49
24.

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
CAMPUS ARAGON

GENERACION DE ANIMACIONES A PARTIR DE DATOS OBTENIDOS DE UNA DESCRIPCION MATEMATICA DE FENOMENOS FISICOS REALIZADA EN COMPUTADORA"

T E S I S

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACION

P R E S E N T A :

MARCELO PEREZ MEDEL

ASESOR: DR ENRIQUE DALTABUIT GODAS

MEXICO **1997.**

Centro Tecnológico Aragón

Lic. Alberto Ibarra Rosas
Jefe de la Unidad Académica
Escuela Nacional de Estudios Profesionales Aragón
Universidad Nacional Autónoma de México
Presente.

Estimado Lic. Ibarra:

A petición del interesado me permito informarle que el alumno de la carrera de Ingenieria en Computacion con numero de cuenta 8718819-0 Sr. Marcelo Perez Medel ha terminado con esta fecha la tesis que ha registrado con el nombre "Generación de Animaciones a Partir de Datos Obtenidos de una Descripcion Matemática de Fenómenos Físicos Realizada en Computadora".

Hago propicia la ocasión para enviarle un cordial saludo.

A T E N T A M E N T E
"POR MI RAZA HABLARÁ EL ESPÍRITU"
Ciudad Netzahualcoyolt, Estado de México a 25 de agosto de 1997.

Dr. Enrique Daltabuit
Director de Tesis

Vo. Bo.

Ing Juan Gastaldi Perez

Tel. (52)5-623.0960

Av. Hacienda de Rancho Seco s/n
Fracc. Bosques de Aragón
Nezahualcoyotl, Edo. de México
C.P. 57170

Fax (52)5-623.0864

AGRADECIMIENTOS :

Agradezco :

A Dios, por la vida y la oportunidad.

A mi mamá y a mi papá (Raquel Medel y Marcelo Pérez) los cuales me encaminaron en esta senda y de los cuales tomé lo fundamental de mi comportamiento.

A mi esposa Carmen Delia por su apoyo y su comprensión en estos últimos cuatro años, además de ser para mí una gran motivación.

A mis hijos Marcelo Daniel y Dulce Raquel, los cuales son mi orgullo y mi esperanza.

Que Dios los Bendiga.

Agradezco :

A la Universidad Nacional Autónoma de México donde he recibido gran parte de mi formación y en la cual he podido conocer muchos de los conceptos de la cultura y la ciencia.

A mis profesores del Campús Aragón, que se han esforzado en formar profesionistas del más alto nivel.

Al Dr. Enrique Daltabuit Godas, asesor de esta tesis, el cual ha sabido orientar y asesorar de excelente forma el esfuerzo invertido en este trabajo.

TITULO:

GENERACION DE ANIMACIONES A PARTIR DE DATOS OBTENIDOS DE UNA DESCRIPCION MATEMATICA DE FENOMENOS FISICOS REALIZADA EN COMPUTADORA.

HIPOTESIS:

ES POSIBLE REALIZAR ANIMACIONES QUE SIMULEN UN TIRO PARABOLICO CON UN ALTO GRADO DE REALISMO EMPLEANDO PROGRAMACION EN LENGUAJE PASCAL CON EL APOYO DE PROGRAMAS DE DOMINIO PUBLICO Y PROGRAMAS COMPARTIDOS.

OBJETIVOS GENERALES

° CREACION DE UN PROGRAMA EN PASCAL, CAPAZ DE RECIBIR PARAMETROS Y OBTENER COMO SALIDA DATOS, EMPLEANDO FORMULAS DE FISICA CLASICA. EN ESTE TRABAJO SE EMPLEARA COMO EJEMPLO EL TIRO PARABOLICO.

° EVALUAR LAS CARACTERISTICAS DE LOS PROGRAMAS FREEWARE Y SHAREWARE (PREFERENTEMENTE CON CODIGO FUENTE), QUE PUEDAN SER EMPLEADOS EN LA GENERACION DE ANIMACIONES, TALES COMO : GENERADORES Y VISUALIZADORES DE FORMAS, TEXTURAS, BRILLOS, SOMBRAS, REFLEJOS, ETC. CONVERTIDORES DE GRUPOS DE IMAGENES A ANIMACIONES Y VISUALIZADORES DE ANIMACION.

° ELABORAR UNA TESIS QUE PUEDA SERVIR DE BASE A POSTERIORES TRABAJOS DE SIMULACION Y ANIMACION MAS ELABORADOS.

OBJETIVOS PARTICULARES

° DETERMINAR LOS LIMITES FISICOS QUE ES POSIBLE DETERMINAR EN FORMA REALISTA :

A) EN EL TIEMPO
B) EN EL ESPACIO
C) EN LA MENTE DEL OBSERVADOR

° ANALIZAR EL PROBLEMA DE REPRESENTAR EN FORMA DISCRETA UN FENOMENO CONTINUO

° RESOLVER PROBLEMAS RELATIVOS A EL ALMACENAMIENTO Y VISUALIZACION DE LA ANIMACION.

° ELABORACION DE UN MODULO DE ENTRADA DE DATOS, CAPAZ DE VALIDAR DATOS, DE FACIL MANEJO Y CON VARIAS ALTERNATIVAS TALES COMO RESOLUCION DE LA ANIMACION, VARIACION DE POSICION DE LA CAMARA, ETC.

° REALIZACION DE UN MODULO EN LENGUAJE DE PROGRAMACION PASCAL QUE TOME COMO ENTRADA LA VELOCIDAD INICIAL Y EL ANGULO DE LANZAMIENTO Y A PARTIR DE ESTOS DATOS DEVUELVA ALTURA MAXIMA, TIEMPO DE VUELO Y POSICION DEL OBUS EN CADA INSTANTE.

° ELABORACION DE LAS RUTINAS NECESARIAS PARA GENERAR ARCHIVOS QUE CONTENGAN LOS DATOS OBTENIDOS DEL MODULO DE CALCULOS, Y ESTOS ARCHIVOS SE ENCUENTREN EN UN FORMATO ADECUADO PARA SER LEIDO POR UN PROGRAMA DE GENERACION DE REPRESENTACIONES DE SUPERFICIES DE CALIDAD FOTOGRAFICA.

° IMPLEMENTACION EN PASCAL DE RUTINAS QUE REALICEN CICLOS QUE SE REQUIEREN PARA GENERAR TODOS LOS CUADROS DE LA ANIMACION, PARA ESTO SE EMPLEAN LLAMADAS AL MODULO QUE GENERA LOS ARCHIVOS PARA EL RENDERIZADOR, CON LOS PARAMETROS CONVENIENTES.

° EMPLEAR EN SU MAYORIA OPCIONES ESTANDAR DE UN LENGUAJE DE GRAN DIFUSION (PASCAL).

° BUSCAR EN INTERNET O BOLETINES ELECTRONICOS PROGRAMAS DE DOMINIO PUBLICO O COMPARTIDOS, LOS CUALES SEAN FIABLES, Y PREFERENTEMENTE ESTEN ACOMPAÑADOS DE CODIGO FUENTE (SOLO PARA VERIFICAR QUE HAGAN LO QUE TIENEN QUE HACER Y NO SON PROGRAMAS MALICIOSOS), LA POSIBILIDAD DE INTERNET PERMITE QUE MUCHOS PROGRAMADORES PUEDAN COMPARTIR SUS EXPERIENCIAS Y APROVECHANDO ESTO SE PRETENDE APOYARSE EN ALGUNOS DE ESTOS PROGRAMAS PARA REALIZAR UN TRABAJO QUE AGRUPA VARIOS MODULOS QUE PODRIAN REPRESENTAR UNA TESIS CADA UNO. LOS PROGRAMAS A EVALUAR SERAN DE SINTESIS DE IMAGENES, CONVERTIDORES DE GRUPOS DE IMAGENES EN FORMATO DE ANIMACION.

°LOGRAR QUE LA ANIMACION GENERADA TENGA UN GRAN REALISMO, PARA LO CUAL SE TOMARAN EN CUENTA FACTORES TALES COMO:

- PERSISTENCIA DE LA VISION
- ESCALA DE VISUALIZACION
- CALIDAD DE LA REPRESENTACION

TEMARIO

PROLOGO

Los computadoras han tenido un avance vertiginoso, y los programas de animación y simulación han sido muy beneficiados en dicho avance, en el momento actual (Mediados de 1997) la animación tiene múltiples aplicaciones y a corto o mediano plazo tiene perspectivas muy interesantes, por lo que vale la pena elaborar investigaciones al respecto.

Otro aspecto que me ha hecho decidir realizar un proyecto de tesis sobre animación y simulación es el hecho de que son los temas que más me han gustado desde que inicié mi formación en informática, desde mi primera computadora (una Commodore 16 con sólo 16 Kb de RAM) mis programas favoritos fueron los programas gráficos.

El enfoque de este proyecto es diferente a los que normalmente se han realizado, porque es de carácter práctico, pero no busca realizar un sistema completo a partir de cero, sino que busca aglutinar varios programas más para realizar un programa que en conjunto sea más sencillo, escalable y flexible.

INTRODUCCION

INTRODUCCION

¿Quién no ha visto animación realizada en computadora ? la respuesta es sencilla, las personas que nunca ven los programas o anuncios en la televisión y además nunca van al cine.

Las animaciones por computadora han invadido prácticamente todos los lugares, los anuncios, los efectos especiales de las películas (Toy Story es una película hecha casi en su totalidad por computadora) y en general todos aquellos lugares donde se empleen computadoras.

Una animación básicamente es una secuencia de imágenes, para lograrla se emplean modelos a los cuales se les modifican sus propiedades en función del tiempo logrando así la sensación de movimiento, existen varias formas de generar las animaciones :

A) Basadas en guiones : Este tipo de animación consiste en el movimiento de los objetos que se encuentran en escena a través de trayectorias predefinidas, el éxito en la obtención de realismo depende de la habilidad del creador para definir las trayectorias y movimientos.

B) Basados en actores vivos : Para la realización de este tipo de animaciones se requiere de costosos equipos, ya que se requieren de trajes provistos de sensores, que captan los movimientos de los actores que los llevan puestos, se logra mucho realismo para personajes de características humanas, pero los demás objetos no consiguen movimientos tan naturales. Como ventaja tiene un importante ahorro en el tiempo de diseño, ya que todos los movimientos se obtienen de los actores, su principal desventaja que tiene que los movimientos tienen las limitaciones de las personas que sirvieron de modelo.

C) Basados en física : Aquí es donde se logra mayor realismo, ya que los objetos en escena se comportan de forma muy semejante a como lo harían en realidad. A este tipo de animación de simulación se le denomina ciencia numérica y esta comenzando a tener un desarrollo muy impresionante.

Por otra parte la animación es una de las ramas de más rápido crecimiento de la industria de cómputo, generando cantidades millonarias tanto en las industrias del entretenimiento (nintendo 3D, películas, juegos, etc) como en industrias de diseño y manufactura industrial pasando por los centros de investigación y en general cualquier lugar donde se beneficie del manejo de imágenes animadas.

El desarrollo en México tanto de la simulación como de la animación es muy limitado, en su mayoría son programas importados y únicamente hay manejo a nivel de usuario.

CAPITULO 1

ANTECEDENTES DE LA SIMULACION Y LA ANIMACION POR COMPUTADORA.

CAPITULO 1. ANTECEDENTES DE LA SIMULACION Y ANIMACION POR COMPUTADORA

1.1 TERMINOS EMPLEADOS EN SIMULACION Y ANIMACION POR COMPUTADORA

Para poder entender bien este trabajo, primero necesitamos conocer los términos que en éste se emplean, por lo cual he considerado necesario empezar por definir algunos términos que se emplearan a lo largo de todo el trabajo. Las definiciones (Algunas son más bien explicaciones que definiciones) están ordenadas primero las que corresponden a la animación y posteriormente las que se refieren a simulación, y a su vez están ordenadas en un orden que busca ser lógico.

Conceptos de Animación

Pixel : Es la unidad mínima de dibujo, su nombre proviene de la contracción de las palabras anglosajonas "picture element" (Elemento de Dibujo). Para cuestiones prácticas podemos considerar que un pixel es un punto en el monitor o impresora.

Resolución : Es el número de pixeles por unidad de área.
A mayor resolución es mayor la calidad de la imagen, tomemos como ejemplo las imágenes fotográficas : si el grano es pequeño la imagen es más nítida, pero para aquellos que no están muy familiarizados con términos fotográficos pongamos otro ejemplo. Si tomamos una imagen y la desplegamos a una resolución de 32 x 24 (es decir, 32 pixeles de ancho por 24 pixeles de alto), y si esa misma imagen la mostramos a 320 x 240 tendremos una imagen mucho más clara pero que ocupa 100 veces más espacio.

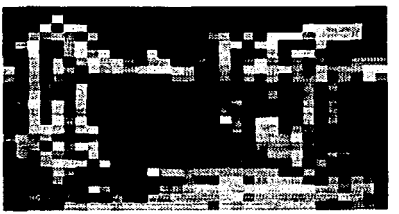


Imagen de 32 x 24

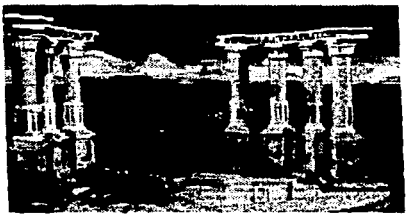


Imagen a 120 x 100

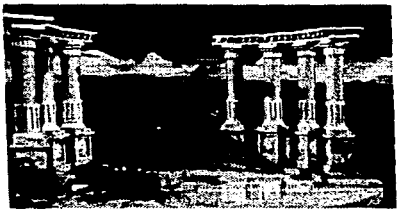


Imagen a 320 x 240

Relación Anchura/altura (Aspec/Ratio) : Es la relación entre la altura y la anchura, por ejemplo los televisores tienen una relación de anchura/altura de 4/3 por que si dividimos el tamaño horizontal entre el vertical obtenemos 4/3.

Este concepto es importante ya que debido a que existen diferentes formatos que se emplean para realizar animación, es necesario realizar adaptaciones al pasar de un formato a otro, para evitar que la animación resultante sufra deformaciones.

Cuadro (Frame) : Se denomina así a una imagen individual que forma parte de una animación. También se le denomina fotograma.

Cuadros Por Segundo (Frames Per Second - FPS) : Los cuadros por segundos indican el número de cuadros (Imágenes Individuales) que son exhibidas en un segundo.

La cantidad de cuadros por segundo es de mucha importancia en la elaboración de una animación, por ejemplo el formato del cine esta estrandarizado a 24 cuadros por segundo, mientras que la televisión lo está a 30 cuadros por segundo. La cantidad de cuadros que se requieren desplegar en un segundo se analiza con detalle en el capitulo 2.

Paleta de Colores : La paleta de colores es el conjunto de las posibles opciones de color: en la computadora se maneja de la siguiente forma :

Paleta de cuatro colores, para codificar cuatro colores se requieren de 2 bits y se codifican de esta forma :

00 Color 1
01 Color 2
10 Color 3
11 Color 4

Antes de emplear una paleta se especifican cuales serán los colores de dicha paleta.

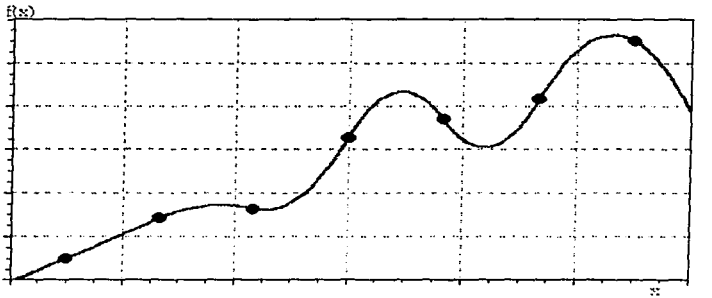
Si queremos manejar una cantidad mayor de colores necesitamos más bits, a continuación una tabla donde se muestran cuantos bits se requieren para varios tamaños de paleta

Número de Bits	Número de Colores Posibles
2	4
4	16
8	256

Cámara : Es la posición imaginaria del observador dentro de una escena.

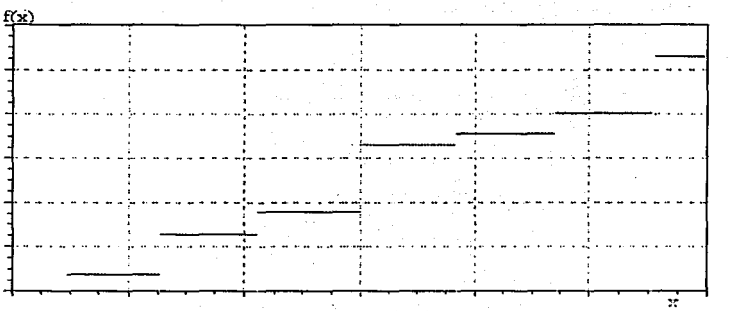
Conceptos de Simulación

Continuo : Cuando hablamos de una función continua nos referimos a una que tiene un número infinito de posibles valores en un rango tan pequeño como se quiera, pongamos como ejemplo la temperatura, ¿Qué temperatura tiene un cuerpo ? bien puede tener 1.5 °C bien y si le aplicamos un poco de calor ¿Qué temperatura tendrá ahora ?, tal vez 1.6 °C pero también podríamos tener 1.55 o cualquier valor entre 1.5 y 1.6, puede existir un número infinito entre una temperatura y otra, otro ejemplo podrían ser las medidas de longitud. En otras palabras (tal vez más simplistas y prácticas para este trabajo) podemos decir que una función es continua si podemos dibujarla sin despegar el lápiz del papel.



Forma de una Gráfica de una Función Continua

Discreto : Las Funciones Discretas son aquellas que aunque x sea continua f(x) no lo es, para un rango de valores de x existe solo un valor para f(x). Una Función es Discreta si no podemos dibujarla sin despegar el lápiz del papel.



Forma de una Gráfica de una Función Discreta

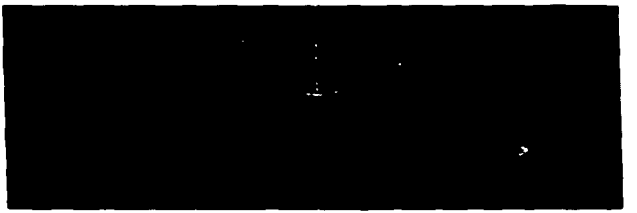
Simulación : Simulación en computadora, consiste en el empleo de modelos matemáticos que puedan emplearse para realizar aproximaciones al comportamiento real.

1.2 ANTECEDENTES DE LA SIMULACION POR COMPUTADORA

La simulación por computadora nació prácticamente desde el momento mismo que nació la computación. La ENIAC (Electronic Numerical Integrator and Calculator) fue construida entre 1943 y 1946 en la escuela de Ingeniería Electrónica Moore, el cual fue un proyecto financiado por el Ballistic Research Laboratory. John W. Mauchly y J. Presper Eckert fueron los responsables de los trabajos de realización.

La ENIAC funcionaba con bulbos (tubos de vacío) a diferencia de sus antecesores, los cuales trabajaban con reles (relevadores electromagnéticos) , una de sus principales aplicaciones fue la realización de grandes tablas de balística, lo cual es sin duda de los primeros manejos de modelos matemáticos en computadora.

En la actualidad la simulación de fenómenos físicos ha evolucionado mucho, de hecho una gran parte de la investigación actual se apoya en modelos matemáticos mas que en los fenómenos reales



1.3 ANTECEDENTES DE LA ANIMACION POR COMPUTADORA

Al principio las primeras computadoras no tenían terminales gráficas y eran muy lentas, de este modo los únicos gráficos que se podían mostrar en aquellos primeros días de la informática eran imágenes simples formadas a partir de caracteres de texto.

Las IBM PC XT (Aparecieron en el mercado alrededor de 1980) poseían un monitor CGA (Color Graphics Array) el cual permitía una resolución de 320 x 200 con una paleta de 4 colores, estaban pues muy limitados, pero ya se podían visualizar algunas formas. Las computadoras personales de esta época eran de capacidades un tanto inferiores (aunque varias tenían mucha mayor capacidad gráfica).

La compañía Commodore tenía en el mercado las computadoras Commodore 64, Commodore 16 y Vic 20, (de hecho mi primera computadora fue una Commodore 16), las cuales se conectaban directamente a la televisión, lo cual de muchas formas era una ventaja, sobre todo si se quería grabar alguna imagen en video. Esta compañía sacó al mercado una nueva serie de ordenadores en 1985, la computadora Amiga, la cual permitía un manejo gráfico impresionante, recuerdo haber visto una animación de un muñeco que realizaba malabares con unos pequeños objetos (una pelota y un cubo si no mal recuerdo).

Los avances han sido enormes desde entonces, algunos de los avances más relevantes fue en el procesamiento, la capacidad de almacenamiento y dispositivos de despliegue, a continuación una tabla aproximada de las diferencias entre equipos :

Aspecto	1980-1982	1997 (Mayo - Junio)
Procesamiento	Micro de 8 Bits 4.7 MHz	Micro de 64 Bits a 200 MHz
Almacenamiento	10 Mbytes	1.7 Gbytes
Dispositivo de Despliegue	320 x 200 4 colores	1024 x 768 16.7 Millones de Colores

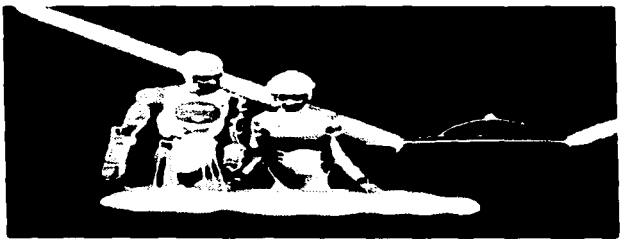
Como se aprecia en la tabla de arriba los avances son enormes, esto es solo en el aspecto de computadoras personales, en cuanto a estaciones de trabajo los avances han sido similares, las capacidades de una estación de trabajo de nivel bajo son típicamente :

Microprocesador de 150 MHz, entrada y salida de sonido y video , monitores de 1280 x 1024 e incluso una pequeña cámara de video. (esto en una estación de trabajo Indy de Silicon Graphics)

La evolución de la animación por computadora puede observarse en las películas, a continuación se muestran algunas imágenes de algunas películas muy representativas.

