



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE INGENIERÍA
DIVISIÓN DE INGENIERÍA ELÉCTRICA
INGENIERÍA EN COMPUTACIÓN
COMPUTACIÓN GRÁFICA e INTERACCIÓN HUMANO
COMPUTADORA



Reporte de práctica 2: Proyecciones y puertos de vista. Transformaciones Geométricas

NOMBRE COMPLETO: Gonzalez Villalba Bryan Jesus

Nº de Cuenta: 421530869

GRUPO DE LABORATORIO: 11

GRUPO DE TEORÍA: 4

SEMESTRE 2025-1

FECHA DE ENTREGA LÍMITE: 28 de Agosto del 2024

CALIFICACIÓN: _____

Introducción:

Para esta práctica utilizamos los archivos de los shaders para crear nuestros propios shaders para aplicarle color al cubo y a la pirámide de nuestro código, en donde además de dibujar estas figuras, también dibujamos nuestras iniciales de la practica pasada, colocándolas arriba de la casa que dibujaremos con os cubos y las pirámides.

Desarrollo:

1. Creamos nuestros archivos para los shaders de cada color que ocuparemos, primero crearemos el shader de color “Rojo” como se ve en la imagen (1.0), luego el shader de color “Verde” imagen (1.1), después el shader de color “Azul” imagen (1.2), después el shader de color “Café” imagen (1.3), y por ultimo el shader de color “Verde Oscuro” imagen (1.4).

```
VVerdeOscuro.vert  VVerde.vert  VRojo.vert  VCafe.vert  VAzul.vert  shadercolor.vert  shadercolor.frag  shader.vert  shader.frag
1  #version 330
2  layout (location =0) in vec3 pos;
3  out vec4 vColor;
4  uniform mat4 model;
5  uniform mat4 projection;
6  void main()
7  {
8      gl_Position=projection*model*vec4(pos,1.0f);
9      vColor=vec4(1.0f,0.0f,0.0f,1.0f);
10     //vColor=vec4(clamp(pos,0.0f,1.0f),1.0f); //aquí está el clamp
11 }
```

(Imagen 1.0)

```
VVerdeOscuro.vert  X VVerde.vert  VCafe.vert  VAzul.vert  shadercolor.vert  shadercolor.frag  shader.vert
C:\Users\bryan\OneDrive\PW\Visual Studio Community\Proyecto practica 2 CGelHC\Proyecto practica 2 CGelHC\shaders\VVerdeOscuro.vert
2  layout (location =0) in vec3 pos;
3  out vec4 vColor;
4  uniform mat4 model;
5  uniform mat4 projection;
6  void main()
7  {
8      gl_Position=projection*model*vec4(pos,1.0f);
9      vColor=vec4(0.0f,1.0f,0.0f,1.0f);
10     //vColor=vec4(clamp(pos,0.0f,1.0f),1.0f); //aquí está el clamp
11 }
```

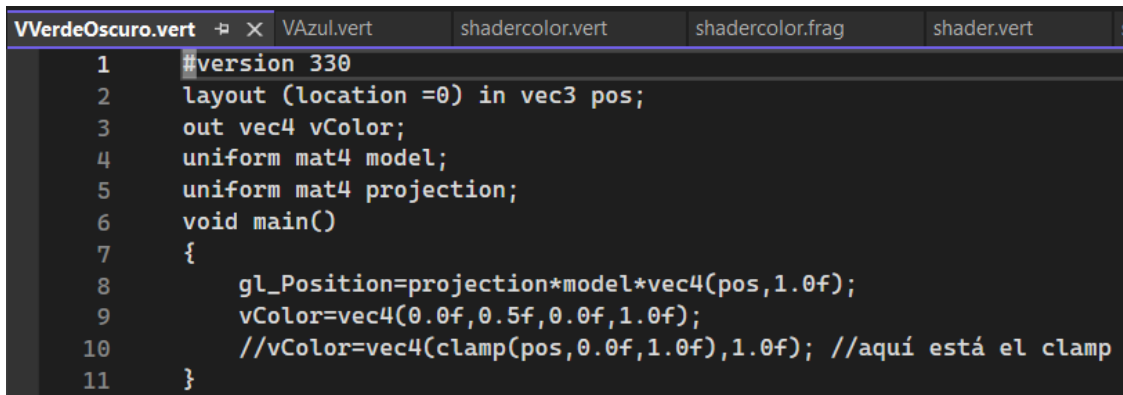
(Imagen 1.1)

```
VVerdeOscuro.vert  VCafe.vert  VAzul.vert  shadercolor.vert  shadercolor.frag  shade
1  #version 330
2  layout (location =0) in vec3 pos;
3  out vec4 vColor;
4  uniform mat4 model;
5  uniform mat4 projection;
6  void main()
7  {
8      gl_Position=projection*model*vec4(pos,1.0f);
9      vColor=vec4(0.0f,0.0f,1.0f,1.0f);
10     //vColor=vec4(clamp(pos,0.0f,1.0f),1.0f); //aquí está el clamp
11 }
```

