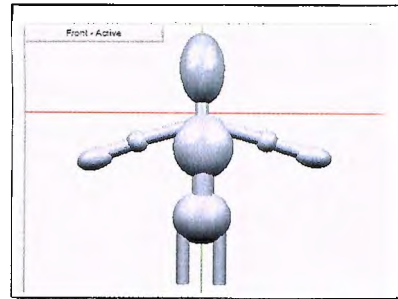




I) **Pierna**, se crearan dos y se nombraran según su posición Pierna más Izq si es izquierda o Der si es Derecha (Ej.: Piernalzq).



Radius 0.030  
Length 0.240

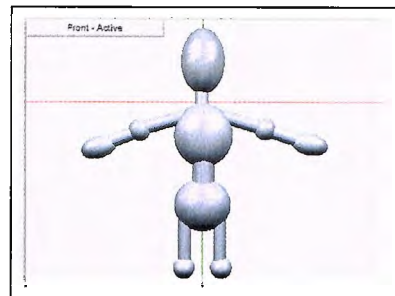


---

J) **Rodilla**, se crearan dos y se nombraran según su posición Rodilla más Izq si es izquierda o Der si es Derecha (Ej.: Rodillalzq).



X Factor 0.10  
Y Factor 0.10  
Z Factor 0.10

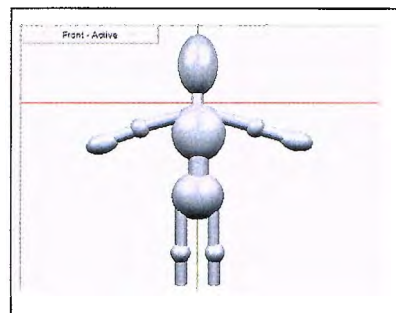


---

K) **Tobillo**, se crearan dos y se nombraran según su posición Tobillo más Izq si es izquierda o Der si es Derecha (Ej.: TobilloIzq).



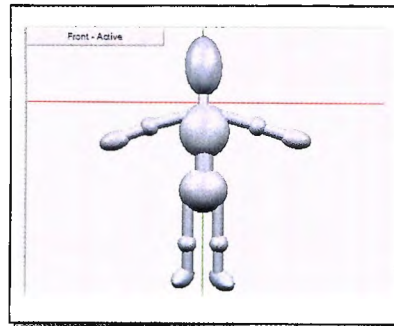
Radius 0.030  
Length 0.160



**L) Pié,** se crearan dos y se nombraran según su posición Pie más Izq si es izquierda o Der si es Derecha (Ej.: Pielzq).



X Factor 0.10  
Y Factor 0.10  
Z Factor 0.22



- **Jerarquía de los objetos**

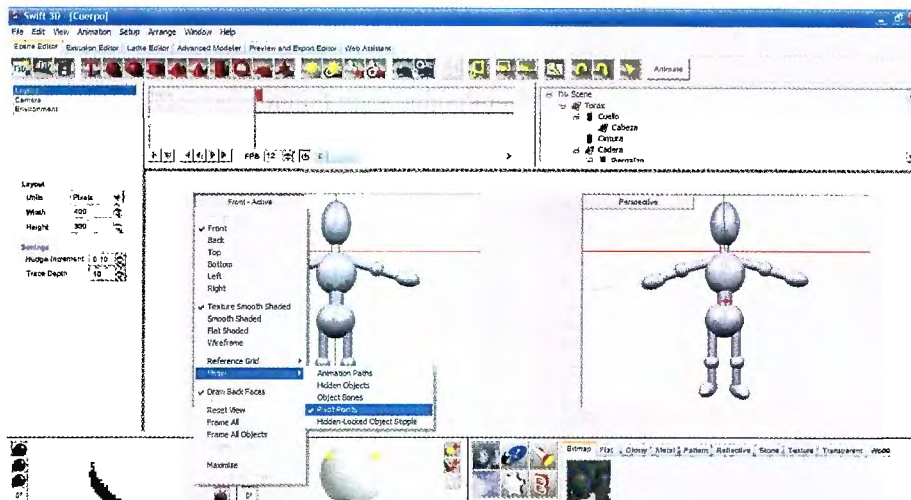
1. A continuación se creara una relación padre – hijo con los objetos. Esta relación sirve para que al mover o rotar un objeto Padre, este cambio también afecte al Objeto Hijo. Se creara arrastrando el objeto hijo sobre el objeto padre. Un objeto padre puede ser hijo de otro objeto padre.

2. En controles de animación, se hará la siguiente jerarquía.

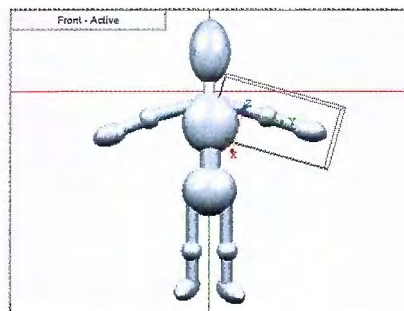


- Articulaciones del muñeco

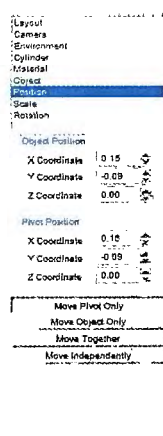
1. Para las articulaciones se deberá modificar los puntos pivote, para verlos se debe activar dando clic en la vista, ir al menú **Show** y seleccionar **Pivot Points**.



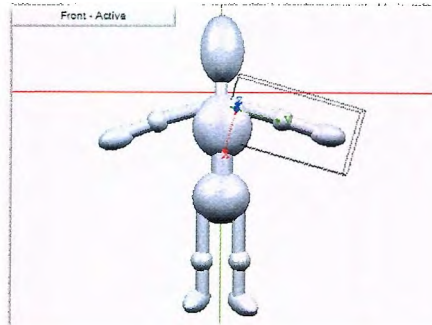
2. Para modificar la posición de un pivote se seleccionara un objeto, en este caso el brazo izquierdo, como se puede ver el pivote esta en el centro del objeto.



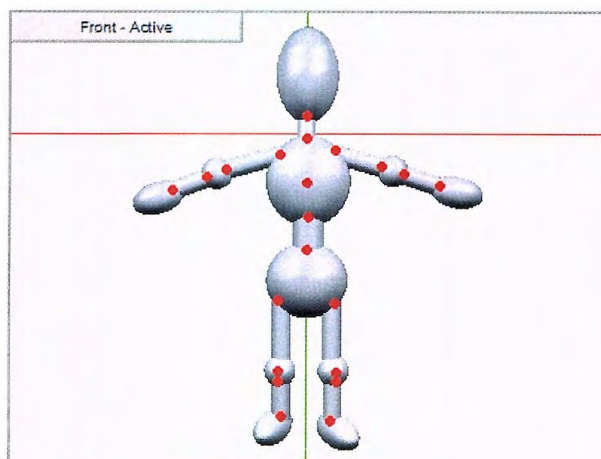
3. En las herramientas generales, seleccionaremos **Position** y daremos un clic al botón **Move Pivot Only**.



