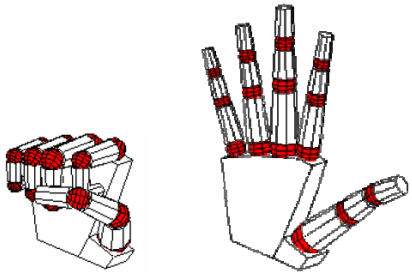


Laboratorio 04

Transformaciones geométricas con OpenGL

Empleando primitivas gráficas y transformaciones geométricas se pueden dar movimiento a los objetos gráficos.
http://www.frontiernet.net/~eugene.ressler/sketch.html

Objetivo: El objetivo de esta práctica, es que el alumno conozca y sepa cómo aplicar los principales comandos de transformaciones geométricas en OpenGL, esto es para dar movimiento a los objetos (traslación y rotación) previa modificación de las características graficas de la escena.

Duración de la Práctica: 2 Horas.

Lugar de realización: Laboratorio de cómputo.

El conocimiento requerido para realizar esta Práctica es de haber asimilado los conceptos básicos de C/C++ y OpenGL; además de la teoría necesaria para transformaciones geométricas en 2D y 3D.

El desarrollo tendrá la siguiente estructura de temas:

1. Introducción
2. Transformaciones geométricas en OpenGL
3. Interacción y animación a través GLUT
   a. Programación guiada por eventos
   b. Control y uso del mouse
   c. Control y uso del teclado
4. Programa ejemplo con OpenGL
5. Algunas primitivas gráficas 3D
6. Ejercicios propuestos
7. Referencias

1. INTRODUCCIÓN

En nuestro mundo 3D tenemos una cantidad increíble de objetos geométricos mayor a la que tendríamos en una representación 2D. Su representación matemática se complica y esto nos perjudica si deseamos crear una aplicación eficiente computacionalmente hablando, además que no deseamos perder demasiado tiempo calculando valores para situar nuestra geometría en pantalla.

Existen **representaciones** gráficas basadas en curvas y superficies (Figura No. 1), al fin y al cabo, todo se acaba reduciendo a un objeto cuyas fronteras son planas (Figura No. 2) y por tanto descriptibles usando:

Puntos, entidades mínimas, de dimensión 0.
Lados, o uniones entre dos puntos, de dimensión 1.
Caras, o uniones sucesivas de segmentos, de dimensión 2.

Por tanto, modelar, o sea crear modelos geométricos de aquellos objetos que deseamos simular, se reduce a diseñar vértices, lados y caras, teniendo en cuenta una apropiada estructura de datos.

Cabe notar que a un sistema gráfico le es enormemente cómodo trabajar con triángulos, dado que representa mejor a las superficies y resulta ser más simple de manipularla; si todas las caras de nuestros objetos están definidas por uniones de triángulos conseguiremos buenos rendimientos. Por ejemplo, las tarjetas aceleradoras de gráficos 3D del mercado, ellas generan millones de triángulos por segundo en pantalla.

Generando triángulos estamos seguros de tener siempre vértices coplanarios y de hecho la descarga de trabajo computacional que eso le supone a la CPU es impresionante.

En caso de tener una figura modelada con otro tipo de polígonos que no sean triángulos, siempre podemos usar un el proceso denominado de **tesselation**[1], o sea descomposición en triángulos; donde cualquier polígono puede descomponerse literalmente en triángulos e incluso dada una triangulación esta puede reordenarse de tal forma que se optimice la estructura y que esta muestre una mejor perspectiva de la geometría.

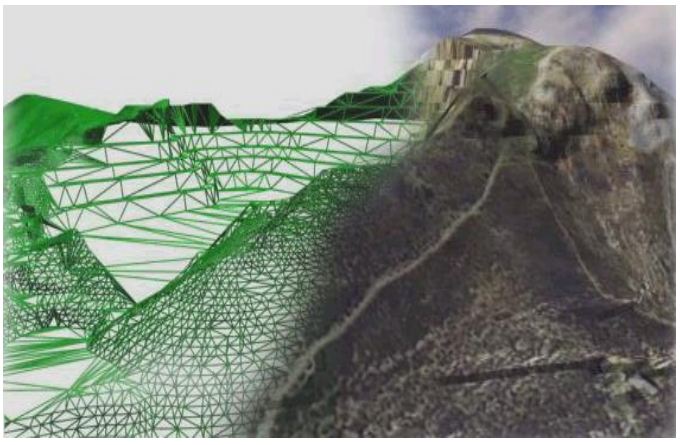


Figura No. 1. Representación de superficies.

