Benemérita Universidad Autónoma de Puebla
Facultad de Ciencias de la Computación

GRAFICACIÓN

TAREA 6: MATRIZ DE MODELADO GLOBAL

Docente:
Prof. Iván Olmos Pineda

Alumno:
Jesús Huerta Aguilar

Matricula:
202041509

NRC: 10592
Sección: 001

CUARTO SEMESTRE

Puebla, Pue. Fecha de entrega: 26/10/2022

INDICE

INTRODUCCIÓN

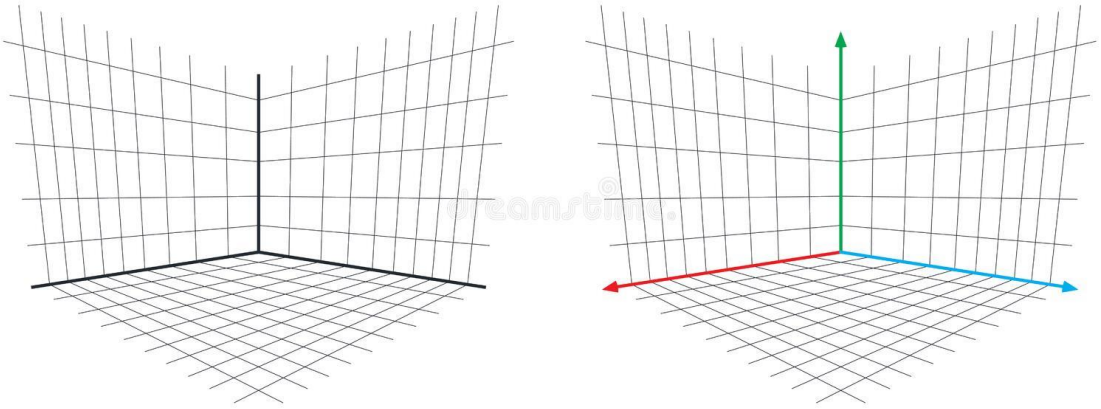
Anteriormente hemos visto que OpenGL guarda la transformación de los objetos en una matriz. A esta matriz se le denomina matriz de visualización/modelado, porque se emplea para estas dos funciones. Además de esta transformación, OpenGL posee otra matriz muy importante, que es la matriz de proyección, en la que se guarda la información relativa a la "cámara" a través de la cual vamos a visualizar el mundo. Al realizar operaciones que modifiquen alguna de estas dos matrices, tendremos que cambiar el "modo de matriz", para que las operaciones afecten a la matriz que nos interesa.

CONCEPTOS DESARROLLADOS

Si bien es cierto que OpenGL proporciona acceso a funciones de dibujado 2D, en este curso nos vamos a centrar en el espacio 3D... OpenGL trabaja, a grandes rasgos, en un espacio de tres dimensiones, aunque veremos que realmente, trabaja con coordenadas homogéneas (de cuatro dimensiones).

Las tres dimensiones que nos interesan ahora son las especificadas por un sistema 3D ortonormal. Es decir, sus ejes son perpendiculares, y cada unidad en uno de ellos está representada por un vector de módulo 1 (si nos alejamos una unidad, nos alejamos la misma distancia del eje de coordenadas, da igual la dirección).

Finalmente, es recomendable desempolvar nuestros algebraicos básicos (vectores, normales, etc.), porque como veremos, nos van a resultar de gran utilidad a la hora de programar en 3D... La situación de los ejes de coordenadas se refleja en la matriz de transformación. Esta matriz representa la transformación que se aplicará a todos los vértices que se dibujen mientras ella esté activa.