4. A continuación moveremos al pivote a la articulación donde se une el tórax y el brazo.

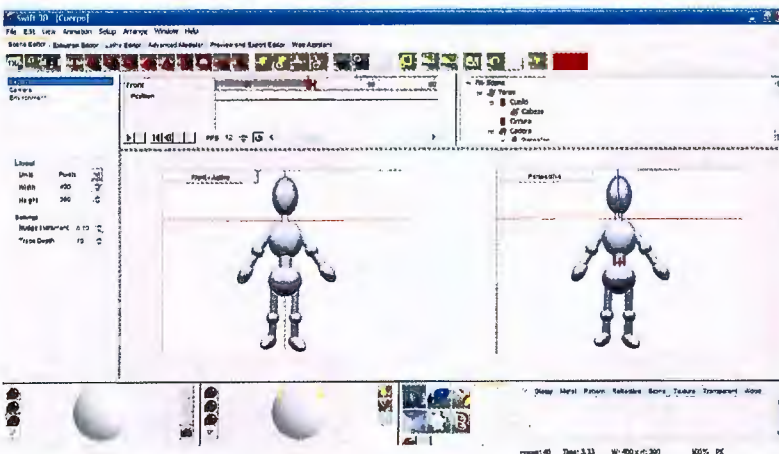
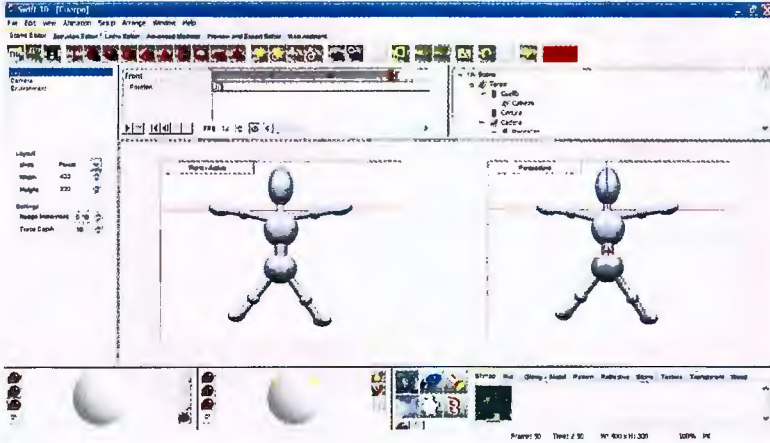
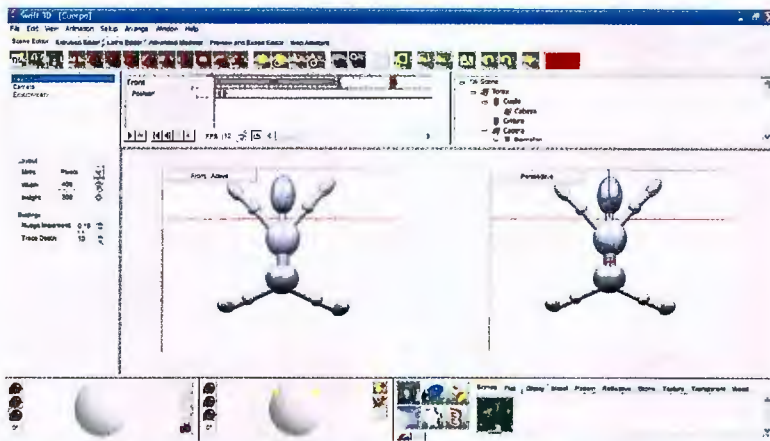
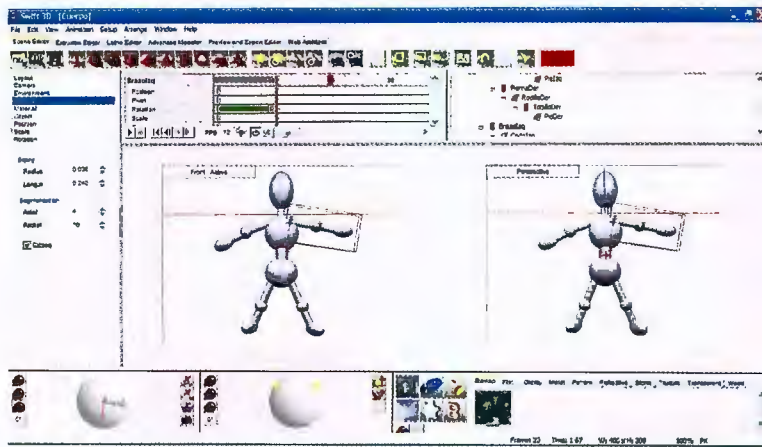


5. Se hará lo mismo para todas las partes de la figura.



- **Animación del Muñeco**

1. Para la animación del muñeco, solo se debe crear un esquema de movimiento, como saltar, correr y otros movimientos. Este muñeco se puede utilizar como que fuera un muñeco de cartón para entender el movimiento de los humanos.



## BIBLIOGRAFIA

- [1]. Donald Hearn, M. Pauline Baker. Gráficas por Computadora. Prentice-Hall Capítulo 1 pp 13.
- [2]. Norton. Introducción a la Computación 3ª edición. Editorial McGraw Hill. Capítulo 1 pp 18.
- [3]. Plastock, Roy. Teoría y Problemas de Gráficas por Computadora—1987 Capítulo 9 pp 348.
- [4]. IEEE, Computer Graphics and Applications Septiembre/Octubre,2005.
- [5]. Rafael Lahoz-Beltran. Bioinformática. Simulación, vida artificial e inteligencia artificial. 2004.
- [6]. Dr. Pere Marqués Graells. Los Videojuegos. 2001
- [7]. Rafael Lahoz-Beltran. Bioinformática. Simulación, vida artificial e inteligencia artificial. 2004.
- [8]. Serway, Beichner. Física Para Ciencias e Ingeniería. McGraw-Hill tomo 1 5ta edición Capítulo 3 pp 58-67
- [9]. Vector: todo segmento de recta dirigido en el espacio. Villalta Roberto Herrera Carlos Matemática Básica, 1998. Capítulo 4 pp 91.
- [10] Serway, Beichner. Física Para Ciencias e Ingeniería. McGraw-Hill tomo 1 5ta edición Capítulo 3 pp 64-65
- [11] Resnick,Halliday,Krane. Física Volumen 1 4ta Edición, Capítulo 3 pp 41-44
- [12] Villalta Roberto Herrera Carlos Matemática Básica, 1998. Capítulo 4 pp
- [13] Serway, Beichner. Física Tomo 1, 5ta Edición, McGraw-Hill, Capítulo 3 pp 61-64.
- [14] Igualdad de Vectores o Vectores Equipolentes, para mejor referencia consultar: D.C
- [15] Murdoch Geometría Analítica Con Vectores
- [16] Matrices. Limusa Noriega 1990. Capítulo 4 pp 78.
- [17] Kaplan, Wilfred, Cálculo y Algebra Lineal : Vectores del plano y cálculo de una variable Vol. I. Limusa 1978
- [18] Mosquera Carlos, Magnitudes Escalares y Vectoriales, Editorial CECSA pp 8-15.
- [19] D.C Murdoch. Geometría Analítica con Vectores y Matrices. capítulo 8 pp191
- [20] Norton, Meter. Introducción a la Computación. Mc-Graw Hill 2000. capítulo 5 pp 188.
- [21] Rosa Barbolla, Paloma Sanz. Algebra Lineal y Teoría de Matrices. Capítulo 2, pp 41-43.
- [22] Barbolla, Rosa, Algebra Lineal y Teoría de Matrices. Prentice Hall 1998
- [23] Corcobado Cartes Juan Luís MATEMÁTICA 1, Editorial CEI Cáceres, capítulo 2 pp 23-33
- [24] Dorf Richard C. Introducción al Álgebra de Matrices, Capítulo 1 pp 51 – 53.
- [25] Kaplan Matemáticas avanzadas Capitulo 5 pp 242.



