Cubo OpenGL

Main.cpp

```
#include <GL/freeglut.h>

#include <stdio.h>

#include <stdlib.h>

#include <iostream>

#include "Mesh.h"

#include "Vector3D.h"

#include <vector>


// Lista de colores RGB representados como Vector3D

std::vector<Vector3D> colorList = {

        Vector3D(1.0f, 0.0f, 0.0f), // Rojo

        Vector3D(0.0f, 1.0f, 0.0f), // Verde

        Vector3D(0.0f, 0.0f, 1.0f), // Azul

        Vector3D(1.0f, 1.0f, 0.0f), // Amarillo

        Vector3D(0.0f, 1.0f, 1.0f), // Cian

        Vector3D(1.0f, 0.0f, 1.0f), // Magenta

        Vector3D(0.5f, 0.5f, 0.5f), // Gris

        Vector3D(1.0f, 0.5f, 0.0f), // Naranja

        Vector3D(0.5f, 0.0f, 0.5f), // Púrpura

        Vector3D(0.0f, 0.5f, 0.0f),  // Verde oscuro

        Vector3D(1.0f, 0.8f, 0.0f), // Amarillo claro

        Vector3D(1.0f, 1.0f, 1.0f), // Blanco

        Vector3D(0.5f, 0.25f, 0.0f), // Marrón

        Vector3D(0.75f, 0.75f, 0.75f), // Gris claro

        Vector3D(0.0f, 0.0f, 0.5f), // Azul oscuro

        Vector3D(0.5f, 0.5f, 0.0f), // Verde oliva

        Vector3D(0.8f, 0.0f, 0.8f), // Púrpura claro

        Vector3D(0.0f, 0.8f, 0.8f), // Cian claro
```

```
        Vector3D(0.8f, 0.8f, 0.0f)  // Amarillo oscuro
};

//Window Height
float width = 800.0f;
float height = 600.0f;



Mesh globalTorus;









void display(void)
{
        /*  clear all pixels  */
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        globalTorus.drawMesh(colorList);



        glutSwapBuffers();
        glFlush();
}

void init(void)
{
        /*  select clearing (background) color      */
        glClearColor(0.0, 0.0, 0.0, 0.0);
```

```
	/*  initialize viewing values  */
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();

	//glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
	//gluOrtho2D(0.0, 1.0, 0.0, 1.0);

	gluPerspective(45.0, 800.0 / 600.0, 0.1, 100.0);

	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();

	gluLookAt(5.7f, 3.0f, 7.0f,    0.0f, 0.0, 0.0f,    0.0, 1.0, 0.0);

	glEnable(GL_DEPTH_TEST);
}

/*
 *  Declare initial window size, position, and display mode
 *  (single buffer and RGBA).  Open window with "hello"
 *  in its title bar.  Call initialization routines.
 *  Register callback function to display graphics.
 *  Enter main loop and process events.
 */
int main(int argc, char** argv)
{

	globalTorus.loadVertices("cube.obj");
	globalTorus.createFaces("cube.obj");

	glutInit(&argc, argv);
```

```
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
	glutInitWindowSize(800, 600);
	glutInitWindowPosition(100, 100);
	glutCreateWindow("Cube");

	init();

	glutDisplayFunc(display);
	glutIdleFunc(display);
	glutMainLoop();

	return 0;   /* ISO C requires main to return int. */
}
```

Mesh.h

```
#pragma once
#include"Vector3D.h"
#include<vector>
#include <GL/freeglut.h>
#include <math.h>
#include "matrixTransformations.h"



class Mesh
{
private:
	std::vector<Vector3D>vertices;
	std::vector<std::vector<Vector3D>>faces;
	float angleX = 0.0;
	float angleY = 0.0;
	float angleZ = 0.0;
```

```
public:
        Mesh() = default; //Constructor
        ~Mesh() = default; //Deconstructor
        void loadVertices(const char *obj);
        void createFaces(const char* obj);
        // Getters para acceder a los vértices y caras
        const std::vector<Vector3D>& getVertices() const { return vertices; }
        const std::vector<std::vector<Vector3D>>& getFaces() const { return faces; }
        void drawMesh(const std::vector<Vector3D>& colors);
        //Metodo para manipular mesh
        void updateVertices();

};
```

