UNIVERSIDAD DE EL SALVADOR

FACULTAD MULTIDISCIPLINARIA ORIENTAL

DEPARTAMENTO DE INGENIERIA Y ARQUITECTURA

INGENIERIA EN SISTEMAS INFORMÁTICOS

ASIGNATURA:

ALGORITMOS GRÁFICOS

DESARROLLO DE UNA SIMULACION DE UN SISTEMA SOLAR EN OPENGL

3D SOLAR SYSTEM SIMULATOR

CATEDRATICO

ING LUDWIN HERNANDEZ

AUTORES

CARLOS EDENILSON ROMERO HERNANDEZ

ANA RUTH SANCHEZ HENRIQUEZ

EL SALVADOR SAN MIGUEL 20 DE JUNIO DE 2017

UNIVERSIDAD DE EL SALVADOR

FMO

DESARROLLO DE UNA SIMULACION DE UN SISTEMA SOLAR EN OPENGL

SOLAR SYSTEM SIMULATOR

TRABAJO FINAL DE CATEDRA

POR

ANA RUTH SANCHEZ HENRIQUEZ

CARLOS EDENILSON ROMERO HERNANDEZ

INDICE

INTRODUCCION

El presente proyecto se basa en la construcción de un Sistema solar utilizando técnicas de programación basadas en OpenGL que es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D.

Para su respectiva construcción utilizaremos herramientas de programación como Geany y librerías basadas en OpenGL como: #include <GL/gl.h>,#include <GL/glut.h>,#include <GL/glut.h>,#include <stdlib.h>,#include <time.h>,#include <stdio.h>,#include <stdarg.h> #include <math.h>,#include <GL/freeglut.h>,#include <cstdio>,#include <iostream>,#include <SOIL/SOIL.h>

Asi como También se aplicaran diferentes texturas para que los planetas se vean de forma real, así de esta forma se le presentará al usuario una información de cada uno de los planetas en el universe.los diferentes tipos de animaciones estaran controladas por el teclado para que se pueda simular mejor cada uno de los planetas.

De igual forma se podra observar respectivamente la distancia de cada uno de ellos, aunque los planetas no van a escala, cada uno tendra su informacion donde se podrá acceder por medio de un menu.

Su funcionamiento estara explicado paso por paso, asi como tambien la definicion de varias funciones en este, esperando que sea de agrado y que su aplicacion sea de mucha ayuda y orientacion a los niños y todas aquellas personas que esten interesadas en conocer hacerca de nuestro Sistema solar Atravez de esta simulacion llamada SOLAR SIMULATOR SYSTEM

Acontinuacion el desarrollo del proyecto

1. El PROBLEMA

1.1-TITULO DESCRIPTIVO DEL PROYECTO

Desarrollo de una simulación de un Sistema Solar
En OpenGL.

1.2-SITUACION PROBLEMÁTICA

El problema que existe sobre la falta de información acerca del conocimiento del espacio cada vez se vuelve más influyente en los niños, ya que ellos desean conocer acerca de lo que existe en nuestro sistema solar.

Se debe tener en cuenta que durante el desarrollo de una actividad lúdica en los niños/as, intervienen todos los potenciales físicos, cognitivos, efectivos y sociales; por lo anterior, es de suma importancia su desarrollo, porque así como ponen en práctica la psicomotricidad, la disociación y la coordinación haciendo uso del esquema corporal, el cual se hace más fácil conocerlo por medio de simulaciones o videojuegos tecnológicos.

De esta manera en este proyecto se trata de poder orientar de una manera más sencilla el aprendizaje de los niños, Atravez de un pequeño simulador el cual servirá como un método educativo de enseñanza e información llamado SOLAR SYSTEM SIMULATOR.

Se decidió hacer este proyecto ya que los niños tiene más interés en la tecnología y esto nos facilita tener un medio por el cual llegar a ellos por medio de la computación grafica basada en OPENGL.

1.3 PLANTEAMIENTO DEL PROBLEMA

Muchas personas están interesadas en el Sistema solar, en especial los niños ya que su curiosidad es muy grande. Se plantea crear una pequeña simulación donde los niños puedan interactuar en ella visitando los planetas y obteniendo información de cada uno de ellos ya sea su nombre, masa, gravedad, referencias históricas, etc.

Se plantea usar OpenGL, Glut.h, SOIL (y otras más).

Para el desarrollo de este programa, el cual estará basado en un Sistema solar con figuras representando los planetas, se utilizaran animaciones, para dar su respectiva animación en sus orbitas así como también la textura indicada para representar más apropiadamente la forma del planeta, también se incluirá una fuente de luz en el sol, la cual iluminara los planetas al rededor.

Se implementara un sistema de vista la cual permitirá seleccionar un planeta y obtener su información. Se implementara una vista libre por lo cual se podrá acercar, alejar, moverse libremente por el sistema solar.

1.4-ENUNCIADO DEL PROBLEMA

¿Por qué se considera que el desarrollo de esta simulación del sistema solar será de mucha importancia?

¿Realmente el uso de este proyecto le dio solución al problema de falta de conocimiento por medio de la tecnología Atravez de un desarrollo de una simulación del sistema solar o por el contrario se sigue buscando más información práctica?

¿Se cuenta con todos los recursos, suficientes para la realización de este proyecto?

Las preguntas anteriormente planteadas serán de investigación para profundizar de una manera teórica y práctica para proponer alternativas de solución.

1.5-JUSTIFICACION

Este proyecto tiene Como objetivo principal desarrollar una simulación de un sistema solar en OPENGL

Esto puede ser útil para las personas que estudian el universo y que tienen interés por saber de cada uno de los planetas.

La importancia de este proyecto radica entonces en poder mostrar de una forma tecnológica a los niños y a todas aquellas personas que quieran conocer más haya de nuestro planeta.

Atravez de esta simulación dar a conocer cada uno de los planetas y su información para que de esta manera puedan informarse más acerca del sistema solar.

1.6-DELIMITACIONES

Delimitación temporal:

La investigación se realizó durante el periodo de mayo a junio del año 2017.

Delimitación espacial:

La investigación se ha llevado a cabo en la Universidad de El Salvador, Facultad Multidisciplinaria Oriental; orientada a los niños, así como también a todos aquellos interesados en saber más acerca de nuestro sistema solar.

Delimitación social:

El estudio fue delimitado y se trabajó con el nivel de educación básica , de niños menores a diez años, donde se realizaron diversas actividades que culminaron con la preparación e implementación de una simulación del sistema solar.

1.6.1-LUGAR O ESPACIO

La preparación de la simulación del sistema solar será de libre acceso y gratuito, por otra parte el lugar donde será implementado será en un servidor web el cual llevará el nombre de SOLAR SYSTEM SIMULATOR y el espacio donde se podrá ejecutara será Atravez de cualquier aparato electrónico que tenga acceso a internet.

1.6.2-TIEMPO

ACTIVIDADES	TIEMPO
DISEÑO DE ESCENAS Y PROGRAMACION	Del 15 de mayo hasta el 14 de junio de 2017
Diseño de figuras	
Textura y animaciones	
PRUEBA	Del 16 de Junio hasta el 21 de junio de 2017
Validaciones y correcciones	
DOCUMENTACION Y CORRECCION	Del 23 de junio hasta el 15 de junio de 2017
Elaboración del documento	
Exposición de la simulación	

1.6.3 TEORIAS

Nuestro proyecto está basado en un sistema solar ya existente, al comienzo planteamos lo que queríamos hacer y llegamos a la conclusión que lo más adecuado sería un sistema solar orientado para los niños.

En nuestra recopilación de información nos encontramos con un simulador similar a lo que nos planteamos creado por INOVE el cual lleva por nombre "Solar System Scope" el cual muestra los distintos planetas del sistema solar, pero no es su única función también simula constelaciones y cometas. El simulador es gratuito pero al adquirirlo se obtienen nuevas herramientas para poder utilizarse, se puede acceder a él por medio de la siguiente dirección web "http://www.solarsystemscope.com/ "

El simulador está desarrollado en HTML y por el motor gráfico Unity3D por lo cual decidimos implementar algo similar con las herramientas que OpenGL nos proporciona para así poder entender y comprender lo que estas herramientas son capas de ser desde cero.

Para mayor información de los creadores del simulador del cual nos basamos se puede acceder a el por medio de la siguiente dirección web "http://www.inove.eu.com/"

1.7-OBJETIVOS DEL PROYECTO

1.7.1. OBJETIVO GENERAL

Desarrollar una simulación en OpenGL que permita al usuario conocer información acerca del Sistema Solar.

1.7.2. OBJETIVOS ESPECÍFICOS.

1. Utilizar cada una de las librerías para el desarrollo de la simulación del sistema solar
2. Distinguir cada uno de los planetas, con sus respectivas texturas.
3. Dar a conocer cada una de las respectivas vistas controladas por el teclado y mouse.
4. Mostrar la información de cada planeta al momento de seleccionarlos

2. FUNDAMENTACION TEÓRICA

OpenGL es una librería de funciones que nos permiten visualizar gráficos 3D en nuestra aplicación. La gran ventaja de esta librería es que nos aísla del hardware disponible; por tanto, si nos ponemos una tarjeta aceleradora, no hará falta que cambiemos nuestro programa para aprovechar la potencia de la tarjeta.

Librerías añadidas

En este proyecto, no sólo vamos a trabajar con la librería OpenGL. Trabajaremos también con la GLU y GLUT.

#include <GL/gl.h> #include <GL/glut.h> y otras mas.

La librería GLU contiene funciones gráficas de más alto nivel, que permiten realizar operaciones más complejas (una cosa así como el ensamblador y el C) En cambio, la librería GLUT es un paquete auxiliar para construir aplicaciones de ventanas, además de incluir algunas primitivas geométricas auxiliares. Además de simplificar mucho el código fuente del programa, como lo veremos en este proyecto.