TRON

Este film es histórico, ya que fue de los primeros que emplean síntesis de imágenes (1982)



Parque Jurásico (Jurassic Park), esta película reúne grandes avances técnicos, donde grandes segmentos de video incluyen síntesis de imágenes en su realización

Fotograma de la Película Parque Jurásico

Pero la película que ha destacado más en la generación de animaciones por computadora es Toy Story, la cual está elaborada casi en su totalidad por medio de equipo de cómputo. Son tan espectaculares sus avances que a continuación se presentan unos cuantos datos .

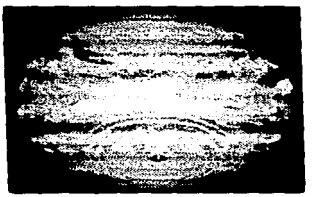
- 79 minutos de animación realizada en computadora
- 114,240 Cuadros de animación computarizada
- Más de 400 modelos empleados
- Más de 800,000 Horas - Maquina para generar los modelos finales
- Se requirieron 4.5 Millones de lineas de código para los modelos. (270 Mbytes)

1.4 ANIMACION DE SIMULACION

La animación de simulación consiste en realizar los cálculos de un fenómeno, del cual se tiene un modelo matemático y los resultados son procesados para generar imágenes de calidad fotográfica, de modo que la animación parezca un video.

Las aplicaciones son muy amplias, en la industria se elaboran modelos de sus productos y se "prueban" en ciertas condiciones simuladas, las cuales muy difícilmente se podrían conseguir en la realidad, o serían extremadamente costosas, en las escuelas se emplean para que los estudiantes visualicen mejor las teorías, y los científicos pueden observar la evolución de casi cualquier fenómeno ya sin las limitaciones de espacio o de tiempo que requiere la naturaleza, un fenómeno físico puede durar millonésimas de segundo o millones de años para realizarse, pero gracias a la animación de simulaciones es posible crear videos que lo reproduzcan en una escala de tiempo razonable.

Las animaciones de simulaciones son muy utilizadas en la ciencia, desde simulación de fenómenos meteorológicos, hasta simulación de colonias de bacterias o de comportamiento de sistemas planetarios. En la siguiente figura se muestra un cuadro de una animación que fue realizada a partir de datos obtenidos de la simulación del impacto de un fragmento del cometa Shoemaker-Levi 9 contra el planeta Júpiter. (Puede observarse en http://www-erl.mit.edu/flolab/cs19press/cs19hj.html



Cuadro de la animacion del impacto de un fragmento de un cometa contra Jupiter.

CAPITULO 2

CONSIDERACION ACERCA DEL TIRO PARABOLICO Y SU IMPLEMENTACION EN UNA COMPUTADORA.

CAPITULO 2 CONSIDERACIONES RELATIVAS A EL TIRO PARABOLICO Y SU IMPLEMENTACION EN UNA COMPUTADORA

2.1 TIRO PARABOLICO SIN RESISTENCIA AL AIRE

El tiro parabólico es un ejemplo simple de la mecánica clásica, el cual es relativamente fácil de implementar : Si despreciamos la resistencia del aire la trayectoria descrita en un tiro parabólico esta definida por dos componentes como se aprecia en el esquema siguiente:

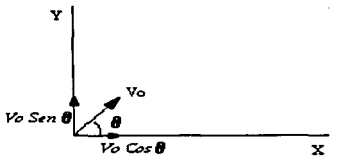


Diagrama de componentes de un tiro parabólico

En el diagrama los componentes estan definidos de la siguientes forma :

Vo Es la Velocidad Inicial
θ Es el ángulo de lanzamiento con respecto a la horizontal

2.1.1 POSICIONES X, Y COMO FUNCION DEL TIEMPO

Componente Horizontal : Es un movimiento rectilíneo uniforme, el cual está definido por :

$x = Vx * t$ Donde $Vx = Vo \cos \theta$

Componente Vertical : Es un movimiento rectilíneo uniformemente acelerado, el cual está definido por :

$y = Vy * t - \frac{1}{2} gt^2$ Donde $Vy = Vo \operatorname{Sen} \theta$

Dados los razonamientos anteriores, se puede implementar una función en Pascal que obtenga la posición de *x* a partir de los siguientes datos:

```
Vini     : Velocidad inicial
Ang      : Angulo de lanzamiento
Tiempo   : Tiempo transcurrido
Xini     : Posición x inicial
masa     : masa (En esta función la masa es ignorada);
```

Primero necesitamos declarar la constante g, de forma:

G = 9.8;

La función queda de la siguiente manera:

```
Function Obten_X (Vini, Ang, Tiempo, Xini, Masa :Real): Real;
 Var
   VoX : Real;
 Begin
   VoX := Vini * Cos(Ang);
   Obten_X := Xini + VoX* Tiempo;
 End;
```

Lo mismo se puede hacer para realizar una función que obtenga la posición de y, donde

```
Function Obten_Y (Vini, Ang, Tiempo, Yini, Masa :Real): Real;
  Var
    VoY : Real;
  Begin
    VoY := Vini *Sin(Ang);
    Obten_Y :=Yini +(VoY * Tiempo)-(0.5)*(g* Tiempo* Tiempo);
  End;
```

2.1.2 ALCANCE MAXIMO Y TIEMPO DE VUELO

Para obtener la altura máxima : sabemos que la altura máxima se alcanza cuando la componente vertical de la velocidad cambia de signo (Deja de subir y comienza a descender) este punto se alcanza cuando Vy=0 y de ahí se obtiene :

$t = (Vo \text{ Sen } 0)/g$ y sustituyendo t en la ecuación :

$y = Vo * t - \frac{1}{2} gt^2$ donde y es considerado h (altura máxima) se obtiene finalmente :

$h = Vo^2 \text{ Sen}^2 0 / 2g$

y la implementación en computadora queda de la siguiente forma :

```
Function Alt_Max(Vini, Ang, masa : Real): Real;
    Var
      temp: Real;
    begin
      temp :=Sin(Ang);
      Temp :=Temp *Temp * Vini * Vini;
      Alt_Max := temp/(2*g);
    end;
```

El tiempo de vuelo se obtiene sustituyendo y = 0:

Tv = 2*Voy/2 y como Voy = Vo * Sen 0 la función en Pascal queda:

```
Function Tiempo_Vuelo(Vini, Ang, masa : Real): Real;
    Var
      Voy : Real;
    begin
      VoY := Vini * Sin(Ang);
      Tiempo_Vuelo:=(2*VoY)/g;
    end;
```

La distancia máxima alcanzada se puede calcular sustituyendo el tiempo de vuelo en la ecuación para obtener la posición en x, la función en Pascal queda:

```
Function Alcance(Vini, Ang, Masa : Real): Real;
   Begin
     alcance :=(Vini * Vini * sin(2*Ang))/g;
   End;
```

Hasta aquí quedan listas las ecuaciones para realizar los cálculos a un tiro parabólico, ahora faltan realizar las funciones donde se tome en cuenta la resistencia del aire.

2.2 TIRO PARABOLICO CON RESISTENCIA AL AIRE

La ecuación para obtener la posición de x tomando en cuenta la resistencia del aire :

La resistencia a el aire produce una fuerza proporcional a la velocidad del cuerpo, y con un sentido opuesto al mismo. Matemáticamente podemos escribirlo como :

$$F_{res} = - bv$$

Donde b es una constante positiva que representa el coeficiente de resistencia. Para poder realizar los cálculos necesarios consideraremos la suma de fuerzas en el cuerpo constante durante todo el recorrido.

2.2.1 POSICIONES X, Y COMO FUNCION DEL TIEMPO

La velocidad terminal del cuerpo se puede expresar de la siguiente forma :

$$Vt = F/b$$

Si a esto aplicamos la 3ª ley de Newton obtenemos :

$$m\frac{d\vec{v}(t)}{dt} = \vec{F} - b\vec{v}(t)$$

Esta ecuación puede ser escrita completamente en términos de velocidad y su primera derivada.

$$\vec{V}(t) = \frac{\vec{F}}{b} + \left(\vec{V}o - \frac{\vec{F}}{b}\right)\exp\left(-\frac{b}{m}t\right)$$

Integrando de nuevo, nosotros encontramos que el vector de posición está dado por :

$$\vec{x}(t) = \vec{x}o + \frac{\vec{F}}{b}t + \frac{m}{b}\left(\vec{v}o - \frac{\vec{F}}{b}\right)\left(1 - \exp\left(-\frac{b}{m}t\right)\right)$$

Y tomando en cuenta que :

$$\vec{F} = -mg\hat{y}$$

Es posible obtener las siguientes ecuaciones :

$$x(t) = \frac{m}{b}VoCos\theta\left(1 - \exp\left(-\frac{b}{m}t\right)\right)$$

$$z(t) = -\frac{mg}{b}t + \frac{m}{b}\left(Vo\,\mathrm{sen}\,\theta + \frac{mg}{b}\right)\left(1 - \exp\left(-\frac{b}{m}t\right)\right)$$

Lo cual se puede convertir en una función en Pascal que obtenga la posición en x

```
Function Obten_XR(Vini, Ang, Tiempo, B, Masa :Real): Real;
   Var
     Alfa : Real;
   Begin
     Alfa :=(1-exp(-b * tiempo/masa));
     Obten_XR :=(masa/b)* Vini *Cos(AngDeg)*alfa;
   End;
```

La función en Pascal que calcula la posición en y queda :

```
Function Obten_YR(Vini, Ang, Tiempo, B, Masa :Real): Real;
Var
  a, b1, b2, c: real;
  Alfa : Real;
Begin
  Alfa:=(1-exp(-b * tiempo/masa));
  a:=-(masa *g * tiempo)/b;
  b1:=(masa/b);
  b2:=(Vini * sin(AngDeg))+(masa * g /b);
  b2:=b1*b2;
  b2:=b2*alfa;
  Obten_YR:=a+b2;
End;
```

2.2.2 UTILERIAS RAD2DEG Y DEG2RAD

Para facilitar algunas operaciones del programa, tales como darle como entrada datos expresados en grados y tomando en cuenta que Pascal realiza sus cálculos de funciones trigonométricas en Radianes se hace conveniente realizar las funciones que conviertan los grados en radianes y viceversa, las funciones se muestran a continuación.

```
Function Deg2Rad(Ang : Real): Real;
Begin
  Deg2Rad:=Ang* 0.01745329251994;
End;

Function Rad2Deg(Ang : Real): Real;
Begin
  Rad2Deg:=Ang* 57.29577951308;
End;
```

3.1 ANALISIS DE CONCENTRACION DE PUNTOS

En el estudio de un tiro parabólico es relativamente fácil observar donde se concentran más los puntos, imaginemos que tomamos una foto a un tiro parabólico una sola película y un flash de tipo estroboscópico, obtendremos un cuadro con varias impresiones. ¿Donde se concentran más los puntos ?.

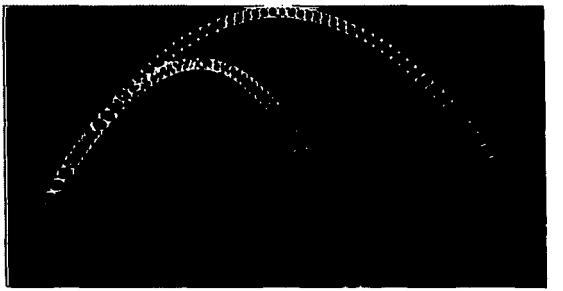


Imagen que muestra la concentración de puntos en un tiro parabólico

Podemos observar que los puntos se distribuyen de forma regular en el eje x, no siendo así en el eje y, en el cual se nota una concentración de puntos alrededor de la altura máxima

Lo anterior es fácil de explicar ya que como al principio se explico el tiro parabólico tiene dos componentes :

El componente x que es un movimiento rectilíneo uniforme, el componente y que es un movimiento uniformemente desacelerado.

De ésto se deduce que la mayor concentración de puntos por unidad de tiempo se encontrará cerca de vy = 0 (punto de altura máxima).

CAPITULO 3

CONSIDERACIONES RELATIVAS A LA ANIMACION EN COMPUTADORA.

CAPITULO 3
CONSIDERACIONES RELATIVAS A LA ANIMACION EN COMPUTADORA

3.1 QUE ES UNA ANIMACION Y PERSISTENCIA DE LA VISION

Todos hemos visto cine y televisión, estos medios emplean animación, lo que no es sino la apariencia de movimiento continuo a partir de un número de imágenes por segundo (no continuo), esto es debido a un fenómeno conocido como persistencia de la visión.

La persistencia de la visión se puede describir como sigue : El ojo humano recibe constantemente información visual, pero no procesa el flujo de manera continua, sino más bien toma "imágenes fijas" cada intervalo de tiempo y en el cerebro se recrea la secuencia de forma que parece un flujo constante de imágenes, gracias a este mecanismo parpadeamos sin que notemos discontinuidad en lo que vemos y no necesitamos procesar cantidades gigantescas de información visual.

La cantidad de imágenes por segundo que somos capaces de percibir en forma consciente ronda entre los 12 y los 16, así que para generar una animación se requieren mostrar por lo menos esta cantidad de imágenes en un segundo, el cine esta estandarizado a 24 cuadros por segundo mientras que la televisión lo está a 30 cuadros por segundo ; si elevamos más la cantidad de cuadros por segundo únicamente desperdiciamos recursos, si a una persona se le muestra una animación tomada a 30 cuadros por segundo y otra realizada a 500 cuadros por segundo (reproducidas a sus respectivas velocidades) no podrá notar diferencia, así pues los cuadros intermedios de una animación pueden ser ignorados (como en realidad pasa con nuestro cerebro).

3.2 RECURSOS DE ANIMACION

Las animaciones por computadora son relativamente recientes ya que requieren cantidades enormes de recursos, pero dado que la evolución de los equipos de cómputo crecen de manera geométrica (doble de capacidad y velocidad cada 18 meses por el mismo precio) es de esperarse que pronto las animaciones sean más accesibles a una gran cantidad de personas.

De los recursos que más requieren las animaciones están :

RECURSOS DE ALMACENAMIENTO

Los recursos necesarios para almacenar una animación son muy altos, ejemplificando:

Una imagen de 640 x 480 ocupa un byte por cada pixel y tiene 307,200 pixeles (640 x 480), ahora bien si queremos almacenar no una imagen sino una secuencia de ellas y en un segundo requerimos 30 imágenes entonces necesitamos un total de 600 imágenes cada una de 307,200 bytes, por lo que obtenemos un total de 184,320,000 bytes o 184 Mbytes.

Si realizamos los cálculos anteriores pero para color verdadero, es decir, 16.7 Millones de colores entonces se necesitan de 3 bytes para almacenar cada pixel, lo cual triplica el total.

RECURSOS DE PROCESAMIENTO

De la revisión de los requerimientos de recursos de almacenamiento nos damos cuenta que la cantidad de puntos o pixeles a calcular es muy elevada, ahora bien para poder generar cada uno de estos puntos se tienen que realizar en ocasiones muchas operaciones que implican por ejemplo el trazado de rayos (Nota : esto se trata con más profundidad en el capitulo 4).

Por ejemplo para calcular un pixel por medio del trazado de rayos requerimos :

1.- Verificación de contacto : Se resuelven las ecuaciones del rayo de traza junto con la del resto de los objetos de la escena a generar y se verifica si intercepta con alguna, si no intercepta con ninguna se dibuja el pixel con el mismo color del fondo.

2.- Si intercepta con mas de un objeto se procede a verificar la distancia de la cámara a los puntos de los objetos interceptados, y se toma en cuenta el más próximo.

3.- Para el punto que resulte interceptado se calculan sus atributos de color, para ello se emplean ciertas fórmulas que implican cálculos con senos y cosenos del rayo con respecto a la superficie normal al plano tangente a la superficie en el punto interceptado, y al rayo o rayos que provienen de la fuente de luz, también se calculan los puntos de otros cuerpos que pudieran reflejar luz a este punto, además de otros aspectos como: refracción y brillo, sombras, texturas, etc.

Así que para poder calcular un solo pixel pueden requerirse varios cientos de operaciones simples, lo cual multiplicado por el número de puntos resulta en varios miles de millones de operaciones, el poder de cálculo de las computadoras actuales esta permitiendo realizar cada vez con mayor rapidez este tipo de trabajo, a continuación una tabla que muestra los rendimientos de varias computadoras:

Plataforma	Tiempo de Generación de la imagen
PC 486 DX2/66 MHz	5 Horas 55 Minutos
PC Pentium 60 MHz	1 Hora 45 Minutos
SUN SPARC CLASSIC	30 Minutos
SUN SPARC 4	12 Minutos 50 Segundos
Silicon Graphics O2	2 Minutos 39 Segundos

Nota: El programa es POV-Ray Versión 2.0 corriendo en forma nativa en todas las plataformas, la imagen generada es el archivo chess.pov que se incluye junto con los ejemplos de POV-Ray, la resolución es de 800 x 600.

RECURSOS DE DESPLIEGUE

Casi cualquiera puede tener actualmente una maquina con la posibilidad de almacenar unos cuantos cientos de Megabytes para animación y habrá muchos con la paciencia para que su computadora pase algunas horas o dias generando las imágenes para una animación, ahora queda un detalle muy importante, se requiere de una gran capacidad de despliegue para mostrar una animación en tiempo real, es por ello que los equipos aptos para este tipo de trabajos sean de muy alta capacidad, por ejemplo una O2 de Silicon Graphics que posee un sistema de comunicación entre el procesador y la memoria de 2.1 Gbytes por segundo.

3.3 FORMATOS DE ANIMACION

Los formatos mas comunes para almacenar animaciones son:

- AVI
- FLI/FLC
- MPEG
- Quicktime (.MOV)

Cada uno de estos presenta sus propias ventajas y desventajas, a continuación se da una breve descripción de algunos de estos formatos y se mencionan algunas de sus caracteristicas principales:

Formato AVI

AVI significa Audio y Video Intercalados (Audio and Video Interhaced), su funcionamiento es en principio relativamente simple:

La parte del video es practicamente una secuencia de imágenes en formato BMP, para esclarecer un poco mas esto, explicaremos el formato BMP.

En los archivos BMP los puntos son almacenados sin compresión y en orden inverso a como aparecen en pantalla, iniciando el archivo con una cabecera de 1078 bytes, de los cuales se emplean solo los primeros 117 bytes y el resto se pone a cero.

Cada línea es almacenada grabando un byte por cada pixel, dentro de la cabecera también son grabados los datos de altura y anchura de la imagen, así como el identificador de archivo, el cual indica el tipo de archivo del cual se trata, todos los archivos .BMP empiezan con BM (en hexadecimal 42 y 4D). Se emplea un byte para cada color debido a que con 8 bits es posible realizar hasta 256 combinaciones diferentes. (los archivos .BMP descritos son de 256 colores paleta idéntica a la de Windows).

Para usar un archivo .BMP se requiere de lo siguiente :

1.- Se leen los datos del archivo de la cabecera.

2.- Se leen todos los datos de la imagen, cuando se han terminado de leer se comienzan a desplegar empezando por los últimos que se hayan leído (en este formato las imágenes se almacenan al revés), es posible también desplegar simultáneamente los archivos mientras se leen pero para esto es necesario comenzar el despliegue de abajo hacia arriba y de derecha a izquierda (mucho mas recomendable).

Un archivo AVI sin sonido básicamente es una secuencia de BMP, con algunas diferencias:

1.- La cabecera del AVI almacena la cantidad de cuadros por segundos que han de desplegarse.

2.- La cabecera contiene la cantidad de cuadros totales

3.- La cabecera contiene la paleta de colores para todos los cuadros, es decir todos los cuadros emplean la misma paleta.

Este formato tal y como está descrito es poco eficiente debido a que requiere de mucho espacio de almacenamiento, diferentes compañías han desarrollado extensiones para este formato donde se emplean diferentes métodos de compresión, de entre los que más destacan se encuentra el Cinepak.

A pesar de no ser muy bueno este formato se emplea mucho debido a que fue desarrollado por Microsoft, que es la empresa que hace Windows, de forma tal que las animaciones en Windows son en .AVI por defecto.

Formato MPEG

El formato MPEG (Motion Picture Expert Group) es uno de los formatos que está ganando mayor aceptación en la actualidad por la gran compresión que realiza. Este formato esta definido en la norma ISO 11.172, este formato incorpora algoritmos bastante ingeniosos (y complejos) para la compresión de las imágenes, algunas de sus características más destacables son:

1.- Compresión con perdidas: La compresión se realiza por medio de algoritmos que se basan en la imperfección de la visión humana por lo que se logra cierto "engaño" al ojo humano por medio de ciertos algoritmos que permiten compresiones sorprendentes.

2.- Algoritmos muy complejos: La complejidad de el formato MPEG se refleja en la gran cantidad de cálculos que son requeridos para descomprimirlos, por lo que en la actualidad cada vez más tarjetas de video incorporan capacidades de descompresión MPEG en la misma tarjeta.

