Programación con OPENGL

Renderizado de un octaedro a una esfera en Python

Miguel Ángel Macías Narváez.
Facultad de Ingeniería, Ingeniería en Sistemas
Universidad de Cuenca
Cuenca, Ecuador
mangel.maciasn@ucuenca.edu.ec

Abstract. -. **Rendering is the automatic process of generating a photorealistic or non-photorealistic image from a 2D or 3D model. Rendering is one of the major sub-topics of 3D computer graphics. On the inside, a renderer is a carefully engineered program, and is possible thanks to OpenGL. OpenGL is a standard specification that defines a multi-language and cross-platform API for writing applications that produce 2D and 3D graphics. In this article we elaborate a program that render a octahedron using Python as programming language and PyOpenGL as the core.**

Keywords-component; OpenGL, Pyopengl, octaedro, renderizar, API, vector, frames, setup.

I. INTRODUCCIÓN.

En la era moderna alguna vez hemos consumido una que otra película animada, en estas películas se deben presentar varias imágenes (frames) y unirlas en un programa capaz de hacer una animación de este tipo. La mayoría de los programas de edición de imágenes 3D pueden hacer esto. En este proceso llamado renderizado, se manejan aspectos como las texturas de mapas de bits, texturas de procedimiento, luces, mapeo de relieve y posición relativa a otros objetos [1].

OpenGL es una especificación que describe un conjunto de funciones y comportamiento exacto que deben tener. Consiste en aceptar primitivas como lo son, puntos, líneas, polígonos para convertirlas en píxeles. La API de OpenGL está basada en procedimientos de bajo nivel que requiere que se dicten los pasos exactos para renderizar una escena [2]. En la actualidad OpenGL ha influido en el desarrollo de tarjetas gráficas, llegando al punto en que se encuentra en los drivers de cada una de estas.

Este artículo se enfoca en describir los pasos y metodología utilizada para el desarrollo de un octaedro y eventualmente convertirlo en una esfera. Este proceso se lleva a cabo en un ambiente de desarrollo basado en Python, haciendo uso de la librería PyOpenGL. En la segunda sección de este artículo se definen los conceptos necesarios para entender el uso de las herramientas utilizadas. En la tercera sección se detalla la metodología empleada para el desarrollo del programa. En la cuarta sección se muestran los resultados y se finaliza con conclusiones en la quinta sección.

II. DEFINICIONES.

A. PyOpenGL.

PyOpenGL es el enlace multiplataforma de Python más común para OpenGL y sus APIs relacionadas. El enlace se crea utilizando la biblioteca de tipos estándar y se proporciona bajo una licencia de código abierto de estilo BSD [3].

PyOpenGL soporta:

- OpenGL versión 1 a la 4.
- GLES.
- GLU, GLUT, FreeGLUT.

PyOpenGL funciona en las versiones de Python 2.6, 2.7, 3.3 y mayores.

B. GLUT.

OpenGL Utility Toolkit es una biblioteca de utilidades para programas OpenGL, que principalmente realizan operaciones de E/S a nivel de sistema con el sistema operativo host. Las funciones realizadas incluyen definición de ventas, control de ventanas y monitoreo de entrada de teclado y mouse. Además, se proporcionan rutinas para dibujar una serie de primitivas geométricas, como cubos, esferas, etc [4].

C. PyGame.

PyGame es un motor gráfico. Un motor de gráficos es un sistema diseñado para crear y desarrollar videojuegos. Es un conjunto multiplataforma de módulos de Python diseñados para escribir videojuegos. Incluye gráficos de computadora y bibliotecas de sonido diseñadas para ser utilizadas con el lenguaje de programación Python [5].

III. METODOLOGÍA.

Crear mallas mediante programación abre muchas posibilidades, por ejemplo, crear objetos paramétricos que respondan a configuraciones del mundo real, arte generativo, formas basadas en matemáticas o concluso contenido para juegos. PyOpenGL permite combinar un conjunto completo de modelado y animación con una API potente. En esta sección se explica la matemática necesaria para poder entender el funcionamiento del algoritmo desarrollado en Python usando la librería PyOpenGL.

