Asignatura	Datos del alumno	Fecha
Informática Gráfica y Visualización	Apellidos: Bocanegra Capera	6 / 02 / 2025
	Nombre: Oscar David	

Actividades

Trabajo: Proyecciones 3D

Crea una ventana OpenGL con cuatro vistas de un mismo cubo, que contendrán respectivamente:

1. Una protección ortogonal.
2. Una protección gabinete.
3. Una proyección perspectiva simétrica.
4. Una proyección perspectiva oblicua.

Entrega

Una vez acabado el trabajo, adjunta:

1. Un ÚNICO fichero `main.c` con la implementación de tu programa.
2. Para realizar la proyección no debemos mover la posición del ojo de la posición por defecto (0,0,0). Es decir, no podemos usar `gluLookAt()` o otra función parecida.
3. Un fichero de texto que contenga:
   a. Un *screenshot* del programa en ejecución.
   b. La respuesta a estas preguntas: ¿Funciona bien tu programa? ¿Qué fallos existen?

Nota: Solo se acepta ficheros editables (.doc, .docx, .odf, .rtf). No se aceptan imágenes escaneadas del ejercicio ni el formato .pdf.

Asignatura	Datos del alumno	Fecha
Informática Gráfica y Visualización	Apellidos: Bocanegra Capera Nombre: Oscar David	6 / 02 / 2025

Solución

```python
import sys
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *
import numpy as np

# Tamaño de la ventana
width, height = 800, 800
```

En este apartado básicamente lo que estoy realizando es la importación de las diferentes librerías que se van a necesitar para el correcto funcionamiento de dicha actividad

- **Sys:** proporciona acceso a variables y funciones que interactúan con el sistema operativo y para este caso en específico lo usamos para inicializar GLUT
- **OpenGL.GL:** Esta librería permite trabajar con la API básica de OpenGL (por ejemplo: glBegin(), glVertex3f(), glColor3f(), glClear(), etc.).
- **OpenGL.GLU:** Con esta librería busco importar funciones de **GLU** (OpenGL Utility Library), que proporciona utilidades para trabajar con transformaciones, perspectivas, etc.
- **OpenGL.GLUT:** Y por último encontramos una librería para la gestión de ventanas, eventos de entrada (teclado, mouse), y el bucle principal de renderizado.
- Y por ultomo tenemos el tamaño de la ventana la cual la defini con una dimensión de 800 px de ancho y 800 pixeles de alto

Después de hacer estas importaciones y de definir el tamaño la ventana lo que se hace es crear la función draw_cube_colored()

Asignatura	Datos del alumno	Fecha
Informática Gráfica y Visualización	Apellidos: Bocanegra Capera Nombre: Oscar David	6 / 02 / 2025

```python
# Dibuja un cubo centrado en el origen con colores en cada cara
def draw_cube_colored():
    """Dibuja un cubo con colores diferentes en cada cara."""
    glBegin(GL_QUADS)
    # Cara frontal (roja)
    glColor3f(1, 0, 0)
    glVertex3f(-0.5, -0.5,  0.5)
    glVertex3f( 0.5, -0.5,  0.5)
    glVertex3f( 0.5,  0.5,  0.5)
    glVertex3f(-0.5,  0.5,  0.5)
    # Cara trasera (verde)
    glColor3f(0, 1, 0)
    glVertex3f(-0.5, -0.5, -0.5)
    glVertex3f(-0.5,  0.5, -0.5)
    glVertex3f( 0.5,  0.5, -0.5)
    glVertex3f( 0.5, -0.5, -0.5)
    # Cara izquierda (azul)
    glColor3f(0, 0, 1)
    glVertex3f(-0.5, -0.5, -0.5)
    glVertex3f(-0.5, -0.5,  0.5)
    glVertex3f(-0.5,  0.5,  0.5)
    glVertex3f(-0.5,  0.5, -0.5)
    # Cara derecha (cian)
    glColor3f(0, 1, 1)
    glVertex3f(0.5, -0.5, -0.5)
    glVertex3f(0.5,  0.5, -0.5)
    glVertex3f(0.5,  0.5,  0.5)
    glVertex3f(0.5, -0.5,  0.5)
    # Cara superior (magenta)
    glColor3f(1, 0, 1)
    glVertex3f(-0.5, 0.5, -0.5)
    glVertex3f(-0.5, 0.5,  0.5)
    glVertex3f( 0.5, 0.5,  0.5)
    glVertex3f( 0.5, 0.5, -0.5)
    # Cara inferior (amarillo)
    glColor3f(1, 1, 0)
    glVertex3f(-0.5, -0.5, -0.5)
    glVertex3f( 0.5, -0.5, -0.5)
    glVertex3f( 0.5, -0.5,  0.5)
    glVertex3f(-0.5, -0.5,  0.5)
    glEnd()
```