3.- Relación de Compresión muy alta: Un archivo MPEG es mucho más pequeño que la secuencia original de imágenes, de los formatos de animación este es el que logra la mayor compresión de todos.

MPEG (Motion Picture Expert Group o Grupo de Expertos de imágenes en movimiento) es el formato más avanzado técnicamente para el almacenamiento de video digital, en la actualidad es el único que permite almacenar una película a pantalla completa en un solo disco compacto.

Existe el estándar MPEG I especifica una resolución de 352 x 288 a 16.7 Millones de Colores ya ha sido desarrollado el MPEG II que especifica una resolución de 768 x 576 Pixeles.

La forma en que se consigue tan elevada compresión es por medio de varios algoritmos, el más importante es uno que aprovecha el principio de que los pixeles de un cuadro y sus adyacentes son muchas veces iguales, de este modo si codificamos un cuadro y el cuadro siguiente es casi igual al anterior solo se graban las diferencias entre ambos con lo cual se logra un ahorro enorme en el almacenamiento, de este modo se pueden definir tres tipos de fotogramas :

Fotogramas infra: Son cuadros que solo contienen información de la imagen, el primer cuadro es un cuadro infra, los siguientes cuadros pueden ser del tipo predicted o cuadros bidireccionales, si un cuadro es muy diferente de los adyacentes es codificado como cuadro infra.

Fotogramas Predicted : Contienen información de cuadros anteriores y las diferencias con respecto de la imagen actual, por ejemplo si un cuadro es codificado y el siguiente es muy semejante al anterior este será un cuadro predicted o de predicción.

Cuadro bidireccional : Contiene información tanto de cuadros anteriores como posteriores.

Otro de los factores que ayuda a la elevada compresión es la forma de codificar la información:

La imagen se descompone en "macrobloques" que son matrices de 16 x 16 pixeles formadas a su vez por 4 bloques de 8 x 8 pixeles, de cada uno de estos bloques es grabada su luminancia y su crominancia dando especial importancia luminancia. Después se transforman en información de coeficientes senoidales, los cuales se seleccionan de un número finito de los mismos, lo cual indica cierta pérdida de la calidad, pero debido a que nuestros ojos no perciben diferencias pequeñas no es perceptible y en cambio permite un gran aumento en la compresión, dado que los coeficientes contienen una gran cantidad de ceros se emplea un algoritmo de compresión que escribe un valor y a continuación el número de veces que se repite dicho valor.

Pero este algoritmo tiene un inconveniente importante: debido a los excelentes algoritmos de compresión requiere de un número de cálculos impresionante, de tal forma se requiere de un poderoso procesador de imágenes en hardware para poder descomprimir en tiempo real, la tendencia de las tarjetas de video indican que cada día serán más las que incluyan descompresión MPEG dentro de la misma tarjeta.

Formato FLI/FLC

Este formato fue desarrollado por Autodesk para sus productos Animator y Animator PRO. emplea algoritmos bastante buenos para la compresión de archivos, pero tiene varias limitantes importantes :

No almacena sonido, aunque se puede leer un archivo de sonido simultáneamente con ayuda de un programa adecuado ; esta limitado a 256 colores.

El formato FLC contiene una cabecera de 128 bytes los cuales se emplean de la siguiente manera :

Posición	Longitud	Descripción
0	4	Tamaño de la animación (incluye cabecera).
4	2	Identificador de Formato de Archivo. Siempre es el número hexadecimal AF12.
6	2	Número de cuadros en el archivo. Los archivos FLC tienen un máximo de 4000 cuadros.
8	2	Ancho de la pantalla en pixeles
10	2	Altura de la pantalla en pixeles
12	2	bits por pixel (siempre 8)
14	2	Se pone en Hexadecimal 0003. Indica que el archivo ha sido adecuadamente cerrado.
16	4	Número de milisegundos de retardo entre cuadros.
20	2	Palabra de 2 bytes sin emplear.
22	4	Fecha de creación MS-DOS
26	4	Número de serie del programa Animator PRO empleado para crear la animación.
30	4	Fecha MS-DOS de última modificación
34	4	Indica que número de serie de programa fue el ultimo en modificar.
38	2	La relación de proporción del eje x con que fue creado el archivo.
42	38	Reservado para aplicaciones futuras. Todos puestos a cero.
84	4	Desplazamiento del archivo hasta el segundo cuadro.
88	40	Reservado para aplicaciones futuras. Puestos a cero.

Los Prefijos de información de FLC

Un bloque opcional de información puede encontrarse inmediatamente después de la cabecera del archivo. Este bloque es empleado para almacenar información adicional. Este bloque inicia con una cabecera de 16 bytes. La cual se describe a continuación :

Desplazamiento	Longitud	Descripción
0	4	Tamaño de el bloque de información adicional
4	2	Identificador del bloque (siempre Hex F1000
6	2	Número de cuadros adicionales de información subordinados
8	8	Reservado para aplicaciones futuras. Puestos a ceros.

Los bloques de datos adicionales contienen información del pixel y color para la animación. Un cuadro de información adicional puede contener varios bloques adicionales subordinados, todos contienen un tipo diferente de datos, cada bloque tiene una cabecera de 16 bytes que se describen a continuación :

Desplazamiento	Longitud	Descripción
0	4	Tamaño de el bloque de información adicional
4	2	Identificador del bloque (siempre Hex F1FA
6	2	Número de cuadros adicionales de información subordinados
8	8	Reservado para aplicaciones futuras. Puestos a ceros.

Si el contador de bloque de datos adicionales es cero indica que el bloque es igual al anterior por lo cual no se realizan cambios ni en la pantalla ni en la paleta. Lo único que se realiza entonces es la pausa entre correspondiente entre cuadros.

Todos los bloques de datos adicionales tienen la siguiente información :

Desplazamiento	Longitud	Descripción
0	4	Tamaño de el bloque de información adicional.
4	2	Identificador del tipo de dato.
6	tamaño-6	Color o dato del pixel.

Los valores del tipo en el bloque indican que tipo de información gráfica contiene el cuadro y que método de compresión fue empleado para codificar los datos la tabla siguiente tabla indica los posibles valores.

Valor	Descripción
4	Paleta de 256 Colores
7	Palabra orientada a compresión por deltas
11	Paleta de 64 Colores
12	Byte orientado a compresion delta
13	El cuadro entero es del color indice 0
15	Byte de longitud de la compresión
16	Indica no compresión
18	Estampa del tamaño de la imagen

Las siguientes secciones describen estos metodos en detalle.

Bloque de datos tipo 4 (FLI de 256 colores)

La información aquí está organizada por paquetes. la primera palabra (16 bits) que sigue a la cabecera es un contador del número de paquetes en el bloque. Cada bloque contiene un byte para identificar el color de la paleta, un byte contador para color y 3 bytes que definen cada color.

Formato FLI

Los archivos .FLI son un subconjunto de los archivos .FLC donde la principal diferencia es que están limitados a una resolución de 320x200.

Descripción de la cabecera de un archivo FLI :

Desplazamiento	Longitud	Descripción
0	4	El tamaño de la animación.
4	2	Identificador del formato (siempre Hex AF11)
6	2	Número de cuadros en el archivo. (Como máximo 4000 cuadros).
8	2	Ancho de la animación en pixeles.
10	2	Altura de la animación en pixeles
12	2	Bits por pixel (siempre 8)
14	2	En los archivos FLI siempre son cero.
16	2	Retardo entre cuadros de animación expresados en 1/70 de segundo
18	110	Reservados para aplicaciones futuras. Puestos a cero.

Bloque de datos adicionales en archivos FLI.

Uno o más cuadros de información adicional pueden seguir a la cabecera. Estos bloques de información adicional son idénticos a los que aparecen en los archivos FLC. excepto porque no contiene la estampa de puesta de la imagen y la palabra de longitud de la compresión no aparecen en los archivos FLI.

CAPITULO 4

CONSIDERACIONES RELATIVAS A LA REPRESENTACION DE SUPERFICIES EN COMPUTADORA.

CAPITULO 4
CONSIDERACIONES RELATIVAS A LA REPRESENTACION DE SUPERFICIES EN COMPUTADORA.

4.1 METODOS DE MODELADO DE OBJETOS

La representación de superficies en computadora puede realizarse de diferentes formas, las más comunes son : Aproximación por medio de superficies formadas por polígonos o por medio de superficies definidas por ecuaciones.

En la representación por polígonos (Que es la más común) los objetos son almacenados por áreas aproximadas por polígonos a las formas reales, por ejemplo si deseamos representar un objeto por medio de polígonos requerimos de lo siguiente :

1.- Tomar varios puntos de la superficie a representar
2.- Definir polígonos que definan áreas uniendo dichos puntos

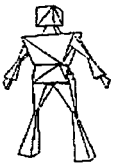
Una esfera por ejemplo se puede definir de 2 formas diferentes si es por ecuaciones se emplea su ecuación paramétrica :

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

o si se emplean aproximaciones por medio de polígonos se emplearía un arreglo de vértices que se aproximarán a dicha ecuación.

4.2 METODOS DE PRESENTACION DE POLIGONOS

Es posible representar cualquier objeto empleando aproximaciones por medio de áreas poligonales, dichas aproximaciones se logran tomando varios puntos de la superficie a representar y dichos puntos se toman como vértices de las áreas encerradas por polígonos, dichas aproximaciones son muy fieles a los modelos originales cuando estos están formados por facetas, pero presentan problemas cuando los modelos a representar son de superficies curvas, otra manera de mejorar el realismo de los modelos es reduciendo el tamaño de las áreas poligonales, es decir, aumentando el número de vértices. Tomemos como ejemplo estas tres imágenes de modelos de alambre (los modelos de alambre son la unión de los vértices por medio de líneas rectas). En estas imágenes se aprecia claramente como al aumentar el número de vértices disminuimos el tamaño de las áreas y por consiguiente el error de aproximación al modelo.



Ahora bien, para que estos modelos puedan tener un realismo aceptable es necesario que puedan ser iluminados de forma que se pueda observar volumen por medio las diferencias de las luces y sombras.

4.2.1 METODO DE SOMBREADO DE INTENSIDAD CONSTANTE

El método de sombreado de intensidad constante consigue visualizaciones muy rápidas, ya que requiere de pocos cálculos, el procedimiento es sencillo :

1.- Se calcula el color del punto central de una faceta (área poligonal).
2.- Esta intensidad de color se establece a todos los puntos visibles del polígono.

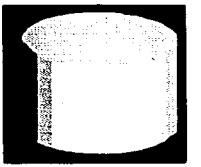
Este método permite presentar resultados muy rapidos, pero es visualmente muy pobre, porque las diferencias de intensidad de luz de una faceta a otra pueden cambiar muy drásticamente dando al modelo final un toque muy "Facetado".

Puede ser muy fiel a la realidad cuando se cumplen los siguientes requisitos :

A) El objeto es un poliedro, es decir, no es una aproximación a una superficie curva.
B) Las fuentes de luz están lo suficientemente lejos como para que la atenuación de la luz sea constante en toda la superficie.
C) La posición de la cámara esta lo suficientemente lejos de la superficie como para que la apariencia de la misma sea constante.

La principal utilización de este algoritmo es la previsualización de modelos antes de la realización definitiva de las imágenes.

También es posible mejorar el aspecto gráfico de este método aumentando el número de facetas, es decir reduciendo el tamaño de los polígonos, esto tiene el inconveniente de aumentar el tamaño de los archivos y hacer más tardado el proceso de cálculo.



Cilindro con sombreado de intensidad constante.

4.2.2 SOMBREADO DE GOURAUD

El método de sombreado de Gouraud es un método más elaborado que el sombreado de intensidad constante, pero requiere de mayores cálculos.

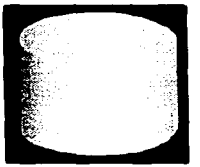
Consiste en la interpolación de intensidades de color a lo largo de una faceta, el algoritmo es sencillo :
Se calculan los vectores normales de cada vértice y se aplica un modelo de color a cada uno, después se realiza un rastreo de cada línea de la faceta y se interpola el color correspondiente.

Tiene mucha mejor calidad que el sombreado de intensidad constante, la principal desventaja de este método es que la iluminación de toques de luz no se realiza adecuadamente, por lo que aparecen bandas de color denominadas bandas de "Mach", las cuales se deben a que los "toques" de luz corresponden a áreas muy pequeñas, las cuales se interpolan también generando una banda de color que puede ser clara u obscura.

Este tipo de sombreado se emplea en juegos que requieren de gráficos vistosos y de una gran velocidad de despliegue, de modo que otro método de presentación no es factible debido a la velocidad de respuesta.

Se realizan los siguientes cálculos para presentar cada superficie :

- Se determina el vector normal unitario promedio en cada vértice del polígono.
- Se aplica un modelo de iluminación en cada vértice para calcular la intensidad del mismo.
- Se interpolan de manera lineal las intensidades de vértice sobre la superficie del polígono.



Cilindro con sombreado de Gouraud

4.2.3 SOMBREADO DE PHONG

El método de sombreado de Phong presenta resultados visuales muy buenos pero consume mucho más tiempo que el sombreado de Gouraud (6 ó 7 veces más)

Básicamente consiste en calcular los vectores normales de los vértices e interpolarlos y aplicar a cada vector interpolado un modelo de iluminación a cada uno de dichos vectores, este método reduce bastante el efecto de bandas de "Mach" .

Los pasos a seguir en este método son :

- Se determina el vector normal unitario promedio en cada vértice del polígono.
- Se interpolan de manera lineal las normales de vértice en la superficie del polígono.
- Se aplica un modelo de iluminación a lo largo de cada línea de rastreo para calcular las intensidades de los pixeles que corresponden a esas posiciones.

Se puede obtener el vector normal N para la intersección de la línea de rastreo a lo largo de la arista entre los vértices 1 y 2 al interpolar en forma vertical entre las normales de los extremos de la arista de la siguiente manera :

$$N = \frac{y - y_2}{y1 - y_2} N_1 + \frac{y_1 - y}{y_1 - y_2} N_2$$

4.3 METODO DE RASTREO DE RAYOS

El algoritmo de rastreo de rayos consta de un plano que será la pantalla, donde se hace coincidir el centro de la imagen con el origen de un sistema coordenado. la cámara que es un punto imaginario observa la escena a través de este plano. dado que es posible trazar una línea recta a partir de dos puntos, es posible determinar un rayo para cada pixel de la pantalla, el algoritmo consiste básicamente en :

1.- Calcular un rayo empleando como puntos la posición de la cámara y la posición relativa del pixel en el eje de coordenadas.

2.- Se determina si el rayo intercepta con alguna superficie, esto se logra por medio de resolución de ecuaciones simultáneas, por ejemplo para una esfera tenemos la ecuación de la esfera y la ecuación del rayo, si encontramos que existe un punto que este sobre el rayo y simultáneamente sobre la esfera decimos que si ha ocurrido una intersección.

3.- Si el rayo intercepta a varias superficies se determina la distancia más corta a la cámara, para esto se emplea la formula de la distancia entre dos puntos, donde el punto inicial es la posición de la cámara y el punto final es el punto de intersección de el rayo con la superficie ; la fórmula de la distancia entre dos puntos en el espacio es :

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Si el rayo intercepta un sólo punto se calcula la intensidad de ese punto por medio de el ángulo de reflexión, se emplea el seno del ángulo que forman el rayo incidente (el cual se calcula por los puntos de la fuente de luz y el punto donde el rayo de luz intercepta la superficie) y el plano normal a la superficie.

Si el rayo no intercepta a ninguna superficie se iguala el pixel a el color del fondo, el cual es generalmente negro.

Si la escena es a color se realizan los cálculos de intensidad por cada componente, pongamos un ejemplo sencillo :

Estamos manejando un algoritmo que da como resultado imágenes a 16.7 millones de colores, para lo cual empleamos 3 bytes, uno para cada componente RGB así tenemos 256 niveles de rojo, 256 niveles de verde y 256 niveles de azul (256 x 256 x 256 = 16.7 millones de combinaciones). Si un pixel tiene una iluminación que hemos calculado como 0.7 (las funciones trigonométricas seno y coseno entregan valores entre 0 y 1), la

fuente de luz es blanca con intensidad 1.0 y el pixel al que nos referimos pertenece a un objeto al que le hemos definido un color Rojo = 1.0, Verde =.0.8 y Azul = 0.5 (en escala de 0 al 1.0) entonces multiplicamos cada componente por la intensidad y obtenemos Rojo = 0.7 Verde = 0.56 y Azul = 0.35 ahora para grabarlo en un archivo que admite 256 niveles de cada color multiplicamos el componente por 255 y el valor obtenido se redonden a el entero más próximo y se graba en el archivo, el resultado final es Rojo = 178, Verde = 143 y Rojo = 89. Si la fuente de luz es de componentes diferentes también se deben realizar cálculos adicionales, en este caso la luz al ser blanca de intensidad 1, tiene implícitamente sus tres componentes con intensidad 1, si esta intensidad varia se multiplica cada componente por el valor de los componentes de color del objeto y después por 255 para obtener el resultado del pixel.

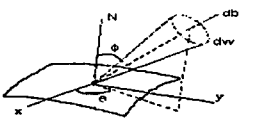
4.- Se graba la cabecera del archivo y se graba la secuencia de pixeles en un formato conveniente, normalmente se guardan a 24 bits (1 byte por componente) dado que esto facilita mucho el proceso, ya que si se quiere grabar en un archivo con una paleta de colores se requiere además implementar un algoritmo de reducción de colores.

4.4 MODELO DE ILUMINACION DE RADIOSIDAD

Generar imágenes donde la iluminación sea difusa en lugar de directa es bastante difícil de realizar con rastreo de rayos, para este tipo de escenas se emplea el algoritmo de radiosidad, el cual se basa en el principio de conservación de la energía .

El algoritmo se puede describir de la siguiente manera : Determinamos la cantidad diferencial de energía radiante (db) que se emite en cada punto de la superficie de la escena y se suman las contribuciones de energía para así obtener la cantidad de transferencia de energía entre las superficies.

En la siguiente figura se observa que (db) es la energía radiante visible que se emite en el punto sobre la superficie en la dirección que dan los ángulos θ y ϕ en el ángulo sólido diferencial $d\omega$ por unidad de tiempo por unidad de superficie, de este modo (db) tiene unidades que se expresan en joules/(segundo * metro^2) o watts/metro^2.



Podemos calcular la intensidad por medio de la formula :

$$I = \frac{dB}{d\omega \cos\phi}$$

Si consideramos como constante la intensidad I para todas las direcciones de vista necesitamos sumar la radiación para todas las direcciones para de este modo obtener el índice total de energía.

4.5 MODELOS DE COLOR

Para utilizar la síntesis de imágenes es preciso emplear un modelo de color : es posible entender la luz por medio de ciertos modelos que se han desarrollado a partir del estudio de las propiedades de la luz, pero ningún modelo cumple con todos los comportamientos de la misma. Varios de estos modelo se emplean para diferentes aplicaciones.

4.5.1 MODELO DE COLOR RGB

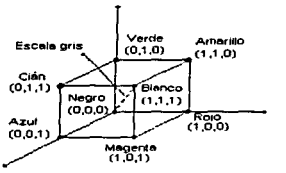
El modelo RGB es uno de los más empleados en dispositivos de salida basados en síntesis aditiva, por ejemplo los monitores de computadora, televisores, proyectores, etc. Se denomina síntesis aditiva a la generación de imágenes por medio de agregar colores.

Los colores primarios para este modelo son rojo, verde y azul (de ahí su nombre RGB = Red, Green, Blue) y con una adecuada selección de los componentes se pueden obtener un número infinito de colores, si nos limitamos a 256 niveles de cada color entonces tenemos 16.7 Millones de colores, como casi nadie es capaz de distinguir diferencias menores a 1/256 de un tono esto es más que suficiente para la mayoría de las aplicaciones.

La forma en que se forman colores a partir de los primarios es sencilla, ejemplificando con un cubo donde los ejes corresponden a los valores de los primarios con una escala del 0.0 al 1.0 obtendríamos lo siguiente :

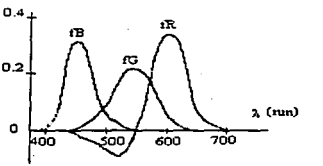
El negro se encuentra en el origen (0,0,0), el blanco se encuentra en el vértice opuesto (1,1,1), la línea que une estos vertices es la escala de grises donde los 3 componentes se encuentran en la misma proporción cada

uno, los colores primarios se encuentran en los ejes y los colores complementarios en los vértices que unen dos ejes. A continuación una imagen que ejemplifica lo anterior :



Cubo que representa la gama de colores generados mediante RGB.

El modelo RGB al igual que cualquier otro modelo de color no es del todo completo, la siguiente gráfica muestra los valores RGB para diferentes longitudes de onda, como se aprecia para longitudes de onda cercanas a los 500 nm se requieren cantidades negativas de color rojo, como esto no es posible el modelo RGB no puede mostrar colores que se aproximen a los 500 nm.

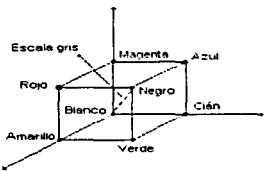


Gráfica que representa las cantidades de componentes primarios para generar el color indicado por la longitud de onda.

4.5.2 MODELO DE COLOR CMY

Si en modelo de color RGB se emplea la síntesis aditiva (suma de colores para generar gamas de colores) para el modelo CMY se emplea el proceso inverso denominado síntesis sustractiva, la cual consiste en restar colores para generar toda la gama de colores, en este modelo los colores primarios son el Magenta, Amarillo y Cián. La aplicación más común de este modelo es en dispositivos de salida impresa.