A. Icosaedro.

Es cualquier poliedro que tiene 20 caras, los cuales son triángulos equiláteros. Es un poliedro regular convexo, tiene 30 aristas y 12 vértices [6]. Una de las formas de construir un icosaedro es considerar sus vértices como las esquinas de tres planos dorados ortogonales. Son planos dorados porque siguen la proporción áurea. Los vértices de estos planos se encuentran en las coordenadas:

$$(0, \pm 1, \pm \varphi), (\pm \varphi, 0, \pm 1), (\pm 1, \pm \varphi, 0)$$

Donde φ representa el valor de la proporción áurea. Estas combinaciones dan como resultado 12 vértices, que crean 20 triángulos equiláteros con 5 triángulos que se encuentran en cada vértice.

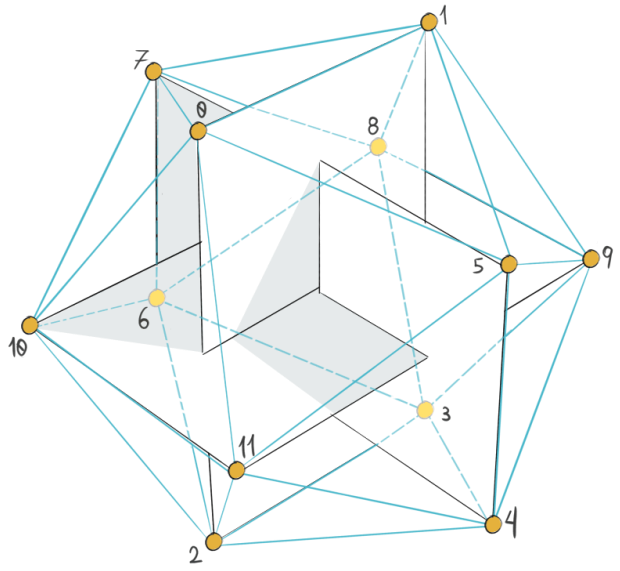

Fig. 1. Diagrama de vértices y aristas de un Icosaedro.

B. Esfera.

La definición de esfera es una superficie 3D cerrada donde cada punto de la esfera se encuentra a la misma distancia de otro punto dado [7]. La ecuación de la esfera se define:

$$x^2 + y^2 + z^2 = r^2$$

Como no se puede dibujar todos los puntos en una esfera, solo se muestrea una cantidad limitada de puntos dividiendo la esfera por sectores (longitud) y pilas (latitud).

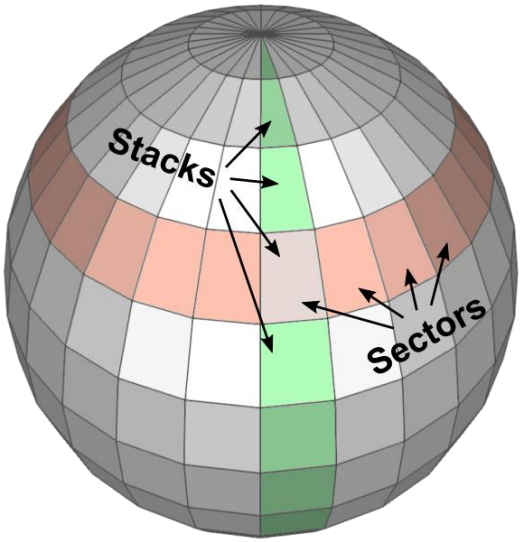

Fig. 2. División de una esfera por sectores y pilas

Un punto cualquiera dentro de una esfera puede ser calculado por sus ecuaciones paramétricas:

$$x = (r \cos \varphi) \cos \theta$$
$$y = (r \cos \varphi) \sin \theta$$
$$z = r \sin \varphi$$

Estas fórmulas y la figura 3 serán esenciales para la codificación del algoritmo que permita renderizar la esfera en Python.

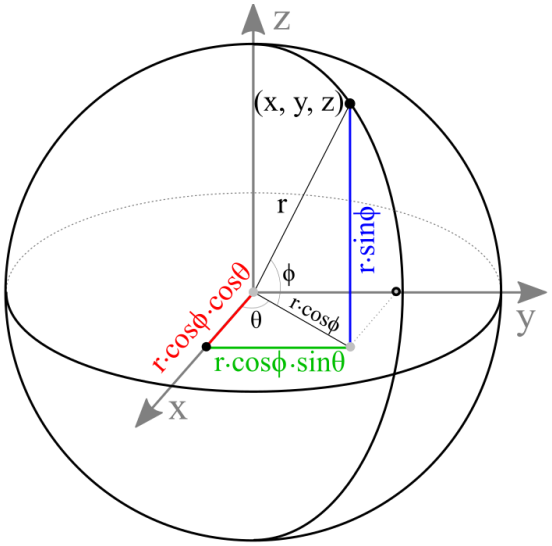

Fig. 3. Cálculo de un punto dentro de una esfera a través de sus ángulos.