ANALISIS EMPIRICO

MAIN:

```
#include "3D_bib.h"
#include "Stage.h"
#include <GL/glut.h>

//Variables dimensiones de la pantalla
int WIDTH=500;
int HEIGTH=500;
//Variables para establecer los valores de gluPerspective
float FOVY=60.0;
float ZNEAR=0.01;
float ZFAR=100.0;
//Variables para definir la posicion del observador
//gluLookAt(EYE_X,EYE_Y,EYE_Z,CENTER_X,CENTER_Y,CENTER_Z,UP_X,UP_Y,UP_Z)
float EYE_X=1.0;
float EYE_Y=5.0;
float EYE_Z=15.0;
float CENTER_X=3;
float CENTER_Y=0;
float CENTER_Z=0;
float UP_X=0;
float UP_Y=1;
float UP_Z=0;
//Variables para dibujar los ejes del sistema
float X_MIN=-20;
float X_MAX=20;
float Y_MIN=-20;
float Y_MAX=20;
float Z_MIN=-100;
float Z_MAX=20;

//Se declara el objeto para utilizar las operaciones 3D
Operaciones3D Op3D;
Stage S(&Op3D);

float Theta1=0,sx1 = 1,sy1 = 1,sz1 = 1, tx1 = 0,ty1 = 0,tz1 = 0;
float Theta2=0,sx2 = 1,sy2 = 1,sz2 = 1, tx2 = 0,ty2 = 0,tz2 = 0;
float Theta3=0,sx3 = 1,sy3 = 1,sz3 = 1, tx3 = 0,ty3 = 0,tz3 = 0;
//Variables para la definicion de objetos
float P1[3]={0.0,0.0,5.0};
float P2[3]={0.0,5.0,0.0};
//float points[5][3]={{0,0,1},{1,0,1},{1,0,0},{0,0,0},{0.5,0.75,0.5}};
float points[8][3]= {{0,0,1},{1,0,1},{1,0,0},{0,0,0},
                     {0,1,1},{1,1,1},{1,1,0},{0,1,0}};
float points2[8][3]={{0,0,1},{1,0,1},{1,0,0},{0,0,0},
                     {0,1,1},{1,1,1},{1,1,0},{0,1,0}};
float points3[8][3]={{0,0,1},{1,0,1},{1,0,0},{0,0,0},
                     {0,1,1},{1,1,1},{1,1,0},{0,1,0}};


//Dibujar ejes
void drawAxis()
{
     glShadeModel(GL_SMOOTH);
     glLineWidth(1.0);
     //X axis in red
     glBegin(GL_LINES);
       glColor3f(1.0f,0.0f,0.0f);
       glVertex3f(X_MIN,0.0,0.0);
       glColor3f(0.5f,0.0f,0.0f);
```

```
61.       glVertex3f(X_MAX,0.0,0.0);
62.     glEnd();
63.     //Y axis in green
64.     glColor3f(0.0f,1.0f,0.0f);
65.     glBegin(GL_LINES);
66.       glColor3f(0.0f,1.0f,0.0f);
67.       glVertex3f(0.0,Y_MIN,0.0);
68.       glColor3f(0.0f,0.5f,0.0f);
69.       glVertex3f(0.0,Y_MAX,0.0);
70.     glEnd();
71.     //Z axis in blue
72.     glBegin(GL_LINES);
73.       glColor3f(0.0f,0.0f,1.0f);
74.       glVertex3f(0.0,0.0,Z_MIN);
75.       glColor3f(0.0f,0.0f,0.5f);
76.       glVertex3f(0.0,0.0,Z_MAX);
77.     glEnd();
78.     glLineWidth(1.0);
79. }
80.
81. //------------------------------------------------------------------------
82. //funciones callbacks
83. void idle(void)
84. {
85.     glutPostRedisplay();
86. }
87.
88. void reshape(int width, int height)
89. {
90.     glViewport(0, 0, width, height);
91. }
92.
93.
94.
95.
96. static void keys(unsigned char key, int x, int y)
97. {
98.     switch(key){
99.         ///OBJETO 1
100.             case '1': // ---> ROTAR +
101.                 Theta1=10;  sx1 = 1;    sy1 = 1;    sz1 = 1,    tx1 = 0,
    ty1 = 0,    tz1 = 0;
102.                 Theta2 = 0; sx2 = 1;    sy2 = 1;    sz2 = 1,    tx2 = 0,
    ty2 = 0,    tz2 = 0;
103.                 Theta3 = 0; sx3 = 1;    sy3 = 1;    sz3 = 1,    tx3 = 0,
    ty3 = 0,    tz3 = 0;
104.                 break;
105.             case '2': // ---> ROTAR -
106.                 Theta1=-10; sx1 = 1;    sy1 = 1;    sz1 = 1,    tx1 = 0,
    ty1 = 0,    tz1 = 0;
107.                 Theta2 = 0; sx2 = 1;    sy2 = 1;    sz2 = 1,    tx2 = 0,
    ty2 = 0,    tz2 = 0;
108.                 Theta3 = 0; sx3 = 1;    sy3 = 1;    sz3 = 1,    tx3 = 0,
    ty3 = 0,    tz3 = 0;
109.                 break;
110.             case '3': // ---> ESCALAR +
111.                 Theta1 = 0; sx1 = 1.1;  sy1 = 1.1;  sz1 = 1.1,  tx1 = 0,
    ty1 = 0,    tz1 = 0;
112.                 Theta2 = 0; sx2 = 1;    sy2 = 1;    sz2 = 1,    tx2 = 0,
    ty2 = 0,    tz2 = 0;
113.                 Theta3 = 0; sx3 = 1;    sy3 = 1;    sz3 = 1,    tx3 = 0,
    ty3 = 0,    tz3 = 0;
114.                 break;
```

```
115.              case '4': // ---> ESCALAR -
116.                   Theta1 = 0; sx1 = 0.9;  sy1 = 0.9;  sz1 = 0.9,  tx1 = 0,
    ty1 = 0,    tz1 = 0;
117.                   Theta2 = 0; sx2 = 1;    sy2 = 1;    sz2 = 1,    tx2 = 0,
    ty2 = 0,    tz2 = 0;
118.                   Theta3 = 0; sx3 = 1;    sy3 = 1;    sz3 = 1,    tx3 = 0,
    ty3 = 0,    tz3 = 0;
119.                   break;
120.              case '5': // ---> TRASLACIÓN X
121.                   Theta1 = 0; sx1 = 1;    sy1 = 1;    sz1 = 1,    tx1 =
    0.1,  ty1 = 0,    tz1 = 0;
122.                   Theta2 = 0; sx2 = 1;    sy2 = 1;    sz2 = 1,    tx2 = 0,
    ty2 = 0,    tz2 = 0;
123.                   Theta3 = 0; sx3 = 1;    sy3 = 1;    sz3 = 1,    tx3 = 0,
    ty3 = 0,    tz3 = 0;
124.                   break;
125.              case '6': // ---> TRASLACIÓN Y
126.                   Theta1 = 0; sx1 = 1;    sy1 = 1;    sz1 = 1,    tx1 = 0,
    ty1 = 0.1,  tz1 = 0;
127.                   Theta2 = 0; sx2 = 1;    sy2 = 1;    sz2 = 1,    tx2 = 0,
    ty2 = 0,    tz2 = 0;
128.                   Theta3 = 0; sx3 = 1;    sy3 = 1;    sz3 = 1,    tx3 = 0,
    ty3 = 0,    tz3 = 0;
129.                   break;
130.              case '7': // ---> TRASLACIÓN Z
131.                   Theta1 = 0; sx1 = 1;    sy1 = 1;    sz1 = 1,    tx1 = 0,
    ty1 = 0,    tz1 = -0.1;
132.                   Theta2 = 0; sx2 = 1;    sy2 = 1;    sz2 = 1,    tx2 = 0,
    ty2 = 0,    tz2 = 0;
133.                   Theta3 = 0; sx3 = 1;    sy3 = 1;    sz3 = 1,    tx3 = 0,
    ty3 = 0,    tz3 = 0;
134.                   break;
135.              ///OBJETO 2
136.              case 'q': // ---> ROTAR +
137.                   Theta1 = 0; sx1 = 1;    sy1 = 1;    sz1 = 1,    tx1 = 0,
    ty1 = 0,    tz1 = 0;
138.                   Theta2 =10; sx2 = 1;    sy2 = 1;    sz2 = 1,    tx2 = 0,
    ty2 = 0,    tz2 = 0;
139.                   Theta3 = 0; sx3 = 1;    sy3 = 1;    sz3 = 1,    tx3 = 0,
    ty3 = 0,    tz3 = 0;
140.                   break;
141.              case 'w': // ---> ROTAR -
142.                   Theta1 = 0; sx1 = 1;    sy1 = 1;    sz1 = 1,    tx1 = 0,
    ty1 = 0,    tz1 = 0;
143.                   Theta2 =-10; sx2 = 1;    sy2 = 1;    sz2 = 1,    tx2 =
    0,    ty2 = 0,    tz2 = 0;
144.                   Theta3 = 0; sx3 = 1;    sy3 = 1;    sz3 = 1,    tx3 = 0,
    ty3 = 0,    tz3 = 0;
145.                   break;
146.              case 'e': // ---> ESCALAR +
147.                   Theta1 = 0; sx1 = 1;    sy1 = 1;    sz1 = 1,    tx1 = 0,
    ty1 = 0,    tz1 = 0;
148.                   Theta2 = 0; sx2 = 1.1;  sy2 = 1.1;  sz2 = 1.1,  tx2 = 0,
    ty2 = 0,    tz2 = 0;
149.                   Theta3 = 0; sx3 = 1;    sy3 = 1;    sz3 = 1,    tx3 = 0,
    ty3 = 0,    tz3 = 0;
150.                   break;
151.              case 'r': // ---> ESCALAR -
152.                   Theta1 = 0; sx1 = 1;    sy1 = 1;    sz1 = 1,    tx1 = 0,
    ty1 = 0,    tz1 = 0;
153.                   Theta2 = 0; sx2 = 0.9;  sy2 = 0.9;  sz2 = 0.9,  tx2 = 0,
    ty2 = 0,    tz2 = 0;
```

```
154.                 Theta3 = 0; sx3 = 1;    sy3 = 1;    sz3 = 1,    tx3 = 0,
    ty3 = 0,    tz3 = 0;
155.                 break;
156.             case 't': // ---> TRASLACIÓN X
157.                 Theta1 = 0; sx1 = 1;    sy1 = 1;    sz1 = 1,    tx1 = 0,
    ty1 = 0,    tz1 = 0;
158.                 Theta2 = 0; sx2 = 1;    sy2 = 1;    sz2 = 1,    tx2 =
    0.1,  ty2 = 0,    tz2 = 0;
159.                 Theta3 = 0; sx3 = 1;    sy3 = 1;    sz3 = 1,    tx3 = 0,
    ty3 = 0,    tz3 = 0;
160.                 break;
161.             case 'y': // ---> TRASLACIÓN Y
162.                 Theta1 = 0; sx1 = 1;    sy1 = 1;    sz1 = 1,    tx1 = 0,
    ty1 = 0,    tz1 = 0;
163.                 Theta2 = 0; sx2 = 1;    sy2 = 1;    sz2 = 1,    tx2 = 0,
    ty2 = 0.1,  tz2 = 0;
164.                 Theta3 = 0; sx3 = 1;    sy3 = 1;    sz3 = 1,    tx3 = 0,
    ty3 = 0,    tz3 = 0;
165.                 break;
166.             case 'u': // ---> TRASLACIÓN Z
167.                 Theta1 = 0; sx1 = 1;    sy1 = 1;    sz1 = 1,    tx1 = 0,
    ty1 = 0,    tz1 = 0;
168.                 Theta2 = 0; sx2 = 1;    sy2 = 1;    sz2 = 1,    tx2 = 0,
    ty2 = 0,    tz2 = 0.1;
169.                 Theta3 = 0; sx3 = 1;    sy3 = 1;    sz3 = 1,    tx3 = 0,
    ty3 = 0,    tz3 = 0;
170.                 break;
171.             ///OBJETO 3
172.             case 'a': // ---> ROTAR +
173.                 Theta1 = 0; sx1 = 1;    sy1 = 1;    sz1 = 1,    tx1 = 0,
    ty1 = 0,    tz1 = 0;
174.                 Theta2 = 0; sx2 = 1;    sy2 = 1;    sz2 = 1,    tx2 = 0,
    ty2 = 0,    tz2 = 0;
175.                 Theta3 =10; sx3 = 1;    sy3 = 1;    sz3 = 1,    tx3 = 0,
    ty3 = 0,    tz3 = 0;
176.                 break;
177.             case 's': // ---> ROTAR -
178.                 Theta1 = 0; sx1 = 1;    sy1 = 1;    sz1 = 1,    tx1 = 0,
    ty1 = 0,    tz1 = 0;
179.                 Theta2 = 0; sx2 = 1;    sy2 = 1;    sz2 = 1,    tx2 = 0,
    ty2 = 0,    tz2 = 0;
180.                 Theta3 =-10;sx3 = 1;    sy3 = 1;    sz3 = 1,    tx3 = 0,
    ty3 = 0,    tz3 = 0;
181.                 break;
182.             case 'd': // ---> ESCALAR +
183.                 Theta1 = 0; sx1 = 1;    sy1 = 1;    sz1 = 1,    tx1 = 0,
    ty1 = 0,    tz1 = 0;
184.                 Theta2 = 0; sx2 = 1;    sy2 = 1;    sz2 = 1,    tx2 = 0,
    ty2 = 0,    tz2 = 0;
185.                 Theta3 = 0; sx3 = 1.1;  sy3 = 1.1;  sz3 = 1.1,  tx3 = 0,
    ty3 = 0,    tz3 = 0;
186.                 break;
187.             case 'f': // ---> ESCALAR -
188.                 Theta1 = 0; sx1 = 1;    sy1 = 1;    sz1 = 1,    tx1 = 0,
    ty1 = 0,    tz1 = 0;
189.                 Theta2 = 0; sx2 = 1;    sy2 = 1;    sz2 = 1,    tx2 = 0,
    ty2 = 0,    tz2 = 0;
190.                 Theta3 = 0; sx3 = 0.9;  sy3 = 0.9;  sz3 = 0.9,  tx3 = 0,
    ty3 = 0,    tz3 = 0;
191.                 break;
192.             case 'g': // ---> TRASLACIÓN X
193.                 Theta1 = 0; sx1 = 1;    sy1 = 1;    sz1 = 1,    tx1 = 0,
    ty1 = 0,    tz1 = 0;
```

```
194.                 Theta2 = 0; sx2 = 1;    sy2 = 1;    sz2 = 1,    tx2 = 0,
    ty2 = 0,    tz2 = 0;
195.                 Theta3 = 0; sx3 = 1;    sy3 = 1;    sz3 = 1,    tx3 =
    0.1,  ty3 = 0,    tz3 = 0;
196.                 break;
197.             case 'h': // ---> TRASLACIÓN Y
198.                 Theta1 = 0; sx1 = 1;    sy1 = 1;    sz1 = 1,    tx1 = 0,
    ty1 = 0,    tz1 = 0;
199.                 Theta2 = 0; sx2 = 1;    sy2 = 1;    sz2 = 1,    tx2 = 0,
    ty2 = 0,    tz2 = 0;
200.                 Theta3 = 0; sx3 = 1;    sy3 = 1;    sz3 = 1,    tx3 = 0,
    ty3 = 0.1,  tz3 = 0;
201.                 break;
202.             case 'j': // ---> TRASLACIÓN Z
203.                 Theta1 = 0; sx1 = 1;    sy1 = 1;    sz1 = 1,    tx1 = 0,
    ty1 = 0,    tz1 = 0;
204.                 Theta2 = 0; sx2 = 1;    sy2 = 1;    sz2 = 1,    tx2 = 0,
    ty2 = 0,    tz2 = 0;
205.                 Theta3 = 0; sx3 = 1;    sy3 = 1;    sz3 = 1,    tx3 = 0,
    ty3 = 0,    tz3 = 0.1;
206.                 break;
207.             default:
208.                 Theta1 = 0; sx1 = 1;    sy1 = 1;    sz1 = 1,    tx1 = 0,
    ty1 = 0,    tz1 = 0;
209.                 Theta2 = 0; sx2 = 1;    sy2 = 1;    sz2 = 1,    tx2 = 0,
    ty2 = 0,    tz2 = 0;
210.                 Theta3 = 0; sx3 = 1;    sy3 = 1;    sz3 = 1,    tx3 = 0,
    ty3 = 0,    tz3 = 0;
211.                 break;
212.         }
213.         glutPostRedisplay();
214.     }
215.     //-------------------------------------------------------------------
    --------
216.
217.     void display()
218.     {
219.         glClear(GL_COLOR_BUFFER_BIT);
220.         drawAxis();
221.         glColor3f(1.0f,1.0f,1.0f);
222.         //se rota la piramide Theta grados con respecto al eje de
    rotacion
223.         //a una distancia definida por el usuario
224.         ///S.RotacionPiramide('Y',Theta,0,0);
225.         ///S.RotacionPiramide(Theta,P1,P2);
226.         ///S.ImprimePiramide();
227.         S.ShowStage();
228.         glutSwapBuffers();
229.     }
230.
231.     void init()
232.     {
233.         glMatrixMode(GL_PROJECTION);
234.         glLoadIdentity();
235.         gluPerspective(FOVY, (GLfloat)WIDTH/HEIGTH, ZNEAR, ZFAR);
236.         glMatrixMode(GL_MODELVIEW);
237.         glLoadIdentity();
238.
    gluLookAt(EYE_X,EYE_Y,EYE_Z,CENTER_X,CENTER_Y,CENTER_Z,UP_X,UP_Y,UP_Z);
239.         glClearColor(0,0,0,0);
240.         ///RESET DE VALORES
241.         Theta1=0;Theta2=0; Theta3=0;
242.
```

```
243.          sx1 = 1; sy1 = 1; sz1 = 1;
244.          sx2 = 1; sy2 = 1; sz2 = 1;
245.          sx3 = 1; sy3 = 1; sz3 = 1;
246.
247.          tx1 = 0,ty1 = 0,tz1 = 0;
248.          tx2 = 0,ty2 = 0,tz2 = 0;
249.          tx3 = 0,ty3 = 0,tz3 = 0;
250.
251.      }
252.
253.      int main(int argc, char **argv)
254.      {
255.          glutInit(&argc, argv);
256.          glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
257.          glutInitWindowPosition(50, 50);
258.          glutInitWindowSize(WIDTH, HEIGTH);
259.          glutCreateWindow("Matriz de Modelado Global | Jesus Huerta
    Aguilar");
260.          init();
261.          //Op3D.LoadIdentity(Op3D.A);
262.          glutDisplayFunc(display);
263.          //glutIdleFunc(idle);
264.          glutKeyboardFunc(keys);
265.          glutReshapeFunc(reshape);
266.          glutMainLoop();
267.          return 0;
268.      }
```