Ya que este modelo es opuesto al RGB el cuadro donde se muestran la gama de colores para CMY es también opuesto :



Cuadro que representa la gama de colores generados por el modelo CMY.

Existen otros muchos modelos de colores, pero sobrepasa los objetivos de este trabajo el abordarlos a fondo.

CAPITULO 5

DESCRIPCION DE LAS CARACTERISTICAS Y MANEJO BASICO DE POVRAY.

CAPITULO 5
DESCRIPCION DE LAS CARACTERISTICAS Y MANEJO BASICO DE POVRAY.

5.1 ¿QUE ES POVRAY ?

El programa Persistence of Vision Ray Tracer (POV-Ray) es un programa libre con derechos de autor. Este programa crea imágenes fotorrealistas usando como entrada archivos de texto ASCII, donde se describen los objetos, tales como esferas, luces, la posición de la cámara, cilindros, así como también las características de los objetos, tales como textura, reflexión, refracción, etc.

La forma de manejar las estructuras de datos es muy semejante a la forma en que se manejan los programas del lenguaje "C", de forma que es necesario tener ciertas precauciones, así pues :

1.- Las mayúsculas son diferentes de las minúsculas.
2.- No se pueden emplear palabras reservadas como identificadores.
3.- Deben corresponder las llaves que abren con las que cierran (Lo mismo los paréntesis).

Este programa primero realiza un proceso de análisis lexicográfico de los archivos para determinar si los objetos de la escena fueron correctamente definidos, y después inicia el proceso de generación de la imagen.

La versión 2.0 (Que es la utilizada para este trabajo) puede correr en una computadora con un microprocesador 386 ó 486 y 4 megabytes de memoria. El coprocesador matemático no es indispensable, pero acelera con mucho el proceso de generación de imágenes.

POV-Ray esta escrito en lenguaje "C" estándar y puede ser compilado y usado en muchos tipos de computadoras. De hecho es posible compilarlo y ejecutarlo en cualquier computadora que posea un compilador de lenguaje "C".

5.2 FORMATO DE POVRAY

POV-Ray se escribe en archivos de texto ASCII, algunos conceptos importantes para manejarlo se explican a continuación :

5.2.1 ARCHIVOS "INCLUDE"

Un aspecto interesante de POV-Ray es que puede incluir dentro de si otros archivos en los cuales se hayan escrito previamente definiciones u objetos del estilo de POV-Ray para ello se emplea la directiva "include" la cual es muy semejante a como se manejan las bibliotecas del lenguaje "C" así podemos crear librerías de objetos, texturas o colores. La forma de incluir un archivo en otro es :

```
#include "nombrearchivo.inc"
```

Ejemplo :

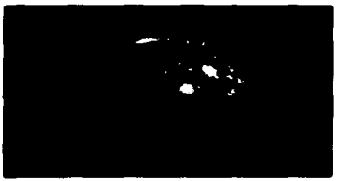
Usando un editor de texto, cree un archivo llamado imagen1.pov :

```
#include "colors.inc"
#include "textures.inc"
#include "stones.inc"

      camera {
        location  <5, 5, -6>
        look_at   <0, 0,  0>
      }

light_source {<10, 10, 5> color rgb <1, 1, 1>}
light_source {<10, 7, -5> color rgb <1, 1, 1>}
light_source {<-10, 10, 5> color rgb <1, 1, 1>}
sphere {<0,0,0>,3  texture {Stone21}}
```

La imagen que se obtiene es :



Los archivos indicados anteriormente contienen : definición de colores, definiciones de formas y definiciones de texturas, estos archivos se distribuyen junto con POV-Ray, usted puede crear los suyos propios.

Si deseamos emplear mucho un color de diseño propio con las siguientes características como el siguiente :

color red 0.6 green 0.9 blue 0.34

lo podemos declarar como :

```
#declare MiColor = color red 0.6 green 0.9 blue 0.34
```

y podemos emplear dicho color donde deseemos.

ejemplo:

```
sphere {
  <0, 0, 0>, 1
  pigment { color red 0.6 green 0.9 blue 0.34 }
}
```

Queda :

```
#declare MiColor = color red 0.6 green 0.9 blue 0.34

sphere {
  <0, 0, 0>, 1
  pigment { color MiColor }
}
```

Realizar declaraciones tiene muchas ventajas, facilita la lectura del archivo, acorta los tamaños de los archivos al declarar en un solo sitio y emplear solo el nombre del objeto en los lugares donde se requiera.

5.2.2 IDENTIFICADORES Y PALABRAS RESERVADAS

POV-Ray define varios identificadores de uso común, también usted puede definir sus propios identificadores. Los identificadores pueden tener hasta cuarenta caracteres de longitud y constan de dígitos , letras mayúsculas, minúsculas y el carácter de subrayado. (El primer carácter de un identificador debe ser un carácter alfabético).

POV-Ray posee varias palabras reservadas, que al ser parte del lenguaje no es posible utilizarlas como identificadores.

Estas son las palabras reservadas de POV-Ray:

adaptive	height_field	rgbf
agate	hexagon	right
agate_turb	iff	ripples
all	image_map	rotate
alpha	include	roughness
ambient	interpolate	scale
area_light	intersection	sky
background	inverse	smooth
bicubic_patch	ior	smooth_triangle
blob	jitter	specular
blue	lambda	sphere
bounded_by	leopard	spotlight
box	light_source	spotted
bozo	location	sturm
brilliance	looks_like	texture
bumps	look_at	tga
bump_map	mandel	threshold
bump_size	map_type	tightness
camera	marble	tile2
checker	material_map	tiles

Palabras reservadas (Continuación)

clipped_by	max_intersections	torus
clock	max_trace_level	translate
color	merge	triangle
color_map	metallic	turbulence
colour	normal	type
colour_map	no_shadow	union
component	object	up
composite	octaves	use_color
cone	omega	use_colour
crand	once	use_index
cubic	onion	u_steps
cylinder	open	version
declare	phase	v_steps
default	phong	water_level
dents	phong_size	waves
difference	pigment	wood
diffuse	plane	wrinkles
direction	point_at	x
disc	poly	y
distance	pot	z
dump	quadric	
falloff	quartic	
filter	quick_color	
finish	quick_colour	
flatness	radial	
fog	radius	
frequency	raw	
gif	red	
gradient	reflection	
granite	refraction	
green	rgb	

5.2.3 COMENTARIOS

Los Comentarios son texto que hace más legibles los archivos. Los comentarios no tienen ninguna acción sobre la imagen final. Se ponen de forma muy semejante a como se luce en "C". si se desea ignorar lo que resta de una línea se pone una doble diagonal "// comentario" y si se desea que sea un párrafo el comentario o este en medio de una línea se coloca entre diagonal - asterisco : /* comentario */

Ejemplo :

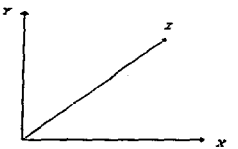
```
// Sencillo programa de prueba
  // Incluye el archivo que define varios colores
  #include "Colors.inc"

  // Define la camara
  camera {
   location <24, -40, 9> /* Ubicación de la camara */
   direction <0.0, 0.0, 2.5>
   sky <0.0, 0.0, 1.0> /* Define Sistema Derecho de */
   up <0.0, 0.0, 1.0> // coordenadas (Z es eje Vertical)
   right <1.3333, 0.0, 0.0> // Configura la relación Alto/Ancho
   look_at <5.75, 0, (3.25+4)> // Hacia donde observa
  }
```

5.2.4 VECTORES

POV-Ray opera en un sistema coordenado en tres dimensiones el cual se determina por medio de 3 vectores perpendiculares entre si : X, Y, Z así que son necesarios para especificar un punto estos tres valores. Un "vector" es un conjunto de 3 valores expresados con números reales. Para definir un vector se expresan tres números reales entre picoparéntesis, los tres valores son separados por comas. Por ejemplo:

< 3.4, -5.32, 12.1 >



Conformación típica del sistema coordenado.

Las comas son necesarias para mantener la legibilidad de el archivo y para evitar que el programa que revisa la sintaxis del archivo se confunda.

5.3 LUCES

Las fuentes de luz no son objetos visibles, pero son necesarios para poder iluminar los otros objetos, sin una fuente de luz no habrá ningún objeto visible.

Las fuentes de luz son puntuales, es decir son infinitamente pequeñas. Los puntos luminosos son tratados como figuras, pero en realidad son puntos invisibles que emiten los rayos luminosos que hacen visibles a otros objetos. Las fuentes de luz crean los brillos, reflejos y sombras, se pueden emplear múltiples fuentes de luz en una escena, pero esto incrementa bastante el tiempo de generación de la imagen.

La intensidad luminosa no disminuye con la distancia, lo cual afecta un poco el realismo, ya que es bien sabido que la iluminación es inversamente proporcional al cuadrado de la distancia, pero el efecto de los rayos luminosos sobre los cuerpos se debe a el ángulo de incidencia, ya que los rayos luminosos tienden a ser paralelos conforme la distancia aumenta, lo cual aumenta el realismo.

La sintaxis para una fuente de luz es :

```
light_source { <x, y, z> color definicion_de_color}
```

El vector entre picoparéntesis indica la posición de la fuente de luz, definicion_de_color debe ser un color previamente definido, o bien se debe definir un color por medio de el indicador rgb y los componentes de dicho color por ejemplo si deseáramos una fuente de luz de color rojo en la posición 10, 5 , 7 escribiríamos :

```
light_source { <10, 5, 7> color rgb <1, 0, 0>}
```

Ejemplo con un color predefinido.

```
light_source { <-5.6, 12,  7.5> color Gris}
```

Donde Gris es un color previamente definido y la fuente de luz se encuentra en X=-5.6, Y=12, Z=7.5.

5.4 CAMARA

Ninguna escena podrá ser observada si no existe una cámara que será el punto donde se observará el resultado de la generación de la imagen.

```
camera {
  location <3,5,-10>
  look_at  <0,2,1>
}
```

Los dos vectores más importantes en POV-Ray son los vectores de localización y el de "punto de mira", ambos vectores son obligatorios, es decir, se requieren forzosamente, el vector de localización indica cual es la posición de la cámara en el espacio, el segundo vector (punto de mira) indica el punto en el espacio hacia el cual la cámara apunta.

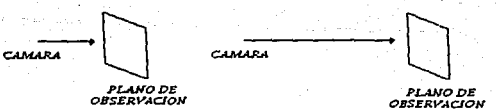
EL VECTOR SKY

El vector Sky o cielo indica la orientación de los ejes de referencia, este vector es opcional, así que puede ser omitido en la declaración de la cámara.

```
camera {
 location <3,5,-10>
 sky     <1,1,0>
 look_at  <0,2,1>
}
```

Vector Direction

La longitud del vector de dirección permite obtener vistas como si empleáramos un telefoto o un gran angular.



Gran Angular Telefoto

Si no es definido valor para el vector de dirección este tomará por defecto el valor <0,0,1>.

Vector UP

El vector UP define cual será el "arriba" de nuestra imagen.

Relación Anchura/Altura

Para obtener un Aspect/Ratio de 4/3 (Tipico en las computadoras) emplee "up" <0,1,0>" y "right <1.33,0,0>" .

5.5 OBJETOS DE USO FRECUENTE

5.5.1 ESFERAS

Uno de los objetos que se utilizan con mayor frecuencia son las esferas, los cuales por cierto son objetos que manejan algoritmos muy optimizados de trabajo. La sintaxis para definirlas es:

```
sphere { <CENTRO>, RADIO }
```

Donde <CENTRO> es un vector de posición que especifica el centro de la esfera y RADIO es el valor del radio de la esfera (número real).
Ejemplo :

```
sphere { <0, 25, 0>, 10
  pigment {Blue}
}
```

5.5.2 CAJAS

Las cajas o paralelepípedos son figuras muy simples que son de gran ayuda a la hora de realizar objetos más complejos.

La sintaxis es :

```
box { <ESQUINA1>, <ESQUINA2> }
```

Donde ESQUINA1 y ESQUINA2 son vectores de posición que definen esquinas diagonalmente opuestas.

Por ejemplo :

```
box { <0, 0, 0>, <1, 1, 1> }
```

Los elementos de ESQUINA1 deben ser menores a sus correspondientes elementos de ESQUINA2.

```
box {< 3,  3,    3>,   < 5, 0.5,  4>
     pigment { DMFWood4 }
}
```

5.5.3 CILINDROS

Los cilindros se definen como:

```
cylinder {<1, -1, 1>, <3,3,3>, 0.7
 open
 textura {New_Brass}
}
```

5.5.4 PLANOS

Los planos son objetos de extensión infinita que están implementados en POV-Ray de una forma muy sencilla, se describen en forma de un vector los componentes de un plano.[1]

```
plane {
     <0, 1, 0>, 0
     pigment {
       checker
         color Red
         color Blue
     }
}
```

[1] Si tiene problemas para entender esto consulte los temas correspondientes a el plano en el espacio, les recomiendo en particular el libro "El cálculo con Geometría Analítica" de Louis Leithold.

El vector que define un plano es el vector normal a dicho plano, así podemos definir un plano empleando un solo vector.

Para ahorrarnos trabajo POV-Ray tiene predefinidos los vectores <1,0,0>, <0,1,0> y <0,0,1> pueden ser referenciados por medio de los identificadores "x", "y", y "z".

Así podemos emplear estos identificadores para definir un plano de la siguiente forma :

```
plane {
      y,0
      pigment {... etc.
```

Un plano que se exprese como una función polinomial de la forma :

$Ax + By + Cz + D = 0$

Puede ser representado en POV-Ray de la forma :

plane { <A, B, C>, D }

5.5.5 CONOS

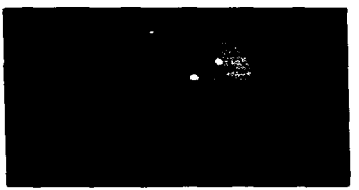
La Figura cono esta definida por el centro y el radio de cada extremo. En este ejemplo uno de los extremos esta en <0,1,0> y tiene un radio de 0.3 mientras el otro extremo se encuentra ubicado en <1,2,3> con un radio 1. Para que un cono tenga punta en uno de los extremos ponga radio 0.

```
cone {
      <0,1,0>,0.3
      <1,2,3>,1.0
      pigment {DMFWood4  scale 4 }
      finish {Shiny}
    }
```

5.6 TEXTURAS

5.6.1 TEXTURAS PREDEFINIDAS

POV-Ray tiene varias texturas predefinidas en los archivos include estándar más específicamente en "textures.inc" y "stones.inc".



```
sphere {
      <0, 1, 0>, 2.5
      texture {
        pigment {
          DMFWood4
          scale 235
        }
        finish {Shiny}
      }
    }
```

La palabra "texture" indica que adicionaremos una textura a un objeto, puede ser una previamente definida o una definida en ese preciso lugar.

La forma de definir una textura es :

```
texture {
  IDENTIFICADOR_DE_LA_TEXTURA
  pigment {...}
  normal {...}
  finish {...}
```

PIGMENTOS

Los colores o patrones de colores de un objeto son definidos por la sentencia "pigment", un pigmento es una parte de la especificación de la textura.

La forma completa de definir un pigmento :

```
pigment {
  IDENTIFICADOR_DEL_PIGMENTO
  TIPO_DE_PATRON
  MODIFICADORES_DEL_PIGMENTO
  TRANSFORMACIONES...
}
```

Color

El tipo más simple de pigmento es el color sólido. Para especificar un color sólido :

```
pigment {color Blue}
```

La manera corta de especificar un color es :

```
color rgb<0.2, 0.5, 0.9>
```

o

```
color rgbf<0.2, 0.8, 1.0, 0.7>
```

POV-Ray permite definir prácticamente cualquier color mediante un modelo RGB. la sintaxis para definir un color es :

color red valor green valor blue valor

o bien
color rgb <1.0, 0.8, 0.8>

Donde valor puede estar en un rango de 0.0 a 1.0, con lo cual es posible generar teóricamente un número enorme de posibles colores. (Aunque en el archivo de salida sólo sea posible grabar 16.7 Millones de Colores).

ACABADO DE SUPERFICIE

Una de las principales caracteristicas de un programa generador de presentaciones es el manejo de acabados. para lo cual empleamos el parámetro "finish".

```
sphere {
      <0, 1, 2>, 2
      texture {
        pigment {color Yellow}
        finish {phong 1}
      }
    }
```

5.7 PARAMETROS DE RENDER

Para generar la imagen de calidad fotográfica de una escena empleamos POV-Ray con los siguientes parámetros:

POVRAY +Inombre.pov +Onombre.tga +FT +W80 +H60

Donde:

nombre.pov es el archivo de texto donde hemos definido nuestra escena

nombre.tga es el archivo de salida en formato TGA.

+FT le indica a POV-Ray que el formato de salida será TGA.

+W80 indica el ancho de la imagen final en pixeles.

+H60 indica el alto de la imagen final en pixeles.

A continuación una tabla que describe los diferentes parámetros:

+Annn	0.0 a 3.0	Genera una imagen con anti-alias o "suavizado", valores pequeños son para más "suavizado".
+A		Usa el valor defecto 0.3 anti-alias
-A		Apaga el anti-alias (defecto)
+Bnnn o -Bnnn		Tamaño del buffer del archivo de salida
+C		Continua una imagen parcial abortada
-C		comienza el la generación de la imagen desde la 1rá. linea
+Dxxx		Despliega gráficamente la imagen mientras se genera.
+FT		Indica la salida en formato TGA
+FD		Indica la salida en formato dump
+FR		indica la salida en formato raw
-F		Desabilita la salida a archivo
+Hnnn	1 a 32.767	Altura de la imagen en pixeles.
+Lruta	+Lruta	Ruta de la libreria
Ruta de la libreria		Nombre del archivo de salida
+Qn	0 a 9	Calidad de la imagen
+Wnnn	1-32.768	Ancho de la imagen

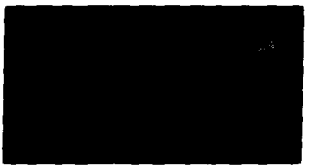
5.8 TRANSFORMACIONES

Es posible realizar transformaciones a los objetos, para que estos se adapten a nuestras expectativas, las transformaciones son :

5.8.1 TRANSLACION

Un objeto puede ser trasladado a otra posición, la cual se indica por medio de un vector.

```
sphere { <10, 10, 10>, 1
    pigment { Green }
}
```



```
sphere { <10, 10, 10>, 1
   pigment { Green }
   translate <-15, 2, 1>
}
```

5.8.2 ESCALA

Es posible cambiar el tamaño de un objeto en cualquiera de sus ejes, las escalas pueden aumentar el tamaño del objeto para lo cual se emplean como factores números mayores que 1.0 y para reducir el tamaño de un objeto se emplean escalas menores que 1.0, se pueden modificar los tamaños de x, y, z independientemente. Para esto se emplea el modificador : scale <x, y ,z>.

5.8.3 ROTACION

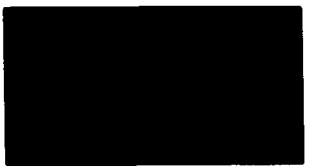
Se puede cambiar la dirección de cualquier objeto definiendo rotación, la cual puede ser con respecto a cualquier eje cualquier número de grados.
Para esto se emplea el modificador : rotate <x, y, z>.

5.9 OPERACIONES ENTRE OBJETOS

5.9.1 UNION

Una de las herramientas que permiten generar imágenes complejas es la unión, la cual se logra al unir 2 o más objetos simples para generar objetos más complicados.
Por ejemplo:

```
union {
  sphere {<0,0,0>,1}
  box {< -4, 0.2, -1>,<5, -0.4,
1> }
 pigment {Blue}
}
```



5.9.2 INTERSECCION

Una intersección es la función lógica de la disyunción, es decir el resultado obtenido de la intersección es el conjunto de puntos que pertenecen a los dos objetos originales.

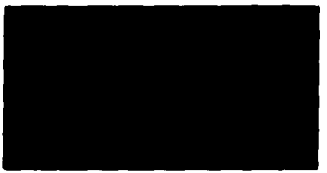
Por ejemplo:

```
intersection {
  sphere {<0,0,0>,1}
  box {< -4, 0.2, -1>,<5, -0.4, 1> }
 pigment {Blue}
}
```

5.9.3 DIFERENCIA

La diferencia de dos conjuntos es el conjunto de todos los puntos que pertenecen al primer conjunto pero no al segundo.

Por ejemplo:



```
difference {
      sphere {<0,0,0>,1}
      box {< -4, 0.2, -1>,<5, -0.4, 1> }
     pigment {Blue}
    }
```

CAPITULO 6

EVALUACION DE LOS PROGRAMAS : DTA, VFD, AAPLAY, F3D-DOS.

CAPITULO 6
EVALUACION DE LOS PROGRAMAS : DTA, VFD, AAPLAY, F3D-DOS

Las descripciones de los siguientes programas fue obtenida de los archivos de documentación que acompañan dichos programas, no es una traducción de los mismos, y la documentación original incluye más información que la aquí mencionada. La información aquí contenida es la que se consideró más relevante para el presente trabajo.

Nota : Para ampliar la información al respecto de estos programas consulte la bibliografía donde se indican las fuentes.

6.1 PROGRAMA DTA

DTA es un programa compartido, es decir, puede ser empleado para probarse sin compromiso, y después de un cierto período de prueba se paga si se desea continuar con su uso o se borra, todo esto es únicamente de buena fé.

