BÀI THỰC HÀNH SỐ 3

Các phép biến đổi và tạo mô hình 3D cơ bản

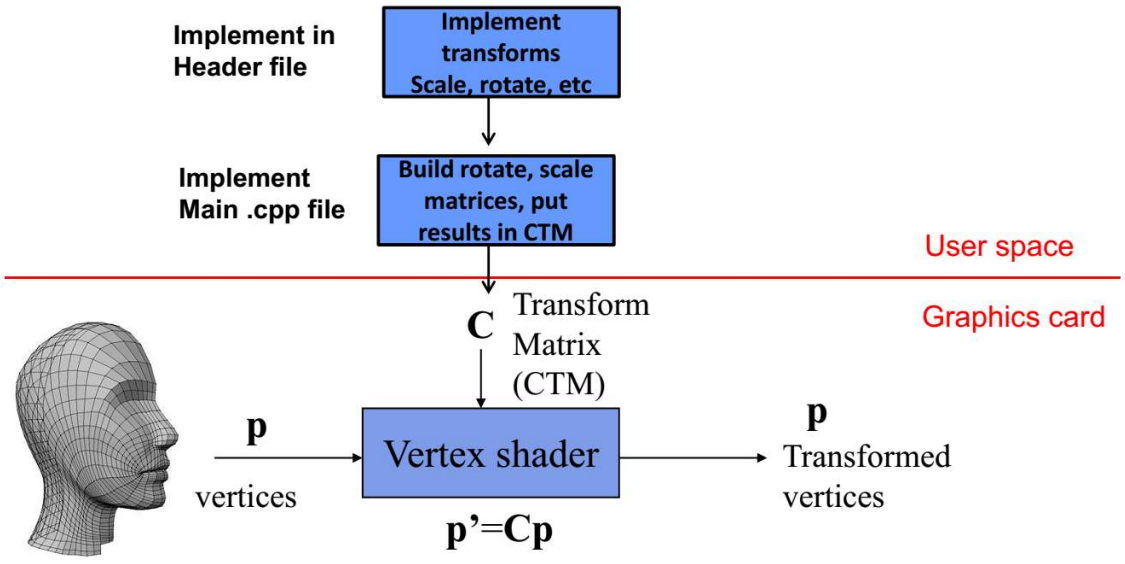
A. Cài đặt các phép biến đổi

1. Nhắc lại kiến thức tại bài thực hành số 2:

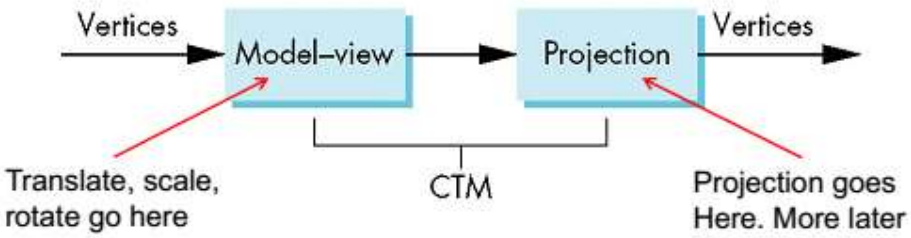
- Trong kiến trúc OpenGL trước phiên bản 3.0, các phép biến đổi được thực hiện bởi các lệnh: glTranslate, glScale, glRotate, ma trận modelview, …
- Từ phiên bản 3.1 trở đi, với kiến trúc lập trình OpenGL hiện đại, nếu lập trình viên cần các phép biến đổi, các ma trận biến đổi thì phải cài đặt chúng.

a) Ma trận biến đổi hiện hành (CTM – Current Transformation Matrix)

- Ma trận biến đổi được cài đặt là một ma trận trong hệ tọa độ thuần nhất với kích thước 4 x 4.
- CTM được định nghĩa và cập nhật trong chương trình của người dùng. Thường được cài đặt như sau:
  - Trong thư viện Header file (mat.h): Cài đặt các phép biến đổi như tịnh tiến, quay, biến đổi tỉ lệ, …
  - Trong chương trình ứng dụng file .cpp: Gọi các phép biến đổi, cài đặt ma trận CTM phù hợp.
  - Trong Vertex Shader: Cài đặt thi hành, tạo ảnh của đỉnh qua phép biến đổi được truyền CTM vào để tính.



