Maio 2023…..

Práctica

1. **(2 puntos) Co ue debuxa as caras 1 (frontal) e 5 (abaixo) da seguinte figura de forma que ao poñerlle a textura que se ve á dereita na cara 1 se vexa unha A e na cara 5 se vexa o G. É obrigatorio darlle cor aos vértices e hai que poñer as normais.´-**
   (Por se non se entende a figura era medio cubo cortado e pedía completar o código para debuxar a cara frontal e a de abaixo)

   Código que daba con ocos:

```
unsigned int VBO;

float vertices[] = {
       _______,          _______,          _______,     _______,
       _______,          _______,          _______,     _______,
       _______,          _______,          _______,     _______,
       _______,          _______,          _______,     _______,
       _______,          _______,          _______,     _______,
       _______,          _______,          _______,     _______,

       _______,          _______,          _______,     _______,
       _______,          _______,          _______,     _______,
       _______,          _______,          _______,     _______
    };

    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);

    glBindVertexArray(VAO);

    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, _______, _______, GL_STATIC_DRAW);

    glVertexAttribPointer(__, __, GL_FLOAT, GL_FALSE, __, __);

    // position Normal
    glVertexAttribPointer(__, __, GL_FLOAT, GL_FALSE, __, __);

    // position Color
    glVertexAttribPointer(__, __, GL_FLOAT, GL_FALSE, __, __);
```

```
    // position Texture
    glVertexAttribPointer(__, __, GL_FLOAT, GL_FALSE, __, __);
```

```
glBindVertexArray(VAO);
glDrawArrays(GL_TRIANGLES, __, __);
glDrawArrays(GL_TRIANGLES, __, __);
```

Solución:

```
unsigned int VBO;

float vertices[] = {
       //Vertices        // Normais       // Cores    //Texturas
(creo que las texturas están   mal puestas?)
       0, 0, 1,          1, 0, 1,         1, 1, 1,    1.0f,  1.0f,
       1, 0, 0,          1, 0, 1,         1, 1, 1,    0.5f,  1.0f,
       0, 1, 1,          1, 0, 1,         1, 1, 1,    1.0f,  0.5f,
       0, 1, 1,          1, 0, 1,         1, 1, 1,    1.0f,  0.5f,
       1, 0, 0,          1, 0, 1,         1, 1, 1,    0.5f,  1.0f,
       1, 1, 0,          1, 0, 1,         1, 1, 1,    1.0f,  0.5f,

       0, 0, 1,          0, -1, 0,        1, 1, 0,    0.5f,  0.5f,
       0, 0, 0,          0, -1, 0,        1, 1, 0,    0.75f, 1.0f,
       1, 0, 0,          0, -1, 0,        1, 1, 0,    1.0f,  0.5f

    };

    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);

    glBindVertexArray(VAO);

    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices,
GL_STATIC_DRAW);

    // position vertex
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 11 *
sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);

    // position Normal
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 11 *
sizeof(float), (void*)(3 * sizeof(float)));
```

```
    glEnableVertexAttribArray(1);


    // position Color
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, 11 *
sizeof(float), (void*)(6 * sizeof(float)));
    glEnableVertexAttribArray(2);

    // position Texture
    glVertexAttribPointer(3, 2, GL_FLOAT, GL_FALSE, 11 *
sizeof(float), (void*)(9 * sizeof(float)));
    glEnableVertexAttribArray(3);
```

```
glBindVertexArray(VAO);
glDrawArrays(GL_TRIANGLES, 0, 6);
glDrawArrays(GL_TRIANGLES, 6, 3);
```

2. **(1,5 puntos) Escribir o shader.vert e o shader.frag para debuxar a anterior figura combinando a cor dos vértices e a textura. Usar luz difusa** (dáballe igual que fora unha luz direccional ou un punto de luz e creo que había que usar cor da luz).