3D_BIB.CPP

```
1. #include "3D_bib.h"
2. #include <cmath>
3.
4.
5.
6. #include <iostream>
7. using namespace std;
8.
9. Operaciones3D::Operaciones3D()
10.      {
11.              //Variables para operaciones trigonometricas
12.              pi = 3.14159265359;
13.              LoadIdentity(A);
14.      }
15.
16.
17.      //recordar que (pi/180 = r/g) donde "r" son radianes y "g"
   grados
18.      //se aplica la formula r
19.      float Operaciones3D::RadToDeg(float r)
20.      {
21.            return ((180*r)/pi);
22.      }
23.
24.      float Operaciones3D::DegToRad(float g)
25.      {
26.            return ((g*pi)/180);
27.      }
28.
29.      void Operaciones3D::LoadIdentity(float M[][4])
30.      {
31.        int i,j;
32.        for(i=0;i<4;i++)
33.          for(j=0;j<4;j++)
34.            if(i==j)
35.               M[i][j]=1;
36.            else
37.               M[i][j]=0;
38.       }
39.
40.      void Operaciones3D::translate(float x, float y, float z)
41.      {
42.        LoadIdentity(T);
43.        T[0][3]=x;
44.        T[1][3]=y;
45.        T[2][3]=z;
46.       }
47.
48.      void Operaciones3D::rotateX(float deg)
49.      {
50.        LoadIdentity(R);
51.        R[1][1] = cos(deg);
52.        R[1][2] = -1*sin(deg);
53.        R[2][1] = sin(deg);
54.        R[2][2] = cos(deg);
```

```
55.       }
56.
57.      void Operaciones3D::rotateY(float deg)
58.      {
59.        LoadIdentity(R);
60.        R[0][0] = cos(deg);
61.        R[0][2] = sin(deg);
62.        R[2][0] = -1*sin(deg);
63.        R[2][2] = cos(deg);
64.       }
65.
66.      void Operaciones3D::rotateZ(float deg)
67.      {
68.        LoadIdentity(R);
69.        R[0][0] = cos(deg);
70.        R[0][1] = -1*sin(deg);
71.        R[1][0] = sin(deg);
72.        R[1][1] = cos(deg);
73.       }
74.
75.      void Operaciones3D::MultM(float M1[][4], float M2[][4], float
   Res[][4])
76.      {
77.        float tmp[4][4];
78.        int i,j,k;
79.        for(i=0; i<4;i++)
80.           for(j=0;j<4;j++){
81.              tmp[i][j]=0;
82.              for(k=0;k<4;k++)
83.                 tmp[i][j]+=M1[i][k]*M2[k][j];
84.           }
85.        for(i=0;i<4;i++)
86.           for(j=0;j<4;j++)
87.              Res[i][j] = tmp[i][j];
88.      }
89.
90.      //multiplica la matriz m por el punto p y regresa el
   resultado en el punto p
91.      void Operaciones3D::MatPoint(float m[][4], float p[3])
92.      {
93.        float tmp[4];
94.        int i,j;
95.        for(i=0; i<3; i++)
96.          { tmp[i] = p[i];
97.            p[i] = 0;
98.          }
99.        tmp[3]=1;
100.       for(i=0;i<3;i++)
101.         for(j=0;j<4;j++)
102.             p[i] += m[i][j]*tmp[j];
103.     }
104.
105.     //multiplica la matriz m por cada punto del objeto definido
   por la matriz p de size x 3
106.     void Operaciones3D::MatObject(float m[][4], int size, float
   p[][3])
107.     {
```

```
108.          int i;
109.          for(i=0; i<size; i++)
110.            MatPoint(m,p[i]);
111.     }
112.
113.
114.     void Operaciones3D::Escalado(float sx,float sy, float sz)
115.     {
116.       LoadIdentity(E);
117.       E[0][0]=sx;
118.       E[1][1]=sy;
119.       E[2][2]=sz;
120.     }
121.
122.
123.     void Operaciones3D::EscaladoGeneral(float sx,float sy, float
   sz,float distA,float distB){
124.          translate(0,-distA,-distB);
125.          Escalado(sx,sy,sz);
126.          MultM(T,E,A);
127.          translate(0,distA,distB);
128.          MultM(A,T,A);
129.     }
130.
131.
132.     //rotacion paralela a uno de los ejes
133.     //theta: angulo de rotacion;
134.     //distA,distB: vector (distA,distB) que separa al eje de
   rotacion del objeto
135.     //con respecto a uno de los ejes del sistema cartesiano. Si
   el eje es:
136.     //X: (distA,distB) es el vector (0,distA,distB)
137.     //Y: (distA,distB) es el vector (distA,0,distB)
138.     //Z: (distA,distB) es el vector (distA,distB,0)
139.     void Operaciones3D::RotacionParalela(char eje, float theta,
   float distA, float distB)
140.     {
141.          switch(eje){
142.             case 'X'://rotacion paralela en "X"
143.                 translate(0,-distA,-distB);
144.                 rotateX(DegToRad(theta));
145.                 MultM(T,R,A);
146.                 translate(0,distA,distB);
147.                 MultM(A,T,A);
148.                 break;
149.             case 'Y'://rotacion paralela en "Y"
150.                 translate(-distA,0,-distB);
151.                 MultM(A,T,A);
152.                 rotateY(DegToRad(theta));
153.                 MultM(A,R,A);
154.                 translate(distA,0,distB);
155.                 MultM(A,T,A);
156.                 break;
157.             case 'Z'://rotacion paralela en "Z"
158.                 translate(-distA,-distB,0);
159.                 rotateZ(DegToRad(theta));
160.                 MultM(T,R,A);
```

```
161.                 translate(distA,distB,0);
162.                 MultM(A,T,A);
163.                 break;
164.          }
165.     }
166.
167.     void Operaciones3D::RotacionLibre(float theta, float p1[3],
   float p2[3]){
168.         float V,a,b,c,d;
169.
170.         /// [1] - CALCULO DEL VECTOR UNITARIO
171.         //Consideramos el eje Z
172.         //Calculo |V|
173.         V = sqrt(pow(p2[0]-p1[0],2) + pow(p2[1]-p1[1],2) +
   pow(p2[2]-p1[2],2));
174.         if(V < 0){
175.             V = -1*V;
176.         }
177.
178.         //Calculo de a,b,c
179.         a = (p2[0]-p1[0]) / V;
180.         b = (p2[1]-p1[1]) / V;
181.         c = (p2[2]-p1[2]) / V;
182.         //Calculo de d (hipotenusa)
183.         d = sqrt(pow(b,2) + pow(c,2));
184.
185.         //Caso para el eje X
186.         if(d == 0){
187.                 translate(-p1[0],-p1[1],-p1[2]);
188.                 rotateX(DegToRad(theta));
189.                 MultM(R,T,A);
190.                 translate(p1[0],p1[1],p1[2]);
191.                 MultM(T,A,A);
192.         }
193.         else{// PARA LOS EJES Y y Z
194.             /// [2] TRASLACION
195.             translate(-p1[0],-p1[1],-p1[2]);
196.             MultM(A,T,A);
197.             /// [3] CALCULO: RX(ALPHA)
198.             LoadIdentity(R);
199.             R[1][1] = c / d;
200.             R[1][2] = -b / d;
201.             R[2][1] = b / d;
202.             R[2][2] = c / d;
203.             //Primera mult para A
204.             MultM(A,R,A);
205.
206.             /// [4] CALCULO: RY(BETA)
207.             LoadIdentity(R);
208.             R[0][0] = d;
209.             R[0][2] = a;
210.             R[2][0] = -a;
211.             R[2][2] = d;
212.             //Segunda mult para A
213.             MultM(A,R,A);
214.
215.             /// [5] CALCULO: RZ(THETA)
```

```
216.            LoadIdentity(R);
217.            R[0][0] = cos(DegToRad(theta));
218.            R[0][1] = -sin(DegToRad(theta));
219.            R[1][0] = sin(DegToRad(theta));
220.            R[1][1] = cos(DegToRad(theta));
221.            //Tercera mult para A
222.            MultM(A,R,A);
223.
224.            /// [6] CALCULO: RY-1(BETA)
225.            LoadIdentity(R);
226.            R[0][0] = d;
227.            R[0][2] = -a;
228.            R[2][0] = a;
229.            R[2][2] = d;
230.            //Cuarta mult para A
231.            MultM(A,R,A);
232.
233.            /// [7] CALCULO: RX-1(ALPHA)
234.            LoadIdentity(R);
235.            R[1][1] = c / d;
236.            R[1][2] = b / d;
237.            R[2][1] = -b / d;
238.            R[2][2] = c / d;
239.            //Quinta mult para A
240.            MultM(A,R,A);
241.
242.            /// [8] TRASLACION INVERSA
243.            translate(p1[0],p1[1],p1[2]);
244.            MultM(A,T,A);
245.        }
246.    }
247.
248.
249.    ///OPERACIONES DE PILA
250.    void Operaciones3D::Push(float mmodel[][4])
251.    {
252.        Matriz *objM = new Matriz();
253.        for(int i = 0;i < 4;i++){
254.            for(int j = 0;j < 4;j++){
255.                objM->M[i][j] = mmodel[i][j];
256.            }
257.        }
258.        pila.push(objM);
259.    }
260.
261.    void Operaciones3D::Pop(float mmodel[][4])
262.    {
263.        Matriz *temp;
264.        temp = pila.top();
265.        for(int i = 0;i < 4;i++){
266.            for(int j = 0;j < 4;j++){
267.                mmodel[i][j] = temp->M[i][j];
268.            }
269.        }
270.        delete temp;
271.        pila.pop();
272.    }
```