Figura No. 2. Origami.

Existen algunas excepciones a este proceso de modelización como la **geometría sólida constructiva** (**Constructive Solid Geometry - CSG**). En esta aproximación, se crean objetos complejos mediante **uniones** e **intersecciones** (álgebra de sólidos) de objetos volumétricos más simples. Los resultados suelen ser muy buenos aunque es una técnica muy complicada y costosa, y por eso muchas veces desechada y dejada para los renderizados finales.

[1] La teselación implica la construcción de mallas para una mejor representación de un modelo. En ingeniería, los Elementos Finitos hace uso de ella. En Geometría computacional el algoritmo de Voronoi está muy ligada a ella.

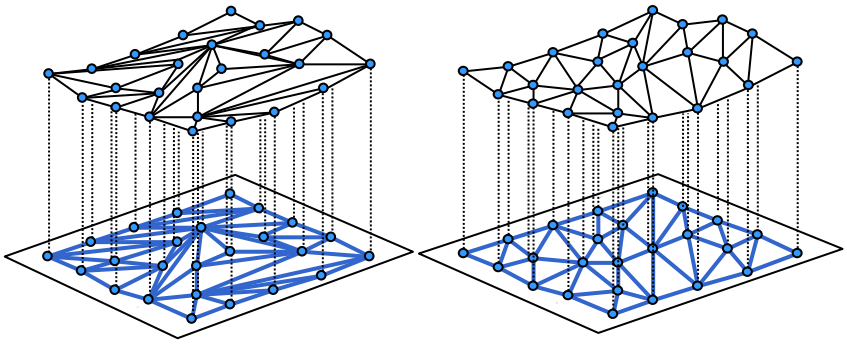


Figura No. 3. Representaciones Alámbricas.

Así, nosotros siempre trabajaremos con las representaciones alámbricas las veces que sea posible (esto es para minimizar el costo computacional).

Pretendemos trabajar la geometría ya modelada (la alámbrica) mediante transformaciones (aquí solamente las usaremos, en teoría se verá al detalle), ya sean traslaciones, rotaciones, escalamientos y composiciones de ellas aplicadas a los vértices (y aristas), dado que para transformar un polígono será suficiente transformar solo sus vértices; todas estas transformaciones son llamadas también de afines y rígidas (según sea el caso), en las primeras los lados pueden sufrir modificaciones pero no los ángulos y en la segunda ninguna de ellas.

2. TRANSFORMACIONES GEOMÉTRICAS EN OPENGL

OpenGL provee las respectivas funciones para efectuar las transformaciones geométricas, las cuales son invocadas empleando las funciones:

- **glScalef(GLfloat sx, GLfloat sy, GLfloat sz):** escalamiento según sean los factores **sx**, **sy** y **sz**, en los respectivos ejes cartesianos.
- **glTranslatef(GLfloat dx, GLfloat dy, GLfloat dz):** traslación según los desplazamientos **dx**, **dy** y **dz**, en las direcciones de los ejes cartesianos.
- **glRotatef(GLfloat angulo, GLfloat vx, GLFloat vy, GLfloat vz):** rotación según un determinado ángulo alrededor de un eje de rotación. Este eje viene definido por el vector dirección (**vx,vy,vz**).

