



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

INGENIERÍA EN COMPUTACIÓN

LABORATORIO DE COMPUTACIÓN GRÁFICA e  
INTERACCIÓN HUMANO COMPUTADORA



## REPORTE DE PRÁCTICA Nº 03

**NOMBRE COMPLETO:** Ian Paul Marentes Degollado

**Nº de Cuenta:** 418046210

**GRUPO DE LABORATORIO:** 01

**GRUPO DE TEORÍA:** 06

**SEMESTRE 2024-2**

**FECHA DE ENTREGA LÍMITE:** 2 de marzo de 2024

**CALIFICACIÓN:** \_\_\_\_\_

1. Generar una pirámide rubik (pyraminx) de 9 pirámides por cara. Cada cara de la pyraminx que se vea de un color diferente y que se vean las separaciones entre instancias (las líneas oscuras son las que permiten diferenciar cada pirámide pequeña). Agregar en su documento escrito las capturas de pantalla necesarias para que se vean las 4 caras de toda la pyraminx o un video en el cual muestra las 4 caras.

A continuación, se muestra la definición de los vértices utilizados para la creación de las pirámides.

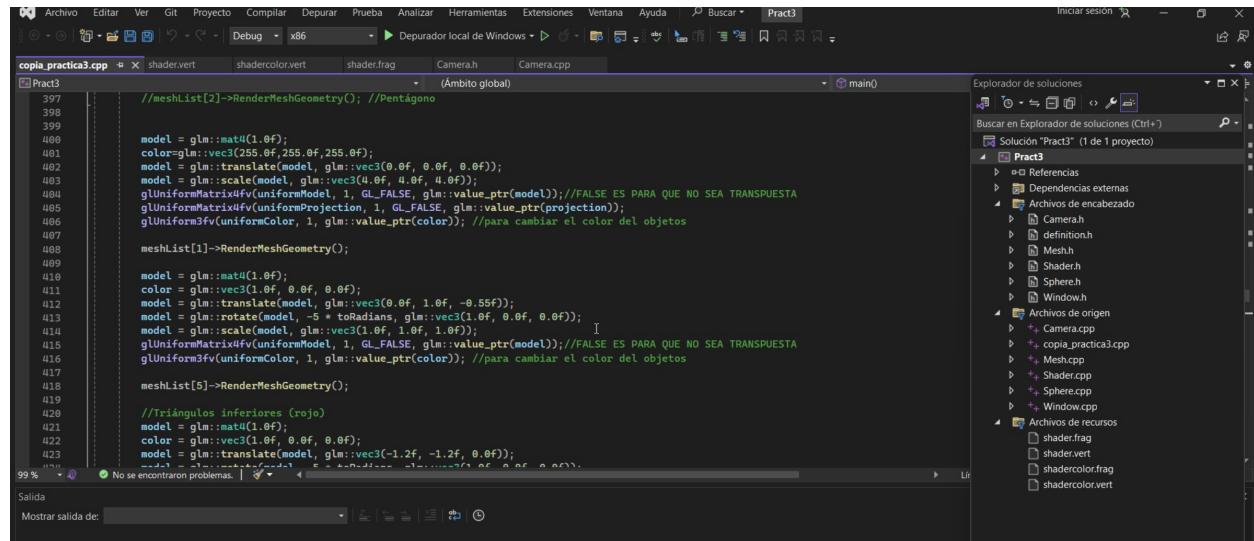
```

Pract3.cpp
100
101
102
103
104
105
106
107
108 void Triangulos_pequeños()
109 {
110     unsigned int indices_piramide_triangular[] = {
111         0,1,2,
112         1,3,2,
113         3,0,2,
114         1,0,3
115     };
116 }
117 GLfloat vertices_piramide_triangular[] = {
118     -0.5f, -0.5f, 0.0f, //0
119     0.5f, -0.5f, 0.0f, //1
120     0.0f, 0.5f, -0.25f, //2
121     0.0f, -0.5f, -0.25f, //3
122 };
123 }

Pract3.h
84
85     // Pirámide triangular regular
86     void CrearPiramideTriangular()
87     {
88         unsigned int indices_piramide_triangular[] = {
89             0,1,2,
90             1,3,2,
91             3,0,2,
92             1,0,3
93         };
94         GLfloat vertices_piramide_triangular[] = {
95             -0.5f, -0.5f, 0.0f, //0
96             0.5f, -0.5f, 0.0f, //1
97             0.0f, 0.5f, -0.25f, //2
98             0.0f, -0.5f, -0.25f, //3
99         };
100     Mesh* obj1 = new Mesh();
101 }

```

Posterior a ello, fui declarando las pirámides que formarían cada una de las caras.



