تاریخ: 1399/01/05

حسام دامغانیان (96216009)

تمرین سری اول رسم اشکال به کمک کتابخانه جی ال

فهرست

تمرین اول–الف)

رسم مکعب بدون آرایه ها

ابتدا کتابخانه های لازم را وارد می کنیم:

```
#include <GL/glew.h>
#include <GL/freeglut.h>
#include <iostream>
using namespace std;
```

سپس متغییر ها را تعریف می کنیم:

```
bool keyboard = true;
float
rotateH = 0,
rotateV = 0,
xrotated, yrotated, zrotated;

int
CurrentWidth = 800,
CurrentHeight = 600;
```

سپس تابع animation (برای چرخش خودکار)، keypress (برای چرخش بوسیله کیبرد)، resizefunction(برای تغییر اندازه صفحه) و تابع renderFunction که در اصل تابع مورد نیاز برای رسم هست را تعریف می کنیم:

```
void keyPress(int key, int x, int y);
void ResizeFunction(int, int);
void RenderFunction(void);
void animation(void);
```

سپس تابع main :

```
int main(int argc, char* argv[])
{
	glutInit(&argc, argv);
	glutInitWindowSize(CurrentWidth, CurrentHeight);
	glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
	glutCreateWindow("P1");
	glutSpecialFunc(keyPress);
	glutReshapeFunc(ResizeFunction);
```

```
	glutDisplayFunc(RenderFunction);
	glutIdleFunc(animation);
1.	glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
	glutMainLoop();
	exit(EXIT_SUCCESS);
}
```

در این تابع ابتدا کتابخانه glut را راه اندازی کرده و سپس اندازه صفحه و حالت و در نهایت با createWindow صفحه ای به نام P1 را می سازیم. با استفاده از کتابخانه تابع keypress را که برای چرخش مکعب توسط کیبرد است را معرفی میکنیم. در خطوط بعد به ترتیب تابع مربوط به تغییر اندازه صفحه و نمایش مکعب و حالت آزاد را معرفی میکنیم. در خط یک صفحه را با رنگ سیاه پاک میکنیم و سپس حلقه ی اصلی را فراخوانی می کنیم. در آخر هم exit() .

تابع تغییر اندازه

```
void ResizeFunction(int Width, int Height)
{
	CurrentWidth = Width;
	CurrentHeight = Height;
	glViewport(0, 0, CurrentWidth, CurrentHeight);
}
```

این تابع در هنگام تغییر اندازه صفحه فراخوانی می شود و viewport را در هر تغییر آپدیت میکند.

تابع نمایش مکعب

```
void RenderFunction(void)
{
	glMatrixMode(GL_MODELVIEW);
	glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glLoadIdentity();

	if (keyboard){
		glRotatef(rotateH, 0.0f, 0.5f, 0.0f);
		glRotatef(rotateV, 0.0f, 0.0f, 0.5f);
	}
	else
	{
		glRotatef(xrotated, 1.0f, 0.0f, 0.0f);
		glRotatef(yrotated, 0.0f, 1.0f, 0.0f);
		glRotatef(zrotated, 0.0f, 0.0f, 1.0f);
	}

	glBegin(GL_QUADS);

	// top
	glColor3f(0.0f, 0.1f, 0.2f);
	glVertex3f(0.5, 0.5, -0.5);
	glColor3f(0.9f, 0.8f, 0.7f);
	glVertex3f(-0.5, 0.5, -0.5);
	glColor3f(0.8f, 0.7f, 0.6f);
	glVertex3f(-0.5, 0.5, 0.5);
	glColor3f(0.7f, 0.6f, 0.5f);
	glVertex3f(0.5, 0.5, 0.5);

	//right
	glColor3f(0.5f, 0.6f, 0.7f);
	glVertex3f(0.5, 0.5, -0.5);
	glColor3f(0.9f, 0.8f, 0.7f);
	glVertex3f(0.5, 0.5, 0.5);
	glColor3f(0.8f, 0.7f, 0.6f);
	glVertex3f(0.5, -0.5, 0.5);
	glColor3f(0.7f, 0.6f, 0.5f);
	glVertex3f(0.5, -0.5, -0.5);

	//left
	glColor3f(0.4f, 0.5f, 0.6f);
	glVertex3f(-0.5, 0.5, 0.5);
	glColor3f(0.9f, 0.8f, 0.7f);
	glVertex3f(-0.5, 0.5, -0.5);
	glColor3f(0.8f, 0.7f, 0.6f);
	glVertex3f(-0.5, -0.5, -0.5);
	glColor3f(0.7f, 0.6f, 0.5f);
	glVertex3f(-0.5, -0.5, 0.5);

	//front
	glColor3f(0.3f, 0.4f, 0.5f);
	glVertex3f(0.5, 0.5, 0.5);
	glColor3f(0.9f, 0.8f, 0.7f);
	glVertex3f(-0.5, 0.5, 0.5);
```