- [26] Ayres jr. Frank. Matrices Editorial McGraw-Hill 1993, Capítulo 2. Matrices especiales.
- [27] Ayres jr. Frank. Matrices Editorial McGraw-Hill 1993, Capítulo 2. Matrices especiales.
- [28] Bourne , D.E. Análisis Vectorial y Tensores Cartesianos, editorial Limusa 1980
- [29] Pita Ruiz, Claudio. Cálculo Vectorial. Prentice Hall 1998.
- [30] Bourne-Kendall. Análisis Vectorial y Tensores Cartesianos. Editorial Limusa Capítulo 1, páginas 14-28
- [31] A. Cayley en 1858 quién define éste cálculo, novedoso en su tiempo. Fundamentos de Matemáticas para Física e Ingeniería. Editorial Limusa. Páginas 14-19.
- [32] Luís Sántalo. Enfoques hacia una Didáctica Humanística de la Matemática. Páginas 45-48
- [33] Scala Estrella Juan José. Análisis Vectorial Vol1. capítulo 2 Cosenos Directores. Páginas 45-53.
- [34] Higdon Archie. Estática Vectorial Tomo1. páginas 38-46
- [35] Alfa Con Estándares 10. Editorial Norma. Capítulo 5 Geometría Analítica Páginas 172-199
- [36] Corcobado Cartes Juan Luis. Matemática 1. Editorial Cáceres páginas 135-137.
- [37] Wong Benjamín. Dibujo Técnico Digital. Editorial Gustavo GILI SA. Capítulo Puntos y trazos páginas 25-27
- [38] Writh Richard S. OPENGL. Capítulo 2 Uso de OpenGL. Páginas 215-218
- [39] Hearn Donald Graficado por Computadora. Editorial Prentice Hall. Capítulo 3. páginas 59-63
- [40] Wong Benjamín. Dibujo Técnico Digital. Capítulo 3 Líneas Rectas páginas 35-38
- [41] Hearn Donal. Gráficas por Computadora. Editorial Prentice Hall Capítulo 3 páginas 61-63
- [42] Donal Hearn Gráficas por Computadora. Editorial Prentice Hall. Capítulo 3 Primitivas de Salida, páginas 63-64.
- [43] D.Foley James. Computer Graphics: Principles and Practice. Editorial Wesley. Capítulo 1. Large scale Computations. Páginas 66-78
- [44] Referencia 1: Tajadura Zapirain J.A / J.López Fernández. Autocad 2000 Avanzado. Editorial Mc.Graw Hill Capítulo 3 Creación y edición de Dibujos páginas 143-192
- [45] D.Foley James. Computer Graphics: Principles and Practice. Editorial Wesley. Capítulo 3. Points and Lines, páginas 133-145
- [46] Donald Hearn. Gráficas por Computadora. Capítulo 4 Atributos de Salida páginas 87-89

- [47] McClelland Decae. Drawing on the Pc. Editorial J.Lightell. Capítulo 2 Basic Drawing Theory. Páginas 44-76
- [48] Definición de Circunferencia, mejor referencia en Matemática Básica Preuniversitaria, Autor Mendoza William. UCA editores.
- [49] Raghavachary Saty. Rendering for Beginners. Editorial Focal Press.Capítulo 4. Geometric Primitives. Páginas 76-88
- [50] Dave shreiner. OpengGI Programing Guide. Editorial Addison Wesley Capítulo 3 Summary of Commands and Routines páginas 30-45
- [51] Thompson. Trigonometría. Aprenda Usted Mismo. Editorial Limusa 1993. Capítulo 1: Ángulos, Circunferencias y triángulos. Páginas 7-13
- [52] D.Foley James. Computer Graphics: Principles and Practice. Editorial Wesley. Capítulo 1. Large scale Computations. Páginas 88-123
- [53] Dave shreiner. OpengGI Programing Guide. Editorial Addison Wesley Capítulo 3 Summary of Commands and Routines páginas 30-45
- [54] Donald Hearn. Gráficas por Computadora. Capítulo 3 Primitivas de Salida, Páginas 74 -76
- [55] Donald Hear. Gráficas por Computadora. Capítulo 3 Primitivas de Salida páginas 75-77.
- [56] HEARN, D., y PAULINE BAKER, Gráficas por Computadora, 1995.
- [57] Dave shreiner. OpengGI Programing Guide. Editorial Addison Wesley Capítulo 3 Summary of Commands and Routines
- [58] Lengyel Eric.Mathematatics for 3D Game Programming And Computer Graphics. Capítulo 3 Transforms, páginas 54-55.
- [59] Hearn Donald Graficado por Computadora. Editorial Prentice Hall. Capítulo 5. página 116
- [60] Larson.Hosteller. Cálculo. Sexta Edición Volumen1.
- [61] Hearn Donald Graficado por Computadora. Editorial Prentice Hall. Capítulo 5. página 119.
- [62] Nemirovsky, The three-dimensional world – manipulations. Capítulos 8 y 9.
- [63] Marco Paluszny, Hartmut Prautzsch Métodos de Bézier y B-splines. Capítulo 1 Nociones básicas y Espacios. Página No. 3-6
- [64] Hearn, Donald, Gráficas Por Computadora.Editorial Prentice Hall. Capítulo 10 Representaciones Tridimensionales, páginas 197-202
- [65] Huw Jones, Computer Graphics. Through Key Mathematics. Editorial Springer Capítulo 7 Vectors: descriptions of spatial Relationship. Páginas 208-213

