Proyecto 01

Motor Grafica

Computación Gráfica

UNAM 2022-2

Gibran Zazueta Cruz

24/marzo/2022

1. Introducción

El programa que se presenta a continuación recibe una malla 3d desde un archivo OBJ y lo renderiza dentro de una escena. Los objetos se definen en 3 archivos diferentes con geometría para un cubo de 8 vértices y 6 caras cuadradas, un cubo de 8 vértices y 12 caras triangulares y una esfera con caras rectangulares.

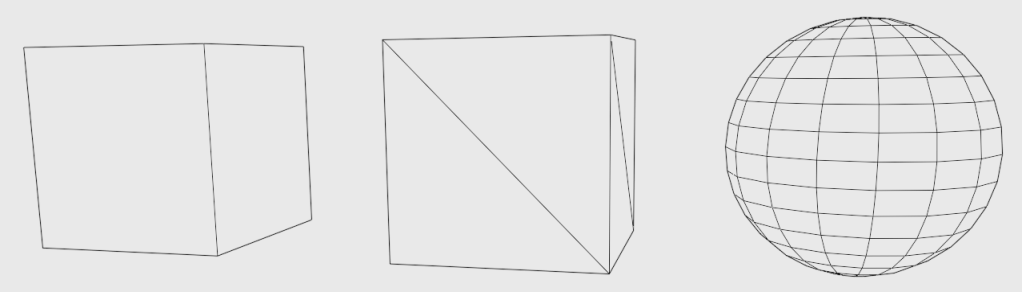

Figura 1: Objetos a renderizar

Como ya se mencionó, la geometría se lee desde un fichero OBJ. Se explica a continuación.

1.1. Archivo OBJ

OBJ es un formato de archivos desarrollado por Wavefront Technologies. Representa la geometría 3d definiendo:

- Posición de los vértivces
- Normal de los vértices
- Coordenadas UV de textura
- Caras (como una lista de vértices)

Para este trabajo solo se hace uso de la posición de los vertices y la definición de las caras.

1.1.1. Librería *assimp*

Para leer los archivos se utiliza la libreria de assimp. *Assimp* es una librería open source que soporta multiples formatos de geometria 3d, entre llos .OBJ. Además funciona con diversos sistemas operativos y provee una interfaz de C++.

También soporta una jerarquía de nodos para mallas, materiales, texturas y animaciones de *bones*. Para el programa de este reporte se hace uso principalmente de la función *import* clase Assimp::Importer que almacena datos es una estructura llamada aiScene.

2. Escena y materiales

Los objetos cuentan con 2 posibles materiales a seleccionar, estos se definen dentro del código como:

Material 1

- Ambiental = 0.0, 0.0, 0.0, 1.0,
- Difusa = 0.50, 0.50, 0.50, 1.0,
- Especular 0.70, 0.70, 0.70, 1.0
- $\rho = 32.0$.

Material 2

- Ambiental = 0.23125, 0.23125, 0.23125, 1.0,
- Difusa = 0.2775, 0.2775, 0.2775, 1.0,
- Especular 0.773911, 0.773911, 0.773911, 1.0
- $\rho = 89.6$.

Finalmente, la escena a renderizar cuenta con 2 luces (blanca y azul) y 3 cámaras.
Se disponen de la siguiente manera:

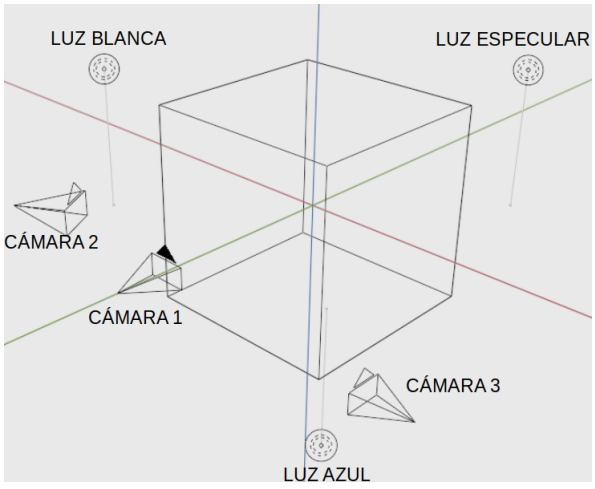

Figura 2: Posición de las componentes de la escena

3. Estructura del código

Desde *mainwindow.cpp* se llama a la función *importFile()* que está definida en *functions.cpp*. La función recibe el path del archivo (como una cadena std::string) y apuntadores al contenedor de vertices y caras del objeto cubeObject, que es el objeto a renderizar en la escena. Dentro de esta función se utiliza la librería de assimp. Se importa la información del fichero con la función *importer* de la clase Assimp::Importer y se guarda la escena en una variable aiScene. La función *importer* necesita que se le especifique un *"Process"*, que es un post-procesado de los datos leidos. En este caso se utiliza *aiProcessJoinIdenticalVertices* para unir los vértices iguales (que son los vértices que se repiten entre las caras).