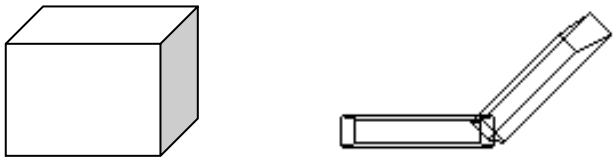


Figura No. 4. Transformaciones Afines y Rígidas

¿Cómo se implementa esto?:

En OpenGL la geometría es afectada por la Matriz de Transformación actual (**CTM Current Transformation Matrix**), que no es más que la composición acumulada de todas las matrices de transformación que se hayan ido llamando. Una forma de proceder sería la siguiente:

```
glMatrixMode(GL_MODELVIEW);    /* Activa la matriz */
glLoadIdentity();              /* Limpia la matriz */
```

```
        /* afectará la geometría que se dibuje a partir de ahora */
glScalef();
glRotatef();
glTranslatef();
glPushMatrix( );
/* salva el estado actual de la matriz, es decir, las 3 transformaciones
anteriores, si lo necesitase en el programa */
glScalef();
/* Renderizado de la geometría que pasará por 4 transformaciones */
dibujo_de_geometría_específica( );
glPopMatrix( );      /* recupera el estado de la matriz anterior */
```

3. INTERACCIÓN Y ANIMACIÓN USANDO GLUT

a) PROGRAMACIÓN GUIADA POR EVENTOS

Un evento es algo que el usuario puede hacer, como por ejemplo maximizar una ventana, redimensionarla, pulsar el botón izquierdo del ratón, o usar una determinada combinación de teclas.

En OpenGL, y gracias a GLUT, se le permite al usuario el control de los botones del ratón, moviéndolo por la pantalla, apretando teclas, cambiando la ventana de la aplicación. Cada vez que éste provoque alguno de estos eventos debemos llamar a una determinada rutina o función para que se haga cargo de la acción a tomar.

Las funciones de GLUT son simples además de tener pocos parámetros, no devuelven punteros y los que lo hacen son punteros pasados a cadenas de caracteres y manejadores de fuentes. Estas funciones se clasifican según su funcionabilidad: inicialización, inicio del procesado de eventos, control de overlay, control de menús, registro de funciones **Callback**, control de mapa de colores, obtención del estado, trazado de fuentes y de formas geométricas. En el transcurso de los talleres iremos disertando acerca de ellas y de cómo usarlas, recordar el taller anterior en la que ya se han visto algunos de ellos, por ejemplo:

- **GlutinitDisplayMode(unsigned int modo):** los posibles valores de modo son:

GLUT_RGBA	Selecciona una ventana en modo RGBA. Es el valor por defecto si no se indican ni GLUT_RGBA ni GLUT_INDEX.
GLUT_RGB	Lo mismo que GLUT_RGBA.
GLUT_INDEX	Selecciona una ventana en modo de índice de colores. Se impone sobre GLUT_RGBA.
GLUT_SINGLE	Selecciona una ventana en modo buffer simple. Es el valor por defecto.
GLUT_DOUBLE	Selecciona una ventana en modo buffer doble. Se impone sobre GLUT_SINGLE.
GLUT_ACCUM	Selecciona una ventana con un buffer acumulativo.
GLUT_ALPHA	Selecciona una ventana con una componente alpha del buffer de color.
GLUT_DEPTH	Selecciona una ventana con un buffer de profundidad.
GLUT_STENCIL	Selecciona una ventana con un buffer de estarcido.
GLUT_MULTISAMPLE	Selecciona una ventana con soporte multimuestra.
GLUT_STEREO	Selecciona una ventana estéreo.
GLUT_LUMINANCE	Selecciona una ventana con un modelo de color de "luminancia".

De entre los controladores de eventos más importantes y usados veremos el control de Ratón (mouse) y de Teclado.

b) CONTROL Y USO DEL MOUSE

- **glutMouseFunc(control_de_Mouse):** cada vez que se pulse uno de los botones del mouse, OpenGL llamará a la función **control_de_Mouse** el cuál es definida por el programador de acuerdo de sus necesidades, siendo el formato como sigue:

```
void control_de_Mouse(int boton, int estado, int x, int y)
{
        /* código */
}
```

Donde los parámetros (que nos dan automáticamente) son:

- **boton:** pueden ser GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON ó GLUT_RIGHT_BUTTON según se halla pulsado el botón izquierdo, del centro o el derecho, todas ellas predefinidas.
- **estado:** GLUT_UP ó GLUT_DOWN, según se halla pulsado o soltado el botón del Mouse.
- **x** e **y**: coordenadas de la ventana de visualización en la que se pulso el botón del Mouse.