- [66] Lengyel Eric. Mathematics for 3D Game Programming and Computer Graphics. Editorial Charles Rivera Media. Capítulo 4. Ingeniería Geométrica. Páginas 83-88.
- [67] D. Foley, James. Computer Graphics: Principles and Practice. Editorial: Addison Wesley. Capítulo 4. Viewing in 3D páginas 230-242
- [68] D. Foley, James. Computer Graphics: Principles and Practice. Editorial: Addison Wesley. Capítulo 4. Viewing in 3D páginas 253-258
- [69] Buss Samuel R. 3D Computer Graphics: A Mathematical Introduction with OpenGL. Editorial. Cambridge University Press. Capítulo 2 Transformations and Viewing. Páginas 34 – 50.
- [70] Schneider Philip, Eberly David H Capítulo 4. Matrices, Vector Algebra and Transformations. Páginas 126-145
- [71] Lengyel Eric. Mathematics for 3D Game Programming and Computer Graphics. Editorial Charles Rivera Media, Inc. Capítulo 3 Transforms, páginas 58-64
- [72] Eberly David H. 3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics Editorial Morgan Kaufmann. Capítulo 2 Geometrical Methods, páginas 8-9
- [73] Jones Huw, Computer Graphics Through Key Mathematics, Capítulo 10 Drawing and Rendering Páginas 319-326
- [74] Lengyel Eric. Mathematics for 3D Game Programming and Computer Graphics. Editorial Charles Rivera Media, Inc. Capítulo 3 Transforms, páginas 60-62
- [75] Hearn Donal, Gráficas por Computadora, Editorial Prentice Hall, Capítulo 11 Transformaciones Tridimensionales, páginas 244-247.
- [76] Govil-Pai Shalini, Principles of Computer Graphics: Theory and Practice Using OpenGL and Maya, Editorial Springer. Capítulo 5 3D Modeling, páginas 104-107
- [77] Zárate Deras Juan Carlos, Alam Zamora Ana Luisa, Planificación de Trayectorias de Objetos Móviles en el Plano y Simulación Tridimensional de las Mismas. Tesis. Páginas 65 -73
- [78] Hearn Donald, Gráficas por Computadora. Editorial Prentice Hall. Capítulo 12: Vista Tridimensional, páginas 253-256
- [79] Lengyel Eric. Mathematics for 3D Game Programming and Computer Graphics. Editorial Charles Rivera Media. Capítulo 4. 3D Engine Geometry. Páginas 120-127.
- [80] Hearn Donald, Gráficas por Computadora. Editorial Prentice Hall, Capítulo 13 Supresión de Superficies y Líneas Ocultas. Páginas 280-294
- [81] Lengyel Eric. Mathematics for 3D Game Programming and Computer Graphics. Editorial Charles Rivera Media. Capítulo 15 Curves and Surfaces. Páginas 453-472
- [82] Hearn Donal. Gráficas por Computadora. Editorial Prentice Hall Capítulo 14 páginas 318.

- [83] D. Foley, James. Computer Graphics: Principles and Practice. Capítulo 13. Achromatic and Colored Light páginas 584-599
- [84] Buss Samuel R. 3D Computer Graphics: A Mathematical Introduction with OpenGL. Capítulo 6 Transformations and Viewing. Páginas 149-153.
- [85] Modelo RGB Hearn Donal. Gráficas por Computadora. Editorial Prentice Hall Capítulo 14 Modelos de Sombreado y Color. Páginas 322-323
- [86] Hearn Donal. Gráficas por Computadora. Editorial Prentice Hall Capítulo 14 páginas 308.
- [87] Eberly David H. 3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics Editorial Morgan Kaufmann. Capítulo 13 Special Effects páginas 430.
- [88] Pocock Lynn, Rosebush Judson The Computer Animator's Technical Handbook. Capítulo 3 The Science of Moving Picture. Páginas 64
- [89] Shalini Govil-Pai Principles of Computer Graphics: Theory and Practice Using OpenGL and Maya. Capítulo 6: Rendering, Shading and Lighting. Páginas 135
- [90] Shalini Govil-Pai Principles of Computer Graphics: Theory and Practice Using OpenGL and Maya. Capítulo 6: Rendering, Shading and Lighting. Páginas 136
- [91] David F Rogers, Rae A Earnshaw. Computer Graphics Techniques: Theory and Practice. Capítulo 2. Color in Computer Graphics. Editorial Springer. Páginas 50-69
- [92] Glassner Andrew S Principles of Digital Image Synthesis. Editorial MK. Capítulo 6 The Human Visual System and Color. Páginas 59-67
- [93] Lengyel Eric. Mathematics for 3D Game Programming and Computer Graphics. Editorial Charles Rivera Media. Capítulo 6. Illuminations. Páginas 163-167.
- [94] Huw Jones, Computer Graphics. Through Key Mathematics. Editorial Springer Capítulo 10 Drawing and Rendering: How to Create Picture. Páginas 321-323
- [95] Huw Jones, Computer Graphics. Through Key Mathematics. Editorial Springer Capítulo 10 Drawing and Rendering: How to Create Picture. Páginas 319-320
- [96] Lengyel Eric, Mathematics for 3D Game Programming and Computer Graphics. Editorial: Charles Rivera Media. Capítulo 6. Illuminations, páginas 166-167
- [97] D. Foley, James. Computer Graphics: Principles and Practice. Editorial: Addison Wesley. Capítulo 16. Illuminations and Shading páginas 745-753
- [98] Buss Samuel R. 3D Computer Graphics: A Mathematical Introduction with OpenGL. Editorial: Cambridge University Press. Capítulo 5. Texture Mapping .Páginas 127 – 138.
- [99] Eberly David H. 3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics Editorial Morgan Kaufmann. Capítulo 3 The Graphics Pipeline, páginas 105-109
- [100] Hearn Donald, Gráficas por Computadora. Editorial Prentice Hall, Capítulo 14 Modelos de Sombreado y Color. Páginas 311-316

- [101] Govil-Pai Shalini, Principles of Computer Graphics: Theory and Practice Using OpenGL and Maya, Editorial Springer. Capítulo 6 Rendering, Shading and Lighths , páginas 154-162
- [102] Dave Astle, Kevin Hawkins, Beginning Opengl Game Programming. Editorial: Thomson Course Technology. Capítulo 7. Texture Mapping. Páginas 169-177.
- [103] Lengyel Eric. Mathematic for 3D Game Programming and Computer Graphics. Editorial: Charles Rivera Media. Capítulo 6 Iluminations. Páginas 140-141
- [104] Zerbst Stefan 3D Game Engine Programming. Editorial: Thomson Course Technology. Capítulo 5: Materials, Textures and Transparency. Páginas 226-230.
- [105] Buss Samuel R. 3D Computer Graphics: A Mathematical Introduction with OpenGL. Capítulo 12 Animations and Kinematics, páginas 289-290.
- [106] Eberly David H. 3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics Editorial Morgan Kaufmann. Capítulo 4 Hierarchical Effects Representations páginas 147-148.
- [107] David F Rogers, Rae A Earnshaw. Computer Graphics Techniques: Theory and Practice. Editorial Springer. Capítulo 4. Animations. Páginas 237-253
- [108] Glassner Andrew S Principles of Digital Image Synthesis. Editorial MK. Capítulo 4 Rendering. Páginas 1057-1058

## **CONCLUSIONES**

1. Una monografía es una herramienta de estudio que requiere para su desarrollo de una investigación exhaustiva. Para este caso, la mayoría de la bibliografía consultada se encuentra en el idioma inglés, por lo tanto el proyecto representa una alternativa más de consulta para usuarios interesados en los conocimientos básicos de gráficas 3D. Además de significar para la Universidad Don Bosco gran respaldo para la cátedra de Gráficas por Computadora, al contar no solo con la monografía, sino también con la guía práctica de aprendizaje que proporciona el tutor web para los laboratorios de la clase.
2. Como segunda etapa del proyecto, se procedió a analizar algoritmos y técnicas más importantes según la temática en las diferentes unidades contenidas en la monografía, luego puestas en práctica bajo ejemplos que permiten ver su utilidad, diseño y aplicación.
3. La complejidad de gráficas por computadora, se ha extendido hasta técnicas de color, iluminación, sombreado, textura y animación. Dichas técnicas se desarrollan para llegar a la creación de objetos orientados a la realidad, por esto el tutor web contiene imágenes interactivas que están formadas con éstas característica.
4. OpenGL es una herramienta para el desarrollo de objetos 3D, creada como un conjunto de librerías gratuitas, las cuales también pueden ser descargadas del tutor web. El estudio monográfico, incluye ejemplos desarrollados bajo éste tipo de código, que también han sido incluidos en el tutor.