3D_BIB.H

```
1.
2. //#include <GL/glut.h>
3. #pragma once
4. #include <stdlib.h>
5. #include <stdio.h>
6. #include <math.h>
7. #include <time.h>
8. #include <stack>
9.
10.      class Matriz{
11.      public:
12.          float M[4][4];
13.      };
14.
15.      class Operaciones3D{
16.            public:
17.              //Variables para matrices de rotacion y traslación
18.              float T[4][4], R[4][4], E[4][4], A[4][4];
19.              //Variables para operaciones trigonometricas
20.              float pi;
21.              std::stack<Matriz *> pila;
22.              Operaciones3D();
23.
24.              //Transformacion de grados a radianes
25.              //recordar que (pi/180 = r/g) donde "r" son radianes
   y "g" grados
26.              //se aplica la formula r
27.              float RadToDeg(float r);
28.              float DegToRad(float g);
29.              //Funcion para cargar la matriz identidad en la
   matriz que se recibe como parametro
30.              void LoadIdentity(float M[][4]);
31.              //Funcion que carga el vector de traslacion en la
   matriz T
32.              void translate(float x, float y, float z);
33.              //Funcion que define la matriz de rotacion con
   rspecto al eje X
34.              void rotateX(float deg);
35.              //Funcion que define la matriz de rotacion con
   rspecto al eje Y
36.              void rotateY(float deg);
37.              //Funcion que define la matriz de rotacion con
   rspecto al eje Z
38.              void rotateZ(float deg);
39.              //Funcion que multiplica la matriz M1 con la matriz
   M2, donde el resultado es Res
40.              void MultM(float M1[][4], float M2[][4], float
   Res[][4]);
41.              //multiplica la matriz m por el punto p y regresa el
   resultado en el punto p
42.              void MatPoint(float m[][4], float p[3]);
43.              //multiplica la matriz m por cada punto del objeto
   definido por la matriz p de size x 3
44.              void MatObject(float m[][4], int size, float p[][3]);
45.
```

```
46.             //Escalado
47.             void Escalado(float ,float ,float);
48.             void EscaladoGeneral(float ,float , float ,float
   ,float );
49.
50.
51.             //Rotacion paralela a uno de los ejes
52.             //eje: eje de referencia
53.             //theta: angulo de rotacion
54.             //(distA,distB): distancia del eje de rotacion al eje
   refrencia del sistema
55.             //La matriz resultante queda almacenada en la matriz
   A
56.             void RotacionParalela(char eje, float theta, float
   distA, float distB);
57.             //Rotacion libre a partir del eje de rotacion
   definido por los puntos
58.             //p1 y p2 y theta grados en sentido contrario a las
   manecillas del reloj
59.             //La matriz resultante queda almacenada en la matriz
   A
60.             void RotacionLibre(float theta, float p1[3], float
   p2[3]);
61.
62.             //OPERACIONES DE PILA
63.             void Push(float [][4]);
64.             void Pop(float [][4]);
65.     };
```