(Imagen 1.2)

```
VVerdeOscuro.vert  VCafe.vert  VAzul.vert  shadercolor.vert  shadercolor.frag  st
1  #version 330
2  layout (location =0) in vec3 pos;
3  out vec4 vColor;
4  uniform mat4 model;
5  uniform mat4 projection;
6  void main()
7  {
8      gl_Position=projection*model*vec4(pos,1.0f);
9      vColor=vec4(0.478f,0.255f,0.067f,1.0f);
10     //vColor=vec4(clamp(pos,0.0f,1.0f),1.0f); //aquí está el clamp
11 }
```

(Imagen 1.3)



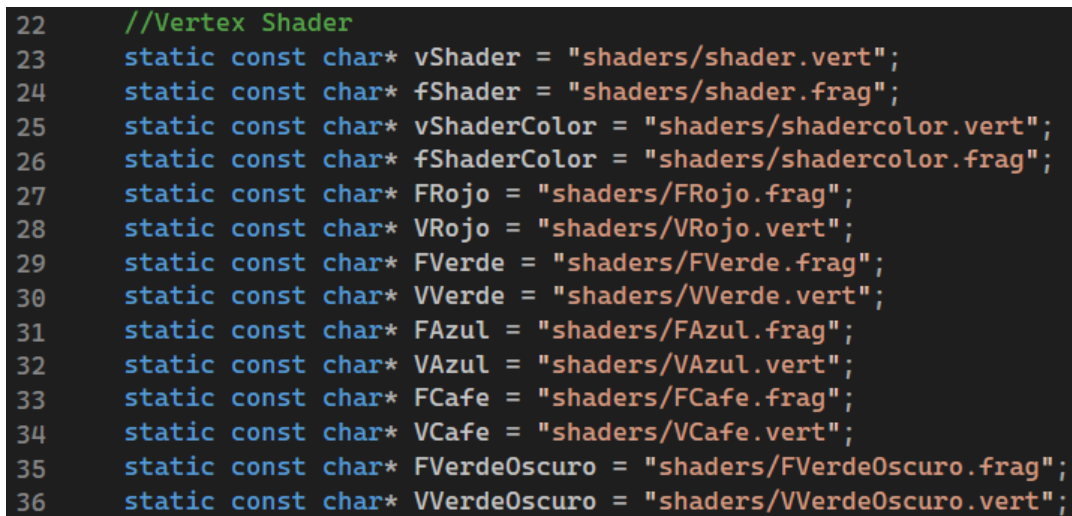
```

1  #version 330
2  layout (location =0) in vec3 pos;
3  out vec4 vColor;
4  uniform mat4 model;
5  uniform mat4 projection;
6  void main()
7  {
8      gl_Position=projection*model*vec4(pos,1.0f);
9      vColor=vec4(0.0f,0.5f,0.0f,1.0f);
10     //vColor=vec4(clamp(pos,0.0f,1.0f),1.0f); //aquí está el clamp
11 }

```

(Imagen 1.4)

