BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

Práctica 5: Implementación de una escena 3D realista con OpenGL.

Graficación

Uriel Lemus Pérez
Matrícula: 202362797

José Emiliano Flores Flores
Matrícula: 202361618

Santiago Silverio Flores
Matrícula: 202367190

Max Ramos

Profesor: LUIS YAEL MÉNDEZ SÁNCHEZ

Fecha: 15/05/2025

Introducción de la práctica

Esta práctica tiene como finalidad aplicar conceptos fundamentales de gráficos por computadora mediante la creación de un visualizador 3D interactivo utilizando Python con las librerías Pygame y PyOpenGL. Se busca que el usuario pueda manipular objetos tridimensionales aplicando transformaciones geométricas, iluminación y texturizado.

Descripción detallada de los objetivos, metodología y resultados

Objetivos

- Implementar un entorno gráfico 3D en tiempo real.
- Aplicar transformaciones básicas: traslación, rotación y escalado.
- Integrar iluminación y texturas para mejorar el realismo.
- Permitir la visualización de múltiples figuras tridimensionales.

Metodología

Se utilizó el lenguaje Python junto a las bibliotecas Pygame y PyOpenGL. Se definió una clase de estado para gestionar las transformaciones, configuración de proyección, luces y texturas. Cada figura cuenta con su propia función de renderizado, empleando primitivas de OpenGL como quads, triángulos y puntos. La interacción del usuario se logra a través del teclado, permitiendo modificar los parámetros de la escena.

Resultados

El resultado fue una aplicación capaz de representar un cubo, pirámide, esfera, cilindro y superelipsoide en un espacio tridimensional. Se logró integrar correctamente iluminación dinámica, texturas y distintos tipos de proyección (ortográfica y perspectiva). El sistema responde de manera fluida a las entradas del usuario.

Explicación técnica de las funciones implementadas

En esta imagen se ilustra cómo la función **aplicar_transformaciones** encadena tres pasos fundamentales:

1. **glTranslatef(x,y,z)**: traslada el sistema de coordenadas actual para posicionar el objeto en X, Y y Z.
2. **glScalef(sx,sy,sz)**: escala el objeto localmente, ajustando su tamaño en cada eje.
3. **glRotatef(ángulo, ex,ey,ez)**: rota el objeto alrededor del eje definido por el vector (ex,ey,ez).

```python
# ------------------- Transformaciones -------------------
def aplicar_transformaciones(estado):
    # Aplica traslación, escala y rotación a la figura
    glTranslatef(*estado.traslacion)
    glScalef(*estado.escala)
    glRotatef(estado.rotacion[0], 1, 0, 0)
    glRotatef(estado.rotacion[1], 0, 1, 0)
    glRotatef(estado.rotacion[2], 0, 0, 1)

def reiniciar_transformaciones():
    glLoadIdentity()
```

(function) rotacion: Any

Figura 1: Funcionamiento de aplicar transformaciones (glTranslatef, glScalef, glRotatef).

Este orden garantiza primero mover al lugar deseado, luego ajustar su tamaño y finalmente orientarlo correctamente en el espacio.

```python
# ------------------- Iluminación -------------------
def configurar_luz(estado):
    glEnable(GL_LIGHTING)
    glEnable(GL_LIGHT0)
    glLightfv(GL_LIGHT0, GL_POSITION, estado.posicion_luz)
    glLightfv(GL_LIGHT0, GL_AMBIENT, estado.luz_ambiental)
    glLightfv(GL_LIGHT0, GL_DIFFUSE, estado.luz_difusa)
    glLightfv(GL_LIGHT0, GL_SPECULAR, estado.luz_especular)

def desactivar_luz():
    glDisable(GL_LIGHTING)
    glDisable(GL_LIGHT0)
```

Figura 2: Configuración e iluminación con glLightfv y glEnable(GL$_{L}IGHTING$).

