UNIVERSIDAD NACIONAL DE SAN AGUSTÍN

ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN

COMPUTACION GRAFICA

Practica 9

Alumnos:
Chayña Batallanes Josnick
Perez Rodriguez Angelo Aldo
Pucho Zevallos Kelvin Paul
Vilcapaza Flores Luis Felipe
Sihuinta Perez Luis Armando

Junio 2021

Índice

1. Implemente todos los ejemplos del capitulo IV, del libro “Computer Graphics Programming in OpenGL with C++” [Gordon and Clevenger(2020)].

En estos ejercicios que se solucionaran posteriormente se necesito de una archivo Utils.cpp, este se usara para las siguientes funciones como leer archivos .glsl que contienen código shaders,compilarlos y vincularlos, y detectar errores GLSL. Dicho archivo es el siguiente :

```
#include <GL\glew.h>
#include <GLFW\glfw3.h>
#include <string>
#include <iostream>
#include <fstream>
#include <cmath>
#include <glm\glm.hpp>
#include <glm\gtc\type_ptr.hpp>
#include <glm\gtc\matrix_transform.hpp>
using namespace std;

class Utils {
public:
    Utils(){}
    //leendo archivos glsl
    static string readShaderSource(const char* filePath) {
        string content;

        ifstream fileStream(filePath, ios::in);
        string line = "";
        while (!fileStream.eof()) {
                getline(fileStream, line);
                content.append(line + "\n");
        }

        fileStream.close();
        return content;
    }

    // functions to catch errors in GLSL
    static void printShaderLog(GLuint shader) {
        int len = 0;
        int chWrittn = 0;
        char* log;
        glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &len);
        if (len > 0) {
                log = (char*)malloc(len);
                glGetShaderInfoLog(shader, len, &chWrittn, log);
                cout << "Shader Info Log: " << log << endl;
                free(log);
        }
    }
    static void printProgramLog(int prog) {
        int len = 0;
        int chWrittn = 0;
        char* log;
```

```
47          glGetProgramiv(prog, GL_INFO_LOG_LENGTH, &len);
48          if (len > 0) {
49                  log = (char*)malloc(len);
50                  glGetProgramInfoLog(prog, len, &chWrittn, log);
51                  cout << "Program Info Log: " << log << endl;
52                  free(log);
53          }
54      }
55      static bool checkOpenGLError() {
56          bool foundError = false;
57          int glErr = glGetError();
58          while (glErr != GL_NO_ERROR) {
59                  cout << "glError: " << glErr << endl;
60                  foundError = true;
61                  glErr = glGetError();
62          }
63          return foundError;
64      }
65
66      static GLuint createShaderProgram(const char* vertShader, const char*
            fragShader) {
67          GLint vertCompiled;
68          GLint fragCompiled;
69          GLint linked;
70
71          string vertShaderStr = readShaderSource(vertShader);
72          string fragShaderStr = readShaderSource(fragShader);
73          GLuint vShader = glCreateShader(GL_VERTEX_SHADER); //generando
               shader for vertex  vacios
74          GLuint fShader = glCreateShader(GL_FRAGMENT_SHADER);//generando
               shader for fragment vacios
75
76          const char* vertShaderSrc = vertShaderStr.c_str();
77          const char* fragShaderSrc = fragShaderStr.c_str();
78
79          glShaderSource(vShader, 1, &vertShaderSrc, NULL);
80          glShaderSource(fShader, 1, &fragShaderSrc, NULL);
81
82          glCompileShader(vShader);
83          checkOpenGLError();
84          glGetShaderiv(vShader, GL_COMPILE_STATUS, &vertCompiled);
85          if (vertCompiled != 1) {
86                  cout << "vertex compilation failed" << endl;
87                  printShaderLog(vShader);
88          }
89          glCompileShader(fShader);
90          checkOpenGLError();
91          glGetShaderiv(fShader, GL_COMPILE_STATUS, &fragCompiled);
92          if (fragCompiled != 1) {
93                  cout << "fragment compilation failed" << endl;
94                  printShaderLog(fShader);
95          }
96          //compiladores
97          GLuint vfProgram = glCreateProgram();
98
99          glAttachShader(vfProgram, vShader);
```

```
100             glAttachShader(vfProgram, fShader);
101             glLinkProgram(vfProgram);
102             checkOpenGLError();
103             glGetProgramiv(vfProgram, GL_LINK_STATUS, &linked);
104             if (linked != 1) {
105                     cout << "linking failed" << endl;
106                     printProgramLog(vfProgram);
107             }
108             return vfProgram;
109         }
110 };
```

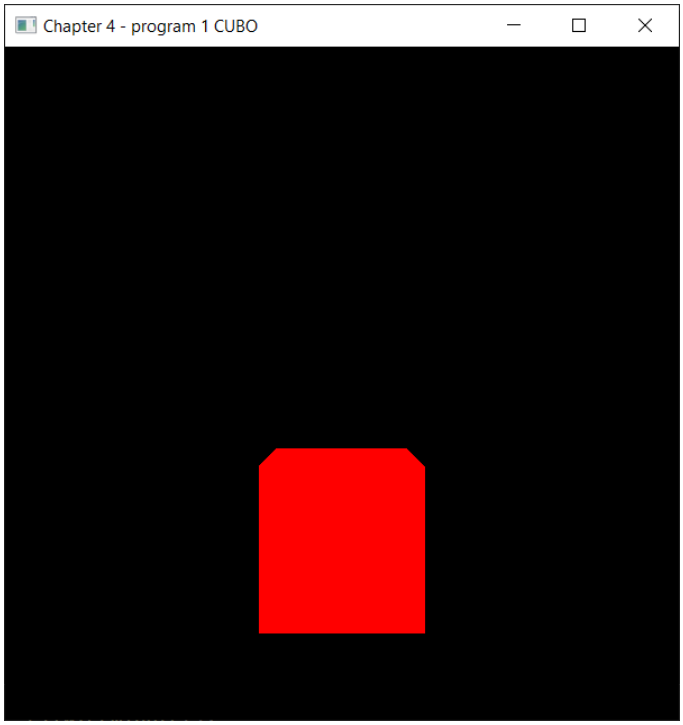
1.1. Programa 4.1: Red Cube (pag. 69, 75, 77 and 78).

En la página 69, demuestra como se construye un programa completo de un cubo en 3D de la siguiente manera:

```
1  #include "Utils.cpp"
2  using namespace std;
3
4  #define numVAOs 1
5  #define numVBOs 2
6
7  float cameraX, cameraY, cameraZ;
8  float cubeLocX, cubeLocY, cubeLocZ;
9
10 GLuint renderingProgram;
11 GLuint vao[numVAOs];
12 GLuint vbo[numVBOs];
13
14 GLuint mvLoc, projLoc;
15 int width, height;
16 float aspect;
17 glm::mat4 pMat, vMat, mMat, mvMat;
18
19 void setupVertices(void) { // 36 v[U+FFFD]rtices, 12 tri[U+FFFD]ngulos, hace un cubo
       de 2x2x2 colocado en el origen
20     float vertexPositions[108] = {
21       -1.0f, 1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 1.0f, -1.0f, -1.0f,//
           traingulo
22       1.0f, -1.0f, -1.0f, 1.0f, 1.0f, -1.0f, -1.0f, 1.0f, -1.0f,
23       1.0f, -1.0f, -1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 1.0f, -1.0f,
24       1.0f, -1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f, -1.0f,
25       1.0f, -1.0f, 1.0f, -1.0f, -1.0f, 1.0f, 1.0f, 1.0f, 1.0f,
26       -1.0f, -1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f,
27       -1.0f, -1.0f, 1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 1.0f, 1.0f,
28       -1.0f, -1.0f, -1.0f, -1.0f, 1.0f, -1.0f, -1.0f, 1.0f, 1.0f,
29       -1.0f, -1.0f, 1.0f, 1.0f, -1.0f, 1.0f, 1.0f, -1.0f, -1.0f,
30       1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 1.0f,
31       -1.0f, 1.0f, -1.0f, 1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 1.0f,
32       1.0f, 1.0f, 1.0f, -1.0f, 1.0f, 1.0f, -1.0f, 1.0f, -1.0f
33      };
34       glGenVertexArrays() y glGenBuffers() crean
35
```

```
36      glGenVertexArrays(1, vao);
37      glBindVertexArray(vao[0]);
38      glGenBuffers(numVBOs, vbo);
39
40      glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
41      glBufferData(GL_ARRAY_BUFFER, sizeof(vertexPositions),
            vertexPositions, GL_STATIC_DRAW);
42
43  }
44
45  void init(GLFWwindow* window) {
46      const char* vertShader = "cube_shader/vertShader.glsl";
47      const char* fragShader = "cube_shader/fragShader.glsl";
48
49      renderingProgram = Utils::createShaderProgram(vertShader, fragShader)
            ;
50      cameraX = 0.0f; cameraY = 0.0f; cameraZ = 8.0f;
51      cubeLocX = 0.0f; cubeLocY = -2.0f; cubeLocZ = 0.0f;
52      setupVertices();
53  }
54
55  void display(GLFWwindow* window, double currentTime) {
56          glClear(GL_DEPTH_BUFFER_BIT);
57          glUseProgram(renderingProgram);
58          mvLoc = glGetUniformLocation(renderingProgram, "mv_matrix");
59          projLoc = glGetUniformLocation(renderingProgram, "proj_matrix");
60
61          // construir la matriz de prespectiva
62          glfwGetFramebufferSize(window, &width, &height);
63          aspect = (float)width / (float)height;
64          pMat = glm::perspective(1.0472f, aspect, 0.1f, 1000.0f); //
                1.0472 radians = 60 degrees
65
66          // construir view matrix, model matrix, and model-view matrix
67          vMat = glm::translate(glm::mat4(1.0f), glm::vec3(-cameraX, -
                cameraY, -cameraZ));
68          mMat = glm::translate(glm::mat4(1.0f), glm::vec3(cubeLocX,
                cubeLocY, cubeLocZ));
69          mvMat = vMat * mMat;
70
71          // copiar matrices de perspectiva y MV a las correspondientes
                variables uniformes
72          glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvMat));
73          glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(pMat));
74
75
76          glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
77          glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
78          glEnableVertexAttribArray(0);
79          /
80
81          // ajustar la configuraci[U+FFFD]n de OpenGL y dibujar el modelo
82          glEnable(GL_DEPTH_TEST);
83          glDepthFunc(GL_LEQUAL);
84          glDrawArrays(GL_TRIANGLES, 0, 36);
85  }
```

