UNIVERSIDAD NACIONAL JORGE BASADRE GROHMANN

Facultad de Ingeniería

Escuela Profesional de Ingeniería en Informática y Sistemas

Informe de trabajo

NOMBRES Y APELLIDOS:	Códigos:
• Rolando Gutierrez Cutipa	2018-119062
• Yunior Copari Copaja	2018-119009

Docente	:	**Ing Omar Latorre Vilca**
Curso	:	**Diseño asistido por computador**
Grupo	:	**2**
Ciclo	:	**8vo Ciclo**
Año de estudios	:	**4to año**

TACNA - PERÚ

2022

Informe de Trabajo

1. **Objetivos:**
   - Implementar un menú donde el usuario pueda elegir qué forma dibujar cubos y pirámides. Al elegir una de estas opciones, el objeto debe dibujarse en el centro de la pantalla con un tamaño fijo. Dibuja la pirámide con una base cuadrada como en la figura de abajo.

   - Implementar transformaciones geométricas 3D de traslación, escala y rotación en los puntos de los objetos creados. Recuerde implementar la traslación, la escala y la rotación con operaciones matriciales. Para la rotación, no use una matriz para cada dirección, esto es más complicado y requiere más operaciones de matriz. Utilice el concepto de Cuaternion, aplicando la matriz que se presenta a continuación para rotar los puntos.

$$M_R = \begin{bmatrix} x^2(1-c)+c & xy(1-c)-zs & xz(1-c)+ys & 0 \\ yx(1-c)+zs & y^2(1-c)+c & yz(1-c)-xs & 0 \\ zx(1-c)-ys & zy(1-c)+xs & z^2(1-c)+c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Donde $c = cos(\theta)$, $s = sin(\theta)$ y $(x; y; z)$ las coordenadas del vector unitario $u\rightarrow$alrededor del cual debe ocurrir la rotación.

Observación: En lugar de rotar con las teclas direccionales, utilice las letras x, s y z para rotar alrededor de los respectivos ejes. Los sentidos de giro se pueden modificar utilizando, por ejemplo, x para el sentido de las agujas del reloj y s para el sentido contrario a las agujas del reloj. Las transformaciones de escala y traslación deben continuar usando los mismos accesos directos (+ y - para la escala y el clic del mouse para la traslación).

2. **Desarrollo:**

```
#include <stdlib.h>
#include <iostream>
#include <stdio.h>
#include <GL\glut.h>
#include <math.h>

static int window;
static int menu_id;
static int submenu_id;
static int value = 0;
```