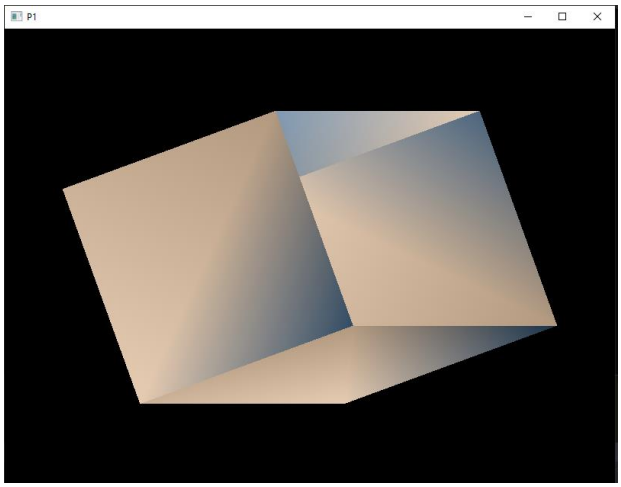
```
        glColor3f(0.8f, 0.7f, 0.6f);
        glVertex3f(-0.5, -0.5, 0.5);
        glColor3f(0.7f, 0.6f, 0.5f);
        glVertex3f(0.5, -0.5, 0.5);

        //back
        glColor3f(0.2f, 0.3f, 0.4f);
        glVertex3f(0.5, -0.5, -0.5);
        glColor3f(0.9f, 0.8f, 0.7f);
        glVertex3f(-0.5, -0.5, -0.5);
        glColor3f(0.8f, 0.7f, 0.6f);
        glVertex3f(-0.5, 0.5, -0.5);
        glColor3f(0.7f, 0.6f, 0.5f);
        glVertex3f(0.5, 0.5, -0.5);

        //bottom
        glColor3f(0.1f, 0.2f, 0.3f);
        glVertex3f(0.5, -0.5, 0.5);
        glColor3f(0.9f, 0.8f, 0.7f);
        glVertex3f(-0.5, -0.5, 0.5);
        glColor3f(0.8f, 0.7f, 0.6f);
        glVertex3f(-0.5, -0.5, -0.5);
        glColor3f(0.7f, 0.6f, 0.5f);
        glVertex3f(0.5, -0.5, -0.5);


        glEnd();

        glFlush();
        glutSwapBuffers();
        glutPostRedisplay();
}
```

در این تابع ابتدا مدل ماتریس را به modelview تبدیل کرده و سپس صفحه را پاک می کنیم. ماتریس همانی را لود میکنیم و در ادامه تنظیمات مربوط به چرخش توسط کیبرد و یا خودکار را انجام می دهیم. سپس تابع glBegin را با فلگ GL_QUADS فراخوانی کرده و راس های ماتریس را با رنگ های مخصوص به خودشان معرفی میکنیم و در آخر با فراخوانی glEnd معرفی مکعب را تمام میکنیم. glFlush, glSwapBuffers, glutPostRedisplayرا فراخوانی میکنیم تا تغییرات رخ داده شده در این فریم را اعمال کنند.

توابع **keypress** و **Animation**:

این دو تابع را برای چرخش مکعب در دو حالت تعریف شده، تعریف میکنیم که توضیح خاصی هم ندارند.

```
void keyPress(int key, int x, int y){

	if (key == 27)
		exit(0);
	if (key == GLUT_KEY_RIGHT)
		rotateH += 1;
	if (key == GLUT_KEY_LEFT)
		rotateH -= 1;
	if (key == GLUT_KEY_UP)
		rotateV += 1;
	if (key == GLUT_KEY_DOWN)
		rotateV -= 1;

	glutPostRedisplay();

}

void animation(void)
{

	yrotated += 0.01;
	xrotated += 0.02;
	RenderFunction();
}
```



خروجی طبق انتظار تولید می شود و با کلید های کیبرد میچرخد

ب)

رسم مکعب با استفاده از آرایه های جی ال

مانند مراحل قسمت الف) کتابخانه ها را وارد کرده و متغیرها را تعریف می کنیم. متغییر ها شامل مقادیر چرخش در هر محور و آرایه های مختصات گره های مکعب، رنگ های گره ها و ایندکس های مختصات داده شده در مکعب مورد نظر است.

توابع `renderFunction, keypress` مانند مرحله ی قبل هستند و تابع `initGl` را نیز برای تبدیل های جبری مورد نیاز تعریف می کنیم.

تابع `main` نیز مانند مرحله قبل است فقط در آن اینبار تابع `intiGl` را نیز فراخوانی می کنیم.