La información del archivo se importa a la estructura de datos *aiScene*. Dentro del programa la información de la geometría se procesa desde un objeto de la clase CubeObject. Esta clase tiene un método para calcular normales de las caras y normales de los vertices, por lo que solo es necesario darle información de lasOBJ coordenadas de los vértices y la composición de las caras.

Después de esto, en mainwindow se crean los objetos de clase *light* para las luces blanca, azul y especular, se especifican intensidad, color y posicionamiento. También se crea el grid de botones que conforma la GUI de la aplicacion. Finalmente se llama a la funcion *drawObject()* con un timer que renderizará al polígono cada 30 ms.

En la funcion *drawObject()* se llama a la función pipeline del objeto raster. En la funcion pipeline primero se proyectan los vertices dentro del objeto de cámara (que contendrá el marco de referencia correspondiente a la cámara 1, 2 o 3), despues se hace el scan conversion e interpolación de los pixeles, zbuffer, normales e información de color(si es que se usa Gouroud).
Si no se utiliza gouroud el último paso es calcular el sombreado de phong en cada pixel.

Finalmente, la informacion de posición de pixeles, color y profundidad se pasa a la clase renderwindow(que hereda de QPainter), donde se pintarán los pixeles sobre el canvas.

4. Ejecutar el programa

En la carpeta de build se puede ejecutar el programa con el archivo renderOBJ-Run. Desde la consola de comandos de linux:

bash

```
./renderOBJ-Run
```

En la carpeta principal está el código fuente. Para generar el ejecutable primero se genera el Makefile con

bash

```
qmake renderOBJ.pro
```

Después se construye el proyecto con *make*

5. Instrucciones de uso

Se presenta la interfaz del programa.

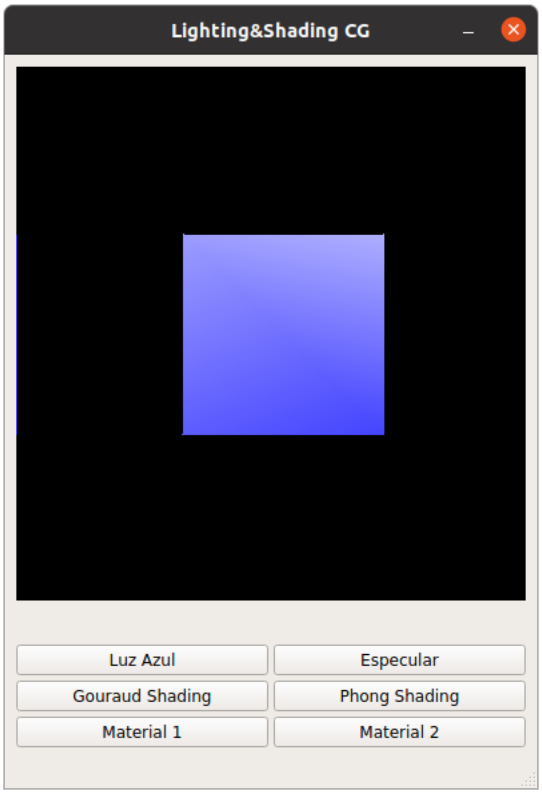

Figura 3: Interfaz gráfica del programa

Para cambiar entre las camaras se utilizan las teclas de los numeros

- "1". Cambia a la cámara 1
- "2". Cambia a la cámara 2
- "3". Cambia a la cámara 3

Para rotar el objeto sobre el eje X se presiona la tecla R. Se vuelve a apresionar para que deje de rotar.

Los botones *Luz Azul* y *Especular* encienden o apagan las luces correspondientes. El programa inicia con todas las luces activas.

Gouraud Shading activa el Sombreado por Gouraud y *Phong Shading* el sombreado de Phong.

Es posible cambiar entre archivos desde el código. En la variable path de mainwindow se cambia la extension por el archivo correspondiete. Por defecto es $Cube\_Triangle$, los otros dos son $Cube\_Quads$ y sphere.