Formato de las funciones

Los nombres de las funciones en estas librerías siguen la siguiente convención:

{gl, glu, glut} <un nombre> [{d, f, u, ... etc}] [v]

El prefijo gl indica que se trata de una función de la librería de OpenGL, el prefijo glu de una función de la librería GLU, y el prefijo glut es para las funciones de la GL

Los sufijos pueden aparecer o no, depende. Si los hay, es que la función puede estar en varias versiones, dependiendo del tipo de datos de los parámetros de ésta. Así, acabará con un sufijo d si recibe parámetros double, y con un sufijo fv si recibe parámetros de tipo *float (es decir, punteros a float). En estas prácticas, por defecto referenciaremos las funciones con sufijo f y fv, aunque hayan más versiones.

Ejemplos de funciones: gluPerspective, glColor3f, glutSwapBuffers, glMaterialf, glMaterialfv

Funcion" Es un conjunto de líneas de código que realizan una tarea específica y puede retornar un valor. "Librería" una biblioteca (del inglés library) es un conjunto de implementaciones funcionales, codificadas en un lenguaje de programación

Funciones básicas para definir objetos

Primitivas geométricas básicas Podemos dibujar puntos, líneas y polígonos. Para definir un punto en el espacio 3D, usaremos la función: glVertex3f(x, y, z). Estos puntos se unen formando estructuras, como líneas y polígonos. La siguiente tabla muestra alguna de las opciones que tenemos:

Modo	Descripción
GL_POINTS	Puntos individuales aislados
GL_LINES	Cada par de puntos corresponde a una recta
GL_LINE_STRIP	Segmentos de recta conectados
GL_LINE_LOOP	Segmentos de recta conectados y cerrados
GL_TRIANGLES	Cada tres puntos un triángulo
GL_QUADS	Cada cuatro puntos un cuadrado
GL_POLYGON	Un polígono

Aunque en nuestro proyecto solo utilizaremos GL_POINTS y GL_LINES.

Primitivas de objetos predefinidos

Hay algunos objetos que vamos a renderizar muy a menudo, y que por tanto, ya vienen definidos pero en nuestro proyecto solamente nos enfocamos a las esferas. Así, disponemos de las siguientes funciones:

- glutWireSphere(radius, slices, stacks), glutSolidSphere(radius, slices, stacks)
- glutWireCube(size), glutSolidCube(size)
- glutWireCone(base, height, slices, stacks), glutSolidCone(base, height, slices, stacks)
- glutWireDodecahedron(void), glutSolidDodecahedron(void)
- glutWireOctahedron(void), glutSolidOctahedron(void)
- glutWireTetrahedron(void), glutSolidTetrahedron(void)
- glutWireIcosahedron(void), glutSolidIcosahedron(void)
- glutWireTeapot(void), glutSolidTeapot(void)

"Primitivas" Representación de un figura u objeto real.
"Renderizar" El proceso de renderizado se desarrolla con el fin de generar en un espacio 3D formado por estructuras poligonales.

Color.

Por defecto, el trazado de las líneas y puntos es blanco, pero podemos cambiarlo. Para hacer esto, usaremos la función glColor3f(r, g, b). El valor de r, g y b debe estar entre 0 y 1.

Definiendo una escena

Hasta ahora hemos construido nuestra escena mediante la especificación directa de las coordenadas 3D, o bien utilizando las primitivas de objetos. Pero, ¿cómo podríamos dibujar, por ejemplo, dos teteras?. Cuando utilizamos glVertex3f, o llamamos a la función glWireTeapot, estamos definiendo un objeto en coordenadas del objeto.

Estas coordenadas son multiplicadas por una matriz, denominada matriz ModelView. Esta matriz es de 4x4 y contiene las transformaciones necesarias de translación, rotación, escalado, etc. que definen la localización de nuestro objeto en el mundo. Por tanto, si llamamos a la función glWireTeapot con dos matrices ModelView diferentes, aparecerán en nuestra escena en dos sitios distintos. OpenGL puede trabajar con varias matrices. Para poder modificar la matriz ModelView, tenemos que decirle a OpenGL que queremos trabajar con ella, porque si no el resultado puede ser impredecible, ya que no sabremos qué matriz estamos modificando. Esto lo haremos haciendo una llamada a glMatrixMode(GL_MODELVIEW).

GlMatrixMode - especifica qué matriz es la matriz actual
Establece el modo de matriz actual. Modo puede asumir uno de cuatro valores

Rotación

Se define mediante la primitiva glRotatef(alpha, x, y, z). Esta función multiplica la matriz actual por una matriz de rotación de alpha grados respecto al eje (x, y, z).

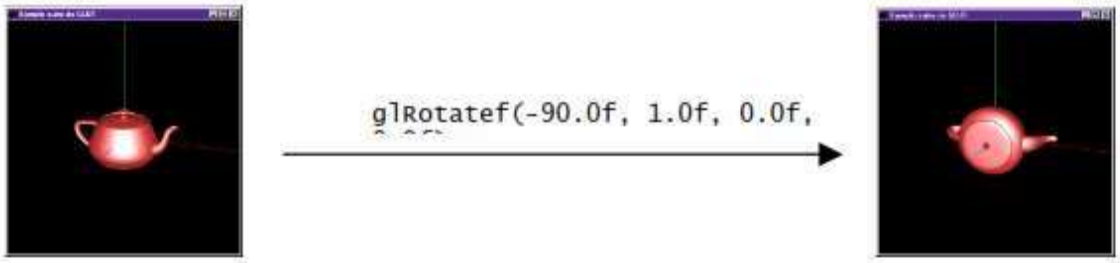


Figura 1.0 "Representación de un objeto aplicándole glRotatedf();"

Translación

Se define mediante la primitiva glTranslatef(x, y, z) Aplica una translación en (x, y, z) sobre la matriz actual.

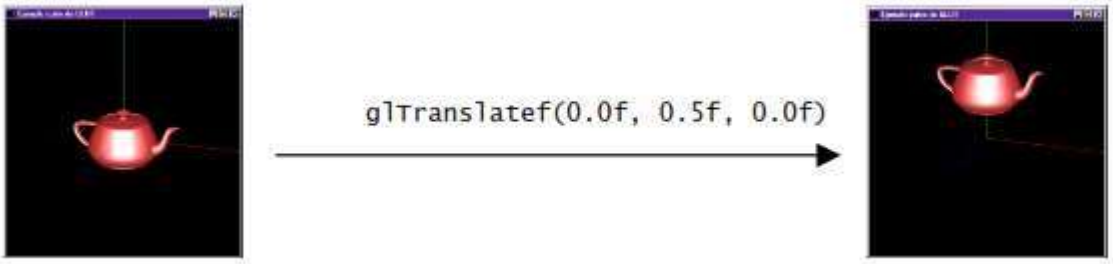


Figura 1.1 "Representación de un objeto aplicándole glTranslatef();"

Escalado

Se define mediante la primitiva glScalef(sx, sy, sz) Escalado de cada uno de los ejes.

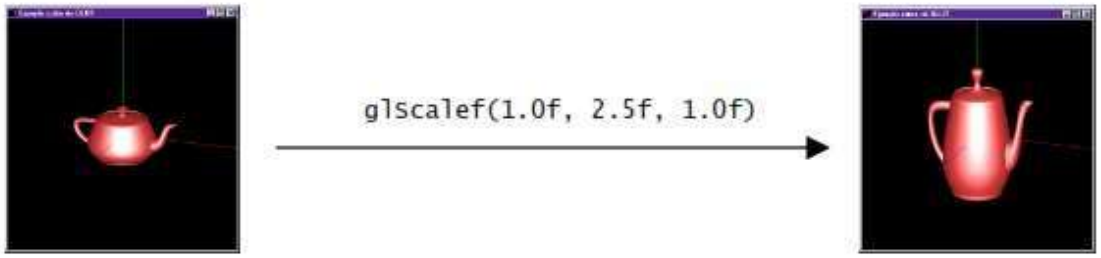


Figura 1.2 "Representación de un objeto aplicándole glScalef();"

Definición y colocación de la cámara

Una vez hemos definido toda nuestra escena en coordenadas mundo, tenemos que "hacerle la foto". Para ello, tenemos que hacer dos cosas: colocar la cámara en el mundo (o sea, en la escena) y definir el tipo de proyección que realizará la cámara.

Posición de la cámara

Tenemos que definir no sólo la posición de la cámara (o donde está), sino también hacia dónde mira y con qué orientación (no es lo mismo mirar con la cara torcida que recta… aunque veamos lo mismo).

Para hacer esto, basta con modificar la matriz ModelView para mover toda la escena de manera que parezca que hemos movido la cámara. El problema de este sistema es que tenemos que pensar bastante las transformaciones a aplicar. Es por ello que la librería GLU viene al rescate con la función gluLookAt.

Su sintaxis es la siguiente:

void gluLookAt(eyeX, eyeY, eyeZ, cenX, cenY, cenZ, vp_X, vp_Y, vp_Z);

Donde eye corresponde a la posición de la cámara, cen corresponde al punto hacia donde mira la cámara y vp es un vector que define la orientación de la cámara. No podemos llamar a gluLookAt en cualquier momento, puesto que tiene postmultiplicar la matriz ModelView (por tanto, conviene llamarla lo primero de todo). El vector vp no puede ser paralelo al vector formado por eye y cen, es más, debería serle perpendicular. Si no, el resultado es impredecible.

MARCO TEORICO

INFORMACION HACERCA DEL SISTEMA SOLAR

El sistema solar es el sistema planetario en el que se encuentran la Tierra y otros objetos astronómicos que giran directa o indirectamente en una órbita alrededor de una única estrella conocida como el Sol.