Un ejemplo simple podría ser:

```
void control_de_Mouse(int boton, int estado, int x, int y )
{
   if (boton==GLUT_LEFT_BUTTON && estado==GLUT_DOWN)
        {
           printf( "trabajando con opengl");
        // podemos imprimirlos valores de x e y
      }
}
```

y luego lo llamamos como:

```
glutMouseFunc(control_de_Mouse);
```

c) CONTROL Y USO DEL TECLADO:

- **glutKeyboardFunc(control_de_Teclado):** esta (función) llama a la función **control_de_Teclado** el cuál debe tener el formato siguiente:

```
void control_de_Teclado(usigned char tecla, int x, int y)
{
        /* código */
}
```

Por ejemplo:

```
void control_de_Teclado(unsigned char key, int x, int y )
{
        /* Según la tecla pulsada realiza una tarea especifica */
        switch(key){
           case 'q' :
              /* tarea 1 */
              break;
           case 'w' :
              /* tarea 2 */
              break;
           case 'e' :
              /* tarea 3 */
```

```
                    break;
                }
        }
```

y luego lo llamamos como:

```
glutKeyboardFunc(control_de_Teclado);
```

Otra función muy útil es:

- **GlutMotionFunc(control_de_movimiento_Mouse):** esta llama a la función **control_de_movimiento_Mouse** el cuál debe tener el formato siguiente:

```
void control_de_movimientoMouse(GLsizei x,GLsizei y)
{
/* por ejemplo el siguiente código */
        printf( "La posición del mouse  es:/n");
        printf( " x = %f/n", (GLfloat)GLsizei x);
        printf( " y = %f/n", (GLfloat)GLsizei y);
}
```

Las funciones **glutReshapeFunc(), glutDisplayFunc()** se han visto en la clase anterior, las restantes deben ser consultadas en la bibliografía mencionada al final de las notas; las cuáles se verán de acuerdo a nuestros temas a desarrollar.

4. PROGRAMA EJEMPLO CON OPENGL

Implemente una aplicación que permita aplicar las transformaciones geométricas mencionadas atrás y que las podamos controlar mediante el uso del teclado. El esqueleto de programa propuesto es el que se muestra a continuación, en ella se sugiere al lector que construya un cubo (de preferencia centrada y unitaria), con las siguientes características:

a) Construir una función ”cara**”** **(**o cuadrilátero**)** a partir de cuatro vértices, que permita obtener una cara rellena.

b) Construir una función ”cubo**”** empleando la función **”cara”** para cubrir las seis caras, incorpore un color diferente en cada cara a sus elección.

c) Incorpore un sistema de ejes coordenados, cada eje debe tener un color diferente (sugerencia: azul, rojo y negro).

```
#include <stdlib.h>
#include<conio.h>
#include <gl/glut.h>
// declaracion de variables
// GLfloat       ...;
// theta[] me indica los ángulos iniciales en los 3 ejes

static GLfloat theta[] = {0.0,0.0,0.0};
// eje es el ángulo a rotar
static GLint eje = 2;

// construya su poligono base
void cara()
{
  glBegin(GL_POLYGON);
        // ... escriba su código
  glEnd();
```

```
}
// construya su objeto geométrico mediante cubo()
void cubo(void)
{
  cara();
  // ... escriba su código
}

// dibujamos nuestra escena
void display(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();

// composicion de rotaciones
  glRotatef(theta[0],1.0,0.0,0.0);
  glRotatef(theta[1],0.0,1.0,0.0);
  glRotatef(theta[2],0.0,0.0,1.0);

  cubo();

  glFlush();
// intercambiamos los buffers, el que se muestra y el que esta oculto
  glutSwapBuffers();
}

// esta función controla el angulo de rotación según el eje de giro
void girar_objeto_geometrico ()
{
  theta[eje] += 2.0;
  if(theta[eje]>360) theta[eje] - = 360.0;
  display();
}

void teclado(unsigned char tecla,int x,int y)
{
        switch(tecla){
        case 'a' : eje = 0; break;
        case 's' : eje = 1; break;
        case 'd' : eje = 2; break;
        case 'f'  : exit(0) ; break;
        }
}

// control de ventana (recuerde el volumen de visualización)
// modifique dicho volumen según su conveniencia
void myReshape(int w, int h)
{
  glViewport(0,0,w,h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  if(w <=h)
        glOrtho(-2.0,2.0,-2.0*(GLfloat)h/(GLfloat)w,
          2.0*(GLfloat)h/(GLfloat)w, -10.0, 10.0);
  else
        glOrtho(-2.0*(GLfloat)w/(GLfloat)h,
          2.0*(GLfloat)w/(GLfloat)h, -2.0,2.0,-10.0,10.0);
  glMatrixMode(GL_MODELVIEW);
}
```

```
void main(int argc, char **argv)
{
 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
 glutInitWindowSize(500,500);
 glutCreateWindow("mi objeto bajo rotaciones");
 glutReshapeFunc(myReshape);
 // invocamos a display() para dibujar nuestra escena
 glutDisplayFunc(display);
 // esta funcion llama a girar_objeto_geométrico() mientras no haya evento
 // alguno ocasionado por el usuario
 glutIdleFunc(girar_objeto_geometrico);
 glutKeyboardFunc(teclado);
 /*glutMouseFunc(mouse);*/
 glEnable(GL_DEPTH_TEST);
 glutMainLoop();
}
```