- CTM sẽ bao gồm các phép biến đổi về model, view và phép chiếu – projection.



b) Các cài đặt trong thư viện mat.h

- Thư viện mat.h là thư viện tự cài đặt (được cung cấp kèm project, do Prof. Angel phát triển)
- Nội dung cài đặt:
  - Kiểu ma trận: mat4 (ma trận 4x4), mat3 (ma trận 3x3), …
  - Các toán tử cho ma trận
  - Các hàm biến đổi: Translate, Scale, Rotate, …
    mat4 Translate(const GLfloat x, const GLfloat y, const GLfloat z)
    mat4 Scale(const GLfloat x, const GLfloat y, const GLfloat z)

Ví dụ: Trong chương trình chính có khai báo #include “mat.h”, ta có:

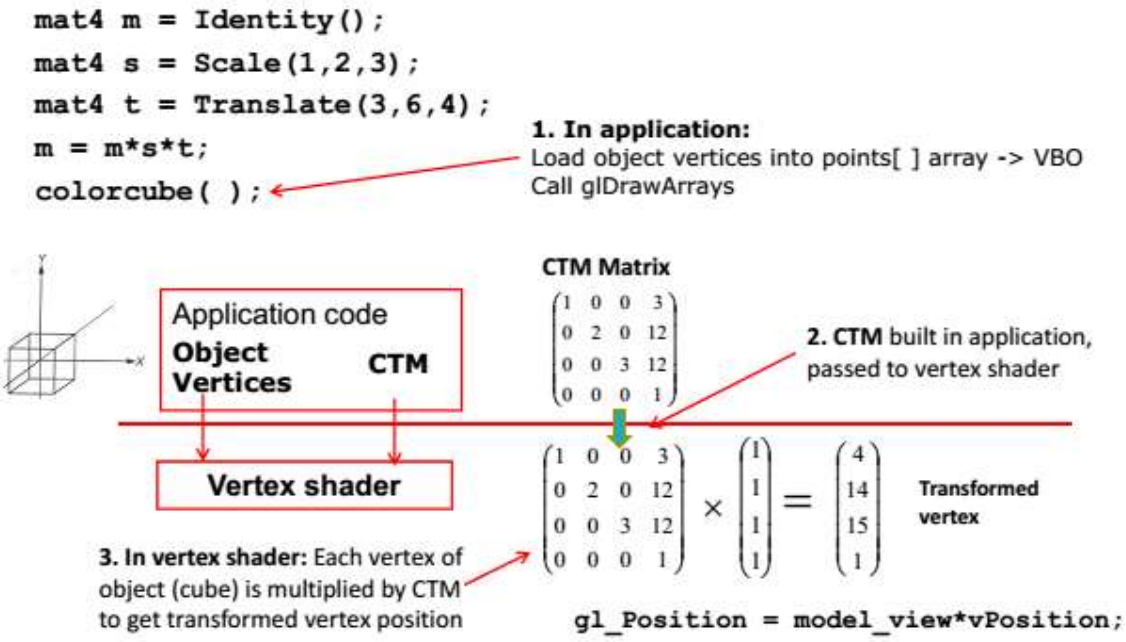
```
mat4 m; //m là một ma trận 4x4 và được gán là ma trận đơn vị

mat4 ctm=Translate(3, 6, 4);
```

Khi đó ta có:

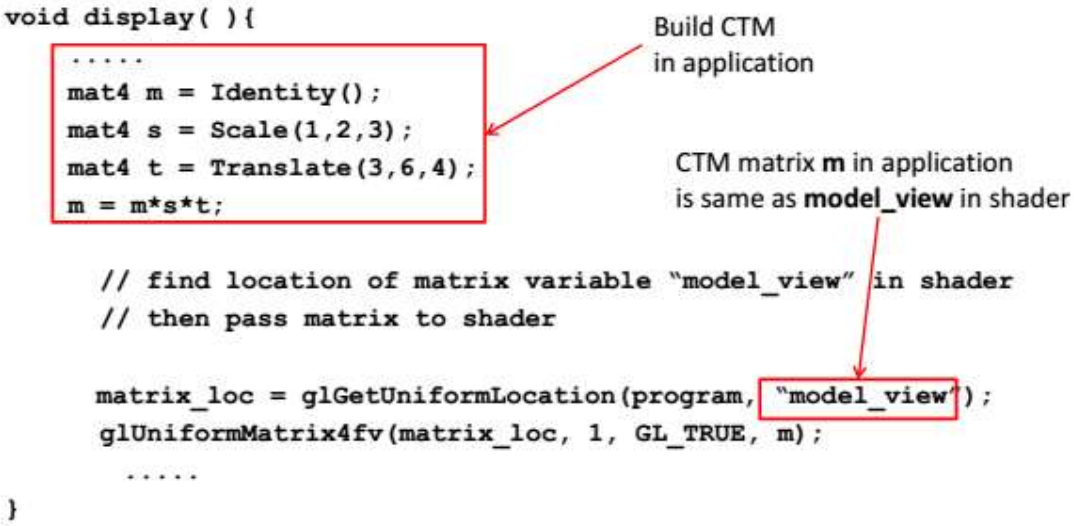
$$\text{CTM} \leftarrow \begin{pmatrix} 1 & 0 & 0 & 3 \\ 0 & 1 & 0 & 6 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$ **Translation Matrix**