La estrella concentra el 99,75 % de la masa del sistema solar, y la mayor parte de la masa restante se concentra en ocho planetas cuyas órbitas son prácticamente circulares y transitan dentro de un disco casi llano llamado plano eclíptico. Los cuatro planetas más cercanos, considerablemente más pequeños Mercurio, Venus, Tierra y Marte, también conocidos como los planetas terrestres, están compuestos principalmente por roca y metal. Mientras que los cuatro más alejados, denominados gigantes gaseosos o "planetas jovianos", más masivos que los terrestres, están compuesto de hielo y gases. Los dos más grandes, Júpiter y Saturno, están compuestos principalmente de helio e hidrógeno. Urano y Neptuno, denominados los gigantes helados, están formados mayoritariamente por agua congelada, amoniaco y metano.[8]

CONCEPCIÓN ARTÍSTICA DE UN DISCO PROTOPLANETARIO.

El Sol es el único cuerpo celeste del sistema solar que emite luz propia, la cual es producida por la combustión de hidrógeno y su transformación en helio por la fusión nuclear. El sistema solar se formó hace unos 4600 millones de años a partir del colapso de una nube molecular.

El material residual originó un disco circunestelarprotoplanetario en el que ocurrieron los procesos físicos que llevaron a la formación de los planetas. El sistema solar se ubica en la actualidad en la nube Interestelar Local que se halla en la Burbuja Local del brazo de Orión, de la galaxia espiral Vía Láctea, a unos 28 000 años luz del centro de esta.

Los ocho planetas que componen el sistema solar son, de menor a mayor distancia respecto al Sol, los siguientes:

- Mercurio
- Venus
- Tierra
- Marte
- Júpiter
- Saturno
- Urano
- Neptuno

Los planetas son cuerpos que giran formando órbitas alrededor de la estrella, tienen suficiente masa para que su gravedad supere las fuerzas del cuerpo rígido, de manera que asuman una forma en equilibrio hidrostático (prácticamente esférica), y han limpiado la vecindad de su órbita de planetesimales (dominancia orbital).

Los planetas interiores son Mercurio, Venus, la Tierra y Marte y tienen la superficie sólida. Los planetas exteriores son Júpiter, Saturno, Urano y Neptuno, también se denominan planetas gaseosos porque contienen en sus atmósferas gases como el helio, el hidrógeno y el metano, y no se conoce con certeza la estructura de su superficie.

El 24 de agosto de 2006, la Unión Astronómica Internacional (UAI) excluyó a Plutón como planeta del sistema solar, y lo clasificó como planeta enano.

A principios del año 2016 se publicó un estudio según el cual puede existir un noveno planeta en el sistema Solar, al que dieron el nombre provisional de Phattie. Dicho estudio se centró en la explicación de las órbitas de muchos de los objetos en el cinturón de Kuiper, que difieren mucho con las órbitas que se calculan, incluidos objetos muy conocidos Sedna. Por tanto se surgió originalmente la idea de la existencia de un objeto no conocido perturbando dichas órbitas. Utilizando modelos matemáticos se realizaron simulaciones en computadora, y se determinó que el posible planeta tendría una órbita excéntrica a una distancia de unas entre 700 y 200 UA del Sol, y tardaría unos diez o veinte mil años en dar una vuelta.[17] [51] [52]

CARACTERÍSTICAS PRINCIPALES

Planeta	Símb.	Diámetro ecuatorial*	Diámetro ecuatorial (km).	Masa*	Radio orbital (UA).	Periodo orbital (años).	Periodo de rotación (días).	Incl.**	Sat.***	Composición de la atmósfera	Imagen
Júpiter	♃	11,2	142984	318	5,20	11,86	0,414	1,3º	63	90 % hidrógeno, 10 % helio, trazas de metano	
Marte	♂	0,53	6787	0,11	1,52	1,88	1,03	1,9º	2	95 % CO_2, 1.6 % argón, 3 % nitrógeno	
Mercurio	☿	0,39	4878	0,06	0,39	0,24	58,6	7º	0	Trazas de hidrógeno y helio	
Neptuno	♆	3,81	49538	17,2	30,06	164,79	0,6745	1,8º	13	74 % hidrógeno, 25 % helio, 1 % metano	
Saturno	♄	9,41	120536	95	9,54	29,46	0,426	2,5º	61	96 % hidrógeno, 3 % helio, 0.5 % metano	
Tierra	⊕	1,00	12756	1,00	1,00	1,00	1,00	0º	1	78 % nitrógeno, 21 % oxígeno, 1 % argón	
Urano	♅	3,98	51108	14,6	19,19	84,01	0,718	0,8º	27	84 % hidrógeno, 14 % helio, 2 % metano	
Venus	♀	0,95	12100	0,82	0,72	0,615	243	3,4º	0	96 % CO_2, 3 % nitrógeno,0.1 % agua	

*El diámetro y masa se expresan en relación a la Tierra ** Inclinación de órbita (en relación con la eclíptica) *** Satélites naturales*

Todo lo anterior explicado del Sistema solar lo aplicamos en nuestro proyecto llamado

SOLAR SYSTEM SIMULATOR de la siguiente manera

FUNCIONABILIDAD DEL PROYECTO

Aplicamos las siguientes librerias

```
#include <GL/gl.h>
#include <GL/glut.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include <stdarg.h>
#include <math.h>
#include <GL/freeglut.h>
#include <cstdio>
#include <iostream>
#include <SOIL/SOIL.h>
```

Luego de esto procedemos al llamando funciones globales

```
void idle(void);
void sistemaSolar(void);
void solInfo(void);
void mercurioInfo(void);
void venusInfo(void);
void tierraInfo(void);
void marteInfo(void);
void jupiterInfo(void);
void saturnoInfo(void);
void uranoInfo(void);
void neptunoInfo(void);
void pauseAnimation(void);
void startAnimation(void);
void mouseCallBack(int btn, int state, int x, int y);
```

/Inicializacion para la variable de rotacion de la luna al rededor de la Tierra.
GLfloat rotLT=0;

Variable para rotacion en su propio Eje.
GLfloat rotateEjeY = 0;

asi como tambien la Inicializacion para los radios de los planetas.

```
GLfloat rdS=20,
      rd1=6,
      rd2=12,
      rd3=9,
      rd4=6,
      rd5=18,
      rd6=15,
      rd7=12,
      rd8=12;
```

Declaramos variables para determinar la posicion inicial de los planetas.

```
GLfloat pos1=40,
      pos2=60,
      pos3=80,
      pos4=100,
      pos5=120,
      pos6=140,
      pos7=160,
      pos8=180,
      posl=95;
```

Variables para determinar la posicion en el eje X y Z

```
GLfloat posx=0,
      posz=0;
```

Variable para la cadena de texto.

```
char text[32];
```

y tambien la Inicializacion para las variables de traslado y escalado.

```
GLfloat rotate_y=0.0f;
GLfloat rotate_x=0.0f;
GLfloat scale = 1.5f;
```

GLRotatef

GlRotatef: multiplica la matriz actual por una matriz de rotación.

Parámetro

Especifica el ángulo de rotación, en grados X y y las coordenadas x, y z de un vector,

respectivamente.

Descripción

GlRotatef produce una rotación de ángulos en torno el vector x,y,z

La matriz actual (véase glMatrixMode) se multiplica por una rotación matriz con el producto

Reemplazando la matriz actual, como si glMultMatrix se llamará .Con la siguiente matriz

como argumento.

Variables para la orbita.
GLfloat calX, calZ;

Implementamos el venctor donde almacenaremos el id de la imagen a imprimir
int vf[5]={2};
int opc;

Variables para la textura.
GLuint texSun;
texMercurio,
texVenus,
texTierra,
texMarte,
texJupiter,
texSaturno,
texUrano,
texNeptuno,
texLuna;

Se hace una Conversion de cuadrados a esferas.

GLUquadricObj *pSphere = NULL;

Para hacer el mapeo de la imagen

Variable para animacion.
int animating = 0;

Variable para aumentar y disminuir distancia.
int count=0;

Variable para la orbita.
int countorb=0;

Definimos el ancho y largo de ventana.

int w_height=700;
int w_width=1000;

int x, y;

CREACION DE LOS PLANETAS.

En estas funciones creamos los planetas, incluyendo su rotacion, traslacion y su nombre en pantalla.