Este programa se emplea desde la línea de comandos para crear animaciones en formato fli o flc, a partir de archivos de archivos de los siguientes tipos:

- o .TGA
- o .PNG
- o .JPG
- o .IMG
- o .PCX
- o algunos .TIF
- o .DIB o .BMP
- o otros .FLI o .FLC

DTA permite una gran variedad de operaciones sobre las imágenes que serán procesadas, incluyendo entre otras:

* Crear una paleta optima para toda la secuencia de imágenes de entrada en color verdadero para crear archivos de animaciones en formato FLI o FLC (Dependiendo de el tamaño de los archivos de entrada o los parámetros)

* Puede leer la paleta de archivos con formato COL o MAP.

* Agrega nuevos cuadros a la animación empleando la paleta de dicho archivo.

* Crea (o agrega a) animaciones de archivos de alto color FLH (16384 o 32768 colores) o archivos de color verdadero FLT (16777216 colores).

* Crea paletas en formato .MAP (PICLAB, FRACTINT) o .COL (Autodesk Animator)

Sintaxis:

Sintaxis: DTA (nombrearchivo) (opciones)
Formatos de Entrada:

.TGA	Targa 8-,16-,24-,32-bit comprimidos/sin compresión.
.PCX	PC Paintbrush 256-color o imágenes PCX de 24 bits
.DIB/.BMP	Archivos de mapa de Bits de Microsoft Windows
.FLI y .FLC	Archivos de animación Autodesk Animator/Animator Pro
.VAN VRLI	Archivos de animación VistaPro de 256 colores
.LZH LHArc	Archivos comprimidos conteniendo archivos en formato gráfico
.ZIP PKZIP	Archivos comprimidos conteniendo archivos en formato gráfico
@guión	Procesa todos los archivos listados en un archivo llamado "guion.SCR"

Por defecto, DTA creará un archivo de animación flic (.FLI/.FLC/.FLH/.FLT). El tipo depende de la resolución y de la cantidad de colores que usted seleccione. 320x200 256-colores producirán un flic, para cualquier otra resolución con 256 colores creará un archivo .FLC 15- y 16-bit de profundidad del color generaran archivos .FLH y archivos de 24-bit de profundidad de color producirán una archivo .FLT.

Con las siguientes opciones se determina el formato de entrada.

/FT (Truevision Targa, .TGA)
/FB (Windows Bitmap, .BMP)
/FM (.MAP paleta)
/FR (MindImage, .RLE)
/FP (ZSoft, .PCX)
/FD (Bitmap de Windows)
/FC (.COL paleta)
/FI (Tagged Image Format, .TIF (En escala de grises sin compresión))
/FJ (JPEG,.JPG)
/FV (animación .VAN)
/FPNG (Portable Network Graphics, .PNG)

Opciones

Por defecto, DTA creará un archivo de salida con las mismas dimensiones que los archivos de entrada. Cuando se crea una animación de imágenes de varios tamaños se emplea la del primer cuadro de entrada.

Si desea determinar un tamaño específico puede hacerlo empleando la siguiente opción:

/R# especifica resolución de salida.
(1) 320x200
(2) 320x240
(3) 320x400
(4) 320x480
(5) 360x480
(6) 640x480
(7) 640x400
(8) 800x600
(10) 1024x768
(12) 1280x1024

/R#,# Especifica exactamente resolución.

Opciones de salida para archivos .FLI

/FLAP Agrega nuevos cuadros a un archivo flic existente
/FLC construye un archivo .FLC cuando el tamaño es 320x200 (solamente a 8 bits)
/Snnn especifica la velocidad de despliegue; default=0
/EO compresión "par-impar"

Opciones de salida para archivos .TGA

/NC no comprime archivos .TGA

Opciones de salida para archivos .PNG

/PB usa el mejor filtro para cada imagen. Este es muy lento, porque DTA debe salvar 5 veces cada imagen para ver que filtro crea el archivo mas pequeño.

Opciones de paleta para mapas de color de salida

/G	Emplea una escala de grises de 256 niveles.
/G32	Emplea una escala de grises de 32 niveles.
/332	Usa la paleta 3/3/2.
/Unombrepal	Usa un archivo existente .MAP o .COL para la paleta.
/M#	Configura el número máximo de colores.
/RO	Genera paleta usando reducción de color por árbol octal.
/FADE#	Difuminar entre dos imágenes por # de cuadros
/REP#	Repite un cuadro # veces
/DF	suavizado con filtro Floyd-Steinberg
/DS	suavizado con filtro Sierra Lite

Guiones:

Como mencionamos anteriormente, usted puede procesar todos los archivos listados en un archivo de texto especificando el nombre precedido por un "@".

6.2 DESCRIPCION DEL PROGRAMA VFD

VFD es un programa para MS-DOS que puede crear Archivos .FLC, .FLI y .AVI a partir de:

Archivos Windows .RDI (RIFF DIB)
Archivos Windows .BMP (1, 4, 8, 16 y 24bits)
Archivos Windows .RLE (4 y 8-bit, incluyendo deltas)
Archivos IBM OS/2 .BMP (1, 4, 8 y 24 bits)
Archivos Truevision .TGA (8, 16, 24 y 32-bit rle y sin compresion)
Archivos Zsoft .PCX (8 y 24 bits)
Archivos Autodesk .FLC y .FLI
Archivos Windows .AVI (RLE y 8, 16, y 24-bit DIB)
Archivos MPEG RAW producidos por DMPEG.EXE v1.1 (8 y 24 bits)

VFD usa un avanzado método de cuantización de color llamado Hilite La promoción Hilite produce video en movimiento de 8 bits con una gran fidelidad de imagen y color a partir de un diverso rango de archivos de imágenes y paletas empleando cuantización escalable cuantización (reducción de color) y difuminado de intensidad variable. Puede ser empleado para crear una paleta optima conteniendo los colores más comunes del archivo de entrada.

PARAMETROS:

Los parámetros siguen a el nombre de archivo de entrada, están separados por espacios y se identifican por medio de slash '/' o un guión '-'. Los parámetros pueden colocarse en cualquier orden.

VFD archivo de entrada [parámetros]

-I	Crea un archivo .FLI (por defecto es .FLC)
-An	Crea un archivo .AVI n=1 8 bits DIB, n=2 RLE
-n=4	8-bit .PCX, n=5,6,7: 8,16,24-bit .TGA sin compresión
-Sn	Velocidad de reproducción en cuadros/segundo n=0.1 a 99
-Ofname	Ruta de salida
-Tn1[,n2]	Total de cuadros
-Mn	Compresión n=1 50%, n=2 67%, n=3 75%, n=4 80%, ..., n=99 99%

VFD puede leer cualquiera de los archivos gráficos listados anteriormente, además es posible emplear comodines ('*' o '?') para identificar series de imágenes para la creación de animaciones.

Se pueden crear guiones mediante archivos de texto con extensión .MAK.

NOTA: En las palabras clave se pueden emplear indistintamente mayúsculas y minúsculas, para colocar un comentario se emplea el punto y coma.

El máximo contador de cuadros es 65535. El tamaño máximo de el archivo makefile es 16kb.

PROMOCION HILITE :

Cualquier combinación de tipos de archivos de imágenes de entrada y formatos de color (de 1 a 32 bit) pueden ser combinados en una makefile VFD. Si el contador total de color excede 256, Promoción Hilite y suavizados serán empleados para obtener la salida óptima para video de 8 bits. VFD construye un histograma de frecuencia de todos los colores de una imagen de entrada. La paleta del video en movimiento consiste en los 256 colores más frecuentes.

Hilite es un método de reducción de colores el cual produce resultados mejores o similares a los métodos estándares, y normalmente con archivos más pequeños de salida.

COMPRESION:

Existen dos formas de comprimir video en movimiento:

1) Reduciendo la calidad de la imagen de cada cuadro.
2) Reduciendo el número de cuadros por segundo.

Ejemplo de la compresión obtenida con diferentes valores :

Parámetro			Compresión
-M1	Escribe 1, salta 1, Escribe 1, ...	Velocidad/2	50%
-M2	Escribe 1, salta 2, Escribe 1, ...	Velocidad/3	67%
-M3	Escribe 1, salta 3, Escribe 1, ...	Velocidad/4	75%
...			
-M99	Escribe 1, salta 99, Escribe 1, ...	Velocidad/100	99%

PALETAS:

La paleta del video en movimiento es creada a partir de los colores mas frecuentes, pero puede ser especificada otra mediante el parámetro -pn, donde n indicará el formato de la paleta :

-P1 RIFF .PAL (PalEdit VFW)
-P2 DIB .PAL (Paintbrush)
-P3,4 AA,AAPro .COL (Autodesk Animator, AA Pro)
-P5 ascii .MAP (PicLab - DOS Dominio Público)
-P6 ascii .PAL (Paint Shop Pro)
-P7 ascii .PAL (NeoPaint)
-P8 genérica .PAL (Improcess)

Paletas

La opción de paleta de Windows (-Wn) agrega los 20 colores de sistema reservados de Windows a la paleta del video. Para agregar estos colores a la paleta se le sustraen a la paleta los 20 colores menos empleados de la paleta.

PALETAS PREDEFINIDAS:

-F1 = 256 Paleta empleada por Windows INDEO
-F2 = Paleta de 32 niveles de gris
-F3 = 32 Colores medios tonos idéntica a Windows
-F4 = Paleta de Windows para Blanco y negro

REDIMENSIONADO

El parámetro '-R1' permite redimensionar el tamaño vertical y horizontal de la imagen de entrada Ejemplo: 640x480 -> 320x240.

La resolución máxima de un archivo de imagen de entrada es 1024x768. El tamaño de los archivos de imágenes de entrada deben ser también menores o igual a esta resolución.

Muchos archivos .AVI son comprimidos con formatos para los cuales no hay controladores públicos para DOS. Estos archivos deben ser convertidos a formato RLE o a cuadros completos con VidEdit de Video For Windows con un programa similar.

VELOCIDAD:

La velocidad de reproducción es del rango de 0.1 a 99 cuadros/seg. Ejemplo:

-S.1 = 10 segundos por cuadro
-S.2 = 5 segundos por cuadro
-S10 = 10 cuadros por segundo

VELOCIDAD DE TRANSFERENCIA:

La velocidad de transferencia requerida por la velocidad de ejecución es desplegada para los archivos .FLC, .FLI y .AVIs cuando estos son generados por VFD. El primer cuadro de los .FLCs y .FLIs es omitido de los cálculos debido a que solo es desplegado una vez y después es reemplazado por un cuadro "delta" que es la diferencia entre cuadros.

AUDIO:

VFD salta sobre las pistas de audio de los archivos de entrada. Los programas de Video For Windows VidEdit y WavEdit pueden ser usadas para agregar audio a los archivos creados por VFD.

MEMORIA Y REQUIRIMIENTOS DE SISTEMA:

- 500Kb de Memoria convencional y 96Kb de memoria XMS para procesar imágenes de 320 x 200 para imágenes mayores se requiere de más memoria XMS.

- Sistema operativo DOS 3.1 o superior

- Procesador 80286 o superior

- Una tarjeta de video SVGA es requerida para reproducir videos a 256 colores. Los archivos FLI emplean los modos estándar del VGA.

6.3 PROGRAMA AAPLAY

El reproductor de Animaciones para Windows de Autodesk es un versátil programa para reproducir animaciones en formato .FLI o .FLC. Es de libre distribución, sólo requiere conservar todos los archivos inalterados, la documentación legal acompaña a la distribución. Esta es la apariencia del programa :

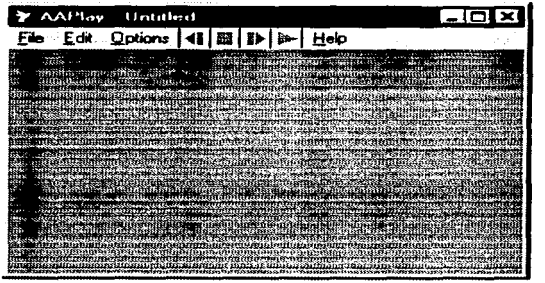


Imagen del reproductor de animaciones de Autodesk.

Los controles para manejarlo son muy semejantes a los de una reproductora de video :

 Regresa un cuadro de la animación.

 Detiene la ejecución de la animación.

 Avanza un cuadro la animación.

 Ejecuta la animación.

Los parámetros de despliegue se configuran en la opción anim settings.. del menú file.

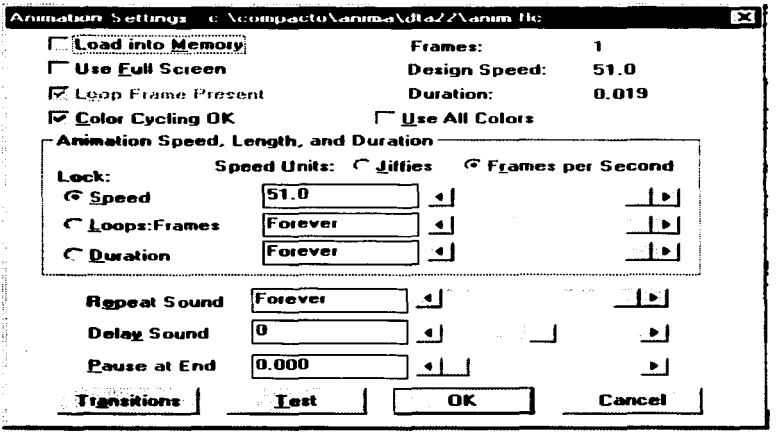


Imagen del cuadro de diálogo de configuración de la reproducción.

En la opción de configuración es posible establecer diferentes valores dependiendo de lo que esperemos de la animación, las opciones son :

Cargar en memoria
Cuando configuramos de esta forma todos los cuadros de la animación son cargados en memoria. Puede ser empleado para aumentar la velocidad de reproducción de animaciones de gran tamaño.

Pantalla Completa
Esta opción le indica a el programa si debe o no reproducir la animación a pantalla completa, de no ser así emplea una ventana para realizar el despliegue.

Usar todos los colores
Permite reservar toda la paleta de colores para la animación.

Velocidad.
La velocidad de la reproducción se expresa en cuadros por segundo.

Ciclo

Repite la animación cuando termina, de este modo la animación es constante

6.4 Programa F3D-DOS

Este es un programa de libre distribución, se puede obtener de varios lugares en Internet, (yo lo baje de (www.cdrom.com/pub/povray/). La función de este programa es generar archivos .INC propios para POV-Ray a partir de archivos .TTF los cuales contienen definiciones de letras para Windows para generar letras en 3D con las texturas que deseemos.

La sintaxis es la siguiente :

/inombrearchivo	Nombre del archivo del fuente true type a emplear
/onombrearchivo	Nombre del archivo de salida.
/ncadena	Nombre del objeto a generar (como se referenciará en POV-Ray.
/cnnn	Código del caracter ASCII a generar.
/rnnn	Resolución de la curva.
/dfloat	Profundidad del objeto a generar.
/eb	Bordes planos.
/er	Bordes redondeados.
/tnnn	Angulos para los triángulos de salida.

CAPITULO 7

PROGRAMA GENERADOR DE ANIMACIONES DE TIRO PARABOLICO.

CAPITULO 7 PROGRAMA GENERADOR DE ANIMACIONES

Para la elaboración del programa generador de animaciones seleccione el lenguaje Pascal debido a que es el que mejor manejo y tiene amplia difusión, aunque es fácil migrar a lenguaje "C".

7.1 LA INTERFACE DE USUARIO

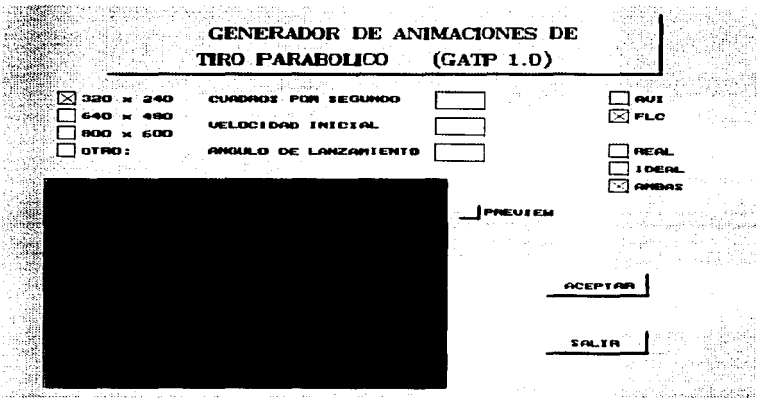
La interface de usuario es la única parte de todo el programa que no emplea funciones estándar de Pascal, pero puede ser fácilmente modificable para solo mostrar pantallas de texto explicativo y entrada de datos, así que con gran facilidad se puede convertir en un programa de Pascal estándar.

Propuestas para la interface de usuario

La primera propuesta para la interface es una relativamente sencilla, la cual puede fácilmente simplificarse para migrar a ANSI es decir a lenguaje estándar

Primera propuesta de Interface de Usuario

La segunda propuesta es una un poco más elaborada, ya que emplea el ratón y modo gráfico, de modo que explota muchas opciones que existen solo en PC's, esta interface pudiera ser implementada en Xwindows pero requeriría ser reescrita por completo :



Segunda propuesta para la Interface de Usuario

7.2 EL PROGRAMA PRINCIPAL

El programa principal inicia con una directiva al compilador para que este reserve espacio en la pila de memoria para poder emplear las llamadas al sistema operativo y verificar el uso del ratón. Esta línea sólo se empleará en caso de usar la segunda alternativa de interface de usuario.

```
{$M $4000,0,45536 }
```

La cabecera del programa indica el identificador del programa :

```
PROGRAM GATP10;
```

Para indicar que librerías serán empleadas hacemos uso de la cláusula USES :

```
USES
  GRAPH, CRT, U3D, POV, DOS, RATON;
```

Como se observa se emplearan las rutinas incluidas en Turbo Pascal versión 7.0 : GRAPH, CRT, DOS y las rutinas propias : U3D, POV y RATON.

UNIDAD	DESCRIPCION
GRAPH	Unidad que proporciona rutinas para el manejo de gráficos.
CRT	Unidad para el manejo de la pantalla en modo texto.
DOS	Unidad para el uso de pilas, y llamadas al sistema operativo.
U3D	Unidad propia que incluye algunas funciones y procedimientos para el manejo de vectores.
POV	Unidad propia que incluye funciones y procedimientos para la escritura de archivos .POV
RATON	Unidad propia para el manejo del ratón.

Dado que algunos valores empleados no varían dentro de todo el programa es conveniente el empleo de constantes, a continuación se muestran las empleadas por el programa, así como una breve descripción de su funcionamiento.

```
CONST
  AREAS : ARRAY[1..16,1..4] OF WORD =
    ( (60,120,75,135) , (60,140,75,155) , (60,160,75,175) ,
      (60,180,75,195) , (360,120,400,140) , (360,150,400,170) ,
      (360,180,400,200) , (500,120,515,135) , (500,140,515,155) ,
      (500,180,515,195) , (500,200,515,215) , (500,220,515,235) ,
      (450,330,530,355) , (380,250,395,265) , (450,395,530,420) ,
      (0,0,0,0) );

      G = 9.8;


      DIG : SET OF CHAR = [#48..#57];
```

CONSTANTE	DESCRIPCION
Areas	Esta constante es un arreglo que almacena las coordenadas de las áreas de elección para las opciones para la interface de usuario gráfica.
G	Constante de Gravitación Universal. Empleada por las funciones de cálculo de tiro parabólico
DIG	Esta constante define un conjunto de caracteres.

Este programa incluye muchas variables, dado que esta diseñado para incrementarse de forma flexible se declaran variables que posiblemente no se requieran de momento.

```
VAR
  FPS, MAXCOLORS, CONT : WORD;
  NumFrames, X, Xmax, Ymax, CONTROL, MODO : Integer;
```

```
Vini, Ang, ANGDEG, Temporal, Temporal2, Ang1, Temp1, Temp2 : Real;
ConFrames, Cont1, Cont2 : Integer;
A, ProgPov, Parametros, CadAux, NomArch : String;
xint, Yint : integer;
R_plano, Radio, Xalcance, Alt_maxima, TdeV : Real;
Luz_Pos, Color, Camara_L, Camara_D, Camara_Up, Camara_Mira,
Centro, plano_n : Vector;
Textur, textur2 : Atributos;
ANCHO, ALTO : WORD;
FORMATO, TRAYECTO : BYTE;
FACTORX, FACTORY : REAL;
```


VARIABLE	DESCRIPCION
FPS	Cuadros por segundo.
MAXCOLORS	Número máximo de colores
CONT	Variable para un contador.
NUMFRAMES	Numero de cuadros.
X	Variable para almacenar el valor de X.
XMAX	Valor maximo de X
YMAX	Valor Máximo de Y.
CONTROL	Valor para el controlador de Video.
MODO	Valor para el modo gráfico.
VINI	Velocidad inicial del cuerpo para calcular tiro parabolico
ANG	Angulo de lanzamiento del cuerpo
ANGDEG	Angulo en grados
TEMPORAL	Variable de uso temporal. (Se emplea para almacenar valores intermedios de calculos)
TEMPORAL2	Variable de uso temporal. (Se emplean para almacenar valores intermedios de calculos).
ANG1	Valor temporal para un angulo
TEMP1	Variable para almacenar un valor intermedio
TEMP2	Variable para almacenar un valor intermedio
CONTFRAMES	Contador de los cuadros
CONT1	Variable para un ciclo
CONT2	Variable para un ciclo
A	Variable de cadena que almacena valores intermedios para los parametros
PROGPOV	Variable de cadena que tiene el nombre del ejecutable de POV-Ray.
PARAMETROS	Variable de tipo cadena que almacena los parametros para POV-Ray.
CADAUX	Cadena Auxiliar para conformar el nombre del archivo de salida.
NOMARCH	Variable de cadena que almacena el nombre del archivo de salida.
XINT	Variable de tipo entero que almacena el valor redondeado de "X".
YINT	Variable de tipo entero que almacena el valor redondeado de "Y".
R_PLANO	Valor del plano en el eje correspondiente
RADIO	Radio de la esfera que efectua la trayectoria parabolica.
X_ALCANCE	Alcance del proyectil en el eje "X"
ALT_MAXIMA	Altura maxima alcanzada por el proyectil
TDEV	Tiempo de Vuelo del proyectil
LUZ_POS	Posicion de la fuente de luz.
COLOR	Vector que se emplea para almacenar un color en formato RGB.
CAMARA_L	Vector de posicion de la camara

CAMARA_D	Vector de posicion que indica el tipo de lente de la camara. (Vea Capitulo 5)
CAMARA_UP	Vector de posicion que indica la referencia del horizonte. (Vea Capitulo 5)
CAMARA_MIRA	Vector de posicion que indica el punto a donde observa la camara.
CENTRO	Vector de posicion del centro de la esfera.
PLANO_A	Vector de posición para los componentes del plano.
TEXTUR	Variable que almacena los atributos de textura del objeto.
TEXTUR2	Variable que almacena los atributos de textura del objeto.
ANCHO	Ancho de la imagen de salida.
ALTO	Alto de la imagen de salida.
FORMATO	Variable de tipo entero que almacena el tipo de formato para la animación.
TRAYECTO	Tipo de trayecto : real , ideal, ambas.
FACTORX	Factor de multiplicacion para adecuar la escala del eje "X" a las dimensiones del monitor.
FACTORY	Factor de multiplicacion para adecuar la escala del eje "Y" a las dimensiones del monitor.