5. Swift 3D v 4 es una herramienta que permite graficado 3D en la web, lo cual abre nuevas puertas para diseño y aplicaciones con mejor vistosidad, aplicabilidad, tecnología y utilización de pocos recursos. La Unidad 6, ha sido desarrollada como una guía introductoria a este software, contiene el funcionamiento de barras de herramientas de diseño, su aplicabilidad y ejemplos paso a paso que muestran el proceso para realizar publicaciones 3D.
6. Parte importante del aprendizaje son las evaluaciones, para ello el proyecto incluye una aplicación desarrollada con la herramienta freeware Java Struts, donde se cuenta con exámenes a los usuarios y Administración de éstos, este último muy útil para maestros o instructores.

## **RECOMENDACIONES**

1. Se recomienda como propuestas para desarrollo de tesis, la ampliación del contenido de la monografía hacia los temas de Realidad Virtual, Renderización e Interfaz de Programación de Aplicaciones (APIs) entre otros, que no han sido incluidos en este proyecto debido a la cantidad de investigación que requieren, pero que son también parte importante del graficado por computadora, de igual forma la Universidad Don Bosco, no cuenta actualmente con libros al respecto, lo cual representaría para la clase de Gráficas por Computadora una alternativa de complementación y actualización del contenido de la asignatura.
2. El Software Tutor Web cuenta con un módulo de exámenes, que se recomienda pueda ser ampliado o mejorado en el módulo de redacción y aplicación de preguntas, por ejemplo desarrollando el programa para aplicar evaluaciones aleatorias y mejorando la seguridad del mismo.



# ANEXOS



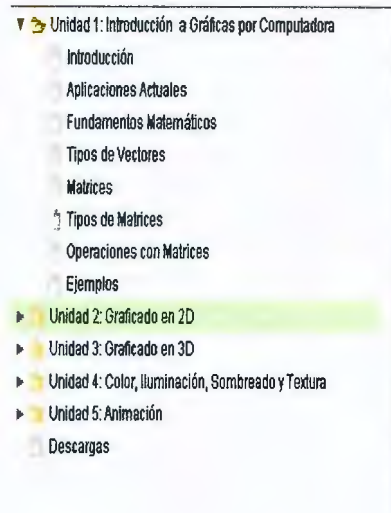
# TUTOR WEB PARA GRÁFICAS POR COMPUTADORA

El siguiente es el entorno principal del tutor:

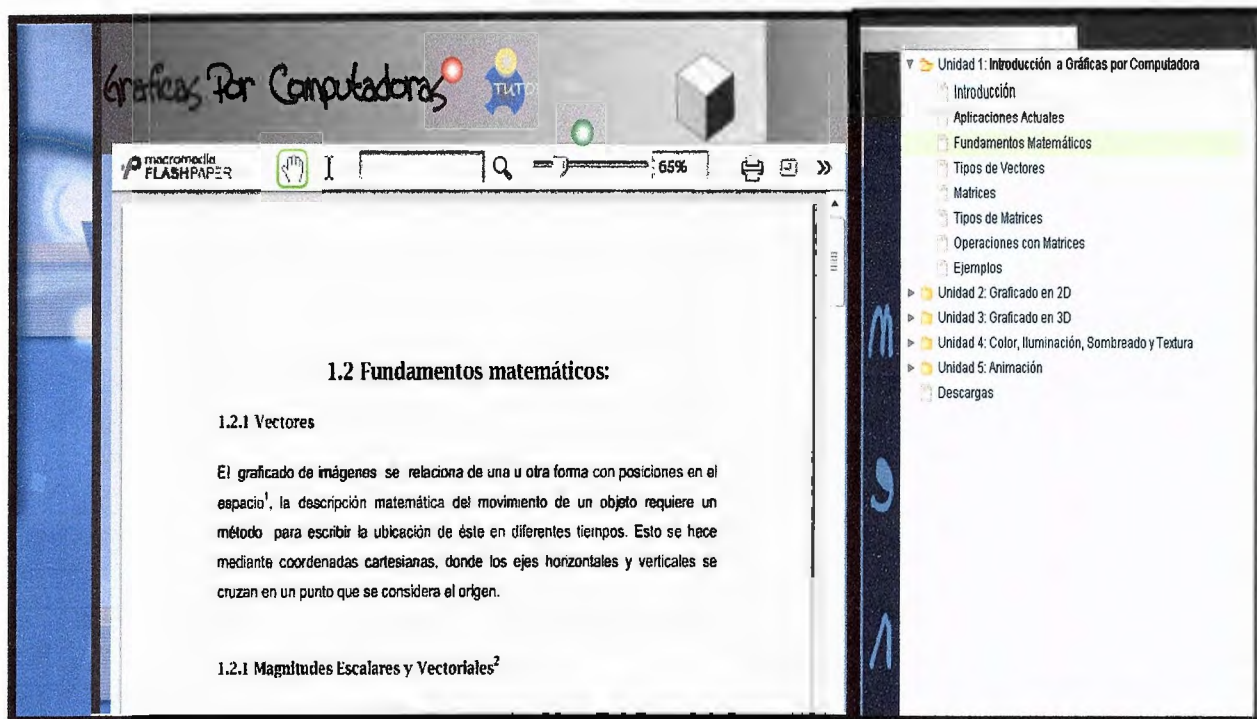


Al lado derecho de la pantalla, está un árbol de exploración conteniendo cada unidad de la monografía.

Al dar clic sobre las carpetas, éstas despliegan el contenido específico.