Sol.

```
void sol(){

   glPushMatrix();
   sprintf(text, "Sol");
   glColor3f(1, 1, 1);
   glRasterPos3f( (rdS/rdS) , 22 , 0);
   for(int i = 0; text[i] != '\0'; i++)
      glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, text[i]);

   glEnable(GL_TEXTURE_2D);
   glBindTexture(GL_TEXTURE_2D,texSun);
   glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
   glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
   gluSphere(pSphere,rdS,60,60);
   glDisable(GL_TEXTURE_2D);
   glPopMatrix();
}
```

```
//Mercurio.
void mercurio(){

   Rotacion y traslacion para Mercurio.

   glPushMatrix();
   glRotatef(rot1,0,1,0);
   glTranslatef(pos1,0,0);

   Nombre del planeta

   sprintf(text, "Mercurio");
   glColor3f(1, 1, 1);
   glRasterPos3f( -3 , 22 , 0);
   for(int i = 0; text[i] != '\0'; i++)
      glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, text[i]);

   Rotacion en su eje.
   glRotatef(rotateEjeY,0,1,0);

   Dibujando Planeta.
   glEnable(GL_LIGHTING);
   glEnable(GL_TEXTURE_2D);
   glBindTexture(GL_TEXTURE_2D,texMercurio);
   glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
   glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
   gluSphere(pSphere,rd1,60,60);
   glDisable(GL_TEXTURE_2D);
   glDisable(GL_LIGHTING);
   glPopMatrix();
}

//Venus.
void venus(){

   //Rotacion y traslacion para Venus.
   glPushMatrix();
   glRotatef(rot2,0,1,0);
   glTranslatef(-pos2,0,0);

   sprintf(text, "Venus");
   glColor3f(1, 1, 1);
   glRasterPos3f( -3 , 22 , 0);
   for(int i = 0; text[i] != '\0'; i++)
```

```
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, text[i]);

   //Rotacion en su eje.
   glRotatef(rotateEjeY,0,1,0);

   //Dibujando Planeta.
   glEnable(GL_LIGHTING);
   glEnable(GL_TEXTURE_2D);
   glBindTexture(GL_TEXTURE_2D,texVenus);
   glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
   glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
   gluSphere(pSphere,rd2,60,60);

   glDisable(GL_TEXTURE_2D);
   glDisable(GL_LIGHTING);
   glPopMatrix();
}

//Tierra.
void tierra(){

   //Rotacion y tralacion para Tierra.
   glPushMatrix();
   glRotatef(rot3,0,1,0);
   glTranslatef(0,0,-pos3);

   sprintf(text, "Tierra");
   glColor3f(1, 1, 1);
   glRasterPos3f( -3 , 22 , 0);
   for(int i = 0; text[i] != '\0'; i++)
     glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, text[i]);

   //Rotacion en su eje.
   glRotatef(rotateEjeY,0,1,0);
   //Dibujando Planeta.
   glEnable(GL_LIGHTING);
   glEnable(GL_TEXTURE_2D);
   glBindTexture(GL_TEXTURE_2D,texTierra);
   glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
   glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
   gluSphere(pSphere,rd3,60,60);
   glDisable(GL_TEXTURE_2D);
```

```
  glDisable(GL_LIGHTING);
    glPopMatrix();
}

//Luna
void luna(){

    //Rotacion y tralacion para Luna.
    glPushMatrix();
    glRotatef(rotL,0,1,0);
    glTranslatef(0,0,-posl);

    sprintf(text, "Luna");
    glColor3f(1, 1, 1);
    glRasterPos3f( -3 , 22 , 0);
    for(int i = 0; text[i] != '\0'; i++)
      glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, text[i]);

    //Rotacion en su eje.
    glRotatef(rotateEjeY,0,1,0);
    //Dibujando Planeta.
    glEnable(GL_LIGHTING);
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D,texLuna);
    glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
    gluSphere(pSphere,rdL,60,60);
    glDisable(GL_TEXTURE_2D);
    glDisable(GL_LIGHTING);
    glPopMatrix();
}

//Marte.
void marte(){

    //Rotacion y traslacion para Marte.
    glPushMatrix();
    glRotatef(rot4,0,1,0);
    glTranslatef(0,0,pos4);

    sprintf(text, "Marte");
    glColor3f(1, 1, 1);
    glRasterPos3f( -3 , 22 , 0);
    for(int i = 0; text[i] != '\0'; i++)
```

```
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, text[i]);

   //Rotacion en su eje.
   glRotatef(rotateEjeY,0,1,0);
   //Dibujando Planeta.
   glEnable(GL_LIGHTING);
   glEnable(GL_TEXTURE_2D);
   glBindTexture(GL_TEXTURE_2D,texMarte);
   glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
   glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
   gluSphere(pSphere,rd4,60,60);
   glDisable(GL_TEXTURE_2D);
   glDisable(GL_LIGHTING);
   glPopMatrix();
}

//Jupiter.
void jupiter(){

   //Rotacion y traslacion para Jupiter.
   glPushMatrix();
   glRotatef(rot5,0,1,0);
   glTranslatef(0,0,-pos5);

   sprintf(text, "Jupiter");
   glColor3f(1, 1, 1);
   glRasterPos3f( -3 , 22 , 0);
   for(int i = 0; text[i] != '\0'; i++)
     glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, text[i]);

   //Rotacion en su eje.
   glRotatef(rotateEjeY,0,1,0);
   //Dibujando Planeta.
   glEnable(GL_LIGHTING);
   glEnable(GL_TEXTURE_2D);
   glBindTexture(GL_TEXTURE_2D,texJupiter);
   glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
   glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
   gluSphere(pSphere,rd5,60,60);
   glDisable(GL_TEXTURE_2D);
   glDisable(GL_LIGHTING);
   glPopMatrix();
```

```
}

//Saturno y su Anillo.
void saturno(){

   //Rotacion y traslacion para Saturno.
   glPushMatrix();
   glRotatef(rot6,0,1,0);
   glTranslatef(0,0,pos6);

   sprintf(text, "Saturno");
   glColor3f(1, 1, 1);
   glRasterPos3f( -3 , 22 , 0);
   for(int i = 0; text[i] != '\0'; i++)
      glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, text[i]);

   //Rotacion en su eje.
   glRotatef(rotateEjeY,0,1,0);
   //Dibujando Planeta.
   glEnable(GL_LIGHTING);
   glEnable(GL_TEXTURE_2D);
   glBindTexture(GL_TEXTURE_2D,texSaturno);
   glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
   glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
   gluSphere(pSphere,rd6,60,60);
   glDisable(GL_TEXTURE_2D);
   glDisable(GL_LIGHTING);
   glPopMatrix();
}

void anillos(){

   //Rotacion y Traslacion para los Anillos.
   glPushMatrix();
   glRotatef(rot6,0,1,0);
   glTranslatef(0,0,pos6);
   glColor3f(0.75f,0.75f,0.75f);
   //Dibujando los anillos.
   glEnable(GL_LIGHTING);
   glRotatef(-65,1,0,1);
   glutSolidTorus(3, 20, 10,50);
   glDisable(GL_LIGHTING);
   glPopMatrix();
```

```
}

//Urano.
void urano(){

   //Rotacion y traslacion para Urano.
   glPushMatrix();
   glRotatef(rot7,0,1,0);
   glTranslatef(-pos7,0,0);

   sprintf(text, "Urano");
   glColor3f(1, 1, 1);
   glRasterPos3f( -3 , 22 , 0);
   for(int i = 0; text[i] != '\0'; i++)
     glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, text[i]);

   //Rotacion en su eje.
   glRotatef(rotateEjeY,0,1,0);
   //Dibujando Planeta.
   glEnable(GL_LIGHTING);
   glEnable(GL_TEXTURE_2D);
   glBindTexture(GL_TEXTURE_2D,texUrano);
   glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
   glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
   gluSphere(pSphere,rd7,60,60);
   glDisable(GL_TEXTURE_2D);
   glDisable(GL_LIGHTING);
   glPopMatrix();
}

//Neptuno.
void neptuno(){

   //Rotacion y traslacion para Neptuno.
   glPushMatrix();
   glRotatef(rot8,0,1,0);
   glTranslatef(pos8,0,0);

   sprintf(text, "Neptuno");
   glColor3f(1, 1, 1);
   glRasterPos3f( -3 , 22 , 0);
   for(int i = 0; text[i] != '\0'; i++)
      glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, text[i]);
```

```
  //Rotacion en su eje.
    glRotatef(rotateEjeY,0,1,0);
    //Dibujando Planeta.
    glEnable(GL_LIGHTING);
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D,texNeptuno);
    glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
    gluSphere(pSphere,rd8,60,60);
    glDisable(GL_TEXTURE_2D);
    glDisable(GL_LIGHTING);
    glPopMatrix();
}
```

FIN DE LA CREACION DE LOS PLANETAS.

GluSphere

GluSphere dibuja una esfera

Especificaciones
gluSphere (GLUquadric * quad, GLdouble radio, rodajas GLint, pilas GLint);

Las variables que toma son las siguientes:

Void gluSphere(GLUquadric* quad, GLdouble radius, GLint slices, GLint stacks);

Los pàrametros que toma

quad: Especifica el objeto quadrics (creado con gluNewQuadric).

Radius: especifica el radio de la esfera.

slices: Especifica el número de subdivisiones alrededor del eje z (Similar a las líneas de longitud).

Stacks: Especifica el número de subdivisiones alrededor del eje z (Similar a las líneas de latitud).

Si la orientación se establece en GLU_OUTSIDE (Con gluQuadricOrientation), entonces

cualquier normal generado Punto lejos del centro de la esfera.De lo contrario, apuntan hacia el centro de la esfera.

Si la texturización está activada (con gluQuadricTexture), entonces textura

Las coordenadas son Generadas de modo que t oscile de 0.0 en Z=-radio

Función donde cargamos la textura.

```
GLuint loadTex(const char* filename){

   GLuint tex_ID = SOIL_load_OGL_texture(filename, SOIL_LOAD_AUTO,
SOIL_CREATE_NEW_ID,(SOIL_FLAG_MIPMAPS | SOIL_FLAG_INVERT_Y |
SOIL_FLAG_NTSC_SAFE_RGB | SOIL_FLAG_COMPRESS_TO_DXT));
      glEnable( GL_TEXTURE_2D );
      glBindTexture( GL_TEXTURE_2D, tex_ID );
      glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
      return tex_ID;
}
```

GlShadeModel

GlShadeModel - seleccione sombreado plano o liso

C Specificationvoid

GlShadeModel (modo GLenum);

Parámetros

modo Especifica un valor simbólico que representa una técnica de sombreado. Los valores aceptados son GL_FLAT y GL_SMOOTH. El valor inicial es GL_SMOOTH

Descripción

Las primitivas GL pueden tener sombreado plano o liso.Sombreado suave, el valor por defecto, Hace que los colores calculados de vértices sean interpolados como el Primitivo es rasterizado. Típicamente asignando diferentes colores a cada fragmento de píxel resultante.