En el programa principal se adoptará la interface de usuario gráfica, de modo que primero se inicializan algunas variables y se inicia el modo gráfico y se dibuja toda la interface, por último se entra al ciclo para determinar las opciones.

```
BEGIN          (*   PROGRAMA PRINCIPAL   *);
  textur2[4]:='texture {pigment { color RGB < 1.0 , 1.0, 1.0>}}';
  textur[4]:='texture {pigment { color RGB < 1.0 , 0.1, 0.1>}}';
  Formato:=1;
  Trayecto:=1;
  CONTROL:=DETECT;
  modo:=0;
  FPS:=18;
  INITGRAPH(CONTROL ,MODO, '');
  XMAX:=GETMAXX;
  YMAX:=GETMAXY;
  SETBKCOLOR(0);
  SETCOLOR(3);
  FOR CONT:=1 TO YMAX DO
    LINE(1,CONT,XMAX,CONT);
  CAJA(1,1,XMAX,YMAX);
  CAJA(100,30,540,100);
  CAJA (AREAS[13,1],AREAS[13,2],AREAS[13,3],AREAS[13,4]);
  CAJA (AREAS[14,1],AREAS[14,2],AREAS[14,3],AREAS[14,4]);
  CAJA (AREAS[15,1],AREAS[15,2],AREAS[15,3],AREAS[15,4]);
  FOR CONT:=1 TO 12 DO
    CUADRO(AREAS[CONT,1],AREAS[CONT,2],AREAS[CONT,3],AREAS[CONT,4],15);
  CUADRO(50,220,370,460,0);
  MOVETO(AREAS[1,1]+20,AREAS[1,2]+5);
  OUTTEXT('320 x 240');
  MOVETO(AREAS[2,1]+20,AREAS[2,2]+5);
  OUTTEXT('640 x 480');
  MOVETO(AREAS[3,1]+20,AREAS[3,2]+5);
  OUTTEXT('800 x 600');
  MOVETO(AREAS[5,1]-180,AREAS[5,2]+5);
  OUTTEXT('CUADROS POR SEGUNDO');
  MOVETO(AREAS[6,1]-180,AREAS[6,2]+5);
```

```
  OUTTEXT('VELOCIDAD INICIAL');
  MOVETO(AREAS[7,1]-180,AREAS[7,2]+5);
  OUTTEXT('ANGULO DE LANZAMIENTO');
  MOVETO(AREAS[4,1]+20,AREAS[4,2]+5);
  OUTTEXT('OTRO: ');
  MOVETO(AREAS[8,1]+20,AREAS[8,2]+5);
  OUTTEXT('AVI');
  MOVETO(AREAS[9,1]+20,AREAS[9,2]+5);
  OUTTEXT('FLC');
  MOVETO(AREAS[10,1]+20,AREAS[10,2]+5);
  OUTTEXT('REAL');
  MOVETO(AREAS[11,1]+20,AREAS[11,2]+5);
  OUTTEXT('IDEAL');
  MOVETO(AREAS[12,1]+20,AREAS[12,2]+5);
  OUTTEXT('AMBAS');
  MOVETO(AREAS[13,1]+15,AREAS[13,2]+10);
  OUTTEXT('ACEPTAR');
  MOVETO(AREAS[14,1]+20,AREAS[14,2]+5);
  OUTTEXT('PREVIEW');
  MOVETO(AREAS[15,1]+20,AREAS[15,2]+10);
  OUTTEXT('SALIR');
  SETCOLOR(1);
  SETTEXTSTYLE(1,0,1);
  MOVETO(180, 40);
  OUTTEXT('GENERADOR DE ANIMACIONES DE');
  MOVETO(170, 70);
  OUTTEXT('TIRO PARABOLICO    (GATP 1.0)');
  SETTEXTSTYLE(1,0,1);
  LEEDATOS;
  READLN;
  CLOSEGRAPH;
END.
```

7.3 CONSIDERACIONES RESPECTO A LAS VARIABLES GLOBALES

Ya que muchas variables se requieren en todo el programa se consideró adecuado colocarlas de forma global, y sólo las variables particulares de cada procedimiento se declararon de forma local, de esta manera podemos realizar la depuración del programa de forma más eficiente.

7.4 VECTORES POR DEFECTO

Dado que es más práctico declarar todos los valores de vectores por defecto en un solo lugar, de este modo se declaran en el procedimiento Vectores_Default, algunos otros valores no pueden ser definidos de inicio, sino hasta que se introducen los datos para los cálculos del tiro parabólico, tales como la posición de la cámara y la fuente de luz , las cuales se obtienen de los resultados del alcance en el eje "X" y de la altura máxima.

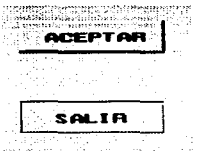
```
Procedure Vectores_Default;
Begin
  Alt_Maxima:=Alt_Max(Vini, Ang, 0);
  Luz_Pos.x:= -xalcance/1.1;
  Luz_Pos.y:= Alt_maxima;
  Luz_Pos.z:= 10;
  Color.x:= 1.0;     Color.y:= 1.0;     Color.z:= 1.0;
  Camara_L.x:= 0;     Camara_L.y:= Alt_maxima/2;
```

```
  Camara_L.z:= - xalcance;
  Camara_D.x:= 0.0;
  Camara_D.y:= 10.0;
  Camara_D.z:= 0;
  Camara_Up.x:=0;
  Camara_Up.y:=1.0;
  Camara_Up.z:=0;
  Camara_Mira.x:=0;
  Camara_Mira.y:=0;
  Camara_Mira.z:=0;
  Plano_a.x:=0.0;
  Plano_a.y:=1.0;
  Plano_a.z:=0;
  r_plano:=0;
End;
```

7.5 PROCEDIMIENTOS PARA LA INTERFACE GRAFICA

7.5.1 PROCEDIMIENTO "CAJA" Y "CAJA2"

El procedimiento caja dibuja un rectángulo que simula volumen, de forma que parece un pequeño botón, mientras que el procedimiento caja2 dibuja un pequeño rectángulo que simula ser un botón oprimido.



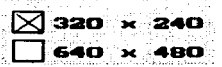

```
PROCEDURE CAJA(X1,Y1,X2,Y2: WORD);
BEGIN
 SETCOLOR(7);
 LINE(X1,Y1,X2,Y1);
 LINE(X1,Y1,X1,Y2);
 SETCOLOR(15);
 LINE(X1+1,Y1+1,X2-1,Y1+1);
 LINE(X1+1,Y1+1,X1+1,Y2-1);
 SETCOLOR(0);
 LINE(X2,Y1,X2,Y2);
 LINE(X1,Y2,X2,Y2);
 SETCOLOR(8);
 LINE(X2-1,Y1,X2-1,Y2);
 LINE(X1,Y2-1,X2-1,Y2-1);
END;

PROCEDURE CAJA2(X1,Y1,X2,Y2: WORD);
BEGIN
 SETCOLOR(0);
```

```
  LINE(X1,Y1,X2,Y1);
  LINE(X1,Y1,X1,Y2);
  SETCOLOR(15);
  LINE(X1+1,Y1+1,X2-1,Y1+1);
  LINE(X1+1,Y1+1,X1+1,Y2-1);
  SETCOLOR(7);
  LINE(X2,Y1,X2,Y2);
  LINE(X1,Y2,X2,Y2);
  SETCOLOR(8);
  LINE(X2-1,Y1,X2-1,Y2);
  LINE(X1,Y2-1,X2-1,Y2-1);
END;
```

7.5.2 PROCEDIMIENTOS CUADRO, MARCA Y DESMARCA

Este procedimiento dibuja un rectángulo que simula volumen, pero no como un botón, sino más bien como un pequeño desnivel en la pantalla. el procedimiento marca dibuja una pequeña cruz en el cuadro correspondiente y el procedimiento desmarca quita dicha cruz. estos procedimientos constan básicamente de líneas y cambios de color.




```
PROCEDURE CUADRO(X1,Y1,X2,Y2, Color: WORD);
VAR
  TEMP : WORD;
BEGIN
  SETCOLOR(color);
  FOR TEMP:=1 TO Y2-Y1 DO
    LINE(X1,Y1+TEMP, X2,Y1+TEMP);
  SETCOLOR(0);
  LINE(X1,Y1,X2,Y1);
  LINE(X1,Y1,X1,Y2);
  SETCOLOR(15);
  LINE(X1+1,Y1+1,X2-1,Y1+1);
  LINE(X1+1,Y1+1,X1+1,Y2-1);
  SETCOLOR(7);
  LINE(X2,Y1,X2,Y2);
  LINE(X1,Y2,X2,Y2);
  SETCOLOR(8);
  LINE(X2-1,Y1,X2-1,Y2);
  LINE(X1,Y2-1,X2-1,Y2-1);
END;

PROCEDURE MARCA(NUMAREA : BYTE);
BEGIN
  SETCOLOR(0);
```

```
  LINE(AREAS[NUMAREA.1]+2,AREAS[NUMAREA.2]+2,AREAS[NUMAREA.3]-
2,AREAS[NUMAREA.4]-2);
  LINE(AREAS[NUMAREA.3]-
2,AREAS[NUMAREA.2]+2,AREAS[NUMAREA.1]+2,AREAS[NUMAREA.4]-2);
 END;

 PROCEDURE DESMARCA(NUMAREA : BYTE);
 BEGIN
  SETCOLOR(15);
  LINE(AREAS[NUMAREA.1]+2,AREAS[NUMAREA.2]+2,AREAS[NUMAREA.3]-
2,AREAS[NUMAREA.4]-2);
  LINE(AREAS[NUMAREA.3]-
2,AREAS[NUMAREA.2]+2,AREAS[NUMAREA.1]+2,AREAS[NUMAREA.4]-2);
 END;
```

7.5.3 PROCEDIMIENTO INDICA

Este es un procedimiento muy sencillo, dependiendo del valor de una bandera cambia el color del primer plano.

```
 PROCEDURE INDICA (X,Y : WORD; ACTIVA :BOOLEAN);
 BEGIN
  IF ACTIVA THEN
    SETCOLOR(0);
  IF NOT(ACTIVA) THEN
    SETCOLOR(3);
    LINE(X+50,Y+10,X+80,Y+10);
 END;
```

7.5.4 PROCEDIMIENTO PREVIEW Y DIBUJA_BALL

El procedimiento preview realiza en una ventana de la pantalla un pequeña simulación de lo que será la animación del tiro parabólico, este procedimiento hace uso de otro denominado dibuja_ball, el cual se encarga de dibujar el pequeño circulo que simula la esfera.

El procedimiento de previsualizar es básicamente un ciclo que llama que itera desde 1 hasta num_frames, es decir, hasta el numero de cuadros que tendrá la animación, también se realizan calculos de escalamiento tanto en "X" como en "Y" para que la previsualización pueda caber en la ventana.

```
 PROCEDURE PREVIEW;
 Var
   Centro2: Vector;
 BEGIN
  Angdeg:=Deg2Rad(Ang);
  xalcance:=alcance(Vini, Angdeg, 0);
  FACTORX:=310/XALCANCE;
  FACTORY:=190/ALT_MAX(VINI, ANGdeg, 0);
  Vectores_Default;
  TdeV:=Tiempo_Vuelo(Vini, Angdeg, 0);
  NumFrames:=Round(TdeV*FPS);
  { 18 Cuadros se Permiten en video sin problemas "}
```

```
  Radio:=2;
  if trayecto=1 then
   For ContFrames:=1 to NumFrames-1 do
   Begin
    Centro.x:=Obten_xR(Vini, Angdeg, ContFrames/FPS, 0.18, 1);
    Centro.y:=Obten_yR(Vini, Angdeg, ContFrames/FPS, 0.18, 1);
    Centro.z:=0;
    setcolor(14);
    DIBUJA_BALL(CENTRO);
   End;
  if trayecto=2 then
   For ContFrames:=1 to NumFrames-1 do
   Begin
    Centro.x:=Obten_x(Vini, Angdeg, ContFrames/FPS, 0, 0);
    Centro.y:=Obten_y(Vini, Angdeg, ContFrames/FPS, 0, 0);
    Centro.z:=0;
    setcolor(blue);
    DIBUJA_BALL(CENTRO);
   End;
  if trayecto=3 then
   For ContFrames:=1 to NumFrames-1 do
   Begin
    Centro.x:=Obten_x(Vini, Angdeg, ContFrames/FPS, 0, 0);
    Centro.y:=Obten_y(Vini, Angdeg, ContFrames/FPS, 0, 0);
    Centro.z:=0;
    setcolor(blue);
    DIBUJA_BALL(CENTRO);
    Centro2.x:=Obten_xR(Vini, Angdeg, ContFrames/FPS, 0.18, 1);
    Centro2.y:=Obten_y(Vini, Angdeg, ContFrames/FPS, 0, 0);
    (*Centro2.y:=Obten_yR(Vini, Angdeg, ContFrames/FPS, 0.18, 1);*)        Centro2.z:=0;
    setcolor(14);
    DIBUJA_BALL(CENTRO2);
   End;
  CUADRO(50,220,370,460,0);
 END;

PROCEDURE DIBUJA_BALL(PUNTO : VECTOR);
  BEGIN
   CIRCLE(60+ROUND(PUNTO.X*FACTORX),420-ROUND(PUNTO.Y*FACTORY),5);
   PUTPIXEL(60+ROUND(PUNTO.X*FACTORX),420-ROUND(PUNTO.Y*FACTORY),10);
   DELAY(100);
 (*  SETCOLOR(0);
   CIRCLE(60+ROUND(PUNTO.X*FACTORX),420-ROUND(PUNTO.Y*FACTORY),5);*)
  END;
```

7.6 PROCEDIMIENTO ANIMA

El procedimiento ANIMA es el corazón del programa, ya que toda la animación será realizada principalmente en este lugar.

```
  PROCEDURE ANIMA;
  BEGIN
   Ang:=Deg2Rad(Ang);
   Salcance:=alcance(Vini, Ang, 0);
```

```
  Vectores_Default;
  TdeV := Tiempo_Vuelo (Vini, Ang, 0);
  NumFrames:=Round(TdeV*FPS);
  { 18 Cuadros se Permiten en video sin problemas '}
  Radio:=2;
  For ContFrames:=1 to NumFrames do
  Begin
   Centro.x:=-(xalcance/2)+Obten_x(Vini, Ang, ContFrames/FPS, 0, 0);
   Centro.y:=Obten_y(Vini, Ang, ContFrames/FPS, 0, 0);
   Centro.z:=0;
   Str(ContFrames, CadAux);
   If CadAux[0]=#1 then CadAux:='000'+CadAux;
   If CadAux[0]=#2 then CadAux:='00'+CadAux;
   If CadAux[0]=#3 then CadAux:='0'+CadAux;
   Inicia_Arch('ani'+CadAux+'.Pov');
   Graba_camara('ani'+CadAux+'.Pov', Camara_L, Camara_D, Camara_Up, Camara_Mira);
   Graba_esfera('ani'+CadAux+'.Pov', Centro, Radio,Textur);
   Graba_plano('ani'+CadAux+'.Pov', plano_a, R_plano.Textur2);
   Graba_Luz('ani'+CadAux+'.Pov', Luz_Pos, Color);
  End;
  For ContFrames:=1 to NumFrames-1 do
  Begin
   Str(ContFrames, CadAux);
   If CadAux[0]=#1 then CadAux:='000'+CadAux;
   If CadAux[0]=#2 then CadAux:='00'+CadAux;
   If CadAux[0]=#3 then CadAux:='0'+CadAux;
   ProgPov:='c:\povray\pempov.exe';
   a:='ani'+CadAux;
   a:=' +i'+a+'.pov +o'+a+'.tga +ft -d +w320 +h240';
   Parametros:=a;
   SwapVectors;
   Exec(ProgPov, Parametros);
   SwapVectors;
   WriteLn('...Regresando de la ejecucion');
   if DosError <> 0 then{ Error? }
     WriteLn('Error del Dos #', DosError)
   else
     WriteLn('Ejecucion Exitosa. ',
     'Codigo de Salida del proceso = ',
      DosExitCode);
  End;
   Parametros:='ani???.tga';
   { SwapVectors;
   Exec('c:\Compacto\animal\dta22\dta.exe','a C:\PERSONAL\NX6\ANI.SCR');
   SwapVectors;
   WriteLn('...Regresando de la ejecucion');
   if DosError <> 0 then{ Error?
     WriteLn('Error del Dos #', DosError)
   else
     WriteLn('Ejecucion Exitosa. ',
     'Codigo de Salida del proceso = ',
      DosExitCode);}
END;
```

7.7 CICLO PRICIPAL DE CAPTURA

Este procedimiento es el encargado de capturar las acciones del ratón y ejecutar las opciones deseadas.

```
FUNCTION CAP_NUM(A, NUMCAR :WORD) : INTEGER;
VAR
  CH : CHAR;
  NUMERO: STRING;
  CONT : BYTE;
  ERROR : INTEGER;
  VALOR : INTEGER;
BEGIN
  CUADRO(AREAS[A,1],AREAS[A,2],AREAS[A,3],AREAS[A,4],15);
  INDICA(AREAS[A,1], AREAS[A,2], TRUE);
  SETCOLOR(0);
  CONT:=1;
  NUMERO:='';
  SETTEXTSTYLE(2,0,5);
  CONT:=1;
  REPEAT
    CH:=READKEY;
    IF CH IN DIG THEN
      BEGIN
        NUMERO:=NUMERO+CH;
        MOVETO(AREAS[A,1]+LENGTH(NUMERO)*8,AREAS[A,2]);
        OUTTEXT(NUMERO[CONT]);
        CONT:=CONT+1;
      END;
  UNTIL (CH=#13) OR (LENGTH(NUMERO) >=NUMCAR);
  INDICA(AREAS[A,1], AREAS[A,2], FALSE);
  VAL(NUMERO, VALOR, ERROR);
  CAP_NUM:=VALOR;
END;


PROCEDURE LEEDATOS;
BEGIN
  set_speed2(100);
  mouse_cursor(true);
  set_mouse_cursor(5,2,hand);
  MARCA(1);
  MARCA(9);
  MARCA(12);
  ANCHO:=340;
  ALTO:=240;
  FORMATO:=2;
  TRAYECTO:=3;
  VINI:=20;
  ANG:=70;
  while true do begin
    while mousebutton=0 do;
    x:=mouse_area(AREAS);
    if x>0 then begin
```

```
mouse_cursor(false);
case x of
  1: begin
       MARCA(1);
       DESMARCA(2);
       DESMARCA(3);
       DESMARCA(4);
       ANCHO:=320;
       ALTO:=240;
     end;
  2: begin
       MARCA(2);
       DESMARCA(1);
       DESMARCA(3);
       DESMARCA(4);
       ANCHO:=640;
       ALTO:=480;
     end;
  3: begin
       MARCA(3);
       DESMARCA(2);
       DESMARCA(1);
       DESMARCA(4);
       ANCHO:=800;
       ALTO:=600;
     end;
  4: begin
       MARCA(4);
       DESMARCA(2);
       DESMARCA(3);
       DESMARCA(1);
     end;
  5: begin
       CUADRO(AREAS[5,1],AREAS[5,2],AREAS[5,3],AREAS[5,4],15);
       FPS:=CAP_NUM(5,2);
     end;
  6: begin
       CUADRO(AREAS[6,1],AREAS[6,2],AREAS[6,3],AREAS[6,4],15);
       VINI:=CAP_NUM(6,3);
     end;
  7: begin
       CUADRO(AREAS[7,1],AREAS[7,2],AREAS[7,3],AREAS[7,4],15);
       ANG:=CAP_NUM(7,2);
     end;
  8: begin
       MARCA(8);
       DESMARCA(9);
     end;
  9: begin
       MARCA(9);
       DESMARCA(8);
     end;
  10: begin
        MARCA(10);
```

```
            DESMARCA(11);
            DESMARCA(12);
            Trayecto:=1;
          end;
        11: begin
            MARCA(11);
            DESMARCA(10);
            DESMARCA(12);
            Trayecto:=2;
          end;
        12: begin
            MARCA(12);
            DESMARCA(10);
            DESMARCA(11);
            Trayecto:=3;
          end;
        13: begin
            CAJA2(AREAS[13,1],AREAS[13,2],AREAS[13,3],AREAS[13,4]);
            DELAY(100);
            CAJA (AREAS[13,1],AREAS[13,2],AREAS[13,3],AREAS[13,4]);
            CLOSEGRAPH;
            ANIMA;
            HALT(0);
          end;
        14: BEGIN
            CAJA2(AREAS[14,1],AREAS[14,2],AREAS[14,3],AREAS[14,4]);
            DELAY(100);
            CAJA (AREAS[14,1],AREAS[14,2],AREAS[14,3],AREAS[14,4]);
            PREVIEW;
          END;
        15: begin
            CAJA2(AREAS[15,1],AREAS[15,2],AREAS[15,3],AREAS[15,4]);
            DELAY(100);
            CAJA (AREAS[15,1],AREAS[15,2],AREAS[15,3],AREAS[15,4]);
            DELAY(50);
          end;
      end;
      mouse_cursor(true);
    end;
  end
END;
```

7.8 UNIDAD POV

La unidad POV incluye varias funciones y procedimientos para la creación de archivos de texto a partir de parámetros de entrada, de modo que los archivos de salida generados de esta manera puedan servir de entrada a POV-Ray.