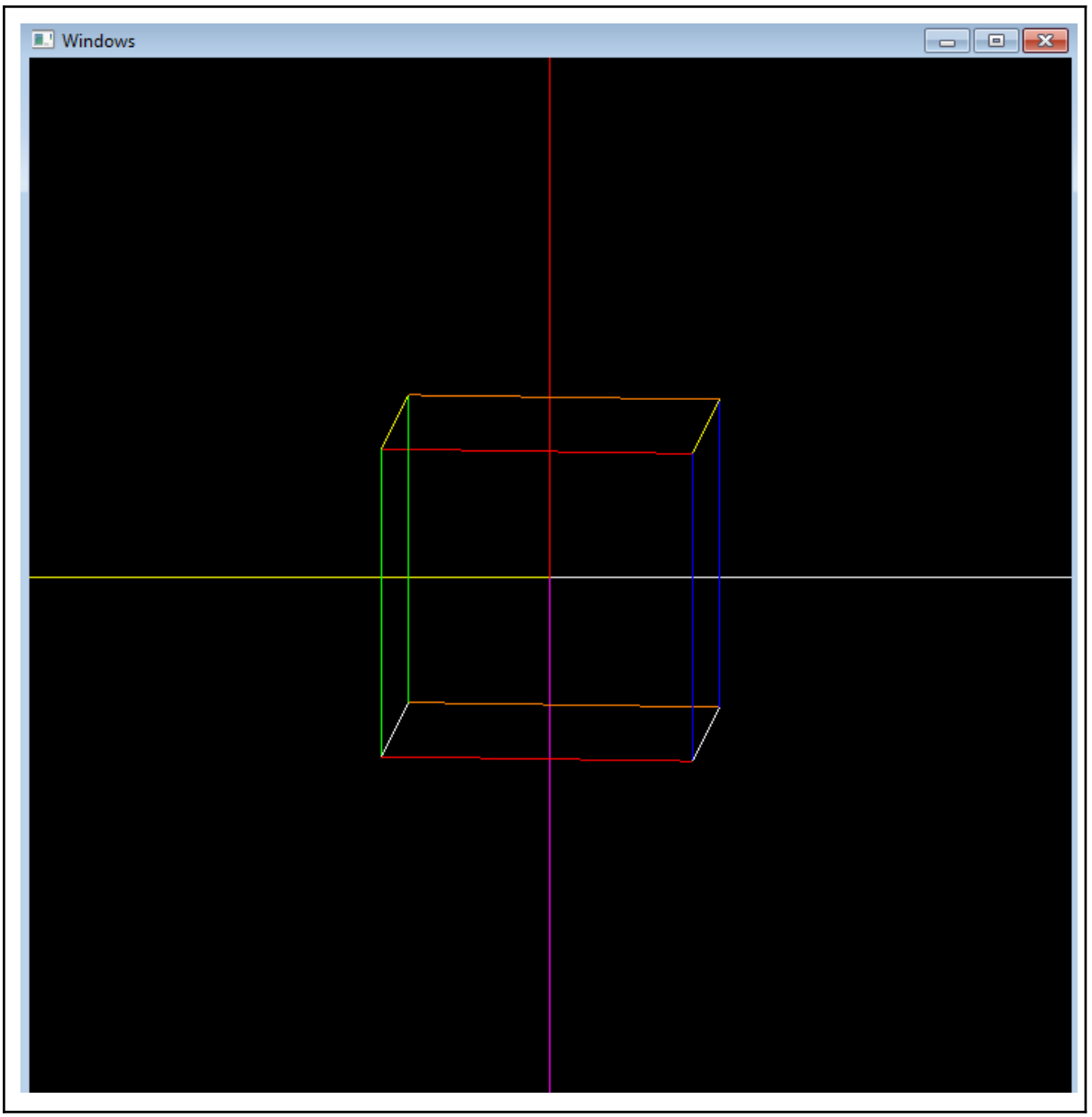
```
void ejes();
void cubo();
void piramide();
void display();
void NormalKeyHandler();
void Mymouse(int boton, int estado, int x, int y);
void val_ini();

double rotate_y=0;
double rotate_x=0;
double escala=1;
double ex=0.0,ey=0.0;

void menu(int num){
  if(num == 0){
    //destruye la ventana especificada.
    glutDestroyWindow(window);
    exit(0);
  }else{
    value = num;
  }
  //marca la ventana actual como que necesita volver a mostrarse.
  glutPostRedisplay();
}
void createMenu(void){
    submenu_id = glutCreateMenu(menu);// crea un nuevo menú emergente.
    glutAddMenuEntry("Cubo", 2);// glutAddMenuEntry agrega una entrada de
menú al final del menú actual .
    glutAddMenuEntry("Priramide", 3);
    glutAddMenuEntry("Solid cubo", 4);

    menu_id = glutCreateMenu(menu);//glutCreateMenu crea un nuevo menú
emergente.
    glutAddSubMenu("Dibujar", submenu_id);//glutAddSubMenu agrega un
activador de submenú en la parte inferior del menú actual .
    glutAddMenuEntry("Clear", 1);// glutAddMenuEntry agrega una entrada
de menú al final del menú actual .
    glutAddMenuEntry("Quit", 0);
    glutAttachMenu(GLUT_RIGHT_BUTTON);// glutAttachMenu adjunta un botón
del mouse para la ventana actual al identificador del menú actual
}
void val_ini(){
    rotate_y=0;
    rotate_x=0;
    escala=1;
    ex=0.0;
    ey=0.0;
}
void ejes(){
    glPushMatrix();
    // Dibujamos un ejes
    glMatrixMode(GL_PROJECTION);
    // Modo proyección
    glLoadIdentity();
    // Cargamos la matriz identidad
    glOrtho(-1.0,1.0,-1.0,1.0,-1.0,1.0);
    // Proyección ortográfica, dentro del cubo señalado
    glMatrixMode(GL_MODELVIEW);
    // Modo de modelado
    glBegin(GL_LINES);
```