El sombreado plano selecciona el color calculado de un solo vértice. Y la asigna a todos los fragmentos de píxeles. Generado por la rasterización de una única primitiva.

En cualquier caso, el color calculado de un vértice es el resultado de la Iluminación si la iluminación está habilitada, O es el color actual en el momento en que se especificó el vértice si.La iluminación está desactivada.

El sombreado plano y liso es indistinguible para los puntos.Se inicia cuando se emite glBegin y se cuentan vértices . Primitivas de 1, el GL da a cada segmento de línea sombreada

Su segundo vértice cuenta de manera similar desde 1,El GL da a cada polígono con sombra plana el color calculado del vértice listado

GLClearColor

GlClearColor - especifica valores claros para los búferes de color

Especificación

GlClearColor (GLclampf rojo, GLclampf verde, GLclampf azul, GLclampf alfa); Parámetros

Rojo, verde, azul, alfa

parametros

los valores de rojo, verde, azul y alfa que se borran los búferes de color.Los valores iniciales son todos 0.

Descripción

GlClearColor especifica el color rojo, verde, azul.Y los valores alfa utilizados por glClear para borrar los búferes de color.Los valores especificados por glClearColor se sujetan al rango

GlClearDepth

GlClearDepth especifica el valor de claridad para el buffer de profundidad

C Specificationvoid glClearDepth (profundidad de GLclampd);

Parámetros

profundidad: Especifica el valor de profundidad utilizado cuando se borra el búfer de profundidad. El valor inicial es 1.

Descripción

GlClearDepth especifica el valor de profundidad utilizado por glClear para borrar el buffer de profundidadad.Los valores especificados por glClearDepth se ajustan al rango (0,1).

Lo siguiente muestra las texturas

Funcion init donde las esfera se trata como un objeto cuadrado.

```
void openGLInit(void){
   glShadeModel( GL_FLAT );
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glClearDepth( 1.0 );
   glEnable( GL_DEPTH_TEST);
   pSphere = gluNewQuadric();
   gluQuadricDrawStyle(pSphere, GLU_FILL);
   gluQuadricNormals(pSphere, GLU_SMOOTH);
   gluQuadricTexture(pSphere, GLU_TRUE);
}
```

GlEnable

GlEnable habilita o deshabilita las capacidades GL del lado del servidor

C Especificación evite glEnable (tapa GLenum); Void glDisable (tapa de GLenum)

Parámetros

Cap: Especifica una constante simbólica que indica una capacidad GL.

Descripción

GlEnable y glDisable

Habilitar y deshabilitar varias capacidades. Utilizar GlIsEnabled o GlSeleccione para la Configuración actual de cualquier capacidad. El valor inicial para cada capacidad con la excepción de

GL_DITHER

GL_MULTISAMPLE

GL_FALSE.

GL_DITHER

GL_MULTISAMPLE

GL_TRUE.

GlBindTexture

GlBindTexture enlaza una textura con nombre a un destino de texturing

Especificación

Void glBindTexture (objetivo GLenum,Textura de GLuint);

Parámetros

Objetivo**:** Especifica el destino al que está ligada la textura. Debe ser GL_TEXTURE_2D.

Textura: Especifica el nombre de una textura.

Descripción

GlBindTexture le permite crear o usar una textura con nombre. Llamar glBindTexture con el objetivo establecido en GL_TEXTURE_2D y la textura establecida en el nombre de la nueva textura vincula el nombre de la textura al destino. Cuando una textura está vinculada a un destino, la vinculación anterior para ese destino se rompe automáticamente.

Los nombres de texturas son enteros sin signo. El valor 0 se reserva para representar la textura por defecto para cada objetivo de textura. Los nombres de textura y el contenido de textura correspondiente son locales al espacio de objeto de textura compartido (véase eglCreateContext) del contexto de representación GL actual.

Puede usar glGenTextures para generar un conjunto de nuevos nombres de textura.

Mientras se une una textura, las operaciones GL en el destino al que está vinculado afectan a la textura enlazada.

Si la textura de mapeo de la dimensionalidad del objetivo al que se une una textura está activa, se utiliza la textura ligada.

De hecho, los objetivos de textura se convierten en alias para las texturas que actualmente están enlazadas a ellos, y el nombre de textura 0 se refiere a las texturas predeterminadas que se enlazaron a ellos en la inicialización.

Un enlace de textura creado con glBindTexture permanece activo hasta que una textura diferente se une al mismo destino, o hasta que la textura enlazada se elimine con glDeleteTextures

Una vez creada, una textura nombrada puede ser re-enlazada al objetivo de la dimensionalidad coincidente tantas veces como sea necesario.

Por lo general, es mucho más rápido usar glBindTexture para enlazar una textura con nombre existente a uno de los objetivos de textura que con recargar la imagen de textura mediante glTexImage2D.

GlTexParameter

GlTexParameter - establece parámetros de textura

C Especificaciones

Void glTexParameterf (GLenum target,GLenum pname, GLfloat param);

Void glTexParameteri (GLenum target,GLenum pname,GLint param);

Parámetros

Objetivo**:** Especifica la textura de destino de la unidad de textura activa, que debe ser GL_TEXTURE_2D o GL_TEXTURE_CUBE_MAP.

Especifica el nombre simbólico de un parámetro de textura de valor único. Pname puede ser uno de los siguientes: GL_TEXTURE_MIN_FILTER, GL_TEXTURE_MAG_FILTER, GL_TEXTURE_WRAP_S o GL_TEXTURE_WRAP_T.

Parámetros

Especifica el valor de pname.

C Especificaciones

Void glTexParameterfv (GLenum target,GLenum pname,Const GLfloat * params);

Void glTexParameteriv (GLenum target, GLenum pname, Const GLint * params);

Parámetros

Objetivo:Especifica la textura de destino de la unidad de textura activa, que debe ser GL_TEXTURE_2D o GL_TEXTURE_CUBE_MAP.

Especifica el nombre simbólico de un parámetro de textura. Pname puede ser uno de los siguientes: GL_TEXTURE_MIN_FILTER, GL_TEXTURE_MAG_FILTER, GL_TEXTURE_WRAP_S o GL_TEXTURE_WRAP_T.

Parametros

Especifica un puntero a una matriz donde se almacena el valor de pname.

Descripción

El mapeado de texturas es una técnica que aplica una imagen a la superficie de un objeto como si la imagen fuera una etiqueta o una envoltura de celofán.

La imagen se crea en el espacio de textura, con un sistema de coordenadas (s, t). Una textura es una imagen bidimensional o cubicada y un conjunto de parámetros que determinan cómo se derivan las muestras de la imagen.

GlTexParameter asigna el valor o los valores en parametros al parámetro texture especificado como pname. Target define la textura de destino de la unidad de textura activa, ya sea GL_TEXTURE_2D o GL_TEXTURE_CUBE_MAP. En pname se aceptan los siguientes símbolos:

GL_TEXTURE_MIN_FILTER

La función de minificación de textura se utiliza siempre que el píxel que se está texturizando se asigna a un área mayor que un elemento de textura. Hay seis funciones de minificación definidas. Dos de ellos usan uno o cuatro elementos de textura más próximos para calcular el valor de textura. Los otros cuatro usan mipmaps.

Un mipmap es un conjunto ordenado de matrices que representan la misma imagen en resoluciones progresivamente más bajas.

Para definir los niveles de mipmap, llame a glTexImage2D, glCompressedTexImage2D o glCopyTexImage2D con el argumento de nivel que indica el orden de los mipmaps.

proporciona una función para minificar la textura como una de las siguientes:

GL_NEAREST

Devuelve el valor del elemento de textura más cercano (en la distancia de Manhattan) al centro del píxel que se está texturizando.

GL_LINEAR

Devuelve el promedio ponderado de los cuatro elementos de textura que están más cerca del centro del píxel que se está texturizando.

GL_NEAREST_MIPMAP_NEAREST

Elige el mapa mip que más se aproxima al tamaño del píxel que se está texturizando y utiliza el criterio GL_NEAREST (el elemento de textura más cercano al centro del píxel) para producir un valor de textura.

GL_LINEAR_MIPMAP_NEAREST

Elige el mipmap que más se aproxima al tamaño del píxel que se está texturizando y utiliza el criterio GL_LINEAR (un promedio ponderado de los cuatro elementos de textura que están más cerca del centro del píxel) para producir un valor de textura.

GL_NEAREST_MIPMAP_LINEAR

Elige los dos mipmaps que más coinciden con el tamaño del píxel que se está texturizando y utiliza el criterio GL_NEAREST (el elemento de textura más cercano al centro del píxel) para producir un valor de textura de cada mipmap. El valor final de la textura es un promedio ponderado de esos dos valores.

GL_LINEAR_MIPMAP_LINEAR

Elige los dos mipmaps que más coinciden con el tamaño del píxel que se está texturizando y usa el criterio GL_LINEAR (un promedio ponderado de los cuatro elementos de textura que están más cerca del centro del píxel) para producir un valor de textura de cada mipmap. El valor final de la textura es un promedio ponderado de esos dos valores.

A medida que se muestrean más elementos de textura en el proceso de minificación, serán menos evidentes los artefactos de aliasing.

Mientras que las funciones de minificación GL_NEAREST y GL_LINEAR pueden ser más rápidas que las otras cuatro, sólo muestrean uno o cuatro elementos de textura para determinar el valor de textura del píxel que se está procesando y pueden producir patrones de moaré o transiciones irregulares. El valor inicial de GL_TEXTURE_MIN_FILTER es GL_NEAREST_MIPMAP_LINEAR.