- Capturas



En la página 75, modifica los shaders para obtener el color en una variable de salida para que en la variable de entrada del fragment shader se disponga el color, para resolver esto se muestra los siguiente codigos shaders:

- Vertex Shader

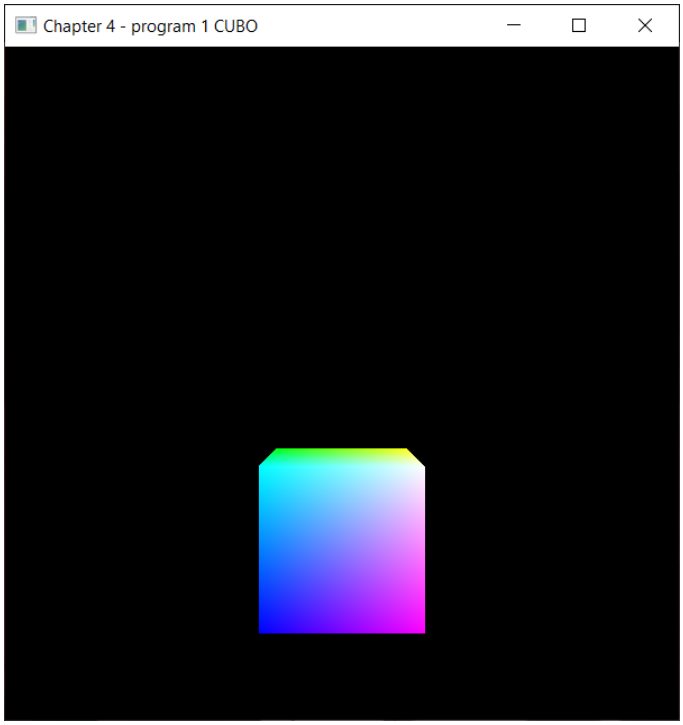
```
#version 430

layout (location=0) in vec3 position;

uniform mat4 mv_matrix;
uniform mat4 proj_matrix;

out vec4 varyingColor;

void main(void)
{
    gl_Position = proj_matrix * mv_matrix * vec4(position,1.0);
    varyingColor = vec4(position,1.0) * 0.5 + vec4(0.5, 0.5, 0.5,
        0.5);
}
```

- Fragment Shader

```
#version 430

in vec4 varyingColor;

```

```
out vec4 color;

uniform mat4 mv_matrix;
uniform mat4 proj_matrix;

void main(void)
{
    color = varyingColor;
}
```

- Capturas



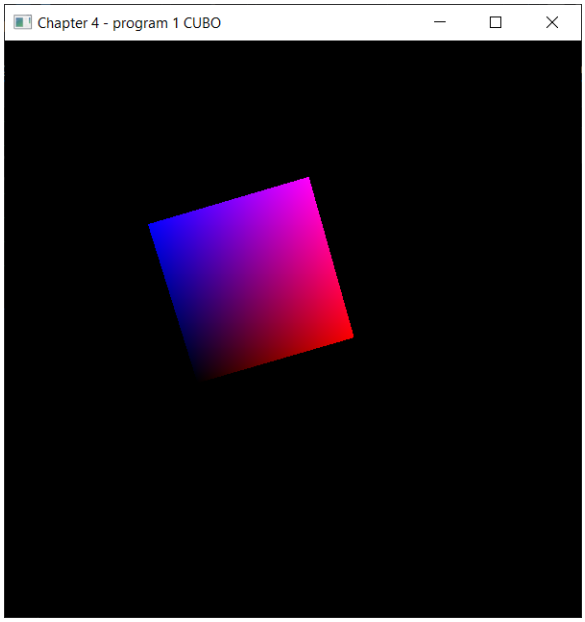
En la página 77, se demuestra la simulacion del movimiento del cubo aplicando transformaciones de traslacion y rotacion.

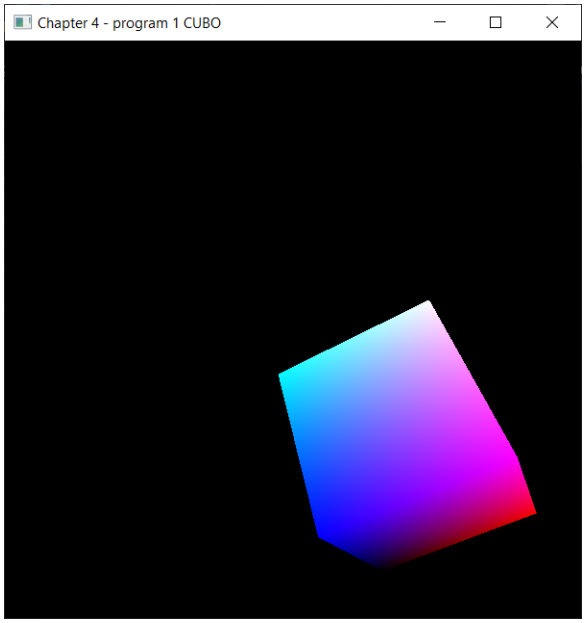
```
#include "Utils.h"
using namespace std;

#define numVAOs 1
#define numVBOs 2

float cameraX, cameraY, cameraZ;
float cubeLocX, cubeLocY, cubeLocZ;

GLuint renderingProgram;
GLuint vao[numVAOs];
GLuint vbo[numVBOs];


GLuint mvLoc, projLoc;
```

```
int width, height;
float aspect;
glm::mat4 pMat, vMat, mMat, mvMat;
glm::mat4 tMat, rMat;

void setupVertices(void) { // 36 v[U+FFFD]rtices, 12 tri[U+FFFD]ngulos, hace un cubo de 2x2x2 colocado en el origen
    float vertexPositions[108] = {
     -1.0f, 1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 1.0f, -1.0f, -1.0f,// traingulo
     1.0f, -1.0f, -1.0f, 1.0f, 1.0f, -1.0f, -1.0f, 1.0f, -1.0f,
     1.0f, -1.0f, -1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 1.0f, -1.0f,
     1.0f, -1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f, -1.0f,
     1.0f, -1.0f, 1.0f, -1.0f, -1.0f, 1.0f, 1.0f, 1.0f, 1.0f,
     -1.0f, -1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f,
     -1.0f, -1.0f, 1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 1.0f, 1.0f,
     -1.0f, -1.0f, -1.0f, -1.0f, 1.0f, -1.0f, -1.0f, 1.0f, 1.0f,
     -1.0f, -1.0f, 1.0f, 1.0f, -1.0f, 1.0f, 1.0f, -1.0f, -1.0f,
     1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 1.0f,
     -1.0f, 1.0f, -1.0f, 1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 1.0f,
     1.0f, 1.0f, 1.0f, -1.0f, 1.0f, 1.0f, -1.0f, 1.0f, -1.0f
    };

    glGenVertexArrays(numVAOs, vao);
    glBindVertexArray(vao[0]);
    glGenBuffers(numVBOs, vbo);

    glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertexPositions), vertexPositions, GL_STATIC_DRAW);

}

void init(GLFWwindow* window) {
    const char* vertShader = "color_cube_shader/vertShader.glsl";
    const char* fragShader = "color_cube_shader/fragShader.glsl";

    renderingProgram = Utils::createShaderProgram(vertShader, fragShader);
    cameraX = 0.0f; cameraY = 0.0f; cameraZ = 8.0f;
    cubeLocX = 0.0f; cubeLocY = -2.0f; cubeLocZ = 0.0f;  //desplaza hacia abajo Y para revelar la perspectiva
    setupVertices();
}

void display(GLFWwindow* window, double currentTime) {
    glClear(GL_DEPTH_BUFFER_BIT);
    glClear(GL_COLOR_BUFFER_BIT);
    glUseProgram(renderingProgram);

    mvLoc = glGetUniformLocation(renderingProgram, "mv_matrix");
    projLoc = glGetUniformLocation(renderingProgram, "proj_matrix");

    // construir la matriz de prespectiva
    glfwGetFramebufferSize(window, &width, &height);
    aspect = (float)width / (float)height;
```

```
67      pMat = glm::perspective(1.0472f, aspect, 0.1f, 1000.0f); // 1.0472
            radians = 60 degrees
68
69      // construir view matrix, model matrix, and model-view matrix
70      vMat = glm::translate(glm::mat4(1.0f), glm::vec3(-cameraX, -cameraY,
            -cameraZ));
71      //mMat = glm::translate(glm::mat4(1.0f), glm::vec3(cubeLocX, cubeLocY
            , cubeLocZ));
72
73      /// usar el tiempo actual para calcular diferentes traslaciones en x,
             y y z
74      tMat = glm::translate(glm::mat4(1.0f),
75      glm::vec3(sin(0.35f * currentTime) * 2.0f, cos(0.52f * currentTime) *
             2.0f, sin(0.7f * currentTime) * 2.0f));
76      rMat = glm::rotate(glm::mat4(1.0f), 1.75f * (float)currentTime, glm::
            vec3(0.0f, -1.0f, 0.0f));
77      rMat = glm::rotate(rMat, 1.75f * (float)currentTime, glm::vec3(1.0f,
            0.0f, 0.0f));
78      rMat = glm::rotate(rMat, 1.75f * (float)currentTime, glm::vec3(0.0f,
            0.0f, 1.0f));
79      // el 1.75 ajusta la velocidad de rotacion
80      mMat = tMat * rMat;
81      mvMat = vMat * mMat;
82
83      // copiar matrices de perspectiva y MV a las correspondientes
            variables uniformes
84      glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvMat));
85      glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(pMat));
86
87      // asociar VBO con el atributo de v[U+FFFD]rtice correspondiente en el
            sombreador de v[U+FFFD]rtices
88      glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
89      glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
90      glEnableVertexAttribArray(0);
91
92      // ajustar la configuraci[U+FFFD]n de OpenGL y dibujar el modelo
93      glEnable(GL_DEPTH_TEST);
94      glDepthFunc(GL_LEQUAL);
95      glDrawArrays(GL_TRIANGLES, 0, 36);
96  }
```