2. Declaramos las variables que mandan a llamar nuestros archivos de shaders, desde la carpeta “Shaders”, declaración de shader.frag y shaders.vert, imagen (2.0).



```

22  //Vertex Shader
23  static const char* vShader = "shaders/shader.vert";
24  static const char* fShader = "shaders/shader.frag";
25  static const char* vShaderColor = "shaders/shadercolor.vert";
26  static const char* fShaderColor = "shaders/shadercolor.frag";
27  static const char* FRojo = "shaders/FRojo.frag";
28  static const char* VRojo = "shaders/VRojo.vert";
29  static const char* FVerde = "shaders/FVerde.frag";
30  static const char* VVerde = "shaders/VVerde.vert";
31  static const char* FAzul = "shaders/FAzul.frag";
32  static const char* VAzul = "shaders/VAzul.vert";
33  static const char* FCafe = "shaders/FCafe.frag";
34  static const char* VCafe = "shaders/VCafe.vert";
35  static const char* FVerdeOscuro = "shaders/FVerdeOscuro.frag";
36  static const char* VVerdeOscuro = "shaders/VVerdeOscuro.vert";

```

(Imagen 2.0)

3. Mandamos a dibujar los colores, guardándolos en la “ShaderList”, que inicia de 0, pero como guardamos 7 datos ahí, tenemos números del 0 al 6, el [0] es la pirámide, el [1] es el cubo, el [2] es el color rojo, el [3] es el color verde, el [4] es el color azul, el [5] es el color café y por ultimo el [6] es el verde olivo, para crear esas llamadas, ocupamos la función “void CreateShaders ()”, como se nos muestra en la imagen (3.0) y la imagen (3.1).

```

337 void CreateShaders()
338 {
339
340     Shader* shader1 = new Shader(); //shader para usar índices: objetos: cubo y pirámide
341     shader1->CreateFromFiles(vShader, fShader);
342     shaderList.push_back(*shader1); //Aquí hace que shaderList[0]
343
344     Shader* shader2 = new Shader(); //shader para usar color como parte del VAO: letras
345     shader2->CreateFromFiles(vShaderColor, fShaderColor);
346     shaderList.push_back(*shader2);
347
348     Shader* shader3 = new Shader();
349     shader3->CreateFromFiles(VRojo, FRojo);
350     shaderList.push_back(*shader3); //2
351
352     Shader* shader4 = new Shader();
353     shader4->CreateFromFiles(VVerde, FVerde);
354     shaderList.push_back(*shader4); //3
355
356     Shader* shader5 = new Shader();
357     shader5->CreateFromFiles(VAzul, FAzul);
358     shaderList.push_back(*shader5); //4
359

```

(Imagen 3.0)

```

359
360     Shader* shader6 = new Shader();
361     shader6->CreateFromFiles(VCafe, FCafe);
362     shaderList.push_back(*shader6); //5
363
364     Shader* shader7 = new Shader();
365     shader7->CreateFromFiles(VVerdeOscuro, FVerdeOscuro);
366     shaderList.push_back(*shader7); //6
367 }

```

(Imagen 3.1)

- Para el dibujado de los cubos y las pirámides, ocupamos el código que el profesor nos indicó en clase, solo copiamos el código, indicamos en meshList si era [0] para la pirámide y [1] para el cubo, después se le daba las coordenadas y al último le fui agregando el “shaderList” para aplicarle el color correspondiente a cada figura, como se muestra en la imagen (4.0), (4.1), (4.2), (4.3) y (4.4).

```

426 //Cubo de casa
427 shaderList[2].useShader();
428 uniformModel = shaderList[2].getModelLocation();
429 uniformProjection = shaderList[2].getProjectLocation();
430 model = glm::mat4(1.0);
431 model = glm::translate(model, glm::vec3(0.0f, -0.3f, -2.0f));
432 //model = glm::rotate(model, glm::radians(angulo), glm::vec3(1.0f, 1.0f, 1.0f));
433 model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
434 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES PARA QUE NO SEA TRANSPUESTA
435 glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
436 meshList[1]->RenderMesh();
437
438 //Piramide de techo
439 shaderList[4].useShader();
440 uniformModel = shaderList[4].getModelLocation();
441 uniformProjection = shaderList[4].getProjectLocation();
442 model = glm::mat4(1.0);
443 model = glm::translate(model, glm::vec3(0.0f, 0.25f, -1.8f));
444 //model = glm::rotate(model, glm::radians(angulo), glm::vec3(1.0f, 1.0f, 1.0f));
445 model = glm::scale(model, glm::vec3(1.2f, 0.3f, 0.3f));
446 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES PARA QUE NO SEA TRANSPUESTA
447 glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
448 meshList[0]->RenderMesh();