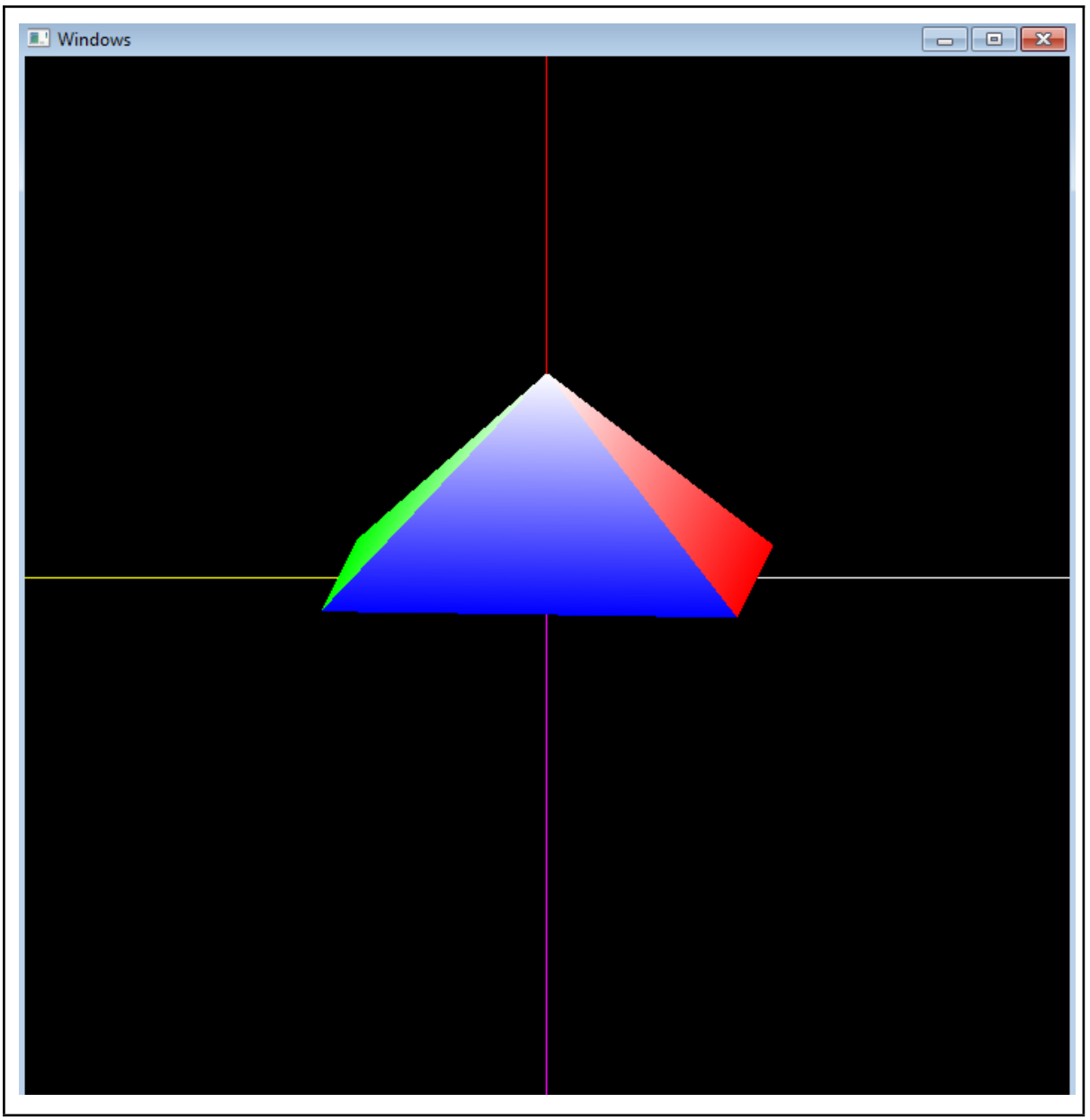
```
    glPushMatrix();
    glColor3f(1.0,1.0,1.0);//blanco
    glVertex3f(0.0,0.0,0.0);//x(0)
    glVertex3f(1.0,0.0,0.0);//x(1)
    glColor3f(1.0,1.0,0.0);//amarillo
    glVertex3f(0.0,0.0,0.0);//x(0)
    glVertex3f(-1.0,0.0,0.0);//x(-1)

    glColor3f(1.0,0.0,0.0);//rojo
    glVertex3f(0.0,0.0,0.0);//y0
    glVertex3f(0.0,1.0,0.0);//y1
    glColor3f(1.0,0.0,1.0);//purpura
    glVertex3f(0.0,0.0,0.0);//y0
    glVertex3f(0.0,-1.0,0.0);//-y1

    glColor3f(0.0,0.0,1.0);//azul
    glVertex3f(0.0,0.0,0.0);//z0
    glVertex3f(0.0,0.0,1.0);//z1
    //glColor3f(0.5,1.0,1.0);//cian
    glColor3f(0.5,0.5,0.5);//violeta
    glVertex3f(0.0,0.0,0.0);//z0
    glVertex3f(0.0,0.0,-1.0);//-z1

    glPopMatrix();
    glEnd();
    // Terminamos de dibujar
    glFlush();
    // Forzamos el dibujado
}
void cubo(){
    //naranja
    glBegin(GL_LINES);
    glColor3f(1.0,0.5,0.0);
    glVertex3f(-0.3,-0.3,-0.3);
    glVertex3f(0.3,-0.3,-0.3);//
    glVertex3f(-0.3,0.3,-0.3);
    glVertex3f(0.3,0.3,-0.3);
    glEnd();

    //ROjo
    glBegin(GL_LINES);
    glColor3f(1.0,0.0,0.0);

    glVertex3f(0.3,-0.3,0.3);
    glVertex3f(-0.3,-0.3,0.3);//
    glVertex3f(0.3,0.3,0.3);
    glVertex3f(-0.3,0.3,0.3);
    glEnd();
    //azul
    glBegin(GL_LINES);
    glColor3f(0.0,0.0,1.0);
    glVertex3f(0.3,-0.3,-0.3);
    glVertex3f(0.3,0.3,-0.3);
    glVertex3f(0.3,0.3,0.3);
    glVertex3f(0.3,-0.3,0.3);
    glEnd();
    //verde
    glBegin(GL_LINES);
    glColor3f(0.0,1.0,0.0);
    glVertex3f(-0.3,-0.3,0.3);
```

```
    glVertex3f(-0.3,0.3,0.3);
    glVertex3f(-0.3,0.3,-0.3);
    glVertex3f(-0.3,-0.3,-0.3);
    glEnd();
    //amarillo
    glBegin(GL_LINES);
    glColor3f(1.0,1.0,0.0);
    glVertex3f(0.3,0.3,0.3);
    glVertex3f(0.3,0.3,-0.3);
    glVertex3f(-0.3,0.3,-0.3);
    glVertex3f(-0.3,0.3,0.3);
    glEnd();