c) Cách áp dụng các ma trận biến đổi

Xây dựng ma trận CTM phù hợp:

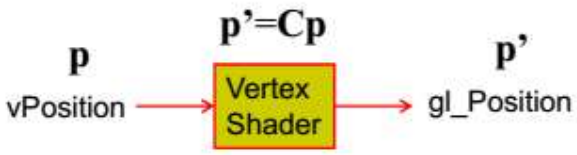


d) Chuyển biến CTM trong ứng dụng đến Vertexshader



e) Cài đặt Vertex Shader

```
in vec4 vPosition;
uniform mat4 model_view;

void main( )
{
   gl_Position = model_view*vPosition;
}
```



Transformed vertex **position**

Contains **CTM**

Original vertex **position**

- Lưu ý**:** Biến **Uniform**

  + Được áp dụng không thay đổi trên toàn bộ các đỉnh của nguyên mẫu

  + Không thay đổi giá trị trong shader

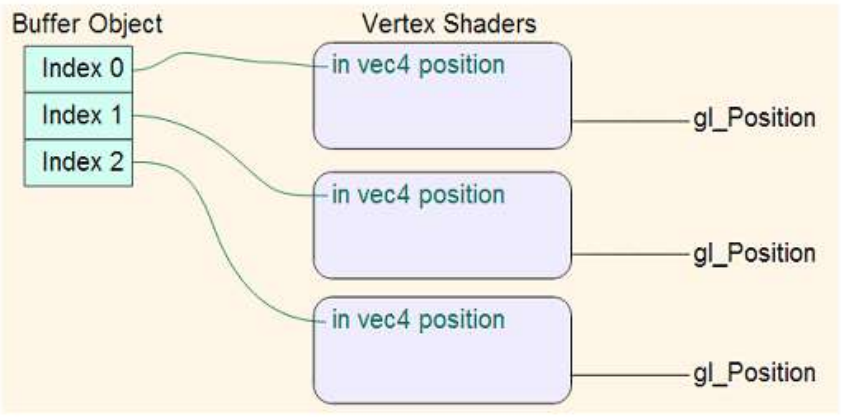
- Khi thực hiện lệnh glDrawArrays(), vertex shader đã gọi với vPosition khác nhau trên shader.

Ví dụ: colorcube() khi vẽ bằng lệnh glDrawArrays có 36 đỉnh, mỗi vertex shader nhận một đỉnh lưu trữ trong vPosition.

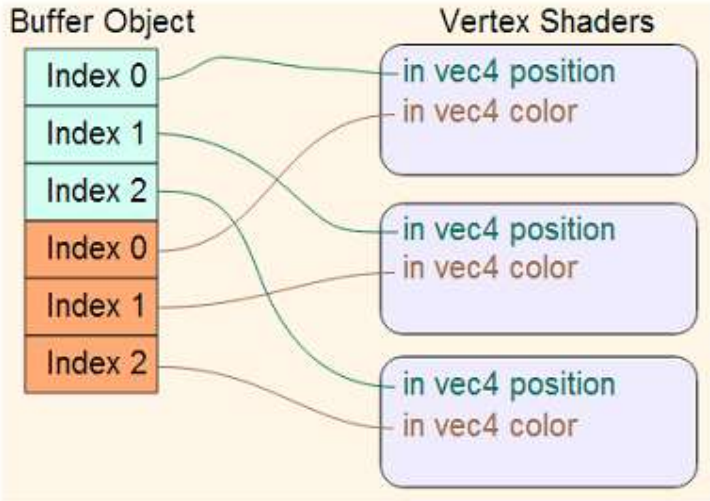
- Shader tính toán vị trí đỉnh được biến đổi và lưu vào gl_Position.