GL_TEXTURE_MAG_FILTER

La función de ampliación de textura se utiliza cuando el píxel que se está texturizando se asigna a un área menor o igual a un elemento de textura. Establece la función de ampliación de textura a GL_NEAREST o GL_LINEAR (véase más adelante). GL_NEAREST es generalmente más rápido que GL_LINEAR, pero puede producir imágenes con textura con bordes más agudos porque la transición entre los elementos de textura no es tan suave. El valor inicial de GL_TEXTURE_MAG_FILTER es GL_LINEAR.

GL_NEAREST

Devuelve el valor del elemento de textura más cercano (en la distancia de Manhattan) al centro del píxel que se está texturizando.

GL_LINEAR

Devuelve el promedio ponderado de los cuatro elementos de textura que están más cerca del centro del píxel que se está texturizando.

Plano Carteciano.

Plano Carteciano. Nos sirve para ubicarnos mejor en la pantalla.

```
void ejes(){

        glColor3f(1,1,1);
   glBegin(GL_LINES);
      glVertex3f(-200,0,0);
      glVertex3f(200,0,0);
      glVertex3f(0,-200,0);
      glVertex3f(0,200,0);
      glVertex3f(0,0,-200);
      glVertex3f(0,0,200);
   glEnd();
}
```

GlVertex

GlVertex especifica un vértice

Especificaciones

Void glVertex2s (GLshort x, GLshort y);

Void glVertex2i (GLint x, GLint y);

Void glVertex2f (GLfloat x, GLfloat y);

Void glVertex2d (GLdouble x,GLdouble y);

Void glVertex3s (GLshort x,GLshort y, GLshort z);

Void glVertex3i (GLint x, GLint y, GLint z);

Void glVertex3f (GLfloat x,GLfloat y,GLfloat z);

Void glVertex3d (GLdouble x, GLdouble y, GLdouble z);

Void glVertex4s (GLshort x,GLshort y,GLshort z,GLshort w);

Void glVertex4i (GLint x, GLint y, GLint z,GLint w);

Void glVertex4f (GLfloat x, GLfloat y, GLfloat z, GLfloat w);

Void glVertex4d (GLdouble x, GLdouble y, GLdouble z, GLdouble w);

Descripción

Los comandos glVertex se utilizan en pares glBegin / glEnd para especificar vértices de puntos, líneas y polígonos. El color actual, normal, coordenadas de textura y coordenada de niebla están asociados con el vértice cuando se llama glVertex.Cuando sólo se especifican x e y, z es el valor predeterminado de 0 y w es el valor predeterminado 1. Cuando se especifican x, yyz, w predeterminado es 1.

Funcion donde creamos las estrellas.

```
void estrellas(){
//Gracias a la funcion rand podemos generar aleatoriamente las estrellas en un cubo, en donde se encuentra el sistema solar.
   int i,j,k;
   srand (time(NULL));
   glBegin(GL_POINTS);
   glPointSize(0.1);

   for(int s=-100; s<100; s++){

      i=(rand()%300);
      j=(rand()%300);
      k=(rand()%300);
```

```
  glColor3f(1,1,1);

      glVertex3f( i, j, k);

      glVertex3f( i, j,-k);

      glVertex3f( i,-j, k);

      glVertex3f( i,-j,-k);

      glVertex3f(-i, j, k);

      glVertex3f(-i, j,-k);

      glVertex3f(-i,-j, k);

      glVertex3f(-i,-j,-k);

   }

   glEnd();

}
```

Funcion para crear las orbitas donde los planetas giran al rededor del sol.

```
void orbita(){
//Orbita para Mercurio.
   glPushMatrix();
   glColor3f(1.0,1.0,1.0);
   glBegin(GL_POINTS);

   for (double i=0.0; i<10; i+=0.001){
      calX=(pos1+1)*cos(i)+0;
      calZ=(pos1+1)*sin(i)+0;
      glVertex3f(calX,0,calZ);
   }
   glEnd();
   glPopMatrix();

//Orbita para Venus.
   glPushMatrix();
   glColor3f(1.0,1.0,1.0);
   glBegin(GL_POINTS);
   for (double i=0.0; i<10; i+=0.001){
```

```
    calX=(pos2+1)*cos(i)+0;
    calZ=(pos2+1)*sin(i)+0;
    glVertex3f(calX,0,calZ);
  }
  glEnd();
  glPopMatrix();

//Orbita para Tierra.
  glPushMatrix();
  glColor3f(1.0,1.0,1.0);
  glBegin(GL_POINTS);
  for (double i=0.0; i<10; i+=0.001){
    calX=(pos3+1)*cos(i)+0;
    calZ=(pos3+1)*sin(i)+0;
    glVertex3f(calX,0,calZ);
  }
  glEnd();
  glPopMatrix();
```

y asi para cada uno de los planetas

GlPushMatrix

GlPushMatrix - empuja y despliega la pila de matriz actual

Especificaciones

Void glPushMatrix (void);

Descripción

Hay una pila de matrices para cada uno de los modos de matriz. En el modo GL_MODELVIEW, la profundidad de la pila es al menos 32. En los otros modos, GL_COLOR, GL_PROJECTION y GL_TEXTURE, la profundidad es como mínimo 2. La matriz actual en cualquier modo es la matriz en la parte superior de la pila para ese modo.

GlPushMatrix empuja la pila de la matriz actual por una, duplicando la matriz actual. Es decir, después de una llamada a glPushMatrix, la matriz en la parte superior de la pila es idéntica a la que está debajo de ella.

GlPopMatrix muestra la pila de matriz actual, reemplazando la matriz actual por la que está debajo de ella en la pila.

Inicialmente, cada una de las pilas contiene una matriz, una matriz de identidad.

Es un error empujar una pila de matriz completa o hacer estallar una pila de matriz que contiene sólo una matriz única. En cualquier caso, el indicador de error se establece y no se realiza ningún cambio en el estado GL

Funcion display. Sonde dibujamos lo principal del proyecto.

```
void display(){
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glLoadIdentity();
   GLfloat lightPosition[] = { 0.0, 0.0, 0.0, 1.0 };
        glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);
   startAnimation();
 sistemaSolar();
  glutSwapBuffers();
}
```

Creamos la funcion donde llamamos todos los planetas.

```
void sistemaSolar(){
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glLoadIdentity();
```

Rotacion de 45 grados en torno al Eje X.

```
glRotated(45.0, 1.0, 0.0, 0.0);
glRotatef( rotate_x, 1.0f, 0.0f, 0.0f );
glRotatef( rotate_y, 0.0f, 1.0f, 0.0f );
glScalef(scale, scale, scale);
```

ejes();

```
estrellas();
if(countorb==0){
orbita();
}
```

Dibujamos los planetas.

```
sol();
mercurio();
venus();
tierra();
luna();
marte();
jupiter();
saturno();
anillos();
urano();  neptuno();
```

Aqui calculamos las variables de rotacion al rededor del sol para cada planeta

```
   rotL=rot3*1;
   rot1=rot3*4.3;
   rot2=rot3*1.63;
   rot3=rot3+0.1;
   rot4=rot3/1.88;
   rot5=rot3/11.9;
   rot6=rot3/29;
   rot7=rot3/84;
   rot8=rot3/165;
   rotateEjeY=rotateEjeY+2;


   glutSwapBuffers();
}
```

Y asi de esta manera para cada uno de los planetas

GlLight

establece los parámetros de la fuente de luz

Especificaciones

Void glLightf (luz de GLenum, GLenum pname, GLfloat param);

Void glLightx (GLenum light, GLenum pname, Param fijado);

Descripción

GlLight establece los valores de los distintos parámetros de la fuente de luz. Luz nombra la luz y es un nombre simbólico de la forma GL_LIGHTi, donde 0 <= i <GL_MAX_LIGHTS. Pname

especifica uno de los diez parámetros de la fuente de luz, de nuevo por nombre simbólico. Params es un valor único o un puntero a una matriz que contiene los nuevos valores.

Para habilitar y deshabilitar el cálculo de iluminación, llame a glEnable y glDisable con argumento GL_LIGHTING. La iluminación está inicialmente desactivada. Cuando está activada, las fuentes de luz activadas contribuyen al cálculo de la iluminación. La fuente de luz i está habilitada y deshabilitada usando glEnable y glDisable con argumento GL_LIGHTi.

algunos parámetros de luz son los siguientes:

GL_AMBIENT

Params contiene cuatro valores de punto fijo o punto flotante que especifican la intensidad RGBA ambiente de la luz. Los valores de punto fijo y punto flotante se asignan directamente. No se fijan valores de punto fijo ni de punto flotante. La intensidad de luz ambiental inicial es (0, 0, 0, 1).

GL_DIFFUSE

Params contiene cuatro valores de punto fijo o punto flotante que especifican la intensidad RGBA difusa de la luz. Los valores de punto fijo y punto flotante se asignan directamente. No se fijan valores de punto fijo ni de punto flotante. El valor inicial para GL_LIGHT0 es (1, 1, 1, 1). Para otras luces, el valor inicial es (0, 0, 0, 0).

GL_SPECULAR

contiene cuatro valores de punto fijo o punto flotante que especifican la intensidad RGBA especular de la luz. Los valores de punto fijo y punto flotante se asignan directamente. No se fijan valores de punto fijo ni de punto flotante. El valor inicial para GL_LIGHT0 es (1, 1, 1, 1). Para otras luces, el valor inicial es (0, 0, 0, 0).

GL_POSITION

contiene cuatro valores de punto fijo o punto flotante que especifican la posición de la luz en coordenadas homogéneas del objeto. Los valores de punto fijo y punto flotante se asignan directamente. No se fijan valores de punto fijo ni de punto flotante.