    //blanco
    glBegin(GL_LINES);
    glColor3f(1.0,1.0,1.0);
    glVertex3f(0.3,-0.3,-0.3);
    glVertex3f(0.3,-0.3,0.3);
    glVertex3f(-0.3,-0.3,0.3);
    glVertex3f(-0.3,-0.3,-0.3);
    glEnd();
}
void piramide(){
    //inicio de dibujo de un cuadrado

    //-----base de la piramide--varios colores
    //se usa GL_QUADS para crear un cuadrilátero
    glBegin(GL_QUADS);
    //vértices en 3d
    glColor3f(1,0,0);//rojo
    glVertex3d(-0.4,0,-0.4);//4
    glColor3f(1,1,0);//amarillo
    glVertex3d(-0.4,0,0.4);//3
    glColor3f(0,0,1);//azul
    glVertex3d(0.4,0,0.4);//2
    glColor3f(0,1,0);//verde
    glVertex3d(0.4,0,-0.4);//1
    glEnd();
    //------------------caras de la pirámide
    //para las caras triangulares
    //se usará GL_TRIANGLES
    //--------Cara 4  amarillo con blanco
    glBegin(GL_TRIANGLES);
    //vértices en 3d
    glColor3f(1,1,1);//blanco
    glVertex3d(0,0.4,0);
    glColor3f(1,1,0);
    glVertex3d(-0.4,0,-0.4);//1
    glVertex3d(0.4,0,-0.4);//4
    glEnd();

    //--------Cara 1 rojo con blanco
    glBegin(GL_TRIANGLES);
    //vértices en 3d
    glColor3f(1,1,1);//blanco
    glVertex3d(0,0.4,0);
    glColor3f(1,0,0);//rojo
    glVertex3d(0.4,0,-0.4);
    glVertex3d(0.4,0,0.4);
    glEnd();
    //--------Cara 3 verde con blanco
```

```
    glBegin(GL_TRIANGLES);
    //vértices en 3d
    glColor3f(1,1,1);//blanco
    glVertex3d(0,0.4,0);
    glColor3f(0,1,0);//verde
    glVertex3d(-0.4,0,0.4);
    glVertex3d(-0.4,0,-0.4);
    glEnd();
    //-------Cara 2 azul con blanco
    glBegin(GL_TRIANGLES);
    //vértices en 3d
    glColor3f(1,1,1);//blanco
    glVertex3d(0,0.4,0);
    glColor3f(0,0,1);//azul
    glVertex3d(0.4,0,0.4);
    glVertex3d(-0.4,0,0.4);
    glEnd();
}
void display(void){
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    ejes();
    glTranslatef(ex,ey,0);
    glRotatef(rotate_x, ex+0.09,ey+0.0,0.0);