- Capturas



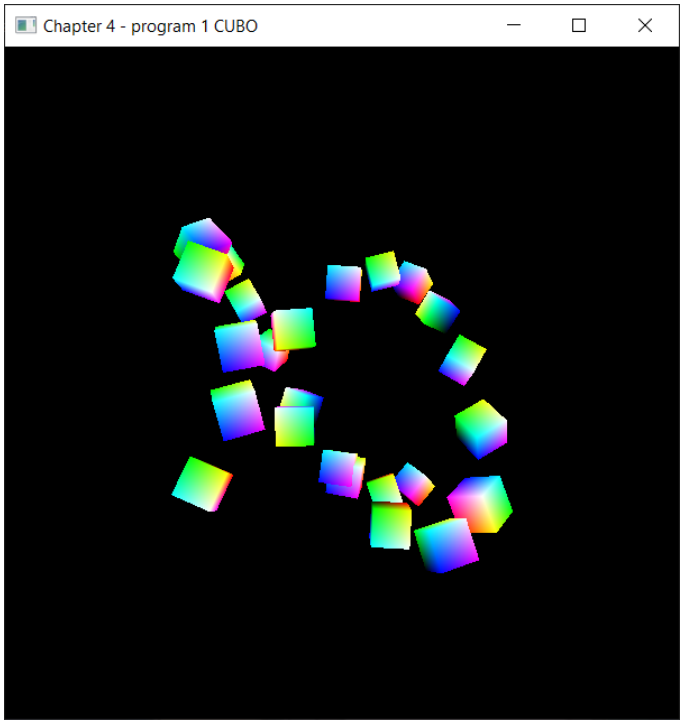


Y finalmente en la página 78, se simula el moviendo de 24 cubos creados dentro de una iteracion que crea 24 veces la matriz de model-view para cada cubo y que por cada uno aplica el metodo glDrawArrays(GL_TRIANGLES, 0, 36) para empezar a graficar.

```

#include "Utils.h"
using namespace std;

#define numVAOs 1
#define numVBOs 2

float cameraX, cameraY, cameraZ;
float cubeLocX, cubeLocY, cubeLocZ;

GLuint renderingProgram;
GLuint vao[numVAOs];
GLuint vbo[numVBOs];

GLuint mvLoc, projLoc;
int width, height;
float aspect;
glm::mat4 pMat, vMat, mMat, mvMat;
glm::mat4 tMat, rMat;

void setupVertices(void) {
    float vertexPositions[108] = {
     -1.0f, 1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 1.0f, -1.0f, -1.0f,//
        traingulo
     1.0f, -1.0f, -1.0f, 1.0f, 1.0f, -1.0f, -1.0f, 1.0f, -1.0f,
     1.0f, -1.0f, -1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 1.0f, -1.0f,
     1.0f, -1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f, -1.0f,
     1.0f, -1.0f, 1.0f, -1.0f, -1.0f, 1.0f, 1.0f, 1.0f, 1.0f,
     -1.0f, -1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f,
     -1.0f, -1.0f, 1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 1.0f, 1.0f,
     -1.0f, -1.0f, -1.0f, -1.0f, 1.0f, -1.0f, -1.0f, 1.0f, 1.0f,
     -1.0f, -1.0f, 1.0f, 1.0f, -1.0f, 1.0f, 1.0f, -1.0f, -1.0f,
     1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 1.0f,
     -1.0f, 1.0f, -1.0f, 1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 1.0f,
     1.0f, 1.0f, 1.0f, -1.0f, 1.0f, 1.0f, -1.0f, 1.0f, -1.0f
    };

    glGenVertexArrays(numVAOs, vao);
    glBindVertexArray(vao[0]);
    glGenBuffers(numVBOs, vbo);

    glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertexPositions),
       vertexPositions, GL_STATIC_DRAW);

}

void init(GLFWwindow* window) {
    const char* vertShader = "color_cube_shader/vertShader.glsl";
    const char* fragShader = "color_cube_shader/fragShader.glsl";

    renderingProgram = Utils::createShaderProgram(vertShader, fragShader)
```

```
        ;
51      cameraX = 0.0f; cameraY = 0.0f; cameraZ = 36.0f;
52      cubeLocX = 0.0f; cubeLocY = -2.0f; cubeLocZ = 0.0f;
53      setupVertices();
54  }
55
56  void display(GLFWwindow* window, double currentTime) {
57      glClear(GL_DEPTH_BUFFER_BIT);
58      glClear(GL_COLOR_BUFFER_BIT);
59      glUseProgram(renderingProgram);//instalando el [U+FFFD] GLSL en la GPU
            .
60
61
62
63      //obtener las variables uniformes para las matrices de MV y
            proyeccion
64      mvLoc = glGetUniformLocation(renderingProgram, "mv_matrix");
65      projLoc = glGetUniformLocation(renderingProgram, "proj_matrix");
66
67      // construir la matriz de prespectiva
68      glfwGetFramebufferSize(window, &width, &height);
69      aspect = (float)width / (float)height;
70      pMat = glm::perspective(1.0472f, aspect, 0.1f, 1000.0f); // 1.0472
            radians = 60 degrees
71
72      // construir view matrix, model matrix, and model-view matrix
73      vMat = glm::translate(glm::mat4(1.0f), glm::vec3(-cameraX, -cameraY,
            -cameraZ));
74
75      for (int i = 0; i < 24; i++)
76      {
77          float tf = currentTime + i; // tf == "time factor", declared as
                type float
78          tMat = glm::translate(glm::mat4(1.0f), glm::vec3(sin(.35f * tf) *
                 8.0f, cos(.52f * tf) * 8.0f,sin(.70f * tf) * 8.0f));
79          rMat = glm::rotate(glm::mat4(1.0f), 1.75f * tf, glm::vec3(0.0f,
                1.0f, 0.0f));
80          rMat = glm::rotate(rMat, 1.75f * tf, glm::vec3(1.0f, 0.0f, 0.0f))
                ;
81          rMat = glm::rotate(rMat, 1.75f * tf, glm::vec3(0.0f, 0.0f, 1.0f))
                ;
82          mMat = tMat * rMat;
83          mvMat = vMat * mMat;
84
85          glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvMat));
86          glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(pMat));
87          glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
88          glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
89          glEnableVertexAttribArray(0);
90          glEnable(GL_DEPTH_TEST);
91          glDepthFunc(GL_LEQUAL);
92          glDrawArrays(GL_TRIANGLES, 0, 36);
93      }
94  }
```