Cụ thể:

+ Các thuộc tính vị trí của đỉnh:

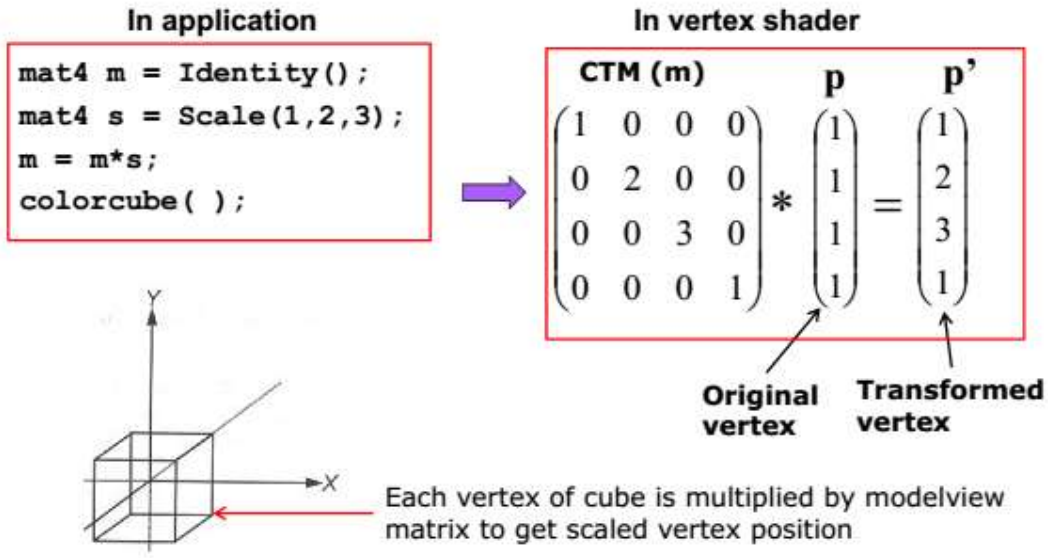


+ Khi VBO lưu trữ nhiều loại thuộc tính đỉnh:

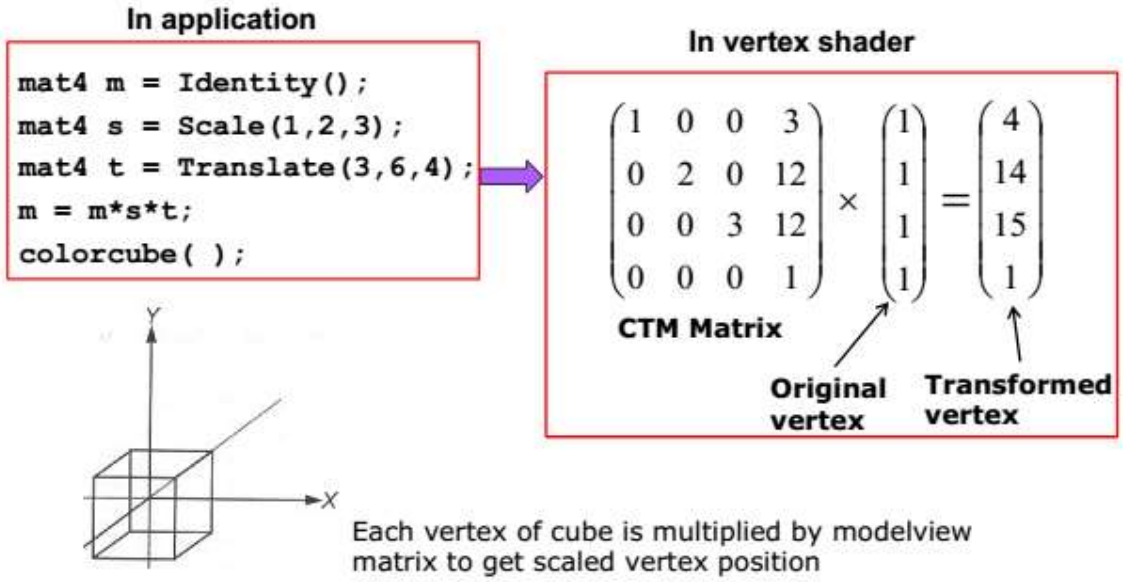


f) Ví dụ

Giả sử có đỉnh (1, 1, 1) là một trong 8 đỉnh của hình lập phương



Hoặc



2. Project vẽ bàn đơn giản

main.cpp

```
//Chương trình vẽ bàn

#include "Angel.h"  /* Angel.h là file tự phát triển (tác giả Prof. Angel), có chứa cả khai báo includes glew và freeglut*/


// remember to prototype
void generateGeometry(void);
void initGPUBuffers(void);
void shaderSetup(void);
void display(void);
void keyboard(unsigned char key, int x, int y);


typedef vec4 point4;
typedef vec4 color4;
using namespace std;

// Số các đỉnh của các tam giác
const int NumPoints = 36;

point4 points[NumPoints]; /* Danh sách các đỉnh của các tam giác cần vẽ*/
color4 colors[NumPoints]; /* Danh sách các màu tương ứng cho các đỉnh trên*/


point4 vertices[8]; /* Danh sách 8 đỉnh của hình lập phương*/
color4 vertex_colors[8]; /*Danh sách các màu tương ứng cho 8 đỉnh hình lập phương*/

GLuint program;

void initCube()
{
	// Gán giá trị tọa độ vị trí cho các đỉnh của hình lập phương
	vertices[0] = point4(-0.5, -0.5, 0.5, 1.0);
	vertices[1] = point4(-0.5, 0.5, 0.5, 1.0);
	vertices[2] = point4(0.5, 0.5, 0.5, 1.0);
```

```
        vertices[3] = point4(0.5, -0.5, 0.5, 1.0);
        vertices[4] = point4(-0.5, -0.5, -0.5, 1.0);
        vertices[5] = point4(-0.5, 0.5, -0.5, 1.0);
        vertices[6] = point4(0.5, 0.5, -0.5, 1.0);
        vertices[7] = point4(0.5, -0.5, -0.5, 1.0);

        // Gán giá trị màu sắc cho các đỉnh của hình lập phương
        vertex_colors[0] = color4(0.0, 0.0, 0.0, 1.0); // black
        vertex_colors[1] = color4(1.0, 0.0, 0.0, 1.0); // red
        vertex_colors[2] = color4(1.0, 1.0, 0.0, 1.0); // yellow
        vertex_colors[3] = color4(0.0, 1.0, 0.0, 1.0); // green
        vertex_colors[4] = color4(0.0, 0.0, 1.0, 1.0); // blue
        vertex_colors[5] = color4(1.0, 0.0, 1.0, 1.0); // magenta
        vertex_colors[6] = color4(1.0, 1.0, 1.0, 1.0); // white
        vertex_colors[7] = color4(0.0, 1.0, 1.0, 1.0); // cyan
}
int Index = 0;
void quad(int a, int b, int c, int d)  /*Tạo một mặt hình lập phương = 2 tam giác, gán
màu cho mỗi đỉnh tương ứng trong mảng colors*/
{
        colors[Index] = vertex_colors[a]; points[Index] = vertices[a]; Index++;
        colors[Index] = vertex_colors[b]; points[Index] = vertices[b]; Index++;
        colors[Index] = vertex_colors[c]; points[Index] = vertices[c]; Index++;
        colors[Index] = vertex_colors[a]; points[Index] = vertices[a]; Index++;
        colors[Index] = vertex_colors[c]; points[Index] = vertices[c]; Index++;
        colors[Index] = vertex_colors[d]; points[Index] = vertices[d]; Index++;
}
void makeColorCube(void)  /* Sinh ra 12 tam giác: 36 đỉnh, 36 màu*/

{
        quad(1, 0, 3, 2);
        quad(2, 3, 7, 6);
        quad(3, 0, 4, 7);
        quad(6, 5, 1, 2);
        quad(4, 5, 6, 7);
        quad(5, 4, 0, 1);
}
void generateGeometry(void)
{
        initCube();
        makeColorCube();
}

GLuint model_loc;
mat4 model;
mat4 instance;
mat4 instance_ban;

void matban(GLfloat w, GLfloat l, GLfloat h)
{
        instance = Scale(w, h, l);
        glUniformMatrix4fv(model_loc, 1, GL_TRUE, model*instance_ban*instance);
        glDrawArrays(GL_TRIANGLES, 0, NumPoints);

}
void chanban(GLfloat w, GLfloat h)
{
        instance = Scale(w, h, w);
        glUniformMatrix4fv(model_loc, 1, GL_TRUE, model*instance_ban*instance);
        glDrawArrays(GL_TRIANGLES, 0, NumPoints);

}
```

```
void ban()
{
	instance_ban = Identity(); /*Sửa identity() trong mat.h thành Identity()*/
	matban(0.6f, 0.4f, 0.01f);
	instance_ban = Translate(0.29f, -0.15f, 0.19f);
	chanban(0.02f, 0.3f);
	instance_ban = Translate(-0.29f, -0.15f, 0.19f);
	chanban(0.02f, 0.3f);
	instance_ban = Translate(-0.29f, -0.15f, -0.19f);
	chanban(0.02f, 0.3f);
	instance_ban = Translate(0.29f, -0.15f, -0.19f);
	chanban(0.02f, 0.3f);
}
void initGPUBuffers(void)
{
	// Tạo một VAO - vertex array object
	GLuint vao;
	glGenVertexArrays(1, &vao);
	glBindVertexArray(vao);

	// Tạo và khởi tạo một buffer object
	GLuint buffer;
	glGenBuffers(1, &buffer);
	glBindBuffer(GL_ARRAY_BUFFER, buffer);
	glBufferData(GL_ARRAY_BUFFER, sizeof(points) + sizeof(colors), NULL,
GL_STATIC_DRAW);

	glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(points), points);
	glBufferSubData(GL_ARRAY_BUFFER, sizeof(points), sizeof(colors), colors);

}


void shaderSetup(void)
{
	// Nạp các shader và sử dụng chương trình shader
	program = InitShader("vshader1.glsl", "fshader1.glsl");   // hàm InitShader khai
báo trong Angel.h
	glUseProgram(program);

	// Khởi tạo thuộc tính vị trí đỉnh từ vertex shader
	GLuint loc_vPosition = glGetAttribLocation(program, "vPosition");
	glEnableVertexAttribArray(loc_vPosition);
	glVertexAttribPointer(loc_vPosition, 4, GL_FLOAT, GL_FALSE, 0, BUFFER_OFFSET(0));

	GLuint loc_vColor = glGetAttribLocation(program, "vColor");
	glEnableVertexAttribArray(loc_vColor);
	glVertexAttribPointer(loc_vColor, 4, GL_FLOAT, GL_FALSE, 0,
BUFFER_OFFSET(sizeof(points)));

	model_loc = glGetUniformLocation(program, "Model");

	glEnable(GL_DEPTH_TEST);

	glClearColor(1.0, 1.0, 1.0, 1.0);        /* Thiết lập màu trắng là màu xóa màn
hình*/
}


void display(void)
{

	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