```

(Imagen 4.0)

```

450 // Piramide de arbol 1
451 shaderList[6].useShader();
452 uniformModel = shaderList[6].getModelLocation();
453 uniformProjection = shaderList[6].getProjectLocation();
454 model = glm::mat4(1.0);
455 model = glm::translate(model, glm::vec3(-0.8f, -0.3f, -1.8f));
456 //model = glm::rotate(model, glm::radians(angulo), glm::vec3(1.0f, 1.0f, 1.0f));
457 model = glm::scale(model, glm::vec3(0.3f, 0.4f, 0.4f));
458 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES PARA QUE NO SEA TRANSPUESTA
459 glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
460 meshList[0]-->RenderMesh();
461
462 //Cubo de arbol 1
463 shaderList[5].useShader();
464 uniformModel = shaderList[5].getModelLocation();
465 uniformProjection = shaderList[5].getProjectLocation();
466 model = glm::mat4(1.0);
467 model = glm::translate(model, glm::vec3(-0.8f, -0.6f, -1.85f));
468 //model = glm::rotate(model, glm::radians(angulo), glm::vec3(1.0f, 1.0f, 1.0f));
469 model = glm::scale(model, glm::vec3(0.1f, 0.2f, 0.1f));
470 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES PARA QUE NO SEA TRANSPUESTA
471 glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
472 meshList[1]-->RenderMesh();

```

(Imagen 4.1)

```

474 // Piramide de arbol 2
475 shaderList[6].useShader();
476 uniformModel = shaderList[6].getModelLocation();
477 uniformProjection = shaderList[6].getProjectLocation();
478 model = glm::mat4(1.0);
479 model = glm::translate(model, glm::vec3(0.8f, -0.3f, -1.8f));
480 //model = glm::rotate(model, glm::radians(angulo), glm::vec3(1.0f, 1.0f, 1.0f));
481 model = glm::scale(model, glm::vec3(0.3f, 0.4f, 0.3f));
482 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES PARA QUE NO SEA TRANSPUESTA
483 glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
484 meshList[0]-->RenderMesh();
485
486 //Cubo de arbol 2
487 shaderList[5].useShader();
488 uniformModel = shaderList[5].getModelLocation();
489 uniformProjection = shaderList[5].getProjectLocation();
490 model = glm::mat4(1.0);
491 model = glm::translate(model, glm::vec3(0.8f, -0.6f, -1.85f));
492 //model = glm::rotate(model, glm::radians(angulo), glm::vec3(1.0f, 1.0f, 1.0f));
493 model = glm::scale(model, glm::vec3(0.1f, 0.2f, 0.1f));
494 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES PARA QUE NO SEA TRANSPUESTA
495 glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
496 meshList[1]-->RenderMesh();

```

(Imagen 4.2)

```

498 //Cubo de ventana 1
499 shaderList[3].useShader();
500 uniformModel = shaderList[3].getModelLocation();
501 uniformProjection = shaderList[3].getProjectLocation();
502 model = glm::mat4(1.0);
503 model = glm::translate(model, glm::vec3(-0.26f, -0.1f, -1.0f));
504 //model = glm::rotate(model, glm::radians(angulo), glm::vec3(1.0f, 1.0f, 1.0f));
505 model = glm::scale(model, glm::vec3(0.17f, 0.17f, 0.1f));
506 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES PARA QUE NO SEA TRANSPUESTA
507 glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
508 meshList[1]-->RenderMesh();
509
510 //Cubo de ventana 2
511 shaderList[3].useShader();
512 uniformModel = shaderList[3].getModelLocation();
513 uniformProjection = shaderList[3].getProjectLocation();
514 model = glm::mat4(1.0);
515 model = glm::translate(model, glm::vec3(0.26f, -0.1f, -1.0f));
516 //model = glm::rotate(model, glm::radians(angulo), glm::vec3(1.0f, 1.0f, 1.0f));
517 model = glm::scale(model, glm::vec3(0.17f, 0.17f, 0.1f));
518 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES PARA QUE NO SEA TRANSPUESTA
519 glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
520 meshList[1]-->RenderMesh();