- Capturas



1.2. Programa 4.2: Instancing Twenty-Four Animated Cubes (pag. 80).

El primer paso que se debe hacer es realizar el mismo procedimiento que el programa anterior de la página 78 pero esta vez de realizarlo en un for de 24 repeticiones, todo lo haremos desde el "Vertex Shader" el cual ahí construiremos nuestras matrices M y V, entonces prácticamente todo lo que una vez estaba en el for de display() de main.cpp se albergará en "vertShader.gsls" por lo que ahí construiremos las matrices de rotación y de traslación como vemos aquí:

Matrix Vertex:

```
#version 430
layout (location=0) in vec3 position;

uniform mat4 m_matrix;
uniform mat4 v_matrix;

uniform mat4 proj_matrix;
uniform float tf;

out vec4 varyingColor;

mat4 buildRotateX(float rad);
mat4 buildRotateY(float rad);
mat4 buildRotateZ(float rad);
mat4 buildTranslate(float x, float y, float z);




void main(void)
```

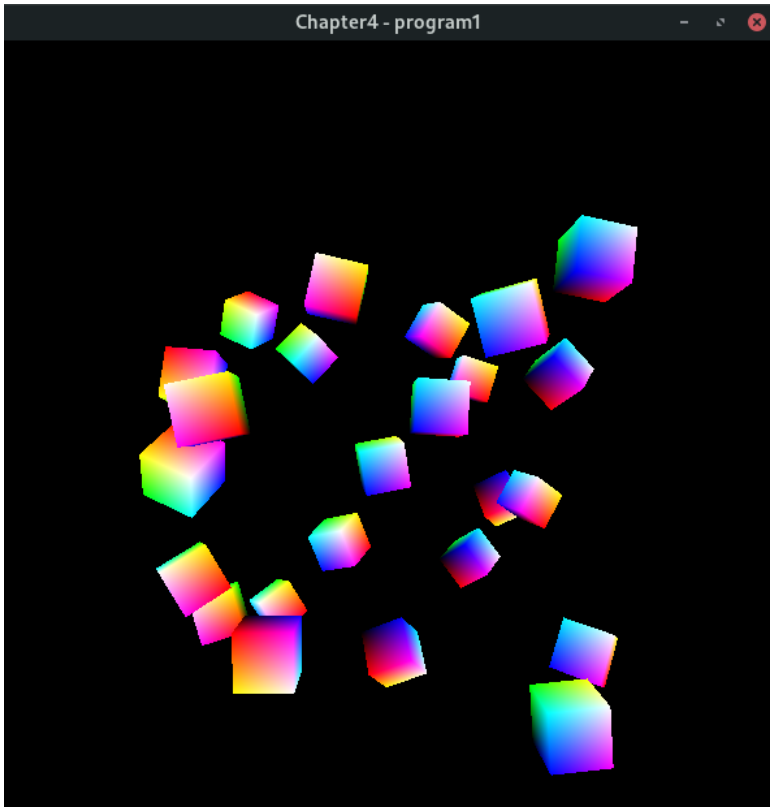
```
{
    float i = gl_InstanceID + tf/2;
    float a = sin(2.0 * i) * 8.0;
    float b = sin(3.0 * i) * 8.0;
    float c = sin(4.0 * i) * 8.0;

    // build the rotation and translation matrices to be applied to this
       model matrix
    mat4 localRotX = buildRotateX(1000*i);
    mat4 localRotY = buildRotateY(1000*i);
    mat4 localRotZ = buildRotateZ(1000*i);
    mat4 localTrans = buildTranslate(a,b,c);

    // build the model matrix and then the model-view matrix
    mat4 newM_matrix = m_matrix * localTrans * localRotX * localRotY *
       localRotZ;
    mat4 mv_matrix = v_matrix * newM_matrix;

    gl_Position = proj_matrix * mv_matrix * vec4(position,1.0);
    varyingColor = vec4(position,1.0)*0.5+vec4(0.5,0.5,0.5,0.5);
}

// utility function to build a translation matrix (from Chapter 3)
mat4 buildTranslate(float x, float y, float z)
{
    mat4 trans = mat4(1.0, 0.0, 0.0, 0.0,
                      0.0, 1.0, 0.0, 0.0,
                      0.0, 0.0, 1.0, 0.0,
                      x, y, z, 1.0 );
    return trans;
}

// builds and returns a matrix that performs a rotation around the X axis
mat4 buildRotateX(float rad)
{ mat4 xrot = mat4(1.0, 0.0, 0.0, 0.0,
                                  0.0, cos(rad), -sin(rad), 0.0,
                                  0.0, sin(rad), cos(rad), 0.0,
                                  0.0, 0.0, 0.0, 1.0 );
return xrot;
}
// builds and returns a matrix that performs a rotation around the Y axis
mat4 buildRotateY(float rad)
{ mat4 yrot = mat4(cos(rad), 0.0, sin(rad), 0.0,
                                  0.0, 1.0, 0.0, 0.0,
                                  -sin(rad), 0.0, cos(rad), 0.0,
                                  0.0, 0.0, 0.0, 1.0 );
return yrot;
}
// builds and returns a matrix that performs a rotation around the Z axis
mat4 buildRotateZ(float rad)
{ mat4 zrot = mat4(cos(rad), -sin(rad), 0.0, 0.0,
                                  sin(rad), cos(rad), 0.0, 0.0,
                                  0.0, 0.0, 1.0, 0.0,
                                  0.0, 0.0, 0.0, 1.0 );
return zrot;
}
```

Y en el "main.cpp"se borra el "glDrawArrays()"por "glDrawArraysInstanced(GL_TRIANGLES, 0, 36, 24);ż si nos fijamos en esta función colocaremos en la ultima variable el n°24 ya que vamos a dibujar 24 cubos, por lo cual lo unico que cambia en todo el codigo principal es la función display:

Función display en main.cpp

```
void display(GLFWwindow* window, double currentTime) {
        glClear(GL_DEPTH_BUFFER_BIT);
        glClear(GL_COLOR_BUFFER_BIT);

        glUseProgram(renderingProgram);
        // get the uniform variables for the MV and projection matrices
        mLoc = glGetUniformLocation(renderingProgram, "m_matrix");
        projLoc = glGetUniformLocation(renderingProgram, "proj_matrix");
        vLoc = glGetUniformLocation(renderingProgram, "v_matrix");
        // build perspective matrix
        glfwGetFramebufferSize(window, &width, &height);
        aspect = (float)width / (float)height;
        pMat = glm::perspective(1.0472f, aspect, 0.1f, 1000.0f); //
            1.0472 radians = 60 degrees
        // build view matrix, model matrix, and model-view matrix

        vMat = glm::translate(glm::mat4(1.0f), glm::vec3(-cameraX, -
            cameraY, -cameraZ));
        mMat = glm::translate(glm::mat4(1.0f), glm::vec3(cubeLocX,
            cubeLocY, cubeLocZ));
        //mvMat = vMat * mMat;

        glUniformMatrix4fv(vLoc, 1, GL_FALSE, glm::value_ptr(vMat));
        glUniformMatrix4fv(mLoc, 1, GL_FALSE, glm::value_ptr(mMat));
        float timeFactor = ((float)currentTime);
        tfLoc = glGetUniformLocation(renderingProgram, "tf");
        glUniform1f(tfLoc, (float)timeFactor);
        glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(pMat));
        // associate VBO with the corresponding vertex attribute in the
            vertex shader
        glBindBuffer(GL_ARRAY_BUFFER,vbo[0]);
        glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
        glEnableVertexAttribArray(0);
        // adjust OpenGL settings and draw model
        glEnable(GL_DEPTH_TEST);
        glDepthFunc(GL_LEQUAL);
        glDrawArraysInstanced(GL_TRIANGLES, 0, 36,24);


}
```

Al final va quedar igual que la modificacion del primer programa osea:
Resultados:



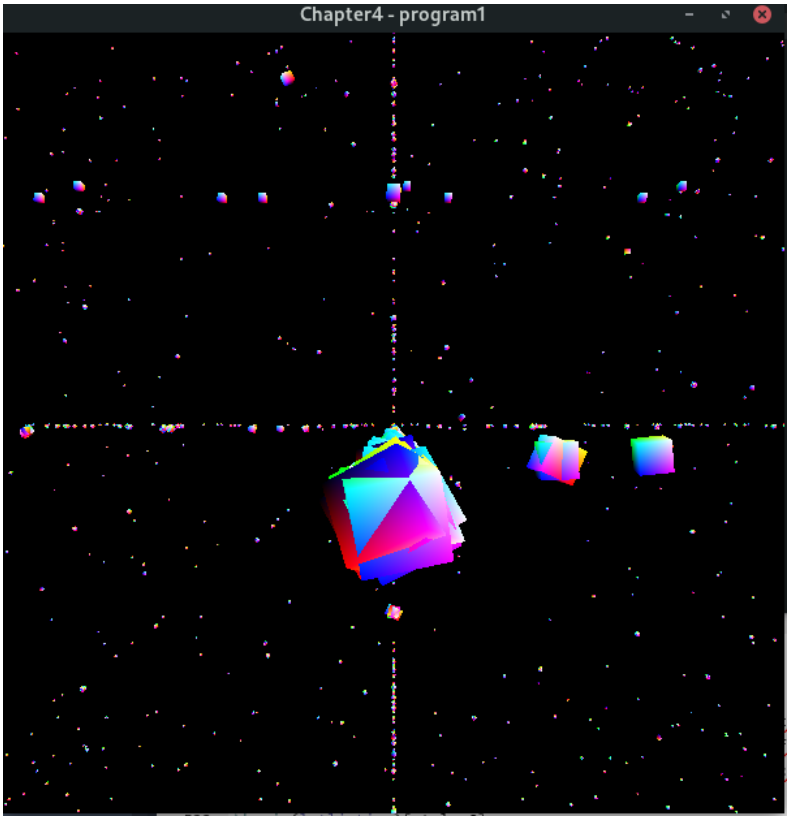
Luego nos piden una modificacíon del programa el cual vamos a dibujar 100000 cubos y para ello modifcamos la función "glDrawArraysInstanced(GL_TRIANGLES, 0, 36,24).ª "glDrawArraysInstanced(GL_TRIANGLES, 0, 36,100000).ªdemás de modificar el Vertex Shaders a:

```
29  #version 430
30  layout (location=0) in vec3 position;
31
32  uniform mat4 m_matrix;
33  uniform mat4 v_matrix;
34
35  uniform mat4 proj_matrix;
36  uniform float tf;
37
38  out vec4 varyingColor;
39
40  mat4 buildRotateX(float rad);
41  mat4 buildRotateY(float rad);
42  mat4 buildRotateZ(float rad);
43  mat4 buildTranslate(float x, float y, float z);
44
45
46
```

```
47
48  void main(void)
49  {
50      float i = gl_InstanceID + tf/2;
51      float a = sin(203.0 * i) * 403.0;
52      float b = sin(301.0 * i) * 401.0;
53      float c = sin(400.0 * i) * 405.0;
54
55      //build the rotation and translation matrices to be applied to this
            cube's model matrix
56      mat4 localRotX = buildRotateX(1000*i);
57      mat4 localRotY = buildRotateY(1000*i);
58      mat4 localRotZ = buildRotateZ(1000*i);
59      mat4 localTrans = buildTranslate(a,b,c);
60
61      // build the model matrix and then the model-view matrix
62      mat4 newM_matrix = m_matrix * localTrans * localRotX * localRotY *
            localRotZ;
63      mat4 mv_matrix = v_matrix * newM_matrix;
64
65      gl_Position = proj_matrix * mv_matrix * vec4(position,1.0);
66      varyingColor = vec4(position,1.0)*0.5+vec4(0.5,0.5,0.5,0.5);
67  }
68
69  // utility function to build a translation matrix (from Chapter 3)
70  mat4 buildTranslate(float x, float y, float z)
71  {
72      mat4 trans = mat4(1.0, 0.0, 0.0, 0.0,
73                                  0.0, 1.0, 0.0, 0.0,
74                                  0.0, 0.0, 1.0, 0.0,
75                                  x, y, z, 1.0 );
76      return trans;
77  }
78
79  // builds and returns a matrix that performs a rotation around the X axis
80  mat4 buildRotateX(float rad)
81  { mat4 xrot = mat4(1.0, 0.0, 0.0, 0.0,
82                                          0.0, cos(rad), -sin(rad), 0.0,
83                                          0.0, sin(rad), cos(rad), 0.0,
84                                          0.0, 0.0, 0.0, 1.0 );
85  return xrot;
86  }
87  // builds and returns a matrix that performs a rotation around the Y axis
88  mat4 buildRotateY(float rad)
89  { mat4 yrot = mat4(cos(rad), 0.0, sin(rad), 0.0,
90                                          0.0, 1.0, 0.0, 0.0,
91                                          -sin(rad), 0.0, cos(rad), 0.0,
92                                          0.0, 0.0, 0.0, 1.0 );
93  return yrot;
94  }
95  // builds and returns a matrix that performs a rotation around the Z axis
96  mat4 buildRotateZ(float rad)
97  { mat4 zrot = mat4(cos(rad), -sin(rad), 0.0, 0.0,
98                                          sin(rad), cos(rad), 0.0, 0.0,
99                                          0.0, 0.0, 1.0, 0.0,
100                                         0.0, 0.0, 0.0, 1.0 );
```

```
101   return zrot;
102   }
```

Resultados:



1.3. Programa 4.3: Cube and Pyramid (pag. 83).

Esta sección habla de como renderizar mas de un modelo a la vez en una sola escena ,donde para cada modelo se necesitaría una nueva matriz y llamadas separadas a varias funciones por cada modelo e incluso el uso de shaders distintos si estos modelos son demasiado diferentes como su construcción base o algunos efectos complicados como por ejemplo la luz ,etc.

```
#include "Utils.h"
using namespace std;

#define numVAOs 1
#define numVBOs 2

float cameraX, cameraY, cameraZ;
float cubeLocX, cubeLocY, cubeLocZ;
float pyrLocX, pyrLocY, pyrLocZ;
GLuint renderingProgram;
GLuint vao[numVAOs];
GLuint vbo[numVBOs];


GLuint mvLoc, tfLoc, projLoc;
int width, height, displayLoopi;
float aspect;
float timeFactor;
glm::mat4 pMat, vMat, tMat, rMat, mMat, mvMat;

void setupVertices(void) {    // 12 triangulos * 3 vertices * 3 valores (
    x, y, z)

    float cubePositions[108] = {
        -1.0f,  1.0f, -1.0f, -1.0f, -1.0f, -1.0f,  1.0f, -1.0f, -1.0f,
         1.0f, -1.0f, -1.0f,  1.0f,  1.0f, -1.0f, -1.0f,  1.0f, -1.0f,
         1.0f, -1.0f, -1.0f,  1.0f, -1.0f,  1.0f,  1.0f,  1.0f, -1.0f,
         1.0f, -1.0f,  1.0f,  1.0f,  1.0f,  1.0f,  1.0f,  1.0f, -1.0f,
         1.0f, -1.0f,  1.0f, -1.0f, -1.0f,  1.0f,  1.0f,  1.0f,  1.0f,
        -1.0f, -1.0f,  1.0f, -1.0f,  1.0f,  1.0f,  1.0f,  1.0f,  1.0f,
        -1.0f, -1.0f,  1.0f, -1.0f, -1.0f, -1.0f, -1.0f,  1.0f,  1.0f,
        -1.0f, -1.0f, -1.0f, -1.0f,  1.0f, -1.0f, -1.0f,  1.0f,  1.0f,
        -1.0f, -1.0f,  1.0f,  1.0f, -1.0f,  1.0f,  1.0f, -1.0f, -1.0f,
         1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f,  1.0f,
        -1.0f,  1.0f, -1.0f,  1.0f,  1.0f, -1.0f,  1.0f,  1.0f,  1.0f,
         1.0f,  1.0f,  1.0f, -1.0f,  1.0f,  1.0f, -1.0f,  1.0f, -1.0f,
    };
    // 6 triangulos * 3 vertices * 3 valores
    float pyramidPositions[54] = {
        -1.0f, -1.0f,  1.0f,  1.0f, -1.0f,  1.0f,  0.0f,  1.0f,  0.0f,
         1.0f, -1.0f,  1.0f,  1.0f, -1.0f, -1.0f,  0.0f,  1.0f,  0.0f,
         1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f,  0.0f,  1.0f,  0.0f,
        -1.0f, -1.0f, -1.0f, -1.0f, -1.0f,  1.0f,  0.0f,  1.0f,  0.0f,
        -1.0f, -1.0f, -1.0f,  1.0f, -1.0f,  1.0f, -1.0f, -1.0f,  1.0f,
         1.0f, -1.0f,  1.0f, -1.0f, -1.0f, -1.0f,  1.0f, -1.0f, -1.0f
    };