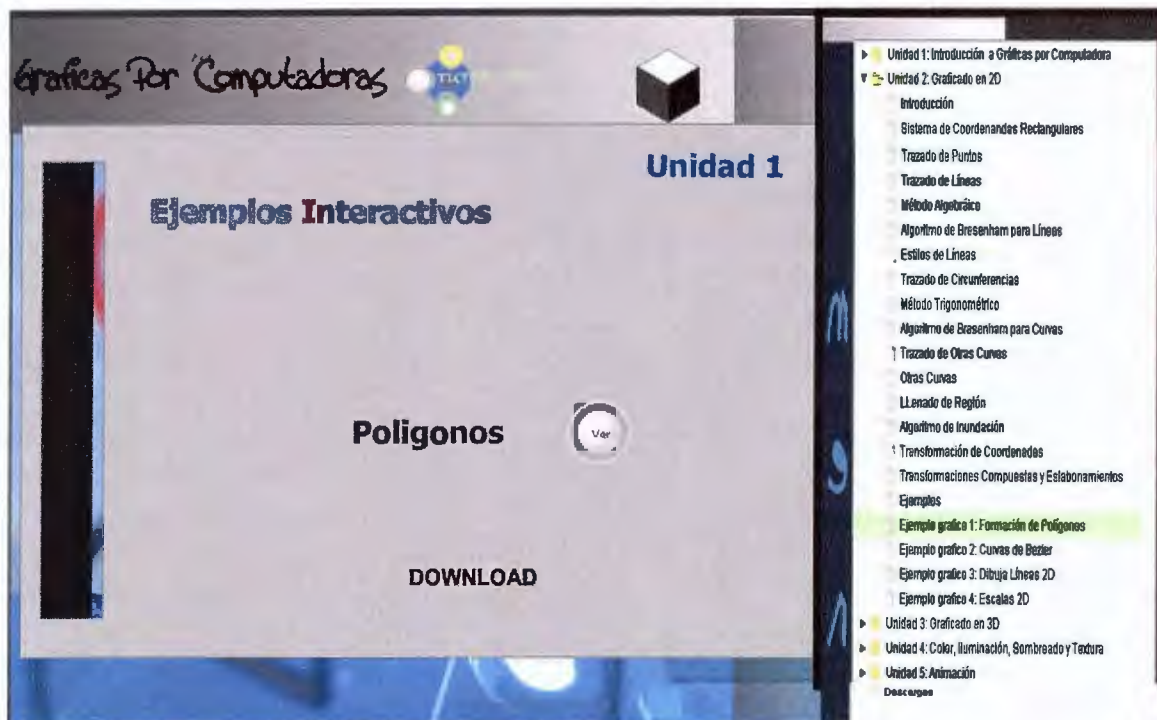


Al seleccionar uno de los temas el contenido es mostrado sobre la parte principal de la pantalla.



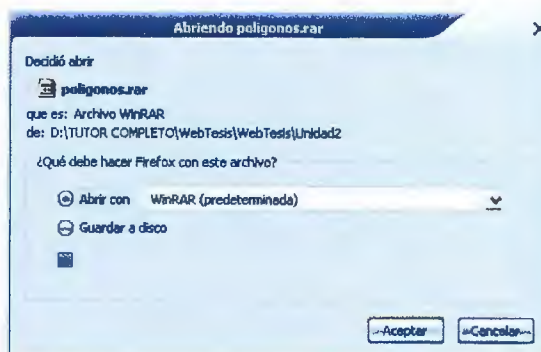
Aquí la información puede enviarse a impresión, o ampliarse según el usuario lo seleccione.

Al final de cada unidad (excepto la unidad uno), se encuentran ejemplos que pueden ser descargables.



Al dar clic sobre el botón VER, aparecerá otra ventana mostrando el ejemplo animado, que en la mayoría de los casos, son interactivos, es decir que el usuario podrá manipularlos y experimentar con ellos.

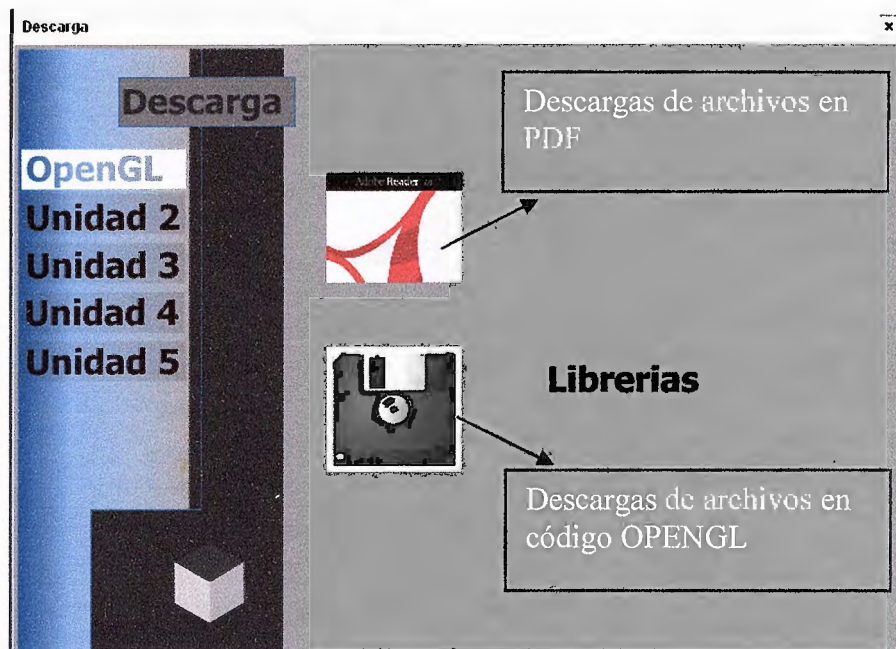
La opción de DOWNLOAD, permite al usuario descargar el ejemplo (en donde aparecerá un cuadro de diálogo), es decir el código principal en el cual se ha programado. Los ejemplos interactivos, han sido desarrollados con la herramienta Macromedia Flash y los ejemplos prácticos con código OpenGL.







Seleccionamos la opción GUARDAR EN DISCO y luego ACEPTAR, entonces aparecerá una ventana de diálogo para que el usuario indique la ubicación del archivo.

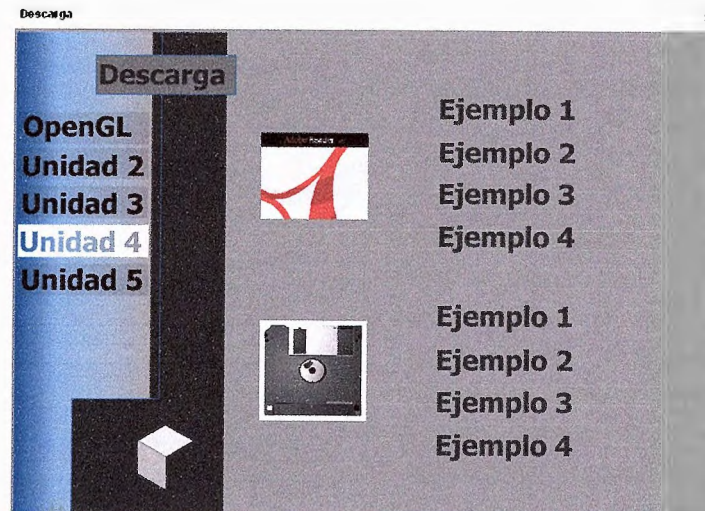
La opción DESCARGAS del menú, presenta una ventana con las siguientes opciones:



- **OPENGL:** para descargar las librerías OpenGL, damos clic sobre la opción **LIBRERIAS** y aparecerá el cuadro de diálogo para que indiquemos la ubicación en la que se guardará la carpeta.
- **Unidad 2:** aparecerá en la ventana, las opciones para visualizar el código de los ejemplos de la unidad en formato PDF, para que el usuario manipule la información según lo crea conveniente.



- Unidad 3, 4 y 5: de igual forma, al dar clic sobre la opción de de Ejemplo que indica la imagen del disco,  se descargarán los ejemplos en código OPENGL, y los que indica la imagen de Acrobat Reader  se visualiza el código de ejemplos en PDF.