Las diferentes funciones y procedimientos

Función	Objetivo

Existe_Archivo	Verifica la existencia de un archivo
Inicia_Archivo	Crea un archivo.
Salva_Ani	Guarda la Animación en formato .ANI
Graba_Blob	Graba en un archivo una descripcion de Blob
Graba_Luz	Graba la fuente de Luz
Graba_Camara	Graba la camara
Graba_Esfera	Graba una Esfera
Graba_Plano	Graba un plano en el archivo de Texto
Genera_Shell	Crea el Shell para Renderisar la animación
Genera_Bat	Crea un Bat para Renderizar la Animación
Real2Cad	Convierte un numero Real en una Cadena

No todas las funciones de esta unidad son empleadas, pero fueron definidas para su posterior ampliación, las funciones que se emplean son:

Inicia_archivo
Graba_Luz
Graba_Camara
Graba_Esfera
Graba Plano

Estos procedimientos reciben parametros clave que pueden ser generados a partir de los cálculos de los procedimientos de tiro parabólico. La estructura es muy sencilla reciben los parámetros correspondientes al objeto a crear asi como el nombre del archivo donde será agregado. asi lo unico que se hace es una vez creado el archivo se va abriendo para cada añadir cada uno de los siguientes objetos

Uno de los objetivos de esta unidad será admitir más funciones para crear escenas más complejas, es por ello que se han considerado ya otros objetos tales como BLOB's (metabolas), y cajas, y se considera la implementación de procedimientos que creen archivos shell o bat para generar las imágenes desde UNIX o plataformas PC, de forma que esta unidad permita portabilidad, de hecho las instrucción son en gran medida ANSI.

```
{ Esta Unidad Define Varias Funciones y Procedimientos
  Para la creación de archivos de Texto  Para POVRAY    }

{  Función            Objetivo
```

```
Existe_Archivo   Verifica la existencia de un archivo
Salva_Ani        Guarda la Animación en formato .ANI
Graba_Blob       Graba en un archivo una descripción de Blob
Graba_Luz        Graba la fuente de Luz
Graba_Camara     Graba la cámara
Graba_Esfera     Graba una Esfera
Graba_Plano      Graba un plano en el archivo de Texto
Genera_Shell     Crea el Shell para Renderisar la animación
Genera_Bat       Crea un Bat para Renderizar la Animación
Real2Cad         Convierte un numero Real en una Cadena
                                        }
```

```
Unit POV;


Interface

Uses
  U3D;

Type
   Atributos  = Array[1..6] of String[65];

   Blob = Record
     Threshold : Real;
     Num       : Word;
     Valor     : Array[1..200,2..5] of real;
     Textura   : Array[1..6] of String[25];
    end;

  Procedure Salva_Ani(nombre:string);
  Procedure Graba_Blob(Archivo:String; Objeto:Blob );
  Procedure Graba_Camara(archivo:String;localizacion,direccion,arriba,mira: Vector);
  Procedure Graba_Luz(Archivo:String; Posicion,Color : Vector);
  Procedure Inicia_arch(Archivo: String);
  Procedure Graba_Esfera(Archivo : String ; posicion: Vector; Radio : Real;A:Atributos);
  Procedure Graba_Caja (Archivo:String; inicio,fin:Vector; A: Atributos);
```

```
Procedure Graba_Plano (Archivo: String; Plano : Vector; r: real; A: Atributos);
{Procedure Genera_Bat(NombreBat : String; NombreAni: String; NumC : Word);}
Function  Existe_Archivo(Nombre: String): Boolean;
Function  Real2Cad(Numero: Real):String;

Implementation

Function  Real2Cad(Numero: Real):String;
Var
  CadAux : String;
Begin
  str(Numero:10:5,CadAux);
  Real2Cad:=CadAux;
End;

Function Existe_Archivo(Nombre: String): Boolean;
Var
   Arch: Text;
Begin
  {$I-}
  Assign(Arch, Nombre);
  FileMode := 0;
  Reset(Arch);
  Close(Arch);
  {$I+}
  Existe_Archivo := (Nombre <> '') and (IOResult = 0);
End;

Procedure salva_ani(nombre:string);
Begin
  if Existe_Archivo(nombre) then
  ;
End;

Procedure graba_blob(archivo:String; Objeto:Blob );
Var
```

```
  Arch : text;
  ContBlob : Word;
 Begin
  Assign(Arch, archivo);
  Append(Arch);
  Writeln(Arch,'blob {');
  Writeln(Arch,'threshold '+Real2Cad(Objeto.Threshold));
  For ContBlob:=1 to Objeto.Num do
   Writeln(Arch,'  component 1, '+
        Real2Cad(Objeto.Valor[ContBlob,2])+', <'+
        Real2Cad(Objeto.Valor[ContBlob,3])+', '+
        Real2Cad(Objeto.Valor[ContBlob,4])+', '+
        Real2Cad(Objeto.Valor[ContBlob,5])+'> ');
  For ContBlob:=1 to 6 do
   Writeln(Arch,Objeto.Textura[ContBlob]);
  Writeln(Arch,'}');
  Close(Arch);
 End;

 Procedure Graba_Camara(archivo:String;localizacion,direccion,arriba,mira: Vector);
 Var Arch : Text;
 Begin
  Assign(Arch, archivo);
  Append(Arch);
  Writeln(Arch,'camera {');
  Writeln(Arch,'  location  <'+Real2Cad(Localizacion.X)+','
        +Real2Cad(Localizacion.Y)+','+Real2Cad(Localizacion.Z)+'>');
  Writeln(Arch,'  direction  <'+Real2Cad(Direccion.X)+','
        +Real2Cad(Direccion.Y)+','+Real2Cad(Direccion.Z)+'>');
  Writeln(Arch,'  up  <'+Real2Cad(arriba.X)+','
        +Real2Cad(arriba.Y)+','+Real2Cad(arriba.Z)+'>');
  Writeln(Arch,'  look_at  <'+Real2Cad(mira.X)+','
        +Real2Cad(mira.Y)+','+Real2Cad(Mira.Z)+'>');
  Writeln(Arch,'}');
  Close(Arch);
 end;
```

```
Procedure Graba_Luz(Archivo:String; Posicion,Color : Vector);
Var
  Arch : Text;
Begin
  Assign(Arch, archivo);
  Append(Arch);
  Writeln(Arch,'light_source { <'+Real2Cad(Posicion.X)+', '+
       Real2Cad(Posicion.Y)+', '+Real2Cad(Posicion.Z)+'> color red'+
       Real2Cad(Color.X)+' green '+Real2Cad(Color.Y)+' blue '+
       Real2Cad(Color.Z)+'} ');
  Close(Arch);
End;

Procedure Inicia_arch(Archivo: String);
Var
  Arch : Text;
Begin
  Assign(Arch, archivo);
  Rewrite(Arch);
  Writeln(Arch,'/* Programa Creado Por Generador de Animaciones de Tiro Parabolico 1.0 */');
  Writeln(Arch);
  Close(Arch);
End;


Procedure Graba_Esfera(Archivo : String ; posicion: Vector; Radio : Real;A;Atributos);
Var
  Arch : Text;
Begin
  Assign(Arch, archivo);
  Append(Arch);
  Writeln(Arch,'sphere {<'+Real2Cad(posicion.x)+', '+
               Real2Cad(Posicion.y)+', '+Real2Cad(Posicion.z)+
               '>, '+Real2Cad(Radio)).
  Writeln(Arch,'      '+a[1]);
```

```
  Writeln(Arch,'      '+a[2]);
  Writeln(Arch,'      '+a[3]);
  Writeln(Arch,'      '+a[4]);
  Writeln(Arch,'      '+a[5]);
  Writeln(Arch,'      '+a[6]);
  Writeln(Arch,'}');
  Close(Arch);
End;

Procedure Graba_Plano (Archivo : String ; plano: Vector; R : Real;A:Atributos);
Var
  Arch : Text;
Begin
  Assign(Arch, archivo);
  Append(Arch);
  Writeln(Arch,'plane {<'+Real2Cad(plano.x)+', '+
                Real2Cad(plano.y)+', '+Real2Cad(plano.z)+
                '>, '+Real2Cad(R));
  Writeln(Arch,'      '+a[1]);
  Writeln(Arch,'      '+a[2]);
  Writeln(Arch,'      '+a[3]);
  Writeln(Arch,'      '+a[4]);
  Writeln(Arch,'      '+a[5]);
  Writeln(Arch,'      '+a[6]);
  Writeln(Arch,'}');
  Close(Arch);
End;



Procedure Graba_Caja (Archivo:String; inicio,fin:Vector; A: Atributos);
Var
  Arch : Text;
Begin
  Assign(Arch, archivo);
  Rewrite(Arch);
```

```
  WriteIn(Arch,'box {<'+Real2Cad(Inicio.x)+', '+
                  Real2Cad(Inicio.y)+', '+Real2Cad(Inicio.z)+
                  '>, <'+ Real2Cad(fin.x)+', '+
                  Real2Cad(Fin.y)+', '+Real2Cad(Fin.z)+
                  '>, ');
  WriteIn(Arch,'      '+a[1]);
  WriteIn(Arch,'      '+a[2]);
  WriteIn(Arch,'      '+a[3]);
  WriteIn(Arch,'      '+a[4]);
  WriteIn(Arch,'      '+a[5]);
  WriteIn(Arch,'      '+a[6]);
  WriteIn(Arch,'}');
  Close(Arch);
End;

(* Procedure Genera_Bat(NombreBat : String; NombreAni: String; NumC : Word);
 Var
  CadAni,CadAux : String;
  Arch1  : Text;
  Cont : Word;
 Begin
  Assign(Arch1,NombreBat);
  Rewrite(Arch1);
  For Cont:=1 to NumC do
  Begin
   Str(Cont,CadAux);
   if CadAux[0]=#1 then CadAux:='000'+CadAux;
   if CadAux[0]=#2 then CadAux:='00'+CadAux;
   if CadAux[0]=#3 then CadAux:='0'+CadAux;
   CadAni:=NombreAni+CadAux;
   WriteIn(Arch1,'povray +i'+CadAni+' +o'+CadAni+' +FT +W'+
        Int2Cad(ResX)+' Int2Cad(ResY));
  End;
  Close(Arch1);
End;*)
```

End.

7.9 LA UNIDAD RATON

Esta unidad permite el manejo de un ratón para la interface gráfica, básicamente consiste en el empleo de la interrupción 33 para sensar el puerto serie, es relativamente sencillo crear una unidad de este tipo, pero tiene la desventaja de no ser portable a otros sistemas, por lo que no se entra en demasiado detalle aquí, en Internet es posible encontrar una gran cantidad de librerías para la escritura de interfaces gráficas con el uso de ratón, esta en particular se basa en una que encontre en http:\\oak.oakland.edu

```
unit RATON;

 interface
  uses dos;
  type array32word = array[0..31] of word;
  const
   mano : array32word = ($ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,
                         $ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,$ffff,
                         $0000,$0000,$0700,$0500,$0500,$05FC,$0554,$0D54
                         $1554,$1004,$0804,$0404,$0208,$0208,$0208,$0208),
  var  mouse_clicked_x,
       mouse_clicked_y  : word;

  function  mouse_area   (var xma)       : word;
  function  mouse_clicked(button:word)   : boolean;
  function  mousex                       : word;
  function  mousey                       : word;
  function  mousebutton                  : word;
  function  mouse_swreset                : boolean;
  procedure get_sensitivity      ( var x          word,
                                   var y          word,
                                    var speed     word      );
  procedure set_sensitivity      (    x          : word;
                                  y               : word;
                                  speed           : word      );

  procedure physical_movement      ( var x         : integer;
                                     var y         : integer    );
  procedure set_ratio              (    x,y        : word       );
  procedure cond_off               (    ux,uy,lx,ly : word      );
  procedure set_speed2             (    threshold  : word       );
  procedure mouse_cursor           (    onoff      : boolean    );
  procedure mouse_position         ( var button_stat : word,
                                     var x         : word;
                                     var y         : word       );
  procedure set_mouse_position     (    x,y        : word       );
  procedure set_mouse_x_bounds     (    xl,xu      : word       );
  procedure set_mouse_y_bounds     (    yl,yu      : word       );
  procedure set_mouse_cursor       (    x,y        : word;
                                     var point      : array32word );

  procedure set_text_cursor        (    aorh, startof,  endof    : word       );

  function  mouseparams            ( var num_buttons : word      ):boolean;
```

```
  function  button_pressed      (   button     : word;
                                    var count  : word;
                                    var x      : word;
                                    var y      : word     ):boolean;

  function  button_released     (   button     : word;
                                    var count  : word;
                                    var x      : word;
                                    var y      : word     ):boolean;

implementation

function mouse_area;
var x:integer;
    mx,my,button : word;
    ma : array[1..100,1..4] of word absolute xma;
begin
 x:=1;
 mouse_position(button,mx,my);
 while ma[x,1]<>0 do begin
   if    (mx>=ma[x,1])
     and (mx<=ma[x,3])
     and (my>=ma[x,2])
     and (my<=ma[x,4]) then begin
    mouse_area:=x;
    exit
    end;
  x:=x+1
  end;
 mouse_area:=0
 end;

 function mouse_swreset;
 var regs:registers;
 begin
  regs.ax:=$21;
  intr($33,regs);
  mouse_swreset:=regs.ax=$ffff
  end;

 procedure get_sensitivity;
 var regs:registers;
 begin
  regs.ax:=$1b;
  intr($33,regs);
  x:=regs.bx;
  y:=regs.cx;
  speed:=regs.dx
  end;

procedure set_sensitivity;
var regs:registers;
begin
  regs.ax:=$1a;
```

```
  regs.bx:=x;
  regs.cx:=y;
  regs.dx:=speed;
  intr($33,regs)
  end;

procedure physical_movement;
var regs:registers;
begin
  regs.ax:=$b;
  intr($33,regs);
  x:=regs.cx;
  y:=regs.dx
  end;

procedure set_ratio;
var regs:registers;
begin
  regs.ax:=$0f;
  x:=x and $7fff;
  y:=y and $7fff;
  regs.cx:=x;
  regs.dx:=y;
  intr($33,regs)
  end;

procedure cond_off;
var regs:registers;
begin
  regs.ax:=$10;
  regs.cx:=ux;
  regs.dx:=uy;
  regs.si:=lx;
  regs.di:=ly;
  intr($33,regs)
  end;

procedure set_speed2;
var regs: registers;
begin
  regs.ax:=$13;
  regs.dx:=threshold;
  intr($33,regs)
  end;

function mouseparams;
var regs: registers;
begin
  regs.ax:=$0;
  intr($33,regs);
  mouseparams:=regs.ax=$ffff;
  num_buttons:=regs.bx
  end;

procedure mouse_cursor;
var regs: registers;
```

```
begin
  if onoff then regs.ax:=$1 else regs.ax:=$2;
  intr($33,regs)
  end;

procedure mouse_position;
var regs: registers;
begin
  regs.ax:=$3;
  intr($33,regs);
  button_stat:=regs.bx;
  x:=regs.cx;
  y:=regs.dx
  end;

function mousex;
var b,x,y : word;
begin
  mouse_position(b,x,y);
  mousex:=x
  end;

function mousey;
var b,x,y : word;
begin
  mouse_position(b,x,y);
  mousey:=y
  end;

function mousebutton;
var b,x,y : word;
begin
  mouse_position(b,x,y);
  mousebutton:=b
  end;

procedure set_mouse_position;
var regs: registers;
begin
  regs.ax:=$4;
  regs.cx:=x;
  regs.dx:=y;
  intr($33,regs)
  end;

procedure set_mouse_x_bounds;
var regs: registers;
begin
  regs.ax:=$7;
  regs.cx:=xl;
  regs.dx:=xu;
  intr($33,regs)
  end;

procedure set_mouse_y_bounds;
var regs: registers;
```

```
begin
  regs.ax:=$8;
  regs.cx:=yl;
  regs.dx:=yu;
  intr($33,regs)
  end;

function button_pressed;
var regs:registers;
begin
  regs.ax:=$5;
  regs.bx:=button;
  intr($33,regs);
  count:=regs.bx;
  x:=regs.cx;
  y:=regs.dx;
  button_pressed:=(regs.ax=1)
  end;

function mouse_clicked;
var trash1 : boolean;
    count : word;
begin
  trash1:=button_pressed(button,count,mouse_clicked_x,mouse_clicked_y);
  mouse_clicked:=count>0
  end;



function button_released;
var regs:registers;
begin
  regs.ax:=$5;
  regs.bx:=button;
  intr($33,regs);
  count:=regs.bx;
  x:=regs.cx;
  y:=regs.dx;
  button_released.=(regs.ax=0)
  end;

procedure set_mouse_cursor;
var regs : registers;
begin
  regs.ax:=$9;
  regs.bx:=x;
  regs.cx:=y;
  regs.es:=seg(point[0]);
  regs.dx:=ofs(point[0]);
  intr($33,regs)
  end;

procedure set_text_cursor;
var regs:registers;
begin
```

```
  regs.ax:=$0a;
  regs.bx:=aorh;
  if aorh=0 then begin
    startof:=startof or $00ff;
    endof:=endof and $ff00
    end;
  regs.cx:=startof;
  regs.dx:=endof;
  intr($33,regs)
  end;

end.
```