Que basicamente de lo que se encarga esta función es de dibujar un cubo 3D centrado en el origen del sistema de coordenadas indicadas en la misma funcion, con colores diferenciados en cada cara para hacer mas clara la visiviliadad de los cuadros.

Ademas de esto el cubo se compone de 6 caras cuadradas, cada una formada por 4 vértices y los colores asignados a las caras permiten distinguirlas claramente al aplicar diferentes proyecciones gráficas.

Para esto se utiliza la primitiva GL_QUADS de OpenGL para el renderizado.

Esta función se invoca desde la rutina de visualización para mostrar el cubo en cada uno de los 4 viewports.

Despues de tener esta funcion clara pasamos a set_ortho() la cual se encarga de definir un volumen de visualización ortográfico con límites personalizados.
Posteriormente a esto lo que hace es que se traslada la escena hacia el eje Z negativo para situar el cubo dentro del volumen de visión.
Así que esta proyección se caracteriza por no producir distorsión de perspectiva, conservando las proporciones originales de los objetos.

```python
# Proyección gabinete
def set_cabinet():
    """Configura una proyección gabinete (oblicua con ángulo de 45° y escala 0.5)."""
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    angle = np.deg2rad(45)
    d = 0.5  # Escala típica para gabinete
    m = np.array([
        [1, 0, d*np.cos(angle), 0],
        [0, 1, d*np.sin(angle), 0],
        [0, 0, 1, 0],
        [0, 0, 0, 1]
    ], dtype=np.float32)
    glMultMatrixf(m.T)
    glOrtho(-2, 2, -2, 2, 1, 10)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
    glTranslatef(0, 0, -5)
```

Después de esto pasamos a la función set_cabinet(), la cual se encarga de configurar la proyección gabinete, la cual es un tipo de proyección oblicua, adicionalmente puedo

decir que esta función se encarga de crear una matriz de transformación la cual se va a proyectar en el eje Z sobre el plano XY con un ángulo de 45 grados como lo indica la variable angle, después de esto se define un volumen ortográfico para así poder completar la configuración de la proyección.

Y con esta proyección pude representar la profundidad de los objetos con una distorsión controlada.

```
def set_perspective():
    """Configura una proyección perspectiva simétrica usando gluPerspective."""
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluPerspective(60, 1, 1, 10)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
    glTranslatef(0, 0, -5)
```

Ahora con esa nueva función set_perspective () lo que busco es configurar una proyección en perspectiva simétrica, así que, esta función genera un volumen de visualización en una forma de pirámide truncada, lo cual hace que la imagen tenga un efecto realista en la perspectiva.

```
# Proyección perspectiva oblicua
def set_oblique():
    """Configura una proyección perspectiva oblicua (oblicua con ángulo de 30° y escala 1.0)."""
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    angle = np.deg2rad(30)
    d = 1.0
    m = np.array([
        [1, 0, d*np.cos(angle), 0],
        [0, 1, d*np.sin(angle), 0],
        [0, 0, 1, 0],
        [0, 0, 0, 1]
    ], dtype=np.float32)
    glMultMatrixf(m.T)
    glOrtho(-2, 2, -2, 2, 1, 10)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
    glTranslatef(0, 0, -5)
```