Para regresar a la pantalla principal, simplemente se da un clic sobre el botón de CERRAR 

## Exámenes Tutor Web

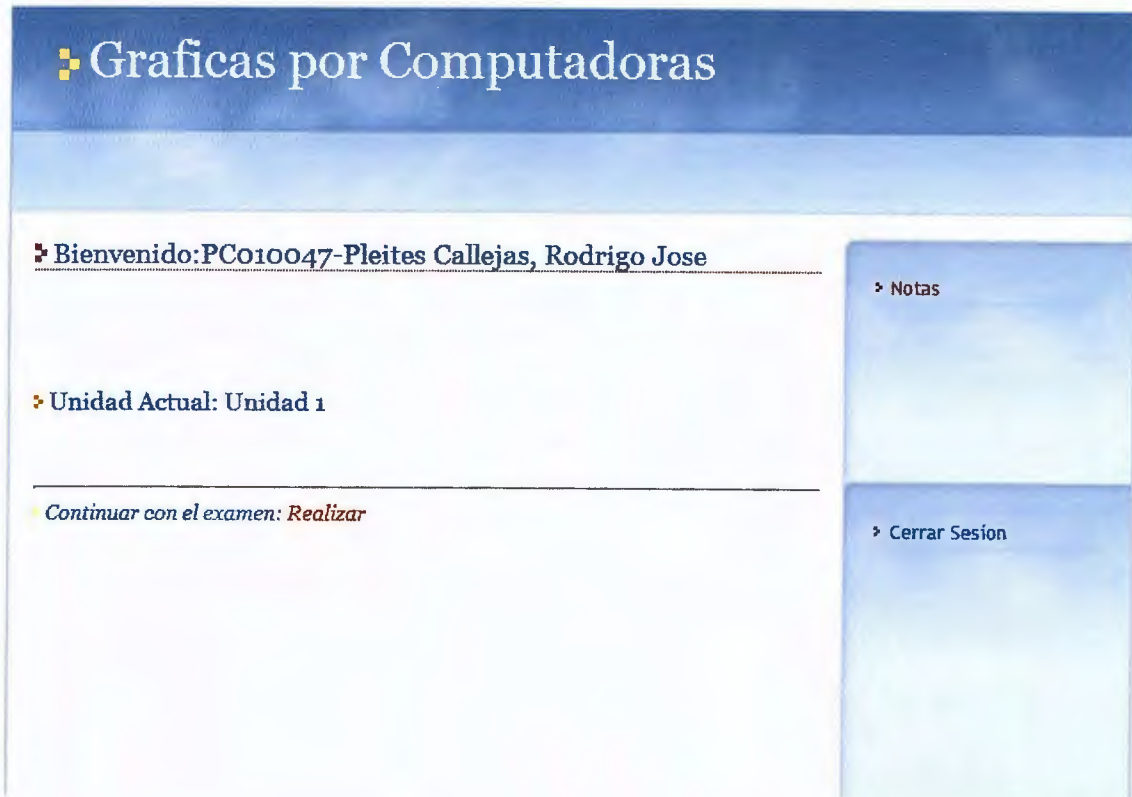
- ***Módo de Usuario.***

La siguiente es la pantalla de entrada a la sección de exámenes, acá se debe introducir el nombre del usuario y la contraseña correspondiente de cada alumno, para poder acceder al contenido del examen a realizar



Pantalla de login

Al acceder a la sección de exámenes se presenta la siguiente pantalla de bienvenida, esta informa sobre la unidad actual a la que se pertenece y el examen correspondiente a realizar de dicha unidad, también se tiene acceso al cuadro de notas y a cerrar la sesión.



Pantalla de bienvenida a la sección de exámenes



Al dar clic en la palabra realizar de la pantalla anterior, da inicio al examen y se presenta una pantalla como la siguiente, esta muestra una pregunta con respuestas de selección múltiple.

**Graficas por Computadoras**

---

**Unidad: 1-Pregunta: 1**

*Pregunta: ¿Cuál es la magnitud que pueden ser especificad completamente mediante un número y unidad de medida. ?*

Seleccione una de las respuestas:

- Respuesta 1: Magnitudes vectoriales.
- Respuesta 2: Magnitudes escalares.
- Respuesta 3: Ambas
- Respuesta 4: Ninguna

[> Notas](#)

[> Cerrar Sesión](#)

Pantalla de presentación de preguntas y respuestas

Al seleccionar la respuesta indicada y dar clic en el botón aceptar pasa a la siguiente pregunta y así consecutivamente hasta finalizar la serie de preguntas y respuestas. Desde aquí también se puede acceder al cuadro de notas y a cerrar la sesión.

Pantalla de presentación de notas por unidad.

# Graficas por Computadoras

Notas

One item found.  
1

Alumno							
PC010047	0	0	0	0	0	0	0

Notas

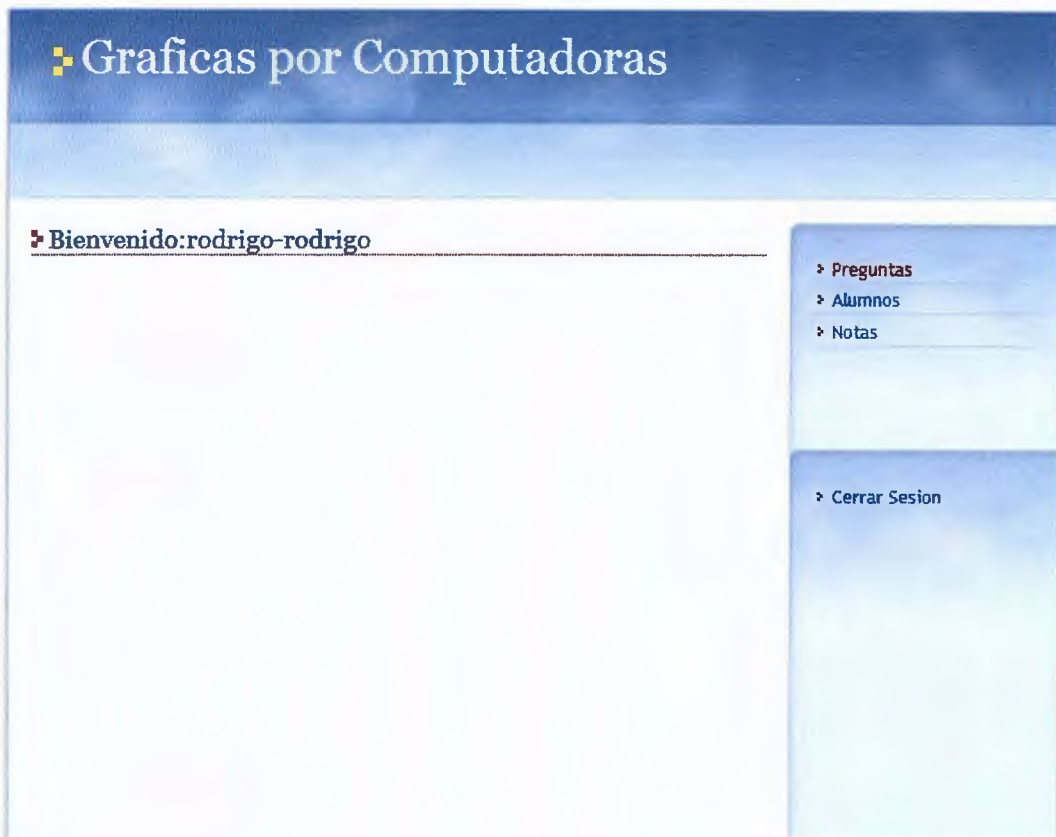
Cerrar Sesión



- **Módo de Administrador**

De la misma manera que entra un usuario como estudiante, el administrador cuenta con una clave especial que debe introducir en el login de la pantalla principal.