El rango de ángulos del sector es de 0 a 360 grados, y los ángulos de la pila son de 90 (arriba) a -90 grados (abajo). El sector y el ángulo de apilamiento se calculan:

$$\theta = 2\pi \frac{SectorPaso}{SectorContador}$$

$$\varphi = \frac{\pi}{2} - \pi \frac{PilaPaso}{PilaContador}$$

IV. EJECUCIÓN DEL ALGORITMO Y RESULTADOS.

El código y los resultados de la ejecución del algoritmo se adjuntan en las carpetas “src” y “media” respectivamente. También se puede acceder a este contenido a través del siguiente enlace: https://github.com/TheWorstOne/Pycosphere.

A. Construcción de la Esfera usando coordenadas polares.

Se necesita crear el setup de la escena y la ventana que contendrá el rendering de la esfera. Para eso primero se importan las librerías necesarias.

```
import pygame
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *
from math import *
```

Fig. 4. Librerías empleadas para el desarrollo del algoritmo en Python.

Se inicializan las variables que definirán las características de la escena, así como también de la esfera.

```
# Last time when sphere was re-displayed
last_time = 0

# Number of latitudes in sphere
lats = 4

# Number of longitudes in sphere
longs = 4

# Direction of light
direction = [0.0, 2.0, -1.0, 1.0]

# Intensity of light
intensity = [0.7, 0.7, 0.7, 1.0]

# Intensity of ambient light
ambient_intensity = [0.3, 0.3, 0.3, 1.0]

# The surface type(Flat or Smooth)
surface = GL_FLAT
```

Fig. 5. Setup de las características de la escena y esfera.

Se comienza con una esfera con longitud y latitud de 4 unidades. El resto de las variables servirán para personalizar aspectos de iluminación y textura.

Por cada unidad de latitud se calculan dos latitudes auxiliares las cuales servirán para generar 2 vértices por iteración. Las variables lat0 y lat1 representan el valor del ángulo phi, el cual se vio en la sección 3, literal B.

```
lat0 = pi * (-0.5 + float(float(i - 1) / float(lats)))
z0 = sin(lat0)
zr0 = cos(lat0)

lat1 = pi * (-0.5 + float(float(i) / float(lats)))
z1 = sin(lat1)
zr1 = cos(lat1)
```

Fig. 6. Cálculo de phi usando latitud.

Una vez calculado el valor de phi, se itera por cada unidad de longitud para calcular los valores de theta y posteriormente generar las coordenadas de dos vértices que serán puntos de la esfera. La variable lng guarda el valor de theta, y los vértices se forman a partir de sus ecuaciones paramétricas, como se explicó en la sección 3, literal B.

```
for j in range(0, longs + 1):
    lng = 2 * pi * float(float(j - 1) / float(longs))
    x = cos(lng)
    y = sin(lng)
    glNormal3f(x * zr0, y * zr0, z0)
    glVertex3f(x * zr0, y * zr0, z0)
    glNormal3f(x * zr1, y * zr1, z1)
    glVertex3f(x * zr1, y * zr1, z1)
```

Fig. 7. Cálculo de theta y vértices usando longitud.

Una vez ejecutado el algoritmo esta es la salida que se muestra:

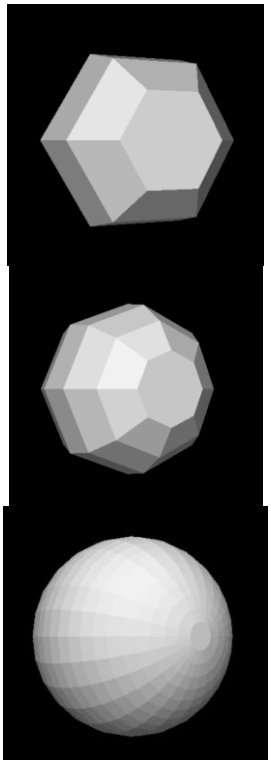

Fig. 8. Frames de la esfera renderizada en Python.