```

(Imagen 4.3)

```

522 //Cubo de puerta
523 shaderList[3].useShader();
524 uniformModel = shaderList[3].getModelLocation();
525 uniformProjection = shaderList[3].getProjectLocation();
526 model = glm::mat4(1.0);
527 model = glm::translate(model, glm::vec3(0.0f, -0.6f, -1.0f));
528 //model = glm::rotate(model, glm::radians(angulo), glm::vec3(1.0f, 1.0f, 1.0f));
529 model = glm::scale(model, glm::vec3(0.17f, 0.19f, 0.1f));
530 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES PARA QUE NO SEA TRANSPUESTA
531 glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
532 meshList[1]-->RenderMesh();

```

(Imagen 4.4)

5. Para el dibujado de las letras, lo único que hicimos fue copiar nuestras coordenadas de la practica anterior y mandarlas a dibujar por medio de “meshColorList”, como se muestra en la imagen (5.0), (5.1), (5.2), (5.3) y (5.4).

```

GLfloat vertices_letras[] = {
    //X      Y      Z      R      G      B
    // Letra B 14 * 18 = 252
    -0.9f,0.9f,0.0f, 0.0f,0.0f,1.0f, //Punto A
    -0.9f,0.7f,0.0f, 0.0f,0.0f,1.0f, //Punto B
    -0.8f,0.7f,0.0f, 0.0f,0.0f,1.0f, //Punto C

    -0.9f,0.9f,0.0f, 0.0f,0.0f,1.0f, //Punto A
    -0.8f,0.7f,0.0f, 0.0f,0.0f,1.0f, //Punto C
    -0.8f,0.9f,0.0f, 0.0f,0.0f,1.0f, //Punto D

    -0.9f,0.7f,0.0f, 0.0f,0.0f,1.0f, //Punto B
    -0.8f,0.7f,0.0f, 0.0f,0.0f,1.0f, //Punto C
    -0.9f,0.5f,0.0f, 0.0f,0.0f,1.0f, //Punto E

    -0.8f,0.7f,0.0f, 0.0f,0.0f,1.0f, //Punto C
    -0.9f,0.5f,0.0f, 0.0f,0.0f,1.0f, //Punto E
    -0.8f,0.5f,0.0f, 0.0f,0.0f,1.0f, //Punto F

```

(Imagen 5.0)

```

167      // Letra J 6 * 18 = 108
168      -0.2f,0.9f,0.0f, 0.0f,1.0f,0.0f, //Punto A
169      0.2f,0.9f,0.0f, 0.0f,1.0f,0.0f, //Punto B
170      -0.2f,0.8f,0.0f, 0.0f,1.0f,0.0f, //Punto C
171
172      0.2f,0.9f,0.0f, 0.0f,1.0f,0.0f, //Punto B
173      -0.2f,0.8f,0.0f, 0.0f,1.0f,0.0f, //Punto C
174      0.2f,0.8f,0.0f, 0.0f,1.0f,0.0f, //Punto D
175
176      0.0f,0.8f,0.0f, 0.0f,1.0f,0.0f, //Punto E
177      0.0f,0.5f,0.0f, 0.0f,1.0f,0.0f, //Punto F
178      0.1f,0.5f,0.0f, 0.0f,1.0f,0.0f, //Punto G
179
180      0.0f,0.8f,0.0f, 0.0f,1.0f,0.0f, //Punto E
181      0.1f,0.5f,0.0f, 0.0f,1.0f,0.0f, //Punto G
182      0.1f,0.8f,0.0f, 0.0f,1.0f,0.0f, //Punto H
183
184      0.0f,0.5f,0.0f, 0.0f,1.0f,0.0f, //Punto F
185      -0.20f,0.5f,0.0f, 0.0f,1.0f,0.0f, //Punto I
186      -0.20f,0.55f,0.0f, 0.0f,1.0f,0.0f, //Punto J