PIRAMIDE.CPP

```
1. #include "Piramide.h"
2.
3.
4.
5. extern float P1[3];
6. extern float P2[3];
7. extern float points[8][3];
8. extern float points2[8][3];
9. extern float points3[8][3];
10.      extern float Theta1,sx1,sy1,sz1;
11.      extern float Theta2,sx2,sy2,sz2;
12.      extern float Theta3,sx3,sy3,sz3;
13.
14.
15.
16.      Piramide::Piramide(Operaciones3D *data)
17.      {
18.          Op3D = data;
19.      }
20.
21.      Piramide::~Piramide()
22.      {
23.          //dtor
24.      }
25.
26.
27.      ////////////////////////////////////////////////////////////
   ////////////////
28.      //funciones de objetos
29.      ////////////////////////////////////////////////////////////
   ////////////////
30.      float Piramide::Norma(float p1[7], float p2[7])
31.      {
32.            float n=0;
33.            int i;
34.            for(i=0;i<3;i++)
35.              n += pow(p2[i]-p1[i],2);
36.            return(sqrt(n));
37.      }
38.
39.      void Piramide::ImprimeMallaPiramide(int k,float pin[][3])
40.      {
41.           int i,j;
42.           float U[3],d,norma;
43.           for(i=0;i<7;i++)
44.           {
45.              norma = Norma(pin[i],pin[i+1]);
46.              d = norma/(float)k;
47.              U[0] = (pin[i+1][0]-pin[i][0])/norma;
48.              U[1] = (pin[i+1][1]-pin[i][1])/norma;
49.              U[2] = (pin[i+1][2]-pin[i][2])/norma;
50.              for(j = 1; j < k; j++)
51.              {
52.                 glBegin(GL_LINES);
53.                   glVertex3f(pin[i+4][0],pin[i+4][1],pin[i+4][2]);
```

```
54.
   glVertex3f(pin[i][0]+U[0]*d*j,pin[i][1]+U[1]*d*j,pin[i][2]+U[2]*d*j
   );
55.                 glEnd();
56.              }
57.           }
58.           norma = Norma(pin[i],pin[0]);
59.           d = norma /(float)k;
60.           U[0] = (pin[0][0]-pin[i][0])/norma;
61.           U[1] = (pin[0][1]-pin[i][1])/norma;
62.           U[2] = (pin[0][2]-pin[i][2])/norma;
63.           for(j = 1; j < k; j++)
64.              {
65.                 glBegin(GL_LINES);
66.                   glVertex3f(pin[i+4][0],pin[i+4][1],pin[i+4][2]);
67.
   glVertex3f(pin[i][0]+U[0]*d*j,pin[i][1]+U[1]*d*j,pin[i][2]+U[2]*d*j
   );
68.                 glEnd();
69.              }
70.
71.       }
72.
73.       void Piramide::ImprimePiramide(float pin[][3])
74.       {   int i;
75.
76.           glBegin(GL_LINE_LOOP);
77.             for(i=0;i<4;i++)
78.               glVertex3f(pin[i][0],pin[i][1],pin[i][2]);
79.           glEnd();
80.
81.
82.           glBegin(GL_LINE_LOOP);
83.             for(i=4;i<8;i++)
84.               glVertex3f(pin[i][0],pin[i][1],pin[i][2]);
85.           glEnd();
86.
87.
88.           glBegin(GL_LINES);
89.             for(i=0;i<4;i++){
90.               glVertex3f(pin[i+4][0],pin[i+4][1],pin[i+4][2]);
91.               glVertex3f(pin[i][0],pin[i][1],pin[i][2]);
92.               }
93.           glEnd();
94.           //ImprimeMallaPiramide(20);
95.           //glColor3f(1,1,1);
96.        }
97.
98.
99.       //Rotacion paralela
100.      //rota a la piramide theta grados, donde el eje de rotacion
   se encuentra
101.      //a una distancia distA-distB del eje seleccionado (ejeXYZ)
102.      void Piramide::RotacionPiramide(char ejeXYZ, float theta,
   float distA, float distB,float pin[][3])
103.      {
104.           //se prepara la matriz de operaciones A: T^(-1)R(T)
```

```
105.          Op3D->RotacionParalela(ejeXYZ,theta,distA,distB);
106.          //se aplica A a cada punto de la piramide
107.          Op3D->MatObject(Op3D->A,5,pin);
108.     }
109.
110.     //Rotacion libre
111.     void Piramide::RotacionPiramide(float theta, float p1[3],
   float p2[3],float pin[][3])
112.     {
113.          //se imprime el eje de rotacion
114.          //glColor3f(1.0,1.0,1.0);
115.          glBegin(GL_LINES);
116.            glVertex3f(p1[0],p1[1],p1[2]);
117.            glVertex3f(p2[0],p2[1],p2[2]);
118.          glEnd();
119.         //Se prepara la matriz de operaciones A
120.         Op3D->RotacionLibre(theta,p1,p2);
121.         //se aplica A a cada punto de la piramide
122.         Op3D->MatObject(Op3D->A,5,pin);
123.     }
```