Con la función set_oblique() lo que busco es generar una proyección oblicua con un ángulo de 30 grados y una escaña de 1.0, para así poder representar la profundidad de

los objetos con un ángulo y una escala personalizable, para así poder generar un efecto visual diferente a la proyección que encontrábamos en gabinete.

```
def display():
    """Función principal de dibujo: divide la ventana en cuatro viewports y dibuja el cubo en cada proyección."""
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glEnable(GL_DEPTH_TEST)

    # Viewport 1: ortogonal
    glViewport(0, height//2, width//2, height//2)
    set_ortho()
    glPushMatrix()
    glRotatef(30, 1, 1, 0)
    draw_cube_colored()
    glPopMatrix()

    # Viewport 2: gabinete
    glViewport(width//2, height//2, width//2, height//2)
    set_cabinet()
    glPushMatrix()
    glRotatef(30, 1, 1, 0)
    draw_cube_colored()
    glPopMatrix()

    # Viewport 3: perspectiva simétrica
    glViewport(0, 0, width//2, height//2)
    set_perspective()
    glPushMatrix()
    glRotatef(30, 1, 1, 0)
    draw_cube_colored()
    glPopMatrix()

    # Viewport 4: perspectiva oblicua
    glViewport(width//2, 0, width//2, height//2)
    set_oblique()
    glPushMatrix()
    glRotatef(30, 1, 1, 0)
    draw_cube_colored()
    glPopMatrix()

    glutSwapBuffers()
```

Ahora con la función display() lo que se busca hacer es el renderizado del programa, en donde se indica que se divida la pantalla en 4 viewports o sectores y se va a dibujar el cubo en cada uno de estos espacios, y como en cada apartado tenemos objeticos diferentes lo que se hace es configurar la proyección necesaria para cada viewport, y finalmente se intercambian los buffers para mostrar la imagen en pantalla.

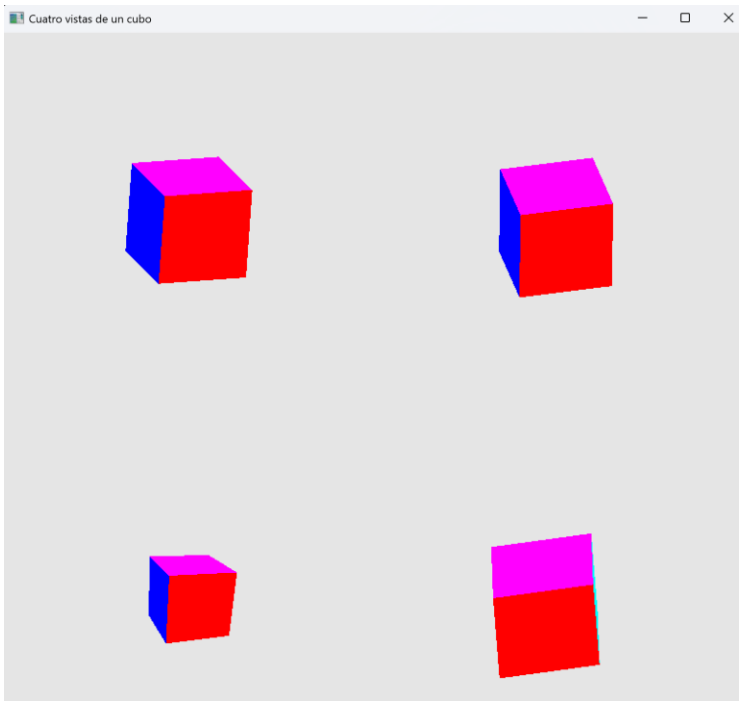
Asignatura	Datos del alumno	Fecha
Informática Gráfica y Visualización	Apellidos: Bocanegra Capera Nombre: Oscar David	6 / 02 / 2025

```python
def main():
    """Inicializa GLUT y ejecuta el bucle principal de la ventana."""
    glutInit(sys.argv)
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH)
    glutInitWindowSize(width, height)
    glutCreateWindow(b"Cuatro vistas de un cubo")
    glutDisplayFunc(display)
    glClearColor(0.9, 0.9, 0.9, 1)
    glutMainLoop()

if __name__ == "__main__":
    main()
```

Y por último encontramos la función main() la cual se encarga de inicializar la librería GLUT, de configurar el modo de visualización, crear la ventana y titulo especificado anteriormente, y de registrar la función display() como la encargada de renderizar el contenido.

Un *screenshot* del programa en ejecución.



Asignatura	Datos del alumno	Fecha
Informática Gráfica y Visualización	Apellidos: Bocanegra Capera	6 / 02 / 2025
	Nombre: Oscar David	

La respuesta a estas preguntas: ¿Funciona bien tu programa? ¿Qué fallos existen?

Para este caso puedo decir que el programa esta funcionando a la perfección, al mostrar las diferentes proyecciones y desde diferentes ángulos, por ende, no presenta ningún inconveniente y corre de la forma esperada