5. ALGUNAS PRMITIVAS GRÀFICAS 3D

Glut proporciona una variedad muy amplia de primitivas 3D, a partir de ellas se pueden construir objetos más elaborados con ayuda de las transformaciones geométricas:

- **void glutWireCube(GLdouble lado**): dibuja un cubo (representación alámbrica) centrado en el origen con aristas de longitud igual a **lado.**

- **void glutSolidCube(GLdouble lado**): dibuja un cubo (representación sólida) centrado en el origen con aristas de longitud igual a **lado.**

- **void glutWireSphere(GLdouble radio, GLint Nmeridianos, GLint Nparalelas**): dibuja una esfera alámbrica centrada en el origen con radio igual a **radio** y número de cortes meridianos y paralelas a **Nmeridianos** y **Nparalelas** respectivamente.

- **void gluSolidSphere(GLdouble radio, GLint Nmeridianos, GLint Nparalelas**): similar a **glutWireSphere**, pero la representación es sólida.

- **void glutWireCone(GLdouble radio, GLdouble altura, GLint Nmeridianos, GLint Nparalelas**): dibuja una cono alámbrica centrada en el origen con radio igual a **radio**, la altura de dicho cono es igual a **altura** y número de cortes meridianos y paralelas a **Nmeridianos** y **Nparalelas** respectivamente.

- **void glutSolidCone(GLdouble radio, GLdouble altura, GLint Nmeridianos, GLint Nparalelas**): similar a **glutSolidCone** , pero el modelado es un sólido.

- **void glutWireTorus(GLdouble rmenor, GLdouble rmayor, GLint Nmeridianos, GLint Nparalelas**): dibuja un toroide alámbrico con radios menor y mayor iguales a **rmenor** y **rmayor** respectivamente; número de cortes meridianos y paralelas a **Nmeridianos** y **Nparalelas** respectivamente.

- **void glutSolidTorus(GLdouble rmenor, GLdouble rmayor, GLint Nmeridianos, GLint Nparalelas**): análogamente, pero el modelado es un sólido.

Consulte los textos y manuales de OpenGL para otras primitivas graficas 3D.

6. EJERCICIOS PROPUESTOS

Ejercicio 01:

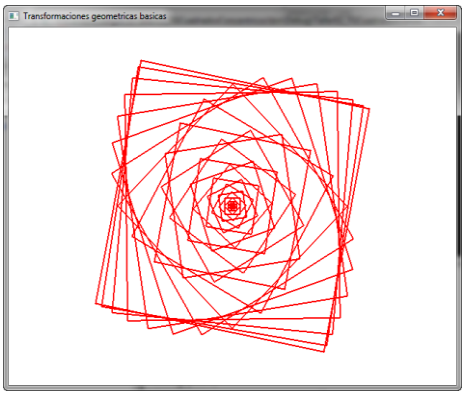
Una vez que su aplicación fuese exitosa, proceda a usar las funciones **glutWireCube** y **glutWireSphere** como el objeto geométrico a tratar, otros objetos geométricos pueden ser consultados en las referencias. También use otro tipo de transformaciones geométricas y composición de ellas.