   Vertex:

```
#version 330 core

layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec3 aColor;
layout (location = 3) in vec2 aTexCoord;

out vec3 Normal;
out vec3 Color;
out vec3 FragPos;
out vec2 TexCoord;

uniform mat4 transform;
uniform mat4 view;
uniform mat4 projection;

void main()
{
   gl_Position = projection * view * transform * vec4(aPos, 1.0f);
   Normal = mat3(transpose(inverse(transform))) * aNormal;
   Color = aColor;
```

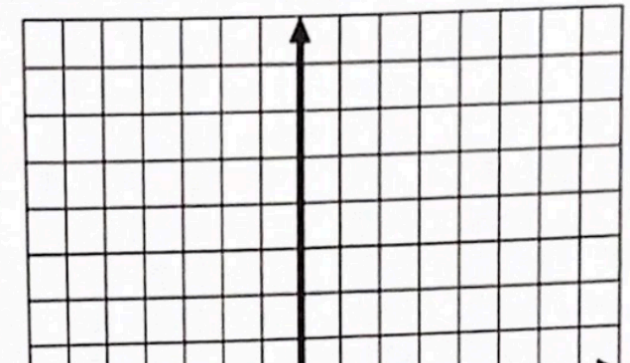
```
    FragPos = vec3(transform * vec4(aPos, 1.0));
    TexCoord = aTexCoord;
}
```

Fragment:

```
#version 330 core

in vec3 Normal;
in vec3 Color;
in vec3 FragPos;
in vec2 TexCoord;

// Usamos unha luz direccional
uniform vec3 lightDirection;
uniform vec3 lightColor;

uniform sampler2D tex;

out vec4 FragColor;

void main()

{

    // Depende de como especifiquemos a direccion da luz
    // temos que considerar -lightDirection ou lightDirection
    float diff = max(dot(normalize(-lightDirection), Normal), 0.0);
    // Tamen se poderían sumar a cor e a textura ou pasarllos
   // como argumento a funcion mix/
    vec3 saida = lightColor * diff * (Color * vec3(texture(tex,
TexCoord)));

    FragColor = vec4 (saida,1.0f);
}
```

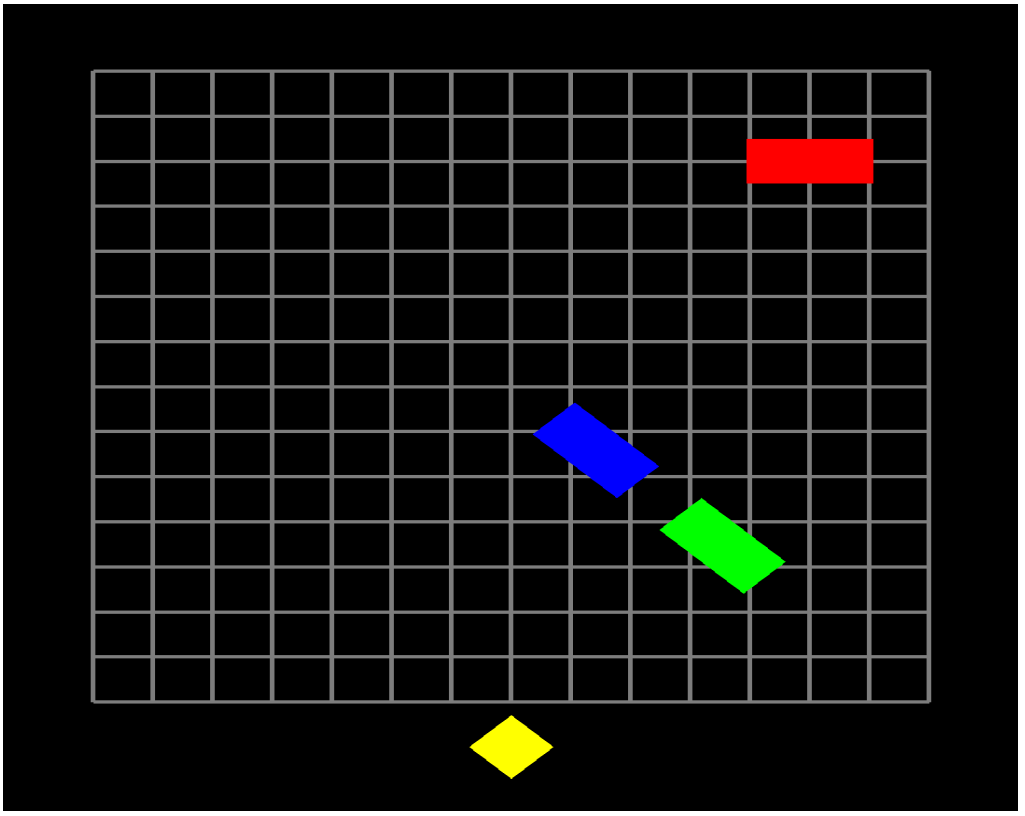
3. **(1,5 puntos) Debuxar a posición na pantalla dos 4 cubos que se debuxan no código amosado a continuación. A dereita do código poñíache unha cuadrícula 7x7 que representaba unha pantalla.**