PIRAMIDE.H

```
1. #include <stdlib.h>
2. #include <stdio.h>
3. #include <math.h>
4. #include <time.h>
5. #include <GL/glut.h>
6. #include "3D_bib.h"
7.
8. #ifndef PIRAMIDE_H
9. #define PIRAMIDE_H
10.
11.
12.      class Piramide
13.      {
14.          public:
15.              Piramide(Operaciones3D *);
16.              ~Piramide();
17.              float Norma(float [], float []);
18.              void ImprimeMallaPiramide(int,float [][3]);
19.              void ImprimePiramide(float [][3]);
20.              void RotacionPiramide(char , float, float,
    float,float [][3]);
21.              void RotacionPiramide(float , float [], float
    [],float [][3]);
22.
23.          private:
24.              Operaciones3D *Op3D;
25.      };
26.
27.      #endif // PIRAMIDE_H
28.
```

STAGE.CPP

```
1. #include "Stage.h"
2. #include "Piramide.h"
3.
4. extern float P1[3];
5. extern float P2[3];
6. extern float points[8][3];
7. extern float points2[8][3];
8. extern float points3[8][3];
9. extern float Theta1,sx1,sy1,sz1,tx1,ty1,tz1;
10.      extern float Theta2,sx2,sy2,sz2,tx2,ty2,tz2;
11.      extern float Theta3,sx3,sy3,sz3,tx3,ty3,tz3;
12.      extern Operaciones3D Op3D;
13.
14.      Piramide Cubo(&Op3D),Cubo2(&Op3D),Cubo3(&Op3D);
15.
16.      Stage::Stage(Operaciones3D *data)
17.      {
18.          Op3D = data;
19.      }
20.
21.      Stage::~Stage()
22.      {
23.          //dtor
24.      }
25.
26.      void Stage::ShowStage(){
27.
28.          Op3D->Push(Op3D->A);
29.          ///CUBO N°1
30.          ///COLOR: ROJO
31.          glColor3f(1,0,0);
32.
33.          Op3D->RotacionParalela('Y',Theta1,0,0);
34.
35.          Op3D->translate(tx1,ty1,tz1);
36.          Op3D->MultM(Op3D->A,Op3D->T,Op3D->A);
37.
38.          Op3D->Escalado(sx1,sy1,sz1);
39.          Op3D->MultM(Op3D->A,Op3D->E,Op3D->A);
40.
41.          Op3D->MatObject(Op3D->A,8,points);
42.          Cubo.ImprimePiramide(points);
43.
44.          //Op3D->Push(Op3D->A);
45.          //Op3D->Pop(Op3D->A);
46.
47.          ///CUBO N°2
48.          ///COLOR: VERDE
49.          glColor3f(0,1,0);
50.          Op3D->RotacionParalela('Y',Theta2,-3,3);
51.
52.          Op3D->translate(tx2,ty2,tz2);
53.          Op3D->MultM(Op3D->A,Op3D->T,Op3D->A);
54.
55.          Op3D->Escalado(sx2,sy2,sz2);
```