Aqui creamos cada una de las funciones que almacenan la informacion del planeta.

```
void solInfo(){
pauseAnimation();
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glLoadIdentity();
   estrellas();
   glTranslatef(-160.0f, 0.0f, 0.0f);
   glPushMatrix();
   glEnable(GL_TEXTURE_2D);
   glBindTexture(GL_TEXTURE_2D,texSun);
   glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
   glRotatef(-90.0f, 1.0f, 0.0f, 0.0f);
   gluSphere(pSphere,100, 60, 60);
   glDisable(GL_TEXTURE_2D);
   glPopMatrix();
glColor3f(1.0f, 1.0f, 1.0f);
   glRasterPos3f( 250 , 250 , 0);
```

```
 sprintf(text, "El Sol.");

   for(int i = 0; text[i] != '\0'; i++)

      glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, text[i]);
```

Funcion de ayuda.

```
void ayuda(){

         pauseAnimation();
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glLoadIdentity();

   glColor3f(1.0f, 1.0f, 1.0f);
   glRasterPos3f( -20 , 250 , 0);
   sprintf(text, "AYUDA.");
   for(int i = 0; text[i] != '\0'; i++)
      glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, text[i]);

   glRasterPos3f( -280 , 230 , 0);
   sprintf(text, "Bienvenido a la pestana ayuda:");
   for(int i = 0; text[i] != '\0'; i++)
      glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, text[i]);
```

GlRasterPos

GlRasterPos especifica la posición ráster para las operaciones de píxeles

Especificaciones
Void glRasterPos2s (GLshort x, GLshort y);

Void glRasterPos2i (GLint x, GLint y);

Void glRasterPos2f (GLfloat x, GLfloat y);

Void glRasterPos2d (GLdouble x, GLdouble y); etc.

Descripción

El GL mantiene una posición 3D en las coordenadas de la ventana. Esta posición, llamada la posición de ráster, se utiliza para posicionar las operaciones de escritura de píxeles y de mapa de bits. Se mantiene con precisión de subpixel. Vea glBitmap, glDrawPixels y glCopyPixels.

La posición ráster actual consta de tres coordenadas de ventana (x, y, z), un valor de coordenada de clip (w), una distancia de coordenadas de ojo, un bit válido y datos de color asociados y coordenadas de textura. La coordenada w es una coordenada de clip, porque w no se proyecta a coordenadas de ventana. GlRasterPos4 especifica explícitamente las coordenadas del objeto x, y, z, yw. GlRasterPos3 especifica explícitamente la coordenada del objeto x, y, yz, mientras que w se establece implícitamente en 1. glRasterPos2 utiliza los valores de los argumentos para x e y mientras que implícitamente z y w son 0 y 1.

Las coordenadas de objeto presentadas por glRasterPos son tratadas como las de un comando glVertex: Son transformadas por la matriz actual y las matrices de proyección y pasadas a la etapa de recorte. Si el vértice no se cierra, entonces se proyecta y se ajusta a las coordenadas de la ventana, que se convierten en la nueva posición ráster actual y se establece el indicador GL_CURRENT_RASTER_POSITION_VALID. Si se elimina el vértice, entonces se borra el bit válido y la posición de ráster actual y las coordenadas de color y textura asociadas no están definidas.

```
void menu (int opc)
{
```

En el vector VF, en su campo 1, se almacenara el id de la imagen

```
  vf[1]=opc;
glPushMatrix();

  switch (opc)
    {
    case 1:
```

```
   pauseAnimation();
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glLoadIdentity();

   Sol_Info();

   glutSwapBuffers();

   break;

   case 2:

   pauseAnimation();
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glLoadIdentity();
```

GlLoadIdentity

GlLoadIdentity - reemplaza la matriz actual por la matriz de identidad

Especificaciones

Void glLoadIdentity (void);

Descripción

GlLoadIdentity reemplaza la matriz actual por la matriz de identidad. Es semánticamente equivalente a llamar a glLoadMatrix con la matriz de identidad(1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1)

Carga cada una de las imagines de texturas de los planetas

```
void initTex(){
   texSun = loadTex("Texturas/Sol.jpg");
   texMercurio = loadTex("Texturas/Mercurio.jpg");
   texVenus = loadTex("Texturas/Venus.jpg");
   texTierra = loadTex("Texturas/Tierra.jpg");
   texMarte = loadTex("Texturas/Marte.jpg");
   texJupiter = loadTex("Texturas/Jupiter.jpg");
   texSaturno = loadTex("Texturas/Saturno.jpg");
   texUrano = loadTex("Texturas/Urano.jpg");
   texNeptuno = loadTex("Texturas/Neptuno.jpg");
   texLuna = loadTex("Texturas/Luna.jpg");
}
```

```
void reshape(int width, int height){

   glViewport(0, 0, width, height);
   Matriz de proyeccion.
   glMatrixMode(GL_PROJECTION);
   Cargamos la matriz identidad.
   glLoadIdentity();
   glOrtho(-300,300,-300,300,-300,300);
   Matriz de modelado.
  glMatrixMode(GL_MODELVIEW);
   Matriz identidad.
   glLoadIdentity();
  Activamos el Test de profundidad.
  glEnable(GL_DEPTH_TEST);
   Funcion de borrrado buffer depth.
glDepthFunc(GL_LEQUAL);
 }
```

Función para controlar teclas especiales

```
void specialKeys( int key, int x, int y )
{
   Flecha derecha: aumentar rotación 7 grados
   if (key == GLUT_KEY_RIGHT)
      rotate_y += 5;

   Flecha izquierda: rotación en eje Y negativo 7 grados
   else if (key == GLUT_KEY_LEFT)
      rotate_y -= 5;

   Flecha arriba: rotación en eje X positivo 7 grados
   else if (key == GLUT_KEY_UP)
```

Función para controlar teclas normales del teclado.

```
void keyboard(unsigned char key, int x, int y)
{
   control de teclas que hacen referencia a Escalar y transladar el cubo en los ejes X,Y,Z.

   switch (key)
   {
   case 'q':

      if(scale >=3){
         scale = 3;
      }
```

```
 else{
          scale +=0.02;
       }

   break;
void pauseAnimation() {
      Llamamos a la función para detener la animación
   animating = 0;
}

void timerFunction(int timerID) {
     Invocamos la funcion para controlar el tiempo de la ejecucion de funciones
   if (animating) {
      //updateFrame();
      glutTimerFunc(30, timerFunction, 0);
      glutPostRedisplay();
   }
}

void startAnimation() {
     llamamos la función para iniciar la animación
  if ( ! animating ) {
     animating = 1;
     glutTimerFunc(30, timerFunction, 1);
  }
}

void init (void){

            glClearColor(0.0, 0.0, 0.0, 0.0);
            glShadeModel(GL_SMOOTH);
            glEnable(GL_TEXTURE_2D);

            // set up lighting
            glEnable(GL_LIGHTING);
            glMatrixMode(GL_MODELVIEW);
            glLoadIdentity();
            GLfloat matSpecular[] = { 1.0, 1.0, 1.0, 1.0 };
            GLfloat matAmbience[] = { 0.3, 0.3, 0.3, 1.0 };
            GLfloat matShininess[] = { 20.0 };
            glClearColor (0.0, 0.0, 0.0, 0.0);
            glShadeModel (GL_SMOOTH);

            glMaterialfv(GL_FRONT, GL_SPECULAR, matSpecular);
            glMaterialfv(GL_FRONT, GL_SHININESS, matShininess);
```

```
glMaterialfv(GL_FRONT, GL_AMBIENT, matAmbience);

        GLfloat lightAmbient[] = { 0.3, 0.3, 0.3, 1.0 };
        GLfloat lightDiffuse[] = { 1.0, 1.0, 1.0, 1.0 };
        GLfloat lightSpecular[] = { 1.0, 1.0, 1.0, 1.0 };

        glLightfv(GL_LIGHT0, GL_AMBIENT, lightAmbient);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, lightDiffuse);
        glLightfv(GL_LIGHT0, GL_SPECULAR, lightSpecular);

        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);
        glDisable(GL_LIGHTING);
}
int main(int argc, char* argv[])
  glutInit(&argc, argv);//Inicializacion del glut.
   glutInitWindowSize(1000,800);//Tamaño de la ventana.
   glutInitWindowPosition(180,0);//Posicion de la ventana en el monitor.
   glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);//Modo de
visualizacion inicial.
   glutCreateWindow("SISTEMA SOLAR");//Nombre de la ventana.
   OpenGLInit();
   init();
   initTex();
   glClearColor(0,0,0,0);//Especificar valores claros para el color del buffer.
   glutDisplayFunc(display);//Establece la devolución de llamada de visualización de la
ventana actual.
   glutReshapeFunc(reshape);//Control de ventanas.

   glutCreateMenu(menu);
   glutAddMenuEntry("Sol",1);
   glutAddMenuEntry("Mercurio",2);
   glutAddMenuEntry("Venus",3);
   glutAddMenuEntry("Tierra",4);
   glutAddMenuEntry("Marte",5);
   glutAddMenuEntry("Jupiter",6);
   glutAddMenuEntry("Saturno",7);
   glutAddMenuEntry("Urano",8);
   glutAddMenuEntry("Neptuno",9);
   glutAddMenuEntry("Salir",10);
  glutAttachMenu(GLUT_RIGHT_BUTTON);

   glutKeyboardFunc(keyboard);//Funcion para teclas normales.
   glutSpecialFunc(specialKeys);//Funcion para teclas especiales.
```

```
//glutMouseFunc(mouse);
 //glutMouseWheelFunc(MouseWheel);
 //glutFullScreen();//Funcion para mostrar pantalla completa.
 startAnimation();//Funcion de animacion.
 glutMainLoop();//Bucle de procesamiento de eventos de GLUT.
 return 0;  }
```

3. ASPECTOS ADMINISTRATIVOS

3.1-RECURSOS HUMANOS

Las personas que participan, y contribuyen en la elaboración del proyecto son muy esenciales e importantes ya que por medio de todas estas partes se llevan a cabo cada uno de los objetivos propuestos, de igual forma son el elemento activo que garantiza los resultados de la Actividad de Investigación y Desarrollo.