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
//myEjes();
```

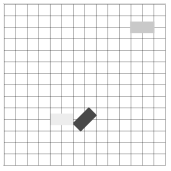


```
glPushMatrix();

glTranslatef(5, 5, 0);

glPushMatrix();

glRotatef(45.0f, 0, 1, 0);
glScalef(2, 1, 1);
glColor3f(1.0, 0, 0);
glutSolidCube(1);

glPopMatrix();

glPopMatrix();

glPushMatrix();

glRotatef(45.0f, 0, 0, 1);
glTranslatef(0, -5, 0);

glPushMatrix();

glRotatef(90.0f, 0, 0, 1); => ROTAR HACIA LA IZQ
glScalef(2, 1, 1);
glColor3f(0, 1, 0);
glutSolidCube(1);

glPopMatrix();

glPushMatrix();

glRotatef(90.0f, 0, 0, 1);
glTranslatef(3, 0, 0);

glPushMatrix();

glRotatef(90.0f, 0, 0, 1);
glScalef(1, 2, 1);
glColor3f(0, 0, 1);
glutSolidCube(1);

glPopMatrix();

glPopMatrix();

glPopMatrix();

glPushMatrix();

glRotatef(90.0f, 0, 0, 1);
```

```
glTranslatef(-8, 0, 0);

glPushMatrix();

glRotatef(45.0f, 0, 0, 1);
glScalef(1, 1, 1);
glColor3f(1, 1, 0);
glutSolidCube(1);

glPopMatrix();
glPopMatrix();