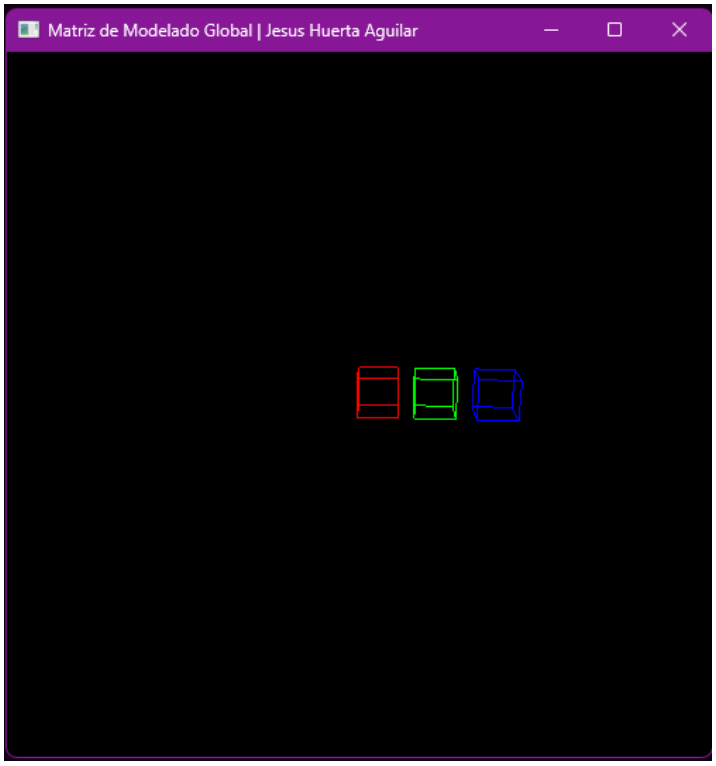
```
56.           Op3D->MultM(Op3D->A,Op3D->E,Op3D->A);
57.
58.           Op3D->MatObject(Op3D->A,8,points2);
59.           Cubo2.ImprimePiramide(points2);
60.
61.
62.
63.           ///CUBO N°3
64.           ///COLOR: AZUL
65.           //O3D.Escalado(sx1,sy1,sz1);
66.           //ImprimePiramide();
67.           glColor3f(0,0,1);
68.
69.           Op3D->RotacionParalela('Y',Theta3,0,0);
70.
71.           Op3D->translate(tx3,ty3,tz3);
72.           Op3D->MultM(Op3D->A,Op3D->T,Op3D->A);
73.
74.           Op3D->Escalado(sx3,sy3,sz3);
75.           Op3D->MultM(Op3D->A,Op3D->E,Op3D->A);
76.
77.           Op3D->MatObject(Op3D->A,8,points3);
78.           Cubo.ImprimePiramide(points3);
79.
80.           Op3D->Pop(Op3D->A);
81.       }
```

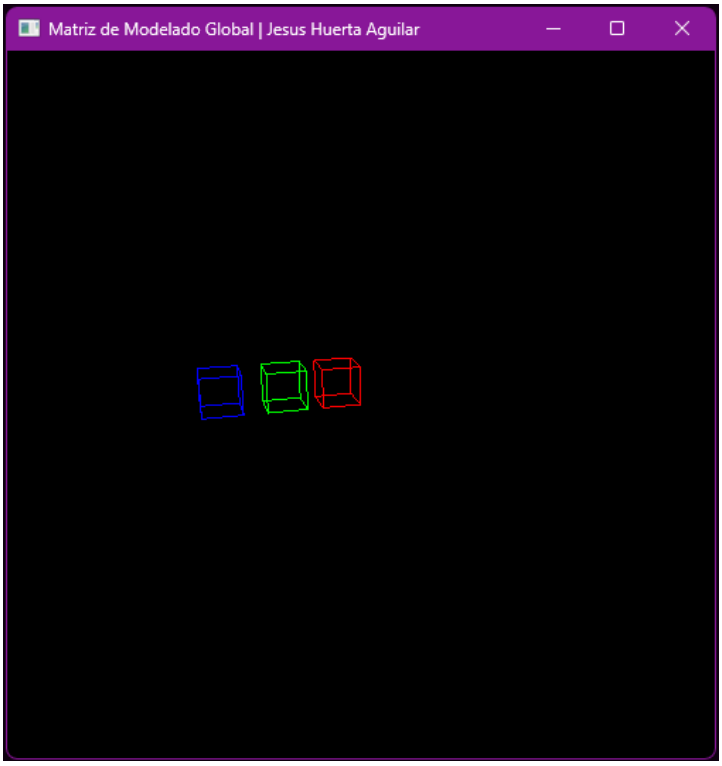
STAGE.H

```
1.  #include <stdlib.h>
2.  #include <stdio.h>
3.  #include <math.h>
4.  #include <time.h>
5.  #include <GL/glut.h>
6.  #include "3D_bib.h"
7.
8.  #ifndef STAGE_H
9.  #define STAGE_H
10.
11.       class Stage
12.       {
13.           public:
14.               Stage(Operaciones3D *);
15.               ~Stage();
16.               void ShowStage();
17.           private:
18.               Operaciones3D *Op3D;
19.       };
20.
21. #endif // STAGE_H
```