La imagen muestra la secuencia de llamadas en la función **configurar_luz**:

- **glEnable(GL_LIGHTING)**: activa el cálculo global de iluminación.
- **glEnable(GL_LIGHT0)**: habilita la primera fuente de luz.
- **glLightfv(GL_LIGHT0, GL_POSITION, vec4)**: posiciona la luz en un punto del espacio.
- **glLightfv(GL_LIGHT0, GL_AMBIENT, vec4)**: define la componente ambiental (luz difusa omnidireccional).
- **glLightfv(GL_LIGHT0, GL_DIFFUSE, vec4)**: configura la luz difusa (impacto directo sobre superficies).
- **glLightfv(GL_LIGHT0, GL_SPECULAR, vec4)**: ajusta la luz especular (brillos y reflejos).

Esta configuración permite un modelado realista de materiales y volúmenes, reaccionando a la posición de la fuente y la orientación de la cámara.

```
def dibujar_esfera():
    esfera = gluNewQuadric()
    gluQuadricTexture(esfera, GL_TRUE)
    gluSphere(esfera, 1, 32, 32)
```

Figura 3: Creación de una esfera con gluSphere.

gluSphere: Permite renderizar una esfera suave utilizando subdivisiones en *slices* y *stacks*, generando triángulos que aproximan la forma curvada.

```
# ------------------- Texturas -------------------
id_textura = None

def inicializar_textura(ruta):
    global id_textura
    try:
        superficie = pygame.image.load(ruta)
    except pygame.error:
        print(f'Error al cargar la textura: {ruta}')
        return
    datos = pygame.image.tostring(superficie, 'RGB', True)
    ancho, alto = superficie.get_size()
    id_textura = glGenTextures(1)
    glBindTexture(GL_TEXTURE_2D, id_textura)
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, ancho, alto, 0, GL_RGB, GL_UNSIGNED_BYTE, datos)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)

def activar_textura():
    if id_textura:
        glEnable(GL_TEXTURE_2D)
        glBindTexture(GL_TEXTURE_2D, id_textura)

def desactivar_textura():
    glDisable(GL_TEXTURE_2D)
```

Figura 4: Aplicación de textura con glTexImage2D.

glTexImage2D: Carga datos de imagen en memoria gráfica para mapear la textura en polígonos. Los parámetros de filtrado (GL_LINEAR) suavizan el muestreo.

Otras funciones importantes:

- **glEnable/glDisable**: Activan o desactivan características de OpenGL (iluminación, texturas, pruebas de profundidad, etc.).
- **gluPerspective**: Establece la matriz de proyección en perspectiva, simulando una cámara con campo de visión.
- **glOrtho**: Define proyección ortográfica para vistas sin distorsión de perspectiva.
- **glBegin/glEnd**: Delimita bloques para dibujar primitivas (GL_TRIANGLES, GL_QUADS, etc.).

- **glTexCoord2f** / **glVertex3f**: Asocian coordenadas de textura y posición de vértices.
- **gluNewQuadric, gluCylinder**: Generan cilindros y otras superficies suavizadas.

Flujo de Usuario

- Al iniciar el programa, se muestra el menú en consola.
- El usuario puede presionar del 1 al 5 para elegir una figura tridimensional o el 6 para salir.
- Una vez seleccionada una figura, se accede al modo de visualización 3D.
- En este modo se pueden usar las siguientes teclas:
  - **Flechas**: mover la figura en X e Y.
  - **W**/**A**/**S**/**D**/**Q**/**E**: rotar la figura en los ejes X, Y y Z.
  - **+** / **-**: escalar la figura.
  - **T**: activar o desactivar la textura.
  - **L**: activar o desactivar la iluminación.
  - **P**: alternar entre proyección perspectiva y ortográfica.
  - **J** / **K**: mover la posición de la fuente de luz.
  - **ESC**: regresar al menú.