B. Construcción de la esfera usando como base un icosaedro.

Las librerías utilizadas y el setup es el mismo que se muestra en las figuras 4 y 5 respectivamente, del literal A. Se calcula el valor de la proporción áurea y se guarda en la variable phi. Luego se utiliza la variable size para guardar el tamaño que tendrá cada triángulo, este valor es opcional, ya que puede simplemente tomarse como 1, como se vio en la sección 3, literal A.

```
phi = (sqrt(5) + 1) / 2
size = sqrt( 1.0 / ( 1 + phi * phi ) )
verts = [
    ( phi * size,        size,         0.0), #0
    ( phi * size,       -size,         0.0), #1
    (-phi * size,       -size,         0.0), #2
    (-phi * size,        size,         0.0), #3
    (      -size,         0.0,  phi * size), #4
    (       size,         0.0,  phi * size), #5
    (       size,         0.0, -phi * size), #6
    (      -size,         0.0, -phi * size), #7
    (        0.0,  phi * size,        size), #8
    (        0.0,  phi * size,       -size), #9
    (        0.0, -phi * size,       -size), #10
    (        0.0, -phi * size,        size), #11
]
```

Fig. 9. Cálculo de phi y vértices del icosaedro.

Para el cálculo de nuevos vértices del icosaedro simplemente se divide cada triángulo (formado por 3 vértices) en cuatro nuevos triángulos (6 vértices).

```
def Icosphere_Subdiv(subdivs):
    global edges
    for i in range(subdivs):
        edges_subdiv = []

        for tri in edges:
            v1 = middle_point(tri[0], tri[1])
            v2 = middle_point(tri[1], tri[2])
            v3 = middle_point(tri[2], tri[0])

            edges_subdiv.append([tri[0], v1, v3])
            edges_subdiv.append([tri[1], v2, v1])
            edges_subdiv.append([tri[2], v3, v2])
            edges_subdiv.append([v1, v2, v3])

        edges = edges_subdiv
```

Fig. 10. Subdivisión de cada triángulo del icosaedro utilizando el punto medio.

El icosaedro solo divide hasta un máximo de 3 veces, ya que entre más subdivisiones tenga mayor será la cantidad de memoria requerida para poder renderizar la esfera. Una vez ejecutado el algoritmo esto es lo que se muestra:

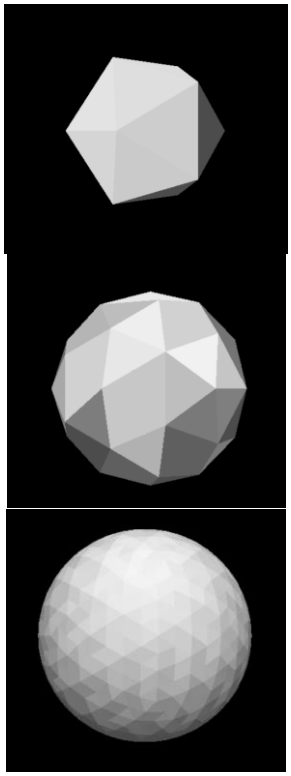

Fig. 11. Frames de la esfera renderizada utilizando un icosaedro.

V. CONCLUSIONES.

Es interesante conocer todo el proceso detrás del renderizado de objetos 2D y 3D, el uso de vértices, y el entendimiento de toda la matemática detrás de ello permite que se puedan crear prototipos para aplicarlos en varios campos. PyOpenGL no es perfecto, pero permite desarrollar aplicaciones increíbles y asentar las bases necesarias para comprender los gráficos por computador.

VI. REFERENCIAS.

[1] "Rendering (computer graphics)", *En.wikipedia.org*. [Online]. Available: https://en.wikipedia.org/wiki/Rendering_(computer_graphics). [Accessed: 28- Mar- 2020].

[2] "OpenGL - The Industry Standard for High Performance Graphics", Opengl.org, 1997. [Online]. Available: https://www.opengl.org/. [Accessed: 28- Mar- 2020].

[3] "PyOpenGL -- The Python OpenGL Binding", Pyopengl.sourceforge.net. [Online]. Available: http://pyopengl.sourceforge.net/. [Accessed: 28- Mar- 2020].

[4] "The freeglut Project :: About", Freeglut.sourceforge.net. [Online]. Available: http://freeglut.sourceforge.net/. [Accessed: 28- Mar- 2020].

[5] "pygame/pygame", GitHub. [Online]. Available: https://github.com/pygame/pygame. [Accessed: 28- Mar- 2020].

[6] "Icosahedron", En.wikipedia.org. [Online]. Available: https://en.wikipedia.org/wiki/Icosahedron. [Accessed: 28- Mar- 2020].

[7] S. Ahn, "OpenGL Sphere", Songho.ca. [Online]. Available: http://www.songho.ca/opengl/gl_sphere.html. [Accessed: 28- Mar- 2020].