6. Programa en ejecución

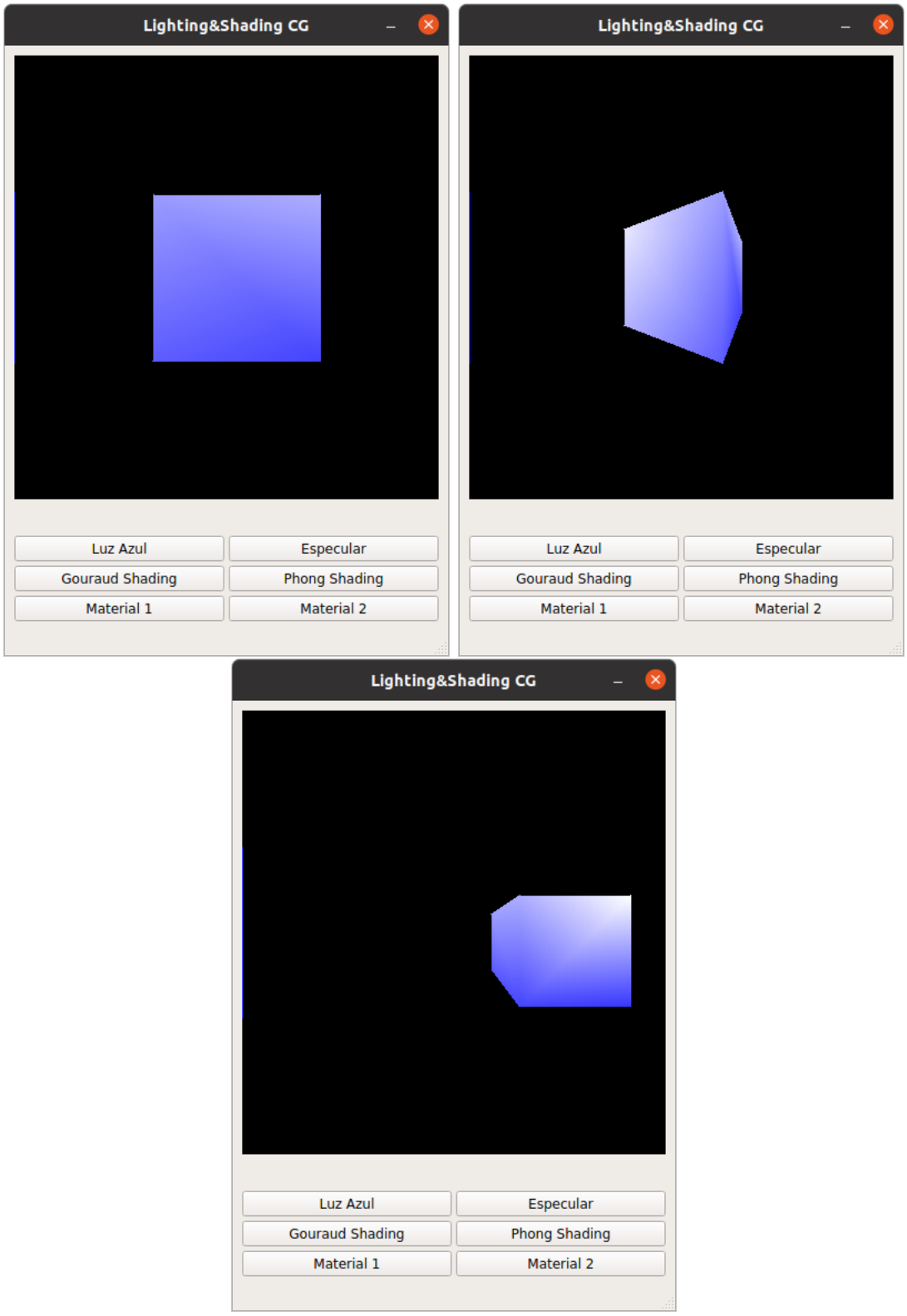

Figura 4: Gouraud Shading. Cámara 1, 2 y 3. Material 1

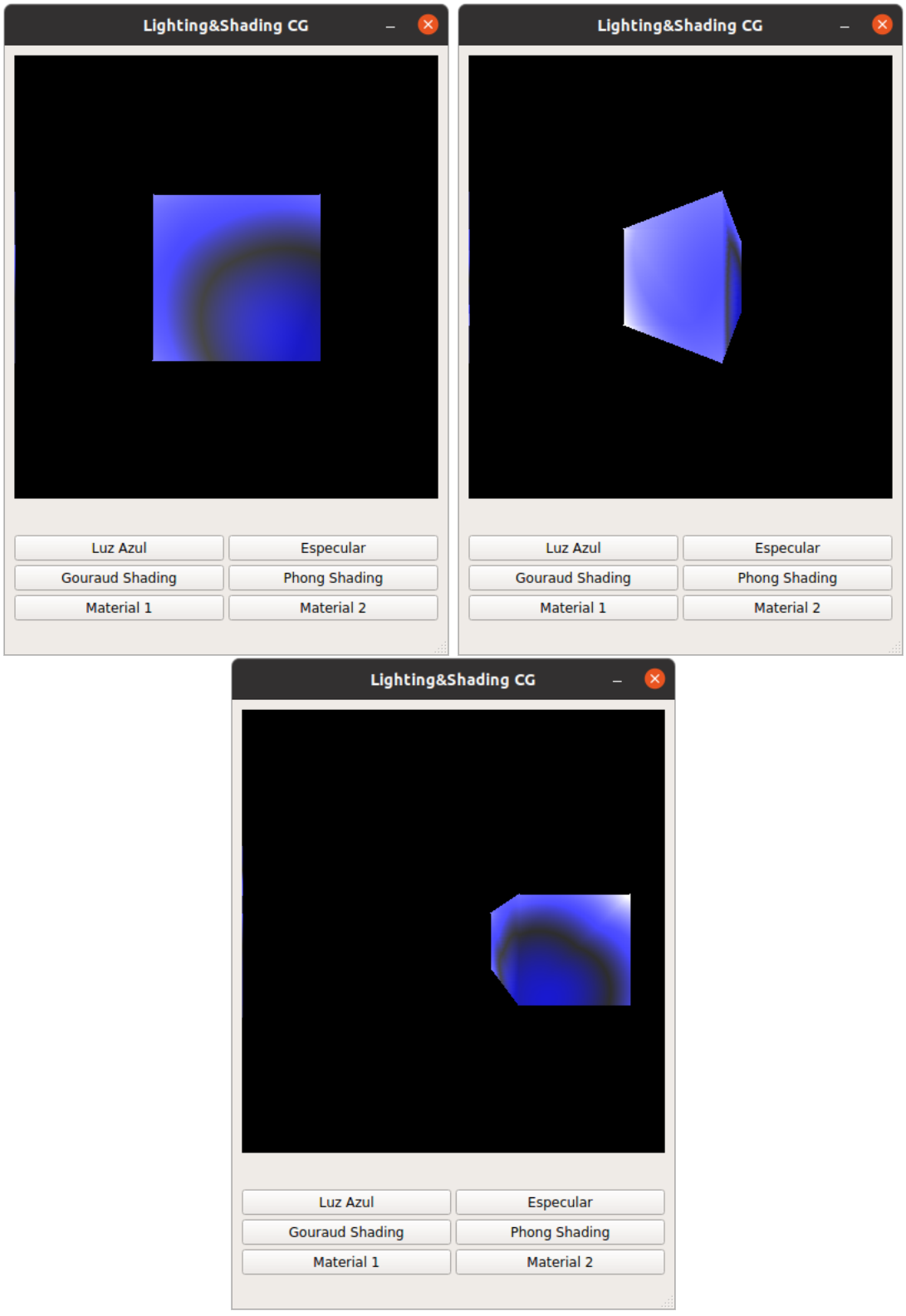