```

```
47
48      glGenVertexArrays(numVAOs, vao);
49      glBindVertexArray(vao[0]);
50      glGenBuffers(numVBOs, vbo);
51
52      glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
53      glBufferData(GL_ARRAY_BUFFER, sizeof(cubePositions), cubePositions,
            GL_STATIC_DRAW);
54
55      glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
56      glBufferData(GL_ARRAY_BUFFER, sizeof(pyramidPositions),
            pyramidPositions, GL_STATIC_DRAW);
57  }
58
59
60  void init(GLFWwindow* window) {
61      renderingProgram = Utils::createShaderProgram(
62          "C:/Users/estilos/source/repos/Grafica_opengl/glsl/vertexShader.
                glsl",
63          "C:/Users/estilos/source/repos/Grafica_opengl/glsl/fragShader.
                glsl");
64
65      glfwGetFramebufferSize(window, &width, &height);
66      aspect = (float)width / (float)height;
67      pMat = glm::perspective(1.0472f, aspect, 0.1f, 1000.0f); // 1.0472
            radians == 60 degrees
68
69      // position the camera further down the positive Z axis (to see all
            of the cubes)
70      cameraX = 0.0f; cameraY = 0.0f; cameraZ = 8.0f;
71      cubeLocX = 0.0f; cubeLocY = -2.0f; cubeLocZ = 0.0f;
72      pyrLocX = 2.0f; pyrLocY = 2.0f; pyrLocZ = 0.0f;
73      setupVertices();
74  }
75
76  void display(GLFWwindow* window, double currentTime) {
77      glClear(GL_DEPTH_BUFFER_BIT);
78      glClearColor(0.0, 0.0, 0.0, 1.0);
79      glClear(GL_COLOR_BUFFER_BIT);
80
81      glUseProgram(renderingProgram);
82      projLoc = glGetUniformLocation(renderingProgram, "proj_matrix");
83      mvLoc = glGetUniformLocation(renderingProgram, "mv_matrix");
84
85      vMat = glm::translate(glm::mat4(1.0f), glm::vec3(-cameraX, -cameraY,
            -cameraZ));
86
87
88      mMat = glm::translate(glm::mat4(1.0f), glm::vec3(cubeLocX, cubeLocY,
            cubeLocZ));
89      mvMat = vMat * mMat;
90
91      glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(pMat));
92      glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvMat));
93
94      glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
```

```
95      glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
96      glEnableVertexAttribArray(0);
97
98      glEnable(GL_DEPTH_TEST);
99      glDepthFunc(GL_LEQUAL);
100     glDrawArrays(GL_TRIANGLES, 0, 36);
101
102
103     mMat = glm::translate(glm::mat4(1.0f), glm::vec3(pyrLocX, pyrLocY,
            pyrLocZ));
104     mvMat = vMat * mMat;
105
106     glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(pMat));
107     glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvMat));
108
109     glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
110     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
111     glEnableVertexAttribArray(0);
112
113     glEnable(GL_DEPTH_TEST);
114     glDepthFunc(GL_LEQUAL);
115     glDrawArrays(GL_TRIANGLES, 0, 18);
116 }
117
118 int main(void) {
119     if (!glfwInit()) { exit(EXIT_FAILURE); }
120     glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
121     glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 1);
122     GLFWwindow* window = glfwCreateWindow(600, 600, "Chapter4 - cubo
            piramide", nullptr, nullptr);
123     glfwMakeContextCurrent(window);
124     if (glewInit() != GLEW_OK) { exit(EXIT_FAILURE); }
125     glfwSwapInterval(1);
126
127     init(window);
128     while (!glfwWindowShouldClose(window)) {
129         display(window, glfwGetTime());
130         glfwSwapBuffers(window);
131         glfwPollEvents();
132     }
133
134     glfwDestroyWindow(window);
135     glfwTerminate();
136     exit(EXIT_SUCCESS);
137 }
```

Captura:

1.4. Programa 4.4: Solar System (pag. 90).

A diferencia de los anteriores ejercicios, este usa una estructura de tipo pila, la cual ayuda a almacenar las matrices de transformacion de cada objeto, que posteriormente se iran dibujando segun como se realiza las operaciones en la pila.

```
#include "Utils.h"
#include <stack>
using namespace std;

#define numVAOs 1
#define numVBOs 2

float cameraX, cameraY, cameraZ;
float cubeLocX, cubeLocY, cubeLocZ;
float pyrLocX, pyrLocY, pyrLocZ;

GLuint renderingProgram;
GLuint vao[numVAOs];
GLuint vbo[numVBOs];

GLuint mvLoc, projLoc;
int width, height;
float aspect;
glm::mat4 pMat, vMat, mMat, mvMat;
stack<glm::mat4> mvStack;

void setupVertices(void) { // 36 vertices, 12 triangulos, hace un cubo de 2x2x2 colocado en el origen
    float cubePositions[108] = {
     -1.0f, 1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 1.0f, -1.0f, -1.0f,//traingulo
     1.0f, -1.0f, -1.0f, 1.0f, 1.0f, -1.0f, -1.0f, 1.0f, -1.0f,
     1.0f, -1.0f, -1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 1.0f, -1.0f,
     1.0f, -1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f, -1.0f,
     1.0f, -1.0f, 1.0f, -1.0f, -1.0f, 1.0f, 1.0f, 1.0f, 1.0f,
     -1.0f, -1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f,
     -1.0f, -1.0f, 1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 1.0f, 1.0f,
     -1.0f, -1.0f, -1.0f, -1.0f, 1.0f, -1.0f, -1.0f, 1.0f, 1.0f,
     -1.0f, -1.0f, 1.0f, 1.0f, -1.0f, 1.0f, 1.0f, -1.0f, -1.0f,
     1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 1.0f,
     -1.0f, 1.0f, -1.0f, 1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 1.0f,
     1.0f, 1.0f, 1.0f, -1.0f, 1.0f, 1.0f, -1.0f, 1.0f, -1.0f
    };

    // pyramid with 18 vertices, comprising 6 triangles (four sides, and two on the bottom)
    float pyramidPositions[54] =
    {
        -1.0f, -1.0f, 1.0f, 1.0f, -1.0f, 1.0f, 0.0f, 1.0f, 0.0f, // front face
        1.0f, -1.0f, 1.0f, 1.0f, -1.0f, -1.0f, 0.0f, 1.0f, 0.0f, // right face
         1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 0.0f, 1.0f, 0.0f, //back face
        -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 1.0f, 0.0f, 1.0f, 0.0f, //
```

```
            left face
        -1.0f, -1.0f, -1.0f, 1.0f, -1.0f, 1.0f, -1.0f, -1.0f, 1.0f, //
            bas[U+FFFD]left front
        1.0f, -1.0f, 1.0f, -1.0f, -1.0f, -1.0f, 1.0f, -1.0f, -1.0f //
            bas[U+FFFD]right back
    };


    glGenVertexArrays(numVAOs, vao);// we need at least 1 VAO
    glBindVertexArray(vao[0]);
    glGenBuffers(numVBOs, vbo);// we need at least 2 VBOs

    glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(cubePositions), cubePositions,
        GL_STATIC_DRAW);

    glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(pyramidPositions),
        pyramidPositions, GL_STATIC_DRAW);

}

void init(GLFWwindow* window) {
    const char* vertShader = "color_cube_shader/vertShader.glsl";
    const char* fragShader = "color_cube_shader/fragShader.glsl";

    renderingProgram = Utils::createShaderProgram(vertShader, fragShader)
        ;
    cameraX = 0.0f; cameraY = 0.0f; cameraZ = 12.0f;
    cubeLocX = 0.0f; cubeLocY = -2.0f; cubeLocZ = 0.0f;
    pyrLocX = 3.0f; pyrLocY = 2.0f; pyrLocZ = 0.0f;
    setupVertices();
}

void display(GLFWwindow* window, double currentTime) {
    glClear(GL_DEPTH_BUFFER_BIT);
    glClear(GL_COLOR_BUFFER_BIT);
    glUseProgram(renderingProgram);
    //obtener las variables uniformes para las matrices de MV y
        proyeccion
    mvLoc = glGetUniformLocation(renderingProgram, "mv_matrix");
    projLoc = glGetUniformLocation(renderingProgram, "proj_matrix");

    // construir la matriz de prespectiva
    glfwGetFramebufferSize(window, &width, &height);
    aspect = (float)width / (float)height;
    pMat = glm::perspective(1.0472f, aspect, 0.1f, 1000.0f); // 1.0472
        radians = 60 degrees

    // push view matrix onto the stack
    vMat = glm::translate(glm::mat4(1.0f), glm::vec3(-cameraX, -cameraY,
        -cameraZ));
    mvStack.push(vMat);

    //matriz de proyeccion de perspectiva
    glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(pMat));
```

```
92
93
94      // ---------------------- pyramid == sun
            ----------------------------------------------
95      mvStack.push(mvStack.top());
96      mvStack.top() *= glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 0.0f
            , 0.0f)); // sun position
97      mvStack.push(mvStack.top());
98      mvStack.top() *= glm::rotate(glm::mat4(1.0f), (float)currentTime, glm
            ::vec3(1.0f, 0.0f, 0.0f));
99      // sun rotation
100
101     glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvStack.top()))
            ;
102
103     glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
104
105     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
106     glEnableVertexAttribArray(0);
107     glEnable(GL_DEPTH_TEST);
108     glEnable(GL_LEQUAL);
109     glDrawArrays(GL_TRIANGLES, 0, 18); // draw the sun
110     mvStack.pop();
111
112     //----------------------- cube == planet
            ----------------------------------------------
113     mvStack.push(mvStack.top());
114     mvStack.top() *= glm::translate(glm::mat4(1.0f), glm::vec3(sin((float
            )currentTime) * 4.0, 0.0f, cos((float)currentTime) * 4.0));
115     mvStack.push(mvStack.top());
116     mvStack.top() *= glm::rotate(glm::mat4(1.0f), (float)currentTime, glm
            ::vec3(0.0, 1.0, 0.0));
117     // planet rotation
118     glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvStack.top()))
            ;
119
120     glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
121     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
122     glEnableVertexAttribArray(0);
123     glDrawArrays(GL_TRIANGLES, 0, 36); // draw the planet
124
125     mvStack.pop(); // remove the planet's axial rotation from the stack
126
127     //----------------------- smaller cube == moon
            ----------------------------------
128     mvStack.push(mvStack.top());
129     mvStack.top() *= glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, sin
            ((float)currentTime) * 2.0,cos((float)currentTime) * 2.0));
130     mvStack.top() *= glm::rotate(glm::mat4(1.0f), (float)currentTime, glm
            ::vec3(0.0, 0.0, 1.0));
131     // moon rotation
132     mvStack.top() *= glm::scale(glm::mat4(1.0f), glm::vec3(0.25f, 0.25f,
            0.25f)); // make the moon smaller
133     glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvStack.top()))
            ;
134
```

```
135     glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
136     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
137     glEnableVertexAttribArray(0);
138     glDrawArrays(GL_TRIANGLES, 0, 36); // draw the moon
139 
140     // remove moon scale/rotation/position, planet position, sun position
            , and view matrices from stack
141     mvStack.pop(); mvStack.pop(); mvStack.pop(); mvStack.pop();
142 }
143 
144 int main(void) {
145     if (!glfwInit()) { exit(EXIT_FAILURE); }
146     glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
147     glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
148     GLFWwindow* window = glfwCreateWindow(600, 600, "Chapter 4 - program
            1 CUBO", NULL, NULL);
149     glfwMakeContextCurrent(window);
150     if (glewInit() != GLEW_OK) { exit(EXIT_FAILURE); }
151     glfwSwapInterval(1);
152     init(window);
153     while (!glfwWindowShouldClose(window)) {
154         display(window, glfwGetTime());
155         glfwSwapBuffers(window);
156         glfwPollEvents();
157     }
158     glfwDestroyWindow(window);
159     glfwTerminate();
160     exit(EXIT_SUCCESS);
161 }
```