```
	model = RotateX(-30)*RotateY(30);
	ban();
	glutSwapBuffers();
}


void keyboard(unsigned char key, int x, int y)
{
	// keyboard handler

	switch (key) {
	case 033:			// 033 is Escape key octal value
		exit(1);		// quit program
		break;
	}
}


int main(int argc, char **argv)
{
	// main function: program starts here

	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
	glutInitWindowSize(640, 640);
	glutInitWindowPosition(100, 150);
	glutCreateWindow("Drawing a Table");


	glewInit();

	generateGeometry();
	initGPUBuffers();
	shaderSetup();

	glutDisplayFunc(display);
	glutKeyboardFunc(keyboard);

	glutMainLoop();
	return 0;
}
```

vshader1.glsl

```
#version 400
in vec4 vPosition;
in vec4 vColor;
out vec4 color;

uniform mat4 Model;
void main()
{
    gl_Position = Model*vPosition;
	color=vColor;
}
```

fshader1.glsl

```
#version 400
in vec4 color;
out vec4  fColor;
```

```
void main()
{
   fColor=color;
   // fColor = vec4( 0.0, 1.0, 0.0, 1.0 );
}
```

3. Bài tập

3.1. Đọc hiểu project vẽ cái bàn, tuỳ chỉnh về màu sắc và độ lớn của bàn.

3.2. Vẽ dãy hàng rào.

3.3. Vẽ mô hình ngăn kéo bàn. Ngăn kéo bàn có thể kéo ra, đẩy vào.