Figura 5: Phong Shading. Cámara 1, 2 y 3. Material 2

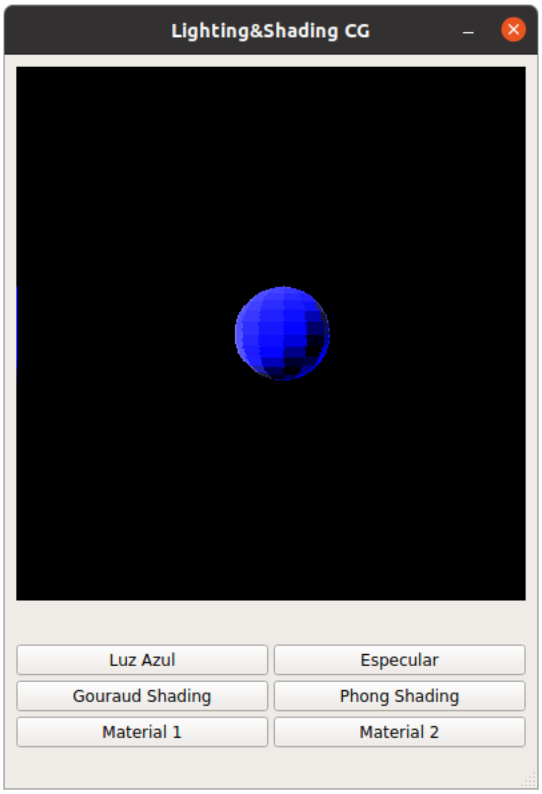

Figura 6: Renderizado esfera

Referencias