La ventana del administrador es la siguiente:



Contiene tres secciones:

- PREGUNTAS
- ALUMNOS
- NOTAS

## SECCIÓN DE PREGUNTAS

# Graficas por Computadoras

### Preguntas

Nuevo

8 items found, displaying 7 to 8.  
[First/Prev] 1, 2, 3 [Next/Last]

Codigo	Preguntas	Unidad	Acción
11	¿Cuál o cuales son las propiedades que se cumplen en el Producto de Matrices por un Escalar?	1	Update  Delete
10	¿Cómo se le llama a una matriz cuadrada que tiene todos sus elementos nulos excepto los de la diagonal principal que son iguales?	1	Update  Delete

- Preguntas
- Alumnos
- Notas

Cerrar Sesión

Esta es la ventana principal donde se crean y se administran las preguntas de los exámenes.

La opción UPDATE, permite modificar la pregunta y sus posibles respuestas.

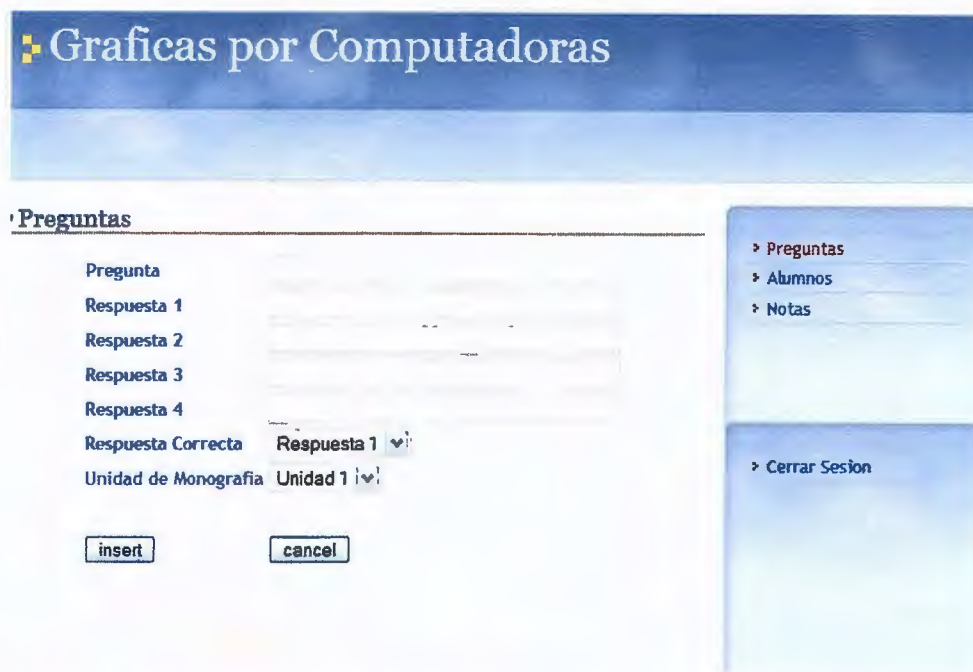
Y la opción DELETE, elimina la pregunta seleccionada, esta opción da antes un aviso de atención para asegurar que la pregunta será eliminada o no.

## AGREGAR PREGUNTAS AL EXAMEN

Para crear una nueva pregunta, el administrador debe dar clic sobre **NUEVO** en la parte superior izquierda de la ventana.

Nuevo

Y se carga la siguiente ventana.



The screenshot shows a web application window titled "Graficas por Computadoras". Below the title bar, there is a sidebar on the right with a tree view containing "Preguntas", "Alumnos", and "Notas". The main content area is titled "Preguntas" and contains a form with the following fields: "Pregunta" (a large text area), "Respuesta 1", "Respuesta 2", "Respuesta 3", "Respuesta 4", "Respuesta Correcta" (a dropdown menu currently showing "Respuesta 1"), and "Unidad de Monografia" (a dropdown menu currently showing "Unidad 1"). At the bottom of the form are two buttons: "insert" and "cancel".

**Pregunta:** Se digita la pregunta que se desea realizar en la evaluación

El sistema da 4 posibles respuestas alternativas. Y la lista desplegable de la opción **RESPUESTA CORRECTA**, como su nombre lo indica, se asigna la respuesta que será la correcta en el examen. Luego se selecciona la opción **UNIDAD DE MONOGRAFÍA**, que indica a cuál unidad corresponde la pregunta que se ha creado.

Luego se da clic sobre el boton

insert

y la pregunta que automáticamente asignada a la unidad a evaluar.

## SECCIÓN DE ALUMNOS

Al dar clic sobre el link de ALUMNOS en la venta principal del administrador, se tiene lo siguiente:

The screenshot shows a web application titled "Graficas por Computadoras". Below the title bar, there is a navigation menu with links: "Preguntas", "Alumnos", and "Notas". The "Notas" link is currently selected. The main content area displays the text "4 items found, displaying all items." followed by a table of student records. The table has five columns: "Codigo", "Alumno", "Materia", "Apellidos", and "Action". There are four rows of data, each representing a student with their ID, name, subject, last name, and a link to view their notes.

Codigo	Alumno	Materia	Apellidos	Action
5	PM981124	Eva Maria	Perez Majano	Notas
4	JM990224	Jessica Mariel	Jovel Mejia	Notas
3	PC010047	Rodrigo Jose	Pleites Callejas	Notas
2	CS990345	Esther Emilia	Castillo Sanchez	Notas

On the right side of the page, there is a sidebar with a "Cerrar Sesion" link.

Si existen alumnos registrados, aparecerá el listado correspondiente, sino la parte central de la venta estará en blanco.

Esta parte del administrador, al igual que en la sección de preguntas, permite adicionar eliminar y modificar los nombres de los alumnos.

## ADICIONAR ALUMNOS

The screenshot shows a web application interface. At the top is a dark blue header with the text 'Graficas por Computadoras' in white. Below the header is a light blue horizontal bar. The main content area has a white background. On the left, there is a section titled 'Alumnos' with a small icon to its left. Below this title are three text input fields labeled 'Nombres', 'Apellidos', and 'Carne'. Below these fields are two buttons: 'insert' and 'cancel'. On the right side of the page, there is a vertical sidebar with a light blue background. It contains a list of links: 'Preguntas', 'Alumnos', 'Notas', and 'Cerrar Sesion'. The 'Alumnos' link is highlighted.

**Graficas por Computadoras**

**Alumnos**

Nombres

Apellidos

Carne

› Preguntas

› Alumnos

› Notas

› Cerrar Sesion

Para agregar uno nuevo registro de alumno, solo se da clic sobre NUEVO y se llenan los campos necesarios, dando clic en INSERT para que se realicen los cambios ya sea de modificación o adición de estudiantes.



## CONTROL DE NOTAS.

**Graficas por Computadoras**

**Notas**

One item found.  
1

Alumno	Unidad 1	Unidad 2	Unidad 3	Unidad 4	Unidad 5	Unidad 6	Promedio
PC010047	2	0	0	0	0	0	0

Navigation links: Preguntas, Alumnos, Notas, Cerrar Sesión

Automáticamente las notas se van reflejando por orden de número de carnet. Para visualizar esto, se da clic sobre NOTAS y aparecen todos los alumnos que han realizado exámenes.

Las notas se visualizan según la unidad evaluada y al final el promedio.