Pensaba en cambiar el color del fondo de la terminal a blanco, pero decidí que la figura principal fuese de color blanco para que contrastara de igual forma con las pirámides de cada una de las caras.

copa\_practica3.cpp

```

418 meshList[5] -> RenderMeshGeometry();
419
420 //Triángulos inferiores (rojo)
421 model = glm::mat4(1.0f);
422 color = glm::vec3(1.0f, 0.0f, 0.0f);
423 model = glm::translate(model, glm::vec3(-1.2f, -1.2f, 0.0f));
424 model = glm::rotate(model, -5 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
425 model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
426 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
427 glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
428
429 meshList[5] -> RenderMeshGeometry();
430
431 model = glm::mat4(1.0f);
432 color = glm::vec3(1.0f, 0.0f, 0.0f);
433 model = glm::translate(model, glm::vec3(1.2f, -1.2f, 0.0f));
434 model = glm::rotate(model, -5 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
435 model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
436 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
437 glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
438
439 meshList[5] -> RenderMeshGeometry();
440
441 model = glm::mat4(1.0f);
442 color = glm::vec3(1.0f, 0.0f, 0.0f);
443 model = glm::translate(model, glm::vec3(0.0f, -1.2f, 0.0f));
444 model = glm::rotate(model, -5 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
445

```

Explorador de soluciones

- Solución "Pract3" (1 de 1 proyecto)
  - Archivos de cabecera
    - Camera.h
    - definition.h
    - Mesh.h
    - Shader.h
    - Sphere.h
    - Window.h
  - Archivos de origen
    - Camera.cpp
    - copa\_practica3.cpp
    - Mesh.cpp
    - Shader.cpp
    - Sphere.cpp
    - Window.cpp
  - Archivos de recursos
    - shader.frag
    - shadercolor.vert
    - shadercolor.frag

copa\_practica3.cpp

```

472
473 //Triángulos volteados
474 model = glm::mat4(1.0f);
475 color = glm::vec3(1.0f, 0.0f, 0.0f);
476 model = glm::translate(model, glm::vec3(0.0f, -0.05f, -0.3f));
477 model = glm::rotate(model, -180 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
478 model = glm::rotate(model, 30 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
479 model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
480 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
481 glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
482
483 meshList[5] -> RenderMeshGeometry();
484
485 model = glm::mat4(1.0f);
486 color = glm::vec3(1.0f, 0.0f, 0.0f);
487 model = glm::translate(model, glm::vec3(0.6f, -1.1f, 0.0f));
488 model = glm::rotate(model, -180 * toRadians, glm::vec3(0.0f, 0.0f, 1.0f));
489 model = glm::rotate(model, 30 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
490 model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
491 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
492 glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
493
494 meshList[5] -> RenderMeshGeometry();
495
496 model = glm::mat4(1.0f);
497 color = glm::vec3(1.0f, 0.0f, 0.0f);
498

```

Explorador de soluciones

- Solución "Pract3" (1 de 1 proyecto)
  - Archivos de cabecera
    - Camera.h
    - definition.h
    - Mesh.h
    - Shader.h
    - Sphere.h
    - Window.h
  - Archivos de origen
    - Camera.cpp
    - copa\_practica3.cpp
    - Mesh.cpp
    - Shader.cpp
    - Sphere.cpp
    - Window.cpp
  - Archivos de recursos
    - shader.frag
    - shadercolor.vert