//glPopMatrix();
```



Solución: creo que esta mal pq antes estaba mal copiado



O último cubo que se debuxaba non se vía (quedaría por debaixo da 5pantalla)

Test **(5 puntos): Respuestas erróneas o en blanco restan (al final decidió que las respuestas incorrectas/en blanco no restasen)**

ESTÁN LAS RESPUESTAS CORREGIDAS POR ÉL

1. Tengo un juego en mi ordenador que tiene las siguientes tasas de refresco, 50f/s durante 100 frames, 25f/s durante 100 frames, 10f/s durante 100

frames, 25f/s durante 100 frames. ¿Cuál es la tasa media de refresco en frames por segundo?

a. =25f/s
b. =20f/s
c. =15f/s
d. =Otra

2. ¿Desde qué comienza el proceso de rasterizado, cuánto tiempo se necesitará para barrer cada columna durante el refresco de pantalla con una resolución de 640*480 y una velocidad de refresco de 30f/s?
    a. (640*480) / (1/30) segundos
    b. (640*480) * (1/30) segundos
    c. (1/30) / (640*480) segundos1
    d. Depende si es la primera o la última columna.

3. ¿Qué son las listas de visualización en OpenGL?
    a. Una herramienta para dibujar objetos en 3D.
    b. Una herramienta para almacenar y reutilizar comandos de dibujo en 3D.
    c. Una herramienta para crear efectos de iluminación en escenas 3D.
    d. Una herramienta para simplificar el código y hacerlo más compacto.

4. ¿Para qué se utiliza la pila de matrices en OpenGL?
    a. Para almacenar las texturas de los objetos en una escena en 3D.
    b. Para almacenar las transformaciones geométricas aplicadas a objetos en una escena 3D.
    c. Para almacenar las transformaciones del world space al clip space.
    d. Para almacenar los vértices de los objetos en una escena 3D.

5. ¿Qué tipo de estructura de datos se utiliza para la pila de matrices de transformación en OpenGL?
    a. FIFO
    b. LIFO
    c. LIFO y FIFO, dependiendo si es de la transformación (LIFO), vista (FIFO).
    d. Se multiplican y solo se almacena la última.

6. Para pasar del view space al clip space se utiliza:

a. La matriz de proyección.
b. La matriz de vista.
c. La matriz de transformación.
d. La matriz de vista por la de proyección por la transformación.

7. En el modelo de iluminación de Phong:
   a. Se considera la escena en un conjunto.
   b. Se consideran los objetos individualmente.
   c. La iluminación entre los objetos se realiza automáticamente, en la CPU o GPU si disponemos una dedicada.
   d. Mejora velocidad de renderizado al optimizar el trazado de rayos.
8. La luz especular:
   a. Mejora el efecto del modelo de iluminación debido al coeficiente de especularidad.
   b. Mejora el efecto del modelo de iluminación debido al coeficiente de reflexión especular.
   c. Mejora el efecto del modelo de iluminación debido al coeficiente de especularidad y coeficiente de reflexión especular.
   d. Mejora el efecto del modelo de iluminación al incluir la posición del usuario en la ecuación.

9. ¿Cuál es la función del modelo de sombreado en OpenGL?
   a. Definir la posición de los vértices en una malla.
   b. Especificar el material y la textura de un objeto.
   c. Calcular la intensidad de la iluminación en cada punto de la superficie de un objeto.
   d. Generar las sombras de un objeto sobre otro.

10. ¿Cuál es la diferencia entre el modelo de sombreado Gouraud y el modelo de sombreado de Phong?
    a. El modelo de sombreado Gouraud es más adecuado para superficies planas, mientras que el modelo de sombreado Phong funciona mejor con superficies curvas.

    b. El modelo de sombreado Gouraud calcula la intensidad de la iluminación en los vértices de un objeto y luego interpola, mientras que el modelo de sombreado Phong calcula la intensidad a partir de la interpolación del valor de las normales de la superficie.

    la CPU o GPU si disponemos una dedicada.
    d. Mejora velocidad de renderizado al optimizar el trazado de

rayos.

c. El modelo de sombreado Gouraud calcula la intensidad de la iluminación en los vértices de un objeto y se interpola a cada fragmento, mientras que el modelo de sombreado Phong calcula la intensidad a partir de la interpolación del valor de los fragmentos adyacentes.
d. Es el mismo, Gouraud es el nombre del inventor y Phong el del algoritmo.

11. El color en el interior de una cara en opengl 1.2, se calcula:
    a. Mediante el modelo de iluminación.
    b. Durante el proceso de rasterizado.
    c. Mediante el proceso de sombreado una vez se determina si la cara es visible o no.
    d. Mediante el color de la cara, y la luz incidente sobre la misma.

12. El color en el interior de una cara en opengl 3.3, se calcula.
    a. Mediante el modelo de iluminación.
    b. Durante el proceso de rasterizado.
    c. En el vertex shader mediante el cálculo de la posición de los fragmentos y las normales.
    d. Mediante el color de cada fragmento.

13. Cuando dibujamos los objetos y teniendo en cuenta el efecto z-buffer. ¿El proceso de render es más efectivo sí?
    a. Dibujamos primero los objetos cercanos y luego los lejanos.
    b. Dibujamos primero los objetos lejanos y luego los cercanos.
    c. El z-buffer resuelve el problema en base, principalmente, a dos ecuacion es.
    d. No se ve afectado por el z-buffer.

14. ¿Cuántos fragmentos pueden existir en escena?
    a. Tantos como pixeles.
    b. Tantos como pixeles por el número de objetos.
    c. Tantos como pixeles por la superficie en pixeles de caras.
    d. Otros.

15. Las variables uniformes:
    a. Similares a las constantes y mantienen fijo su valor en el shader
    b. Permiten enviar valores desde el programa principal a los shaders.
    c. Permiten comunicar los shaders entre si.

d. Son variables normales: int, float, vectores o matrices.

16. En al alpha test:
    a. Se compara el valor de alfa de cada fragmento con un valor umbral predefinido.
    b. Se realizan cálculos matemáticos complejos en la geometría de los objetos para determinar si son visibles o no.
    c. Se aplican texturas RGBA a los objetos en una escena.
    d. Se optimiza el trabajo del fragment Shader.