1.5. Código Vertex Shader

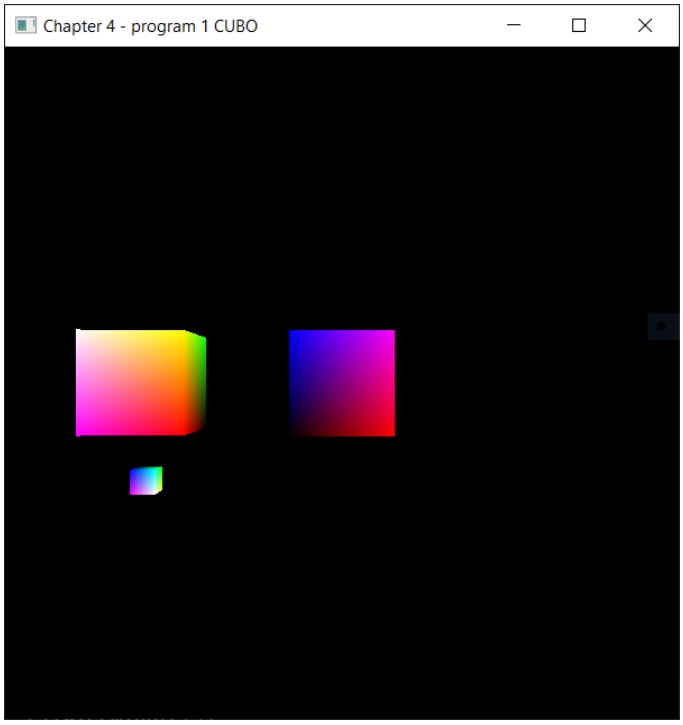
```
1  #version 430
2 
3  layout (location=0) in vec3 position;
4 
5  uniform mat4 mv_matrix;
6  uniform mat4 proj_matrix;
7 
8  out vec4 varyingColor;
9 
10 void main(void)
11 {
12     gl_Position = proj_matrix * mv_matrix * vec4(position,1.0);
13     varyingColor = vec4(position,1.0) * 0.5 + vec4(0.5, 0.5, 0.5, 0.5);
14 }
```

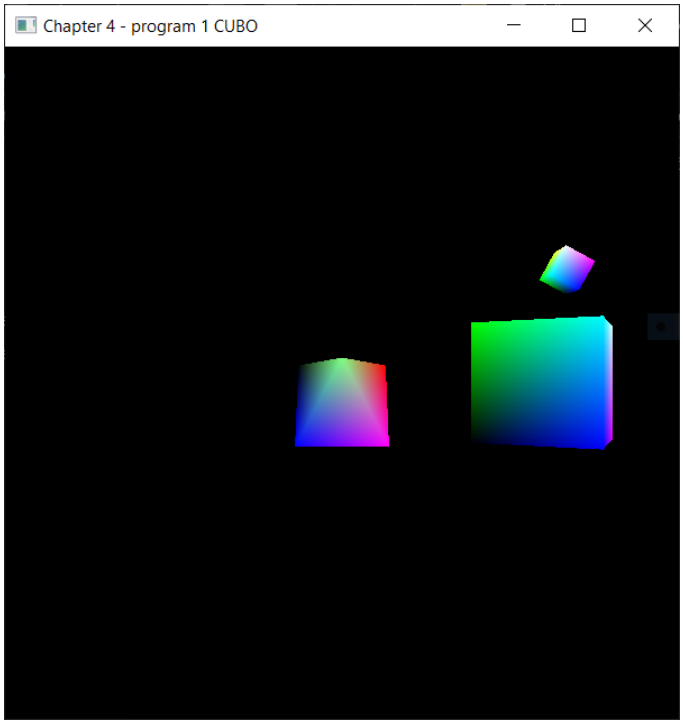
1.6. Código Fragment Shader

```
1  #version 430
2 
```

```
in vec4 varyingColor;

out vec4 color;

uniform mat4 mv_matrix;
uniform mat4 proj_matrix;

void main(void)
{
    color = varyingColor;
}
```