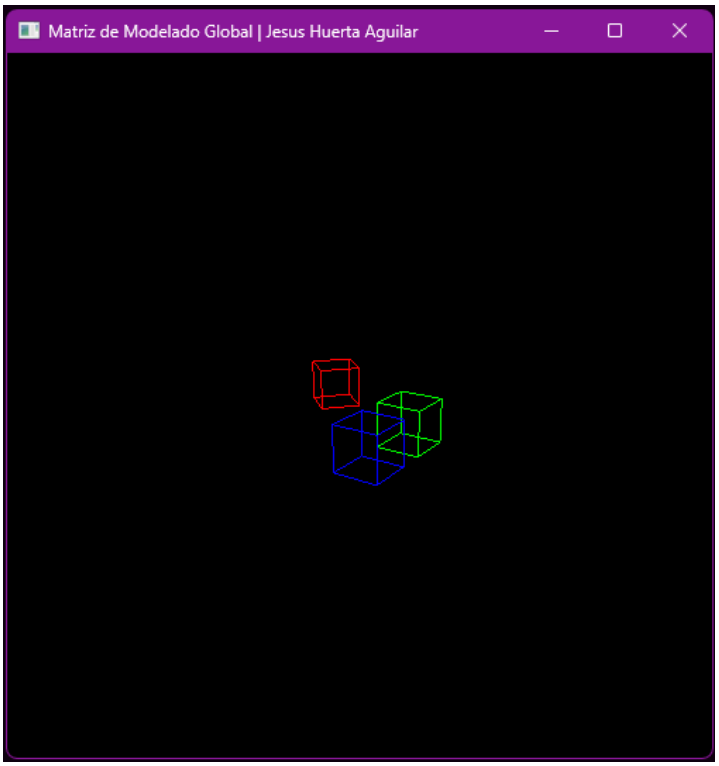
EJECUCIONES

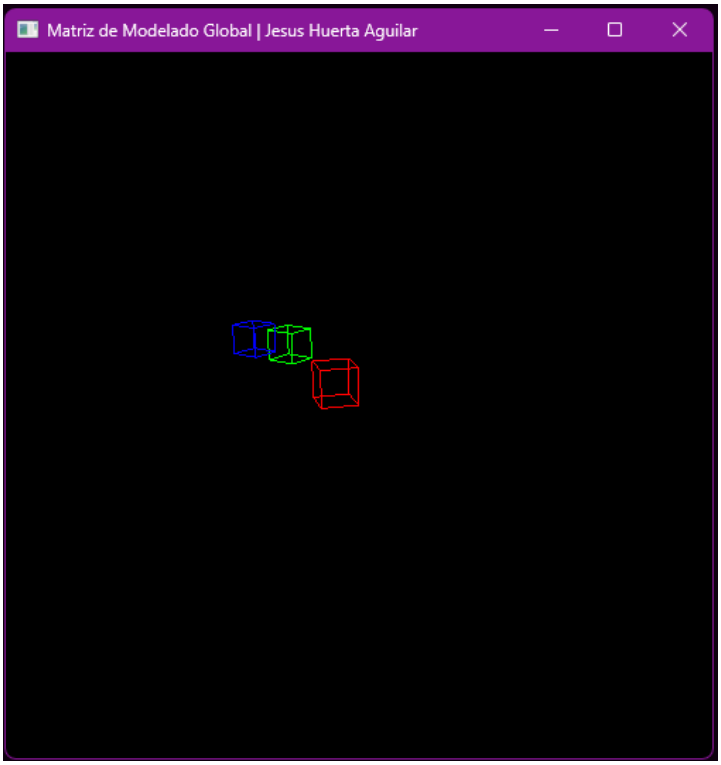
PRECIONANDO “1”:



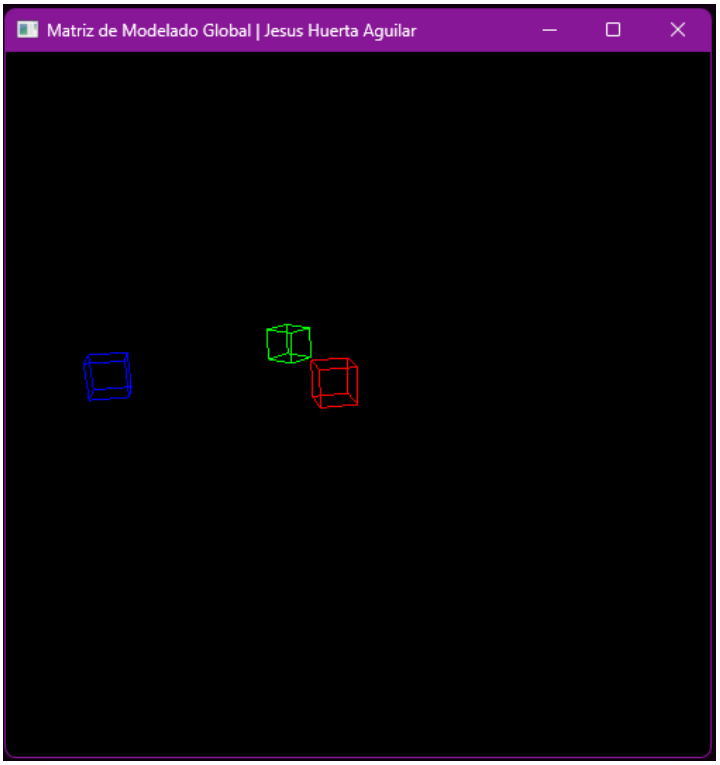


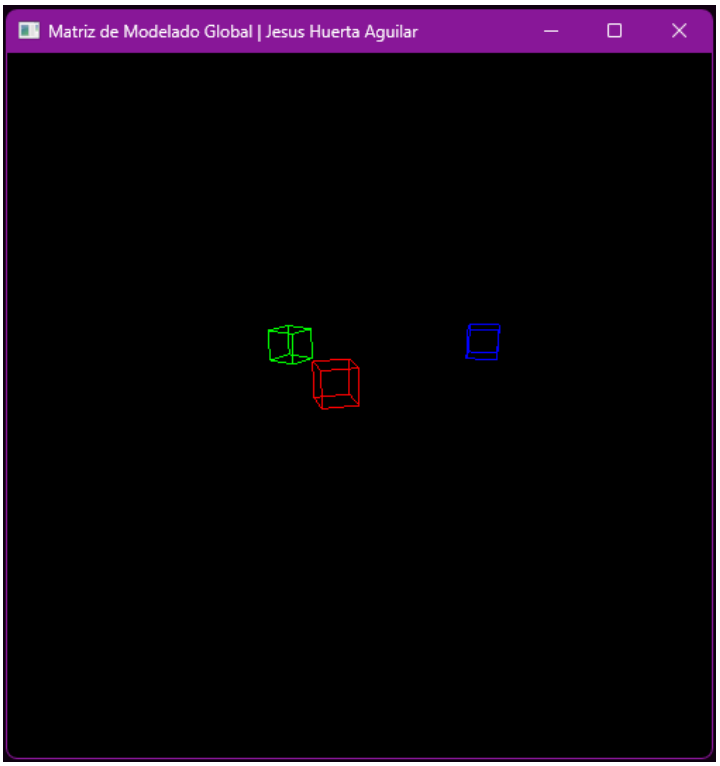
PRECIONANDO “Q”:





PRECIONANDO “A”:





CONCLUSIONES

Las transformaciones de coordenadas se representan por matrices (matrices de dos dimensiones). Para cada uno de los anteriores tipos de transformaciones hay una matriz asociada. Éstas pueden ser especificadas en cualquier momento del programa antes de dibujar la imagen. OpenGL mantiene una pila de matrices de transformación que se han de aplicar sobre cada punto de la escena. Esta es una técnica muy eficiente y útil que exploraremos en futuros artículos. Por el momento vayamos al código fuente, donde se definen algunas de estas transformaciones.

BIBLIOGRAFIA

- Olmos, I. (2022, 6 septiembre). Reunión en «General». sharepoint. Recuperado 9 de octubre de 2022, de https://correobuap.sharepoint.com/sites/Section_202235-CCOS261-10592-001/Shared%20Documents/Forms/AllItems.aspx?FolderCTID=0x01200042EC72C454FA1E45805A79EC6BAB0F21&isAscending=false&sortField=Modified&id=%2Fsites%2FSection_202235-CCOS261-10592-001%2FShared%20Documents%2FGeneral%2FRecordings%2FReunión%20en%20_General_-20221006_070839-Grabación%20de%20la%20reunión%2Emp4&viewid=e5d04394-2fba-4809-9262-9608cc0326d4&parent=%2Fsites%2FSection_202235-CCOS261-10592-001%2FShared%20Documents%2FGeneral%2FRecordings
- Rotación (matemáticas). (s. f.). Wikiwand. Recuperado 10 de octubre de 2022, de https://www.wikiwand.com/es/Rotaci%C3%B3n_(matem%C3%A1ticas)