    glRotatef(rotate_y, ex+0.0,ey+0.09,0.0);
    glScalef(escala, escala, escala);

  if(value == 1){
    glutPostRedisplay();
    val_ini();
  }else if(value == 2){
    glPushMatrix();
    cubo();
    glPopMatrix();
  }else if(value == 3){
    glPushMatrix();
    piramide();
    glPopMatrix();
  }else if(value == 4){
    glPushMatrix();
    glColor3d(1.0, 0.0, 1.0);
    glutWireCube(0.5);
    glPopMatrix();
  }
    glFlush();
    glutSwapBuffers();

}
void NormalKeyHandler (unsigned char key, int x, int y){
    if (key == 'a')
        rotate_y +=10; // do stuff
    else if (key == 'd')
        rotate_y -=5;
    else if (key == 'w')
        rotate_x +=10; // do stuff
    else if (key == 's')
        rotate_x -=5;
    else if (key == '+')
        escala +=0.1;
    else if (key == '-')
```

```
        escala -=0.1;
    glutPostRedisplay();
}
void Mymouse(int boton, int estado, int x, int y){
double a=0;
double b=0;
    if (boton == GLUT_LEFT_BUTTON) // Presione el botón izquierdo del
mouse para comenzar a grabar coordenadas
    {
        a = x-350;
        ex = a/350;
        b = 350-y;
        ey = b/350;
        std::cout<<ex<<" / "<<a<<" // "<<ey<<"\n";
    }
    glutPostRedisplay();
}
int main (int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowPosition (0, 0);
    glutInitWindowSize (700, 700);
    window = glutCreateWindow("Windows");
    createMenu();
    glClearColor(0.0,0.0,0.0,0.0);
    glutDisplayFunc(display);
    glutKeyboardFunc (NormalKeyHandler);
    glutMouseFunc(Mymouse);
    glutMainLoop ();
    return 0;
    glEnable(GL_DEPTH_TEST);
}
```


Windows



3. Conclusiones:

En base a los objetivos trazados inicialmente, se logró implementar. La experiencia de poder implementar, deja muchas enseñanzas, las cuales suman o enriquecen el aprendizaje con todo lo aprendido en las distintas sesiones.