1.7. Capturas





2. Modifique el programa 4.4, para agregar un segundo planeta con un satélite.

En este ejercicio se utilizo la pila de tal manera que se creo un nuevo planeta y su satélite con una matriz de modelo tanto del plantea como del satelite.De la siguiente forma

```
#include "Utils.h"
#include <stack>
using namespace std;

#define numVAOs 1
#define numVBOs 2

float cameraX, cameraY, cameraZ;
float cubeLocX, cubeLocY, cubeLocZ;
float pyrLocX, pyrLocY, pyrLocZ;

GLuint renderingProgram;
GLuint vao[numVAOs];
GLuint vbo[numVBOs];

GLuint mvLoc, projLoc;
int width, height;
float aspect;
glm::mat4 pMat, vMat, mMat, mvMat;
stack<glm::mat4> mvStack;

void setupVertices(void) { // 36 vértices, 12 triángulos, hace un cubo
    de 2x2x2 colocado en el origen
    float cubePositions[108] = {
     -1.0f, 1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 1.0f, -1.0f, -1.0f,//
```

```
        traingulo
25      1.0f, -1.0f, -1.0f, 1.0f, 1.0f, -1.0f, -1.0f, 1.0f, -1.0f,
26      1.0f, -1.0f, -1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 1.0f, -1.0f,
27      1.0f, -1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f, -1.0f,
28      1.0f, -1.0f, 1.0f, -1.0f, -1.0f, 1.0f, 1.0f, 1.0f, 1.0f,
29      -1.0f, -1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f,
30      -1.0f, -1.0f, 1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 1.0f, 1.0f,
31      -1.0f, -1.0f, -1.0f, -1.0f, 1.0f, -1.0f, -1.0f, 1.0f, 1.0f,
32      -1.0f, -1.0f, 1.0f, 1.0f, -1.0f, 1.0f, 1.0f, -1.0f, -1.0f,
33      1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 1.0f,
34      -1.0f, 1.0f, -1.0f, 1.0f, 1.0f, -1.0f, 1.0f, 1.0f, 1.0f,
35      1.0f, 1.0f, 1.0f, -1.0f, 1.0f, 1.0f, -1.0f, 1.0f, -1.0f
36    };
37
38    // pyramid with 18 vertices, comprising 6 triangles (four sides, and
          two on the bottom)
39    float pyramidPositions[54] =
40    {
41        -1.0f, -1.0f, 1.0f, 1.0f, -1.0f, 1.0f, 0.0f, 1.0f, 0.0f, // front
              face
42        1.0f, -1.0f, 1.0f, 1.0f, -1.0f, -1.0f, 0.0f, 1.0f, 0.0f, // right
              face
43         1.0f, -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 0.0f, 1.0f, 0.0f, //
              back face
44        -1.0f, -1.0f, -1.0f, -1.0f, -1.0f, 1.0f, 0.0f, 1.0f, 0.0f, //
              left face
45        -1.0f, -1.0f, -1.0f, 1.0f, -1.0f, 1.0f, -1.0f, -1.0f, 1.0f, //
              bas\uFFFDleft front
46        1.0f, -1.0f, 1.0f, -1.0f, -1.0f, -1.0f, 1.0f, -1.0f, -1.0f //
              bas\uFFFDright back
47    };
48
49
50    glGenVertexArrays(numVAOs, vao);// we need at least 1 VAO
51    glBindVertexArray(vao[0]);
52    glGenBuffers(numVBOs, vbo);// we need at least 2 VBOs
53
54    glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
55    glBufferData(GL_ARRAY_BUFFER, sizeof(cubePositions), cubePositions,
          GL_STATIC_DRAW);
56
57    glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
58    glBufferData(GL_ARRAY_BUFFER, sizeof(pyramidPositions),
          pyramidPositions, GL_STATIC_DRAW);
59
60  }
61
62  void init(GLFWwindow* window) {
63    const char* vertShader = "color_cube_shader/vertShader.glsl";
64    const char* fragShader = "color_cube_shader/fragShader.glsl";
65
66    renderingProgram = Utils::createShaderProgram(vertShader, fragShader)
          ;
67    cameraX = 0.0f; cameraY = 0.0f; cameraZ = 12.0f;
68    cubeLocX = 0.0f; cubeLocY = -2.0f; cubeLocZ = 0.0f;
69    pyrLocX = 3.0f; pyrLocY = 2.0f; pyrLocZ = 0.0f;
```

```
70      setupVertices();
71  }
72
73  void display(GLFWwindow* window, double currentTime) {
74      glClear(GL_DEPTH_BUFFER_BIT);
75      glClear(GL_COLOR_BUFFER_BIT);
76      glUseProgram(renderingProgram);
77      //obtener las variables uniformes para las matrices de MV y
            proyeccion
78      mvLoc = glGetUniformLocation(renderingProgram, "mv_matrix");
79      projLoc = glGetUniformLocation(renderingProgram, "proj_matrix");
80
81      // construir la matriz de prespectiva
82      glfwGetFramebufferSize(window, &width, &height);
83      aspect = (float)width / (float)height;
84      pMat = glm::perspective(1.0472f, aspect, 0.1f, 1000.0f); // 1.0472
            radians = 60 degrees
85
86      // push view matrix onto the stack
87      vMat = glm::translate(glm::mat4(1.0f), glm::vec3(-cameraX, -cameraY,
            -cameraZ));
88      mvStack.push(vMat);
89
90      //matriz de proyeccion de perspectiva
91      glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(pMat));
92
93
94      // ---------------------- pyramid == sun
            ----------------------------------------
95      mvStack.push(mvStack.top());
96      mvStack.top() *= glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 0.0f
            , 0.0f)); // sun position
97      mvStack.push(mvStack.top());
98      mvStack.top() *= glm::rotate(glm::mat4(1.0f), (float)currentTime, glm
            ::vec3(1.0f, 0.0f, 0.0f));
99      // sun rotation
100
101     glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvStack.top()))
            ;
102
103     glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
104
105     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
106     glEnableVertexAttribArray(0);
107     glEnable(GL_DEPTH_TEST);
108     glEnable(GL_LEQUAL);
109     glDrawArrays(GL_TRIANGLES, 0, 18); // draw the sun
110     mvStack.pop();
111
112     //---------------------- cube == planet
            ----------------------------------------
113     mvStack.push(mvStack.top());
114     mvStack.top() *= glm::translate(glm::mat4(1.0f), glm::vec3(sin((float
            )currentTime) * 4.0, 0.0f, cos((float)currentTime) * 4.0));
115     mvStack.push(mvStack.top());
116     mvStack.top() *= glm::rotate(glm::mat4(1.0f), (float)currentTime, glm
```

```
        ::vec3(0.0, 1.0, 0.0));
117     // planet rotation
118     glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvStack.top()))
        ;
119
120     glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
121     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
122     glEnableVertexAttribArray(0);
123     glDrawArrays(GL_TRIANGLES, 0, 36); // draw the planet
124
125     mvStack.pop();
126
127     //---------------------- smaller cube == moon
        ----------------------------------
128     mvStack.push(mvStack.top());
129     mvStack.top() *= glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, sin
        ((float)currentTime) * 2.0,cos((float)currentTime) * 2.0));
130     mvStack.top() *= glm::rotate(glm::mat4(1.0f), (float)currentTime, glm
        ::vec3(0.0, 0.0, 1.0));
131     // moon rotation
132     mvStack.top() *= glm::scale(glm::mat4(1.0f), glm::vec3(0.25f, 0.25f,
        0.25f)); // make the moon smaller
133     glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvStack.top()))
        ;
134
135     glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
136     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
137     glEnableVertexAttribArray(0);
138     glDrawArrays(GL_TRIANGLES, 0, 36); // draw the moon
139
140     // remove moon scale/rotation/position, planet position
141     mvStack.pop(); mvStack.pop();
142
143     //---------------------- cube == planet
        2-------------------------------------------
144     mvStack.push(mvStack.top());
145
146     mvStack.top() *= glm::translate(glm::mat4(1.0f), glm::vec3(sin((float
        )currentTime + 3.50f) * 10.0f, 0.0f, cos((float)currentTime + 3.50
        f) * 10.0f));
147     mvStack.push(mvStack.top());
148     mvStack.top() *= glm::rotate(glm::mat4(1.0f), (float)currentTime, glm
        ::vec3(0.0, 1.0, 0.0));
149     mvStack.top() *= glm::scale(glm::mat4(1.0f), glm::vec3(1.75f, 1.75f,
        1.75f)); // make the planet bigger
150
151     // planet rotation
152     glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvStack.top()))
        ;
153
154     glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
155     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
156     glEnableVertexAttribArray(0);
157     glDrawArrays(GL_TRIANGLES, 0, 36); // draw the planet
158
159     mvStack.pop(); // remove the pla[U+FFFD]s axial rotation from the stack
```

```
160
161     //---------------------- smaller cube == moon
            2---------------------------------
162     mvStack.push(mvStack.top());
163     mvStack.top() *= glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, sin
            ((float)currentTime) * 3.0, cos((float)currentTime) * 3.0));
164     mvStack.top() *= glm::rotate(glm::mat4(1.0f), (float)currentTime, glm
            ::vec3(0.0, 0.0, 1.0));
165     // moon rotation
166     mvStack.top() *= glm::scale(glm::mat4(1.0f), glm::vec3(0.25f, 0.25f,
            0.25f)); // make the moon smaller
167     glUniformMatrix4fv(mvLoc, 1, GL_FALSE, glm::value_ptr(mvStack.top()))
            ;
168
169     glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
170     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
171     glEnableVertexAttribArray(0);
172     glDrawArrays(GL_TRIANGLES, 0, 36); // draw the moon
173     // remove moon scale/rotation/position, planet position, sun position
            , and view matrices from stack
174     mvStack.pop(); mvStack.pop(); mvStack.pop(); mvStack.pop();
175 }
```

2.1. Código Vertex Shader

```
1  #version 430
2
3  layout (location=0) in vec3 position;
4
5  uniform mat4 mv_matrix;
6  uniform mat4 proj_matrix;
7
8  out vec4 varyingColor;
9
10 void main(void)
11 {
12     gl_Position = proj_matrix * mv_matrix * vec4(position,1.0);
13     varyingColor = vec4(position,1.0) * 0.5 + vec4(0.5, 0.5, 0.5, 0.5);
14 }
```

2.2. Código Fragment Shader

```
1  #version 430
2
3  in vec4 varyingColor;
4
5  out vec4 color;
6
7  uniform mat4 mv_matrix;
8  uniform mat4 proj_matrix;
9
```

```
10  void main(void)
11  {
12      color = varyingColor;
13  }
```

2.3. Capturas