```

(Imagen 5.1)


```

193 // Letra G 12 * 18 = 216
194 0.5f,0.9f,0.0f, 1.0f,0.0f,0.0f, //Punto A
195 0.6f,0.9f,0.0f, 1.0f,0.0f,0.0f, //Punto B
196 0.5f,0.5f,0.0f, 1.0f,0.0f,0.0f, //Punto C
197
198 0.6f,0.9f,0.0f, 1.0f,0.0f,0.0f, //Punto B
199 0.5f,0.5f,0.0f, 1.0f,0.0f,0.0f, //Punto C
200 0.6f,0.5f,0.0f, 1.0f,0.0f,0.0f, //Punto D
201
202 0.6f,0.5f,0.0f, 1.0f,0.0f,0.0f, //Punto D
203 0.9f,0.55f,0.0f, 1.0f,0.0f,0.0f, //Punto E
204 0.9f,0.50f,0.0f, 1.0f,0.0f,0.0f, //Punto F
205
206 0.6f,0.5f,0.0f, 1.0f,0.0f,0.0f, //Punto D
207 0.9f,0.55f,0.0f, 1.0f,0.0f,0.0f, //Punto F
208 0.6f,0.55f,0.0f, 1.0f,0.0f,0.0f, //Punto G
209
210 0.9f, 0.55f, 0.0f, 1.0f, 0.0f, 0.0f, //Punto F
211 0.9f,0.65f,0.0f, 1.0f,0.0f,0.0f, //Punto H
212 0.85f,0.65f,0.0f, 1.0f,0.0f,0.0f, //Punto I

```

(Imagen 5.2)

```

242 };
243 MeshColor *letras = new MeshColor();
244 letras->CreateMeshColor(vertices_letras,576);
245 meshColorList.push_back(letras);

```

(Imagen 5.3)

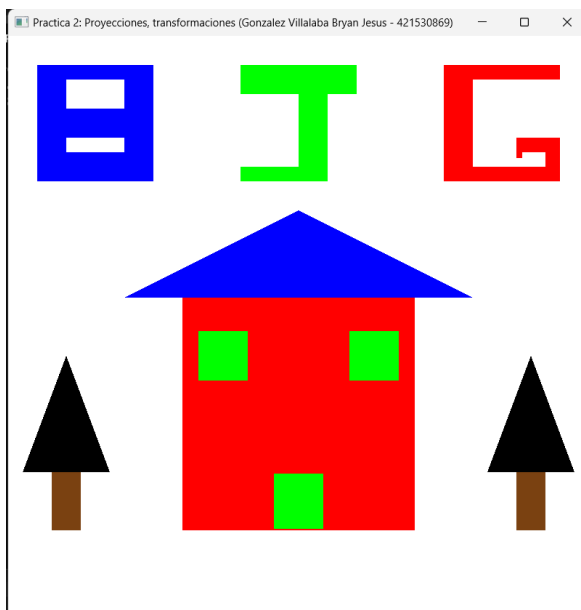
```

417 //Letras
418 model = glm::mat4(1.0);
419 model = glm::translate(model, glm::vec3(0.0f, 0.0f, 1.0f));
420 //model = glm::rotate(model, glm::radians(angulo), glm::vec3(1.0f, 1.0f, 1.0f));
421 model = glm::scale(model, glm::vec3(0.1f, 0.1f, 0.1f));
422 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model)); //FALSE ES PARA QUE NO SEA TRANSPUESTA
423 glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
424 meshColorList[1]->RenderMeshColor();

```

(Imagen 5.4)

6. El resultado de los dos dibujados se ve en la imagen (6.0).



(Imagen 6.0)

Conclusión:

Fue una práctica difícil porque se ejecutaron bien los colores, pero la creación de la casa fue fácil ya que teníamos hechas la figura del cubo y la figura de la pirámide, solo fue poner en posición y colocarle el color, de las letras también fue difícil encontrar la razón del por qué no la podía dibujar, porque no la enviaba a la lista, no imprimió la lista de donde estaba guardada y de las posiciones fue fácil.