Para esto debemos tomar muy encuenta la clasificación de cada una de estas personas, que se harán responsables de cada una de sus partes y llevaran a cabo el ciclo de desarrollo de dicho proyecto

En nuestro proyecto de simulación del sistema solar necesitamos los siguientes recursos humanos personales.

El Diseñador

Es la persona encargada del diseño gráfico, en quien le da forma a la idea llevándola a una simulación física.

El programador

Es quien se encarga de programar las condiciones de la simulación y darle vida al diseño con diferentes texturas y luces así como también animación por medio de funciones controladas por el teclado o mause. Esta persona es la que lleva el control interno de cada una de las diferentes funciones que haga la simulación.

En este proyecto los responsables son los siguientes

Ana Ruth Sanchez Henriquez	Diseñador y documentacion
Carlos Edenilson Romero	Analalista y programador

3.2-PRESUPUESTO

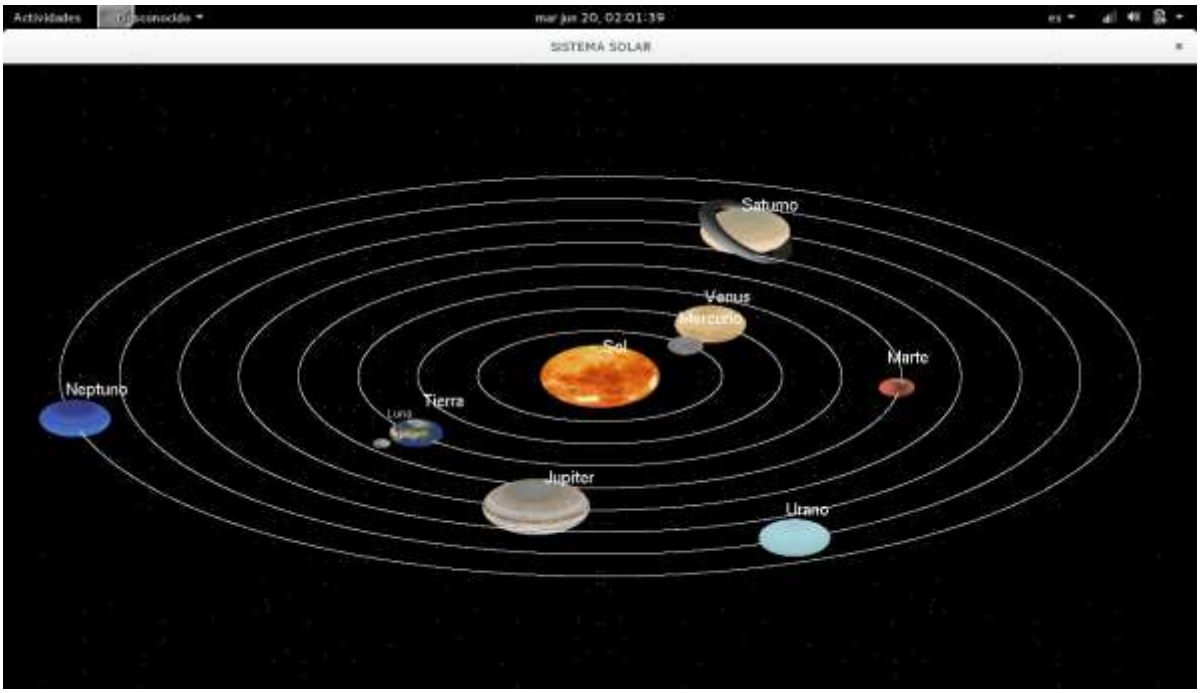
Componentes	**Tiempo (horas)**	**Costo unitario**	**Costo total Mensual**
Diseñador	125h al mes	$7	$875
Animador	140h al mes	$10	$1,400
Programador	190h al mes	$12	$2,280
Gasto de energía eléctrica	25.7kw/h	$0.30ctvs	$7.71
Viáticos			$150.50
Costo general del proyecto			$4,713.21

3.3-CRONOGRAMA

CRONOGRAMA DEL PROYECTO

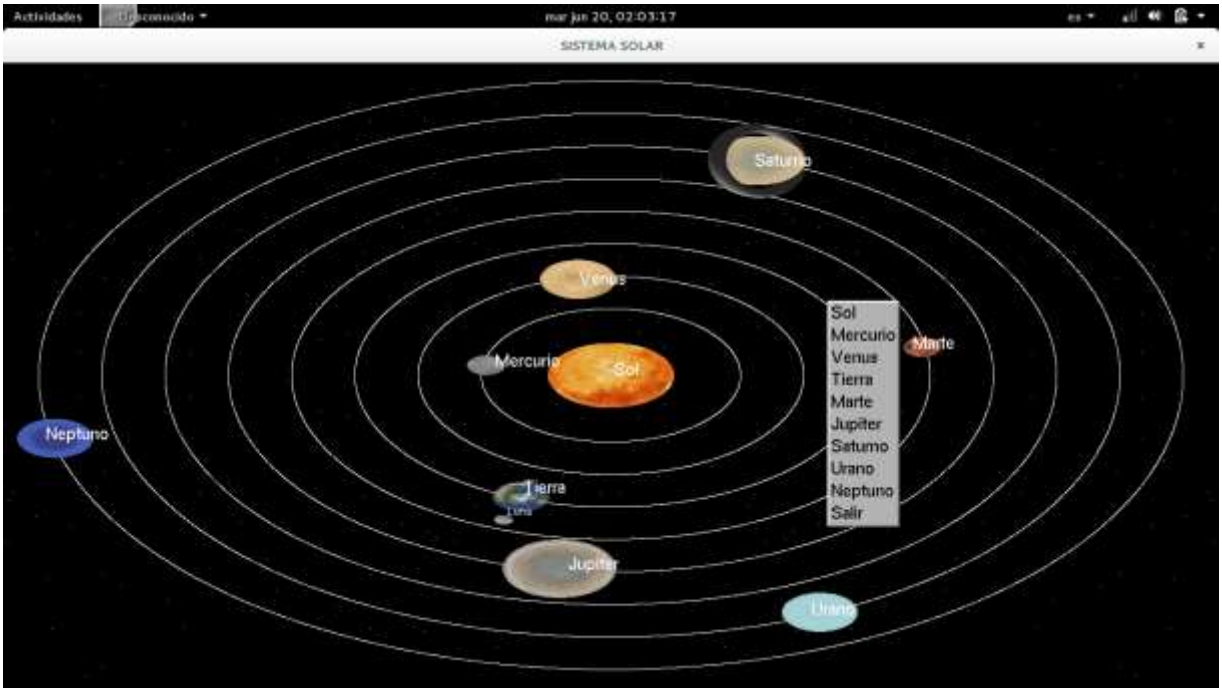
	MESES	Mayo				Junio			
Actividad del Proyecto*	Semana	1	2	3	4	1	2	3	4
Determinar los temas									
Recopilar la información									
Elaboración del reporte									
Análisis del Sistema									
Programación y diseño									
Entrega y defensa									

ANEXOS



SE OBSERVAN CADA UNO DE LOS PLANETAS GIRANDO ALREDEDOR DEL SOL,CON SUS DIFERENTES TEXTURAS

SE PUEDE HACERCAR Y ALEJAR CON LAS DIFERENTES FUNCIONES DE TECLADO



TIENE LA OPCION MENU DONDE SE PUEDE ACCEDER A CUALQUIERA DE LOS PLANETAS

POR MEDIO DEL ,MENÚ PODEMOS ENCONTRAR LA RESPECTIVA INFORMACION DE CADA UNO DE LOS PLANETAS

CONCLUSION

En el anterior proyecto se dio a conocer la simulacion de un sistema solar hecho en openGL, empleando conceptos y definiciones hacerca de nuestro sistema solar, implementrando las diferentes librerias utilizadas para el funcionamiento.

Asi como tambien se pudo observar las diferentes texturas y animaciones que se le implementa, siendo asi el unico proposito de poder mostrar a los niños como es nuestro sistema solar de una forma mas tecnologica y que estos aprendan y conoscan la informacion de cada uno de ellos.

De igual forma en la implementacion de un menu se pudo observar cada uno de los planetas de forma individual con su propia informacion principal.

De esta manera se pudo mostrar la explicacion de cada uno de los procesos de programacion que se fueron dando a lo largo del proyecto,para que puedan observar como va implementado en codigo fuente.

Asi como tambien sus diferentes capturas mostrando la funcionavilidad de este.

Asi es como nuestra simulacion llamada SOLAR SIMULATOR SYSTEM ayuda a todos los niños y personas interesadas a conocer mas sobre nuestro sistema solar Atravez de una forma muy sencilla, atractiva y fundamental; lo que es Atravez de la tecnologia.

Y es asi como damos por concluido nuestro proyecto.

4. REFERENCIAS

- Cañeros Morales, C. (2014). Apuntes de OpenGL y GLUT. Recuperado el 08 de junio de 2017, de http://www.elai.upm.es/webantigua/spain/Asignaturas/InfoInd/teoria/Apuntes%20de%20OpenGL.pdf

- khronos.org/registry/OpenGLRefpages/es2.0/xhtml/glTexParameter.xl

- google.com.sv/search?q=w&oq=w&aqs=chrome..69i60l3j69i57.1568j0j7&sourceid=chrome&ie=UTF-8#q=glLightfv+khronos

- www.khronos.org/registry/OpenGLRefpages/es1.1/xhtml/glLoadIdentity.xml

- https://es.wikipedia.org/wiki/Sistema_solar