7.10 UNIDAD U3D

Esta unidad define algunos procedimientos para el manejo de vectores y el tipo de dato "vector".

```
{  Esta Unidad Define Varias Funciones y Procedimientos
   Para el manejo de Objetos en 3 dimensiones          }

{   Función          Objetivo
    Esfera           genera una esfera de marco de alambre
                                                        }
Unit U3D;


Interface

uses graph;

Const
  pi=3.14159;
  FactorA  = -0.707106781;

Type
  Vector  = Record
    x : Real;
    y : Real;
    z : Real;
  end;

  punto_plano  = record
    x,y : real;
  end;


  procedure p3d_2d(puntoO :vector; var punto1 : punto_plano);
  procedure centra (var punto1 : punto_plano);
  procedure punto3D(punto : vector);
  procedure linea3d(PuntoO,PuntoD : Vector);
  procedure genera_octagono(r,cota :real; origen : vector);
```

```
procedure esfera(origen:vector; radio:real);
procedure cubo(origen:vector;largo:real);

Implementation

Var
  xmaxima, ymaxima : integer;
  octagono, octagono2 : array[1..16] of vector;
  cotas    : array[1..18] of real;
  radios   : array[1..18] of real;

procedure p3d_2d(puntoO :vector; var punto1 : punto_plano);
begin
  punto1.x: = puntoO.y + FactorA * PuntoO.x;
  punto1.y: = puntoO.z + FactorA * PuntoO.x;
  punto1.y: = -punto1.y;
end;

procedure centra (var punto1 : punto_plano);
var
  xmedia,ymedia : integer;
begin
  xmaxima: = getmaxx;
  ymaxima: = getmaxy;
  xmedia: = xmaxima div 2;
  ymedia: = ymaxima div 2;
  punto1.x: = punto1.x + xmedia;
  punto1.y: = punto1.y + ymedia;
end;

procedure punto3D(punto : vector);
var
  punto2 : punto_plano;
begin
  p3d_2d(punto,punto2);
  centra(punto2);
  putpixel(Round(punto2.x),Round(punto2.y),15);
end;

procedure linea3d(PuntoO,PuntoD : Vector);
var
  punto1,punto2 : punto_plano;
begin
  p3d_2d(PuntoO,punto1);   centra(punto1);
  p3d_2d(PuntoD,punto2);   centra(punto2);
  line(Round(punto1.x),Round(punto1.y),Round(punto2.x),Round(punto2.y));
end;

procedure genera_octagono(r,cota :real; origen : vector);
var
  contg : integer;
  v11,v12 : vector;
begin
  for contg: = 1 to 16 do
```

```
  begin
     octagono[contg].x: = r*cos(((contg)/8)*pi);
     octagono[contg].y: = r*sin(((contg)/8)*pi);
     octagono[contg].z: = cota + origen.z;
     octagono[contg].x: = octagono[contg].x + origen.x;
     octagono[contg].y: = octagono[contg].y + origen.y;
  end;
  for contg: = 1 to 15 do
  begin
    v11.x: = octagono[contg].x;   v11.y: = octagono[contg].y;
    v11.z: = octagono[contg].z;   v12.x: = octagono[contg + 1].x;
    v12.y: = octagono[contg + 1].y;   v12.z: = octagono[contg + 1].z;
    linea3d(v11,v12);
  end;
    v11.x: = octagono[1].x;   v11.y: = octagono[1].y;
    v11.z: = octagono[1].z;   v12.x: = octagono[16].x;
    v12.y: = octagono[16].y;   v12.z: = octagono[16].z;
    linea3d(v11,v12);
end;

procedure esfera(origen:vector; radio:real);
var
  contE1, ContE2 : integer;
  inicio : Boolean;
  vo1,vo2  : vector;
begin
  Inicio : =  True;
  for ContE1: = 1 to 8 do
    radios[ContE1]: = radio*cos(contE1/20*pi);
  for ContE1: = 1 to 17 do
  begin
    cotas[ContE1]: = ((2*radio)/16)*(contE1)-radio;
  end;
  for contE1: = 1 to 8 do
    begin
      genera_octagono(radios[9-contE1], Cotas[contE1],origen);
      if not(inicio) then
        for contE2: = 1 to 8 do
        begin
          vo1.x: = octagono[contE2].x;
          vo1.y: = octagono[contE2].y;
          vo1.z: = octagono[contE2].z;
          vo2.x: = octagono2[contE2].x;
          vo2.y: = octagono2[contE2].y;
          vo2.z: = octagono2[contE2].z;
          linea3d(vo1,vo2);
        end;
      inicio: = false;
      octagono2: = octagono;
    end;

   for contE1: = 8 downto 1 do
    begin
      genera_octagono(radios[9-contE1], Cotas[16-contE1],origen);
```

```
      for contE2: = 1 to 16 do
      begin
        vo1.x: = octagono[contE2].x;
        vo1.y: = octagono[contE2].y;
        vo1.z: = octagono[contE2].z;
        vo2.x: = octagono2[contE2].x;
        vo2.y: = octagono2[contE2].y;
        vo2.z: = octagono2[contE2].z;
        linea3d(vo1,vo2);

      end;
      octagono2: = octagono;
    end;
end;

procedure cubo(origen:vector;largo:real);
var
  va,vb,vc,vd,ve,vf,vg,vh : vector;
begin
  va.x: = -largo + origen.x;  va.y: = -largo + origen.y;   va.z: = -largo + origen.z;
  vb.x: = -largo + origen.x;  vb.y: = -largo + origen.y;   vb.z: = largo + origen.z;
  vc.x: = -largo + origen.x;  vc.y: = largo + origen.y;    vc.z: = -largo + origen.z;
  vd.x: = -largo + origen.x;  vd.y: = largo + origen.y;    vd.z: = largo + origen.z;
  ve.x: = largo + origen.x;   ve.y: = -largo + origen.y;   ve.z: = -largo + origen.z;
  vf.x: = largo + origen.x;   vf.y: = -largo + origen.y;   vf.z: = largo + origen.z;
  vg.x: = largo + origen.x;   vg.y: = largo + origen.y;    vg.z: = -largo + origen.z;
  vh.x: = largo + origen.x;   vh.y: = largo + origen.y;    vh.z: = largo + origen.z;
  linea3d(va,vb);   linea3d(va,ve);
  linea3d(va,vc);   linea3d(vb,vf);
  linea3d(vb,vd);   linea3d(ve,vg);
  linea3d(vg,vc);   linea3d(vc,vd);
  linea3d(vh,vf);   linea3d(vh,vd);
  linea3d(vh,vg);   linea3d(ve,vf);
end;
end.
```

CAPITULO 8

LINEAS DE INVESTIGACION.

CAPITULO 8 LINEAS DE INVESTIGACION

El trabajo que se realizo para la creación de esta tesis se basó principalmente en conceptos de animación y en los siguientes recursos:

* Programa libre para la generación de imágenes de calidad fotográfica.

* Un programa compartido para la conversión de secuencias de imágenes a formatos de animación en este caso se empleo el formato FLC.

* El programa de dominio público AAPLAY para Windows para la visualización de imágenes en formato FLI y FLC.

* Un programa propio elaborado en pascal que emplea a los demás programas para la captura de datos, generación de archivos de escenas, generación de las representaciones, la conversión de los grupos de imágenes fijas a animación y la visualización de la misma.

Todo lo anterior se realizó en una PC compatible con IBM, se le puede dar continuidad a este proyecto de varias maneras:

A) Realización del proyecto en otra plataforma de trabajo (preferentemente UNIX o Windows NT)

Existen varios motivos para pensar en migrar este proyecto y de hecho cualquier programa que valga la pena a otra plataforma, uno de los más importantes es el hecho de que el sistema operativo MS-DOS tendera a desaparecer ante la llegada de ambientes más poderosos y amigables, de entre los que destacan por un lado UNIX en estaciones de trabajo y PC's con Linux y por otro Windows de 32 Bits (Windows 95 y próximamente 97) y Windows NT.

El programa generador de representaciones (Imágenes de calidad fotográfica) puede ejecutarse prácticamente en cualquier plataforma, dado que se tienen los fuentes de dicho programa, en la página oficial del programa se encuentran binarios para varias plataformas entre las que destacan:

- Sun
- Silicon Graphics
- HP9000
- Alpha
- RS6000
- Windows NT

Si se desea junto con la distribución estándar del programa se encuentran los programas fuentes y están en "C" ANSI con lo cual se puede compilar en cualquier maquina que tenga un compilador de "C".

- La conversión de grupos de imágenes a formatos de animación puede ser realizada también en UNIX empleando el programa mpeg_encode el cual es del dominio público y del cual se disponen las fuentes en "C" estándar, y es del dominio público. Entre sus principales ventajas se cuenta la generación de animaciones en formato MPEG el cual resulta mucho más eficiente que cualquiera de los formatos actuales en casi cualquier circunstancia. Una copia de este programa puede ser obtenida en la siguiente dirección de Internet :

http: //www.shareware.com

- La visualización de las animaciones se realiza muy eficientemente por medio del programa xanim, del cual existen versiones para cualquier plataforma UNIX con XWindows. Si la visualización se realizara con Windows NT es posible emplear uno de los multiples reproductores de MPEG que acompañan las tarjetas de video de las nuevas computadoras o con el programa compartido VMpeg.

El programa Xanim puede ser obtenido en http: //xanim.va.pubnix.com

y el programa VMpeg se puede bajar de http://www.shareware.com

° El programa GATP10 que actualmente se encuentra en Pascal puede ser fácilmente transportado a "C" ANSI, la única parte que requeriría modificaciones substanciales sería la interface de usuario, la cual por lo demás no es tan indispensable que sea en modo gráfico, se puede realizar en modo texto sin detrimento de la funcionalidad del programa.

B) Respecto al sistema en si podrían cambiarse o ampliarse la librería que corresponde a POV que es la librería que contiene las funciones que permiten generar los archivos de escenas de Pov-ray, de manera que permita más funciones tales como:

° Manejo de BLOB's los cuales resultan ideales para la simulación de formas orgánicas y son muy difíciles de representar por medio de otra forma.

Manejo triángulos: los triángulos pese a su apariencia sencilla permiten una cantidad enorme de posibilidades de trabajo, todos los trabajos que se realizan empleando representación de superficies por medio de aproximaciones a polígonos pueden ser convertidos a imágenes de superficies aproximadas por medio de triángulos, de hecho hay programas como el 3ds2pov que convierte archivos de 3D studio a archivos de Pov-Ray, o el programa vrlm2pov que convierte varios formatos de archivos a formato de pov.

La versión 3.0 de Pov-Ray la cual se liberó mientras esta tesis se escribía presenta características muy interesantes, entre las que destaca el manejo de atmósferas, lo cual abre muchas posibilidades en la generación de imágenes de síntesis.

La función de tiro parabólico puede ser colocada en una librería y pueden crearse otras de modo que se puedan emplear desde un programa principal, de este modo este proyecto se puede ampliar a otros fenómenos físicos uno que puede ser implementado de forma relativamente simple es el de la gravitación universal se podría por ejemplo dar texturas de fotografías a los planeta y verlos de forma muy realista, introducir un elemento nuevo al sistema como un cometa por ejemplo y observar su comportamiento, se necesitarían funciones nuevas para el manejo de la cámara, ya que este programa se manejaría mejor con una cámara en movimiento que con una cámara fija.

C) Desde el punto de vista teórico es mucho lo que se puede mejorar de este trabajo, los puntos que merecen especial interés son:

° Fluídos

Los fluidos son muy difíciles de representar en la actualidad ya que requieren de muchos cálculos y mas reglas que el comportamiento de cuerpos sólidos, los elementos de Pov-Ray que mejor se prestan para generar imágenes que simulen fluidos son los BLOB's o metabolas, otro inconveniente de la representacion de fluidos es que la mayoría de las veces no son completamente opacos si no que tienen cierto grado de transparencia de forma que son necesarios los cálculos de refracción los cuales consumen bastante tiempo de CPU.

° Sistemas de partículas

Estos sistemas presentan problemas muy específicos ya que básicamente son como cualquier objeto que maneje un programa de síntesis de imagen pero es mucho muy pequeño y se encuentra en grupos de varios miles o incluso millones de partículas y su comportamiento es un híbrido entre el comportamiento de los sólidos y el comportamiento de los fluidos por lo cual se emplean algoritmos que permitan generar la imagen de forma realista sin consumir recursos excesivos.

° Verificación de frontera

La verificación de frontera es en esencia la comprobación de que dos cuerpos entran en contacto, en este proyecto se realiza un forma de verificación de frontera de objetos muy simples, si la esfera que es lanzada llega a el nivel del plano del suelo deja de ser visualizada, esto se logra de forma simplista ya que por medio de la formula del tiro parabólico es posible determinar cuando la altura de la esfera es cero con respecto al plano, una verificación de frontera entre varios cuerpos requiere de varias funciones adicionales, por ejemplo:

° Choque de dos esferas

Se verifica que las fronteras de las esferas se han encontrado si la suma de sus radios es igual o menor que la distancia entre sus centros.

° choque de una esfera a un plano

Se puede saber si una esfera a impactado un plano si la distancia del plano a el centro de la esfera es igual o menor a el radio de la esfera.

La verificación de la intersección de fronteras de otros objetos requiere de un poco más de teoría.

* Deformación de Objetos

En este trabajo se emplean objetos sólidos que mantienen su forma todo el tiempo pero si en la vida real un objeto como una esfera choca con otro objeto sufre una deformación proporcional a la fuerza del impacto y variará de acuerdo con la naturaleza del material con el que ha sido creada.

CONCLUSIONES

CONCLUSIONES

El presente trabajo demuestra varias cosas:

Es posible desarrollar en un periodo de tiempo relativamente corto un programa que permita generar animaciones de simulaciones gracias a la recopilación de información y fuentes en Internet, ya que la mayoría de los programas empleados fueron obtenidos de manera gratuita de este lugar.

La correcta selección de programas que sean portables, estables, bien probados y bien documentados en Internet permite la fácil integración por medio de un programa propio, el cual se recomienda sea en un lenguaje portable tal como el Pascal o el lenguaje "C".

Este trabajo fue desarrollado por completo en una PC con Windows 95 y con esto queda demostrado que en las computadoras personales es posible realizar trabajos de animación.

Con las bases adecuadas de física es posible implementar los algoritmos necesarios para generar animaciones que correspondan fielmente a la realidad, y no solo sean bonitas escenas en movimiento.

El realismo no depende de la maquina en que realice la imagen sino de los algoritmos empleados, lo que depende de la computadora que se emplee es el tiempo de generación, así pues es posible crear dos imágenes de la misma calidad en una PC o una estación de trabajo Cray si se emplean los mismos algoritmos, pero en la PC puede demorar varias horas el proceso mientras que en la estación de trabajo puede durar fracciones de segundo.

Si podemos realizar la simulación numérica de un fenómeno, entonces podemos realizar una animación de dicho fenómeno.

Los programas de libre distribución que se pueden obtener de Internet son lo suficientemente flexibles para ser usados como parte de sistemas mayores, de hecho al contar con las fuentes existe la posibilidad de incluso modificarlos o ampliarlos para que satisfagan los requerimientos particulares.

Las ventajas de usar programas libres como base para realización de proyectos mayores presenta varias ventajas :

1.- El tiempo de desarrollo se reduce considerablemente.
2.- El sistema es muy flexible y puede modificarse o ampliarse tanto como se quiera.
3.- Otras personas alrededor del mundo pueden continuar con el trabajo que se realice, de esta forma se contribuye a aumentar el acervo de líneas de código que existen en la red.
4.- Los sistemas libres al ser probados por una amplia comunidad de programadores son muy estables.

Este trabajo intenta ser un pequeño esbozo de Ciencia Numérica, la cual es nueva y supone ventajas sobre los dos tipos tradicionales de hacer ciencia :
La ciencia teórica : presenta el inconveniente de ser hipotética.
La ciencia práctica : presenta varios inconvenientes, tales como limitaciones tanto temporales como espaciales, por ejemplo un investigador difícilmente puede monitorear un proceso que dura una millonésima de segundo, o sólo supone comportamientos de fenómenos que duran millones de años, o de la misma forma están limitados los investigadores cuando estudian hechos cuyos rangos de acción son el mundo de lo núcleos atómicos.
La Ciencia Numérica se basa en la ciencia teorica para elaborar modelos que después se ajustan por medio de experimentación, cuando los modelos son muy fieles a la realidad es posible basarse en ellos como herramienta principal.

El empleo de Ciencia Numérica permite adaptar las escalas tanto temporales como espaciales de forma que sean de utilidad al investigador o al alumno que esta aprendiendo alguna disciplina que implique modelos abstractos.

ANEXO I

IMAGENES A COLOR.

ANEXO 1 IMAGENES A COLOR

Figura 5.2.1

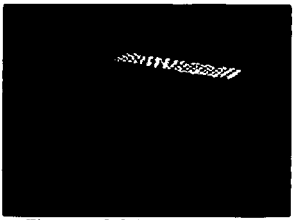


Figura 5.5.2

Figura 5.5.3

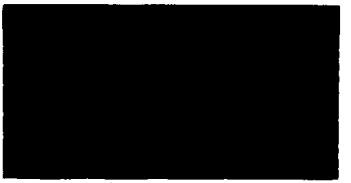


Figura 5.5.4

Figura 5.5.5

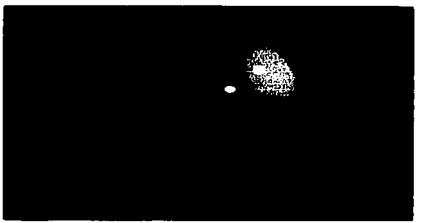


Figura 5.6.1

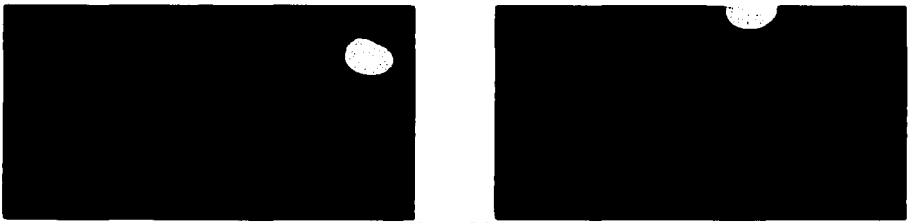


Figura 5.8.1

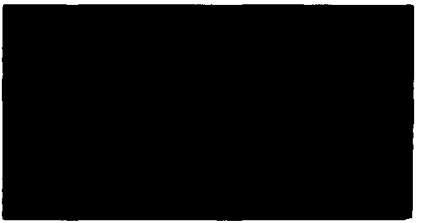


Figura 5.9.1

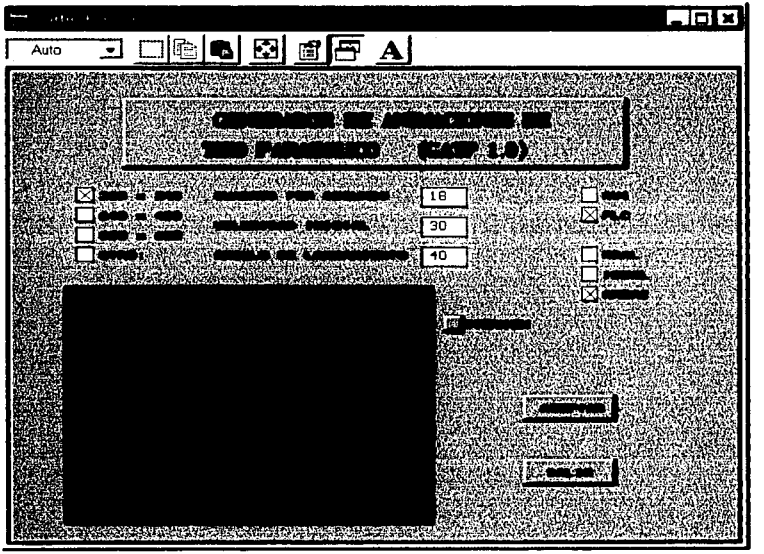
Figura 5.9.2

Figura 5.9.3

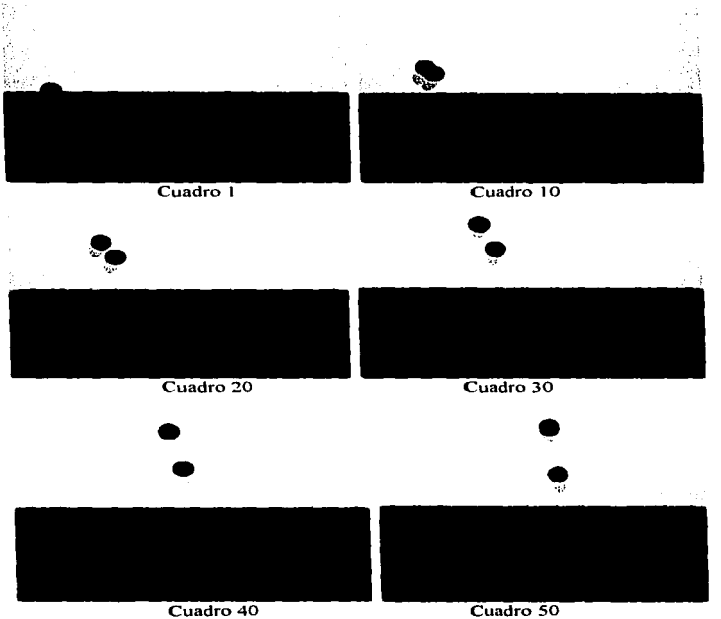
ANEXO II

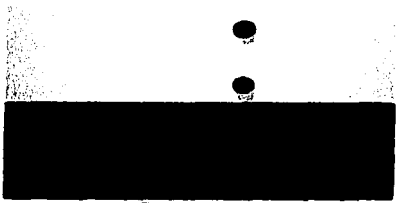
EJEMPLO DEL PROGRAMA.

ANEXO 2 EJEMPLO DEL PROGRAMA



Esta es la pantalla de dialogo para que el usuario escriba sus valores de entrada, los valores que se muestran serán empleados para los cálculos necesarios, en las páginas siguientes se muestran algunos cuadros de la animación resultante.


Cuadro 1
Cuadro 10
Cuadro 20
Cuadro 30
Cuadro 40
Cuadro 50



Cuadro 60

Cuadro 70

Cuadro 80

BIBLIOGRAFIA

BIBLIOGRAFIA

DONALD HEARN, M. PAULINE BAKER

GRAFICAS POR COMPUTADORA
2a ED. MEXICO PRENTICE HALL
1994. 686 Pág.

LEITHOLD, LOUIS

EL CALCULO CON GEOMETRIA ANALITICA
5ta ED. MEXICO ED. HARLA
1014 Pág.

PLASTOCK, ROY A.

TEORIA Y PROBLEMAS DE GRAFICAS POR COMPUTADORA.
MEXICO ED. McGRAW-HILL

GERARDO MARIN

"DE LAS LINTERNAS MAGICAS A LA ANIMACION POR COMPUTADORA".
PERSONAL COMPUTING MEXICO
(MEXICO), NOVIEMBRE 1994 Pág. 32

ARNULFO ZEPEDA

"TECNICAS PARA LA ANIMACION EN PC".
PERSONAL COMPUTING MEXICO
(MEXICO). NOVIEMBRE 1994 Pág. 39

ROBERTO ZARCO

"SOLO AMIGA LO HACE POSIBLE".
PERSONAL COMPUTING MEXICO
(MEXICO). NOVIEMBRE 1994 Pág. 45

INFOGRAFIA DE ALTO NIVEL
REVISTA CD-WARE
ABRIL DE 1995
IMPRESO EN ESPAÑA

SIMON AND SCHUSTER'S INTERNATIONAL DICTIONARY
English/Spanish
Spanish/English
Ed. Prentice Hall
Nueva York

DIRECCIONES DE INTERNET QUE SE EMPLEARON PARA OBTENER INFORMACION.

PARQUE JURASICO :

http://www.geocities.com/Hollywood/Hills/3162/jurassic.html

PELICULA TRON :

http://www.3gcs.com/tron/images/images.htm

PELICULA TOY STORY

http://www.disney.com/DisneyVideos/ToyStory/

SIMULACION DEL SISTEMA SOLAR :

http://medb.physics.utoronto.ca/web/website/JAVA/trajplot/trajplot.html