Mesh.cpp

```
#include "Mesh.h"
#include <GL/freeglut.h>
#include <iostream>
#include"Vector3D.h"
#include<vector>
#include <cstdio>  // For FILE, fopen_s, fclose, fscanf_s
#include "matrixTransformations.h"

int i = 1;

void Mesh::loadVertices(const char* obj) {
   FILE* file = nullptr;

   // Open the .obj file using fopen_s
   if (fopen_s(&file, obj, "r") == 0) {
      std::cout << "File opened successfully" << std::endl;
      char type;
```

```
        float x, y, z;
        int result;

        // Read data from the file using fscanf_s
        while ((result = fscanf_s(file, "%c %f %f %f", &type, sizeof(type), &x, &y, &z)) != EOF) {
            if (result == 4 && type == 'v')
            {
                vertices.push_back(Vector3D(x, y, z));
                //std::cout << "Vertex: (" << x << ", " << y << ", " << z << ")" << std::endl;
            }
        }

        // Close the file
        fclose(file);
    }
    else {
        // File not found
        std::cerr << "The .obj file was not found" << std::endl;
    }
}

void Mesh::createFaces(const char* obj)
{
    FILE* file = nullptr;

    // Open the .obj file using fopen_s
    if (fopen_s(&file, obj, "r") == 0) {
        std::cout << "File opened successfully" << std::endl;
        char type;
        float v1, v2, v3;
        int result;
```

```
        // Read data from the file using fscanf_s
        while ((result = fscanf_s(file, "%c %f %f %f", &type, sizeof(type), &v1, &v2, &v3)) != EOF) {
            if (result == 4 && type == 'f')
            {
                std::vector<Vector3D> face = { vertices[v1 - 1], vertices[v2 - 1], vertices[v3 - 1] };
                faces.push_back(face);
                std::cout << "Face composed by vertices: (" << v1 << ", " << v2 << ", " << v3 << ")" << std::endl;
            }
        }

        // Close the file
        fclose(file);
    }
    else {
        // File not found
        std::cerr << "The .obj file was not found" << std::endl;
    }
}

//Este metodo se dedica unicamente a dibujar los vertices en su posicion actual
void Mesh::drawMesh(const std::vector<Vector3D>& colors)
{
    //std::cout << "New frame"<<std::endl;
    //Actualiza la posicion de los vertices antes de dibujarlos
    //updateVertices();
    glBegin(GL_TRIANGLES);
    for (int face = 0; face < faces.size(); face++)
    {

        //Start new color for the current face
        Vector3D currentColor = colors[face];
        glColor3f(currentColor.getX(), currentColor.getY(), currentColor.getZ());
```

```
      //std::cout << "Face color: (" << currentColor.getX() << ", " << currentColor.getY() << ", " << currentColor.getZ() << ")" << std::endl;

      //std::cout<<"Face "<<face+1 << ":" << std::endl;

      //Draw all the vertices of the current face

      std::vector<Vector3D> currentFace = faces[face];

      for (int i = 0; i < currentFace.size(); i++)

      {

        Vector3D currentVertex = currentFace[i];

        glVertex3f(currentVertex.getX(), currentVertex.getY(), currentVertex.getZ());

        //td::cout << "Vertex Poicion: (" << currentColor.getX() << ", " << currentColor.getY() << ", " << currentColor.getZ() << ")" << std::endl;

      }

    }

    glEnd();

}


//Metodos de Actualizacion de los vertices

void Mesh::updateVertices()

{

    for (int face = 0; face < faces.size(); face++)

    {

      // Access the vertices of the current face

      std::vector<Vector3D>& currentFace = faces[face];  // Use a reference to modify the actual face

      for (int i = 0; i < currentFace.size(); i++)

      {

        // Update the vertex position for the current axis

        Vector3D currentVertex = currentFace[i];

        //currentFace[i] = rotateY(currentVertex, angleY);

        currentFace[i] = rotateX(currentVertex, angleX);


      }

    }}
```

Vector3D.h

```
#pragma once

class Vector3D

{

private:

	float x, y, z;

public:

	Vector3D(float X, float Y, float Z);

	Vector3D();

	float getX();

	float getY();

	float getZ();

};
```

Vector3D.cpp

```
#include "Vector3D.h"

Vector3D::Vector3D()

{

	x = 0.0f;

	y = 0.0f;

	z = 0.0f;

}

Vector3D::Vector3D(float X, float Y, float Z)

{

	x = X;

	y = Y;

	z = Z;

}

float Vector3D::getX()

{

	return x;

}
```

```
float Vector3D::getY()
{
	return y;
}
float Vector3D::getZ()
{
	return z;
}
```

matrixTransformations.h

```
#pragma once
#include"Vector3D.h"
#include <math.h>
Vector3D rotateX(Vector3D vertex,float degrees);
Vector3D rotateY(Vector3D vertex, float degrees);
```

matrixTransformations.cpp

```
#include "matrixTransformations.h"
#include <math.h>
#include <conio.h>
#include <iostream>
#define M_PI 3.14159265358979323846

//Metodos de
Vector3D rotateX(Vector3D vertex,float degrees)
{
	//std::cout << "AngleX: " << degrees<<std::endl;
	float radians = degrees * ( M_PI/ 180.0);
	float newX = vertex.getX();
	float newY = (vertex.getY() * cos(radians))-(vertex.getZ()*sin(radians));
	float newZ = (vertex.getY() * sin(radians)) + (vertex.getZ() * cos(radians));
	return Vector3D(newX, newY, newZ);
```

```
}


Vector3D rotateY(Vector3D vertex, float degrees)
{
	//std::cout << "AngleY: " << degrees << std::endl;
	float radians = degrees * (M_PI / 180.0);
	float newX = (vertex.getX()*cos(radians))-(vertex.getY()*sin(radians));
	float newY = (vertex.getX() * sin(radians)) + (vertex.getY() * cos(radians));
	float newZ = vertex.getZ();
	return Vector3D(newX, newY, newZ);
}
```