copa\_practica3.cpp

```

628 meshList[5] -> RenderMeshGeometry();
629
630 //Triángulos de la cara morada
631 model = glm::mat4(1.0f);
632 color = glm::vec3(0.5f, 0.2f, 1.0f);
633 model = glm::translate(model, glm::vec3(0.4f, 0.0f, -1.3f));
634 model = glm::rotate(model, 2 * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
635 model = glm::rotate(model, -124 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
636 model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
637 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
638 glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
639
640 meshList[5] -> RenderMeshGeometry();
641
642 //Triángulos de la mitad
643 color = glm::vec3(0.5f, 0.2f, 1.0f);
644 model = glm::translate(model, glm::vec3(0.5f, -1.2f, 0.3f));
645 model = glm::rotate(model, glm::vec3(1.0f, 1.0f, 1.0f));
646 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
647 glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
648
649 meshList[5] -> RenderMeshGeometry();
650
651 color = glm::vec3(0.5f, 0.2f, 1.0f);
652 model = glm::translate(model, glm::vec3(-0.6f, 0.1f, -0.1f));
653 model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
654

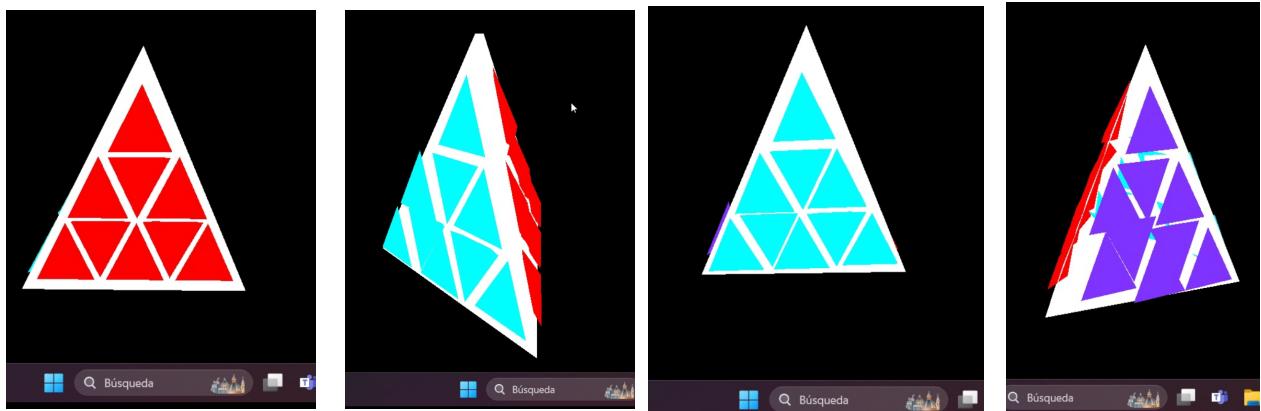
```

Explorador de soluciones

- Solución "Pract3" (1 de 1 proyecto)
  - Archivos de cabecera
    - Camera.h
    - definition.h
    - Mesh.h
    - Shader.h
    - Sphere.h
    - Window.h
  - Archivos de origen
    - Camera.cpp
    - copa\_practica3.cpp
    - Mesh.cpp
    - Shader.cpp
    - Sphere.cpp
    - Window.cpp
  - Archivos de recursos
    - shader.frag

Una complicación que tuve al instanciar las pirámides para cada una de las caras fueron las ubicaciones de las mismas, pues en ocasiones cambiaba el eje de referencia y se volvía un tanto complicado definir la posición dentro del plano tridimensional.

Ejecución del programa.



Cómo se puede observar en la ejecución del programa, en la última cara que definí tuve problemas para ubicar correctamente las pirámides que formaron parte de esta cara de la figura. Para procurar resolverlo fui definiendo la ubicación de cada uno de estos para tener la mejor aproximación a la figura solicitada; sin embargo, aún tuve algunos detalles en la ejecución.

#### Conclusiones

Al finalizar la práctica y el desarrollo del ejercicio en clase, me permitió practicar más y familiarizarme con la ubicación de los vértices en el espacio, aplicando así las funciones de traslación, rotación y escalación para llevar a cabo las figuras solicitadas, aunque me gustaría que fuese una explicación un poco más detenida en función de los shaders y cómo se realiza la construcción de las imágenes en el plano.

#### Referencias consultadas

Deckerix. (s. f.). Transformaciones geométricas en OpenGL. Recuperado de <http://deckerix.com/blog/transformaciones-geometricas-en-opengl/> el 2 de marzo de 2024.