Capturas de pantalla de la ejecución

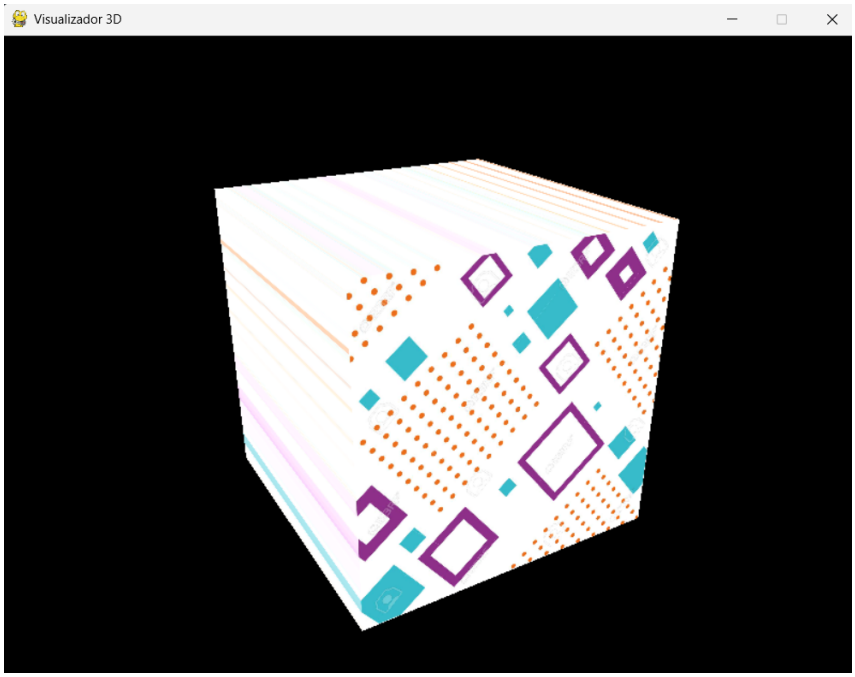

Figura 5: Figura (Cubo) renderizada con textura e iluminación.

Esta captura muestra la aplicación de una textura sobre una figura tridimensional y cómo la iluminación afecta la percepción de profundidad y volumen.

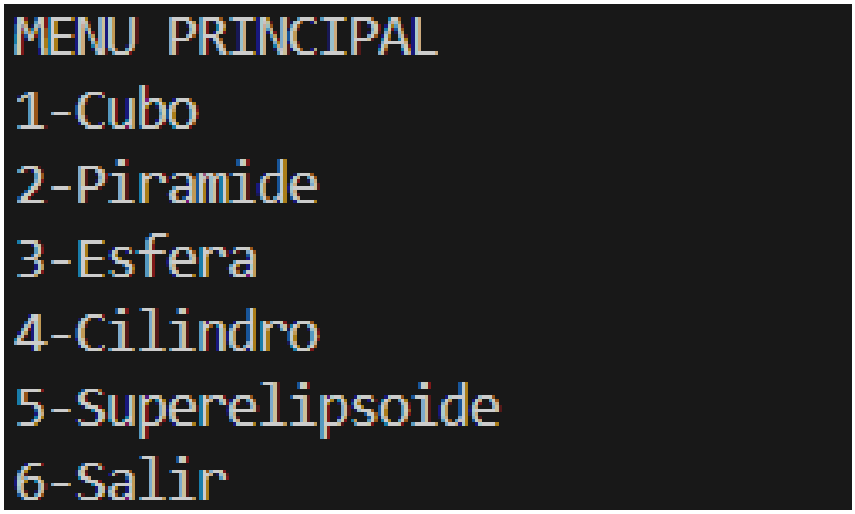

Figura 6: Menú de selección de figuras.

Se observa el menú de inicio en consola con múltiples opciones para seleccionar figuras 3D.

Conclusiones

La implementación del visualizador 3D nos permitió poner en práctica diversos conceptos gráficos, por ejemplo la aplicación de texturas y luces. El uso de PyOpenGL y Pygame nos brindo una excelente introducción a los entornos gráficos interactivos. Se logró un notable nivel de realismo gráfico gracias a las técnicas de iluminación y texturizado aplicadas, permitiendo representar objetos 3D de manera eficiente y visualmente atractiva.