Ejercicio 02:

Diseñe una aplicación en donde se tenga el control de las rotaciones y escalamientos usando el teclado, puede usar el cubo para las pruebas del caso.

Ejercicio 03:

Empleando cuadrados concéntricos obtenga el siguiente, de tal forma que todo el conjunto este girando en sentido antihorario. Elija el ángulo apropiadamente, además de la longitud de la arista.

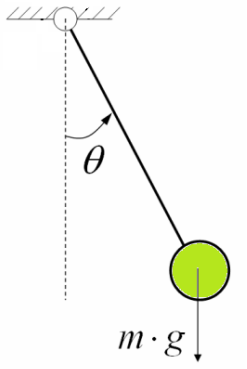


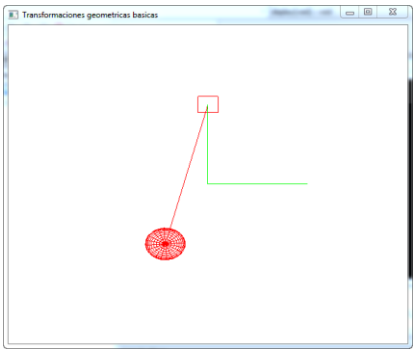
Ejercicio 04:

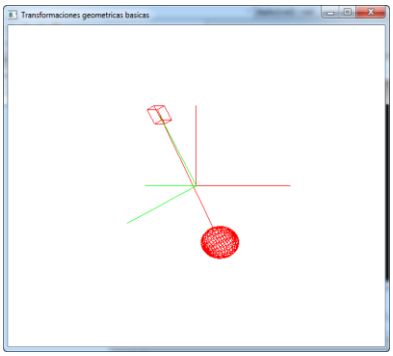
Implemente un menú jerárquico a fin de gestionar diferentes modelos geométricos, además de controlar las rotaciones.

Ejercicio 05:

Diseñe un péndulo simple enmarcado en un sistema 3D, desprecie la masa del objeto, procure las medidas del ángulo y longitud de la cuerda apropiadamente (gráficas referenciales).

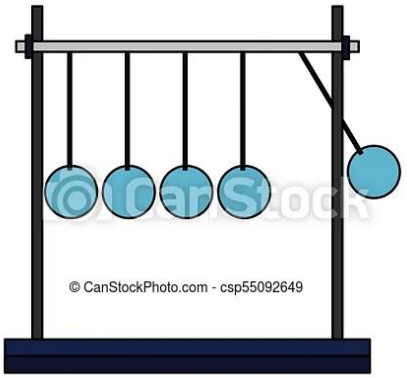






Ejercicio 06:

Diseñe un péndulo de Newton enmarcado en un sistema 3D, procure las medidas del ángulo y longitud de la cuerda apropiadamente, ver imagen referencial adjunta.



Ejercicio 07:

Implemente un programa en la que se muestre una luna girando alrededor de la tierra siguiendo una órbita elíptica, escoja las dimensiones que Ud. crea apropiada.

7. REFERENCIAS

- **Computer Graphics. Principles and Practice (Second Edition).** Foley J. D., A.van Dam, S.K.Feiner and J.F. Hughes. Addison-Wesley. 1990
- **OpenGL Superbible.** R. Wright and M. Sweet. Waite Group, Inc 1996
- **OpenGL Programming Guide: The Official Guide to Learning OpenGL.** D. Sheiner, M. Wood, J. Neider and T. Davis. Addison Wesley, 2007
- **The OpenGL Utility Toolkit (GLUT) Programming Interface (API Version 3**). Mark J. Kilgard, Silicon Graphics, Inc 1996
- **The OpenGL Graphics System: A Specification (Version 1.2).** Mark segal & Kurt Akeley. Silicon Graphics, Inc 1998.