تابع **`renderFunction`**:

```
void renderFunction(void){
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glLoadIdentity();

	glRotatef(rotateZ, 0, 0, 1);
	glRotatef(rotateY, 0, 1, 0);
	glRotatef(rotateX, 1, 0, 0);

	glEnableClientState(GL_VERTEX_ARRAY);
	glEnableClientState(GL_COLOR_ARRAY);


	glVertexPointer(3, GL_FLOAT, 0, vertexCoords);
	glColorPointer(3, GL_FLOAT, 0, vertexColors);

	glDrawElements(GL_QUADS, 24, GL_UNSIGNED_INT, indexArray);
	glutSwapBuffers();
}
```

در این تابع نکته جدید نسبت به مرحله قبل `glEnableClientState` است که بوسیله ی آن اینکه بتوانیم برای گره ها رنگ و مکان را بدهیم فعال می کنیم سپس در

توابع glVertexPointer و glColorPointer آرایه های مد نظر برای ترسیم را می دهیم. و در آخر با فراخوانی glDrawElements با flag مربوط به GL_QUADS و آرایه های ایندکس مکعب را ترسیم می کنیم.

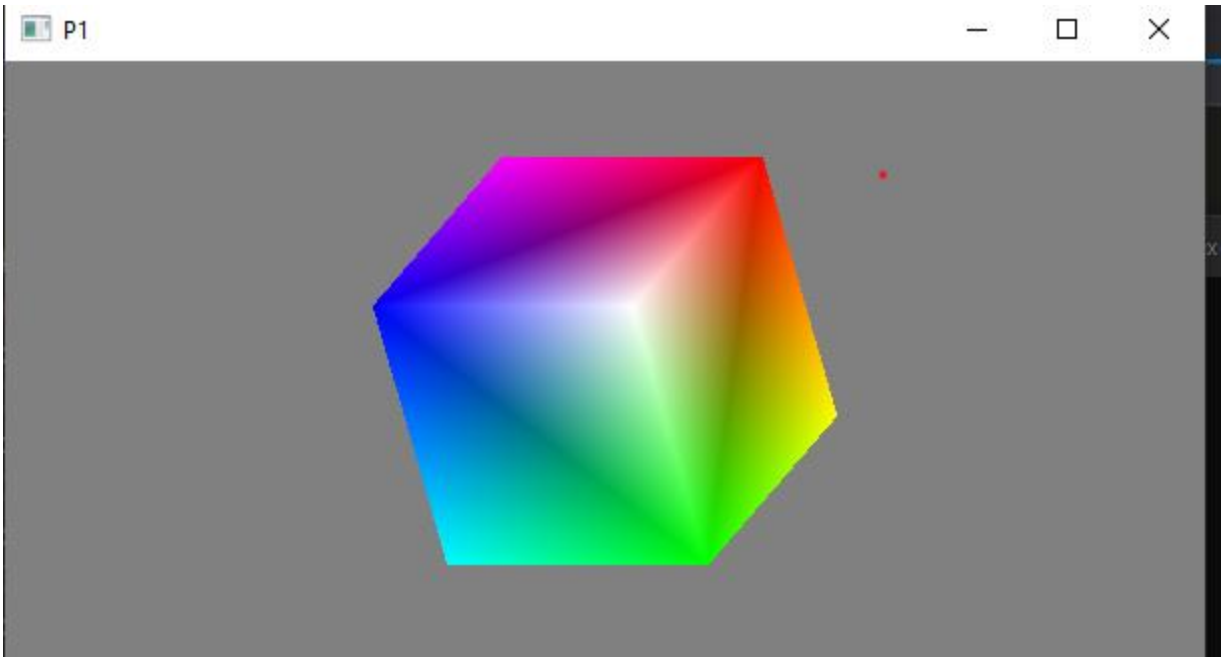
❊ مزیت این تابع نسبت به glDrawArrays این است که در شمارش فراخوانی های glDraw این تابع یکی حساب می شود.

تابع keypress نیز مانند مرحله قبل تعریف می شود.

تابع **initGl**:

```
void initGL(void) {
	glMatrixMode(GL_PROJECTION);
	glOrtho(-4, 4, -2, 2, -2, 2);
	glMatrixMode(GL_MODELVIEW);
	glClearColor(0.5, 0.5, 0.5, 1);
}
```

در این تابع ابتدا حالت ماتریس را با فلگ GL_PROJECTION تغییر داده و مختصات را بوسیله glOrtho از −1 تا 1 به −4 تا 4 و ... می بریم. حالت ماتریس را به GL_MODELVIEW تغییر داده و در آخر نیز صفحه را به رنگ خاکستری در آوریم.



خروجی طبق انتظار تولید می شود. و با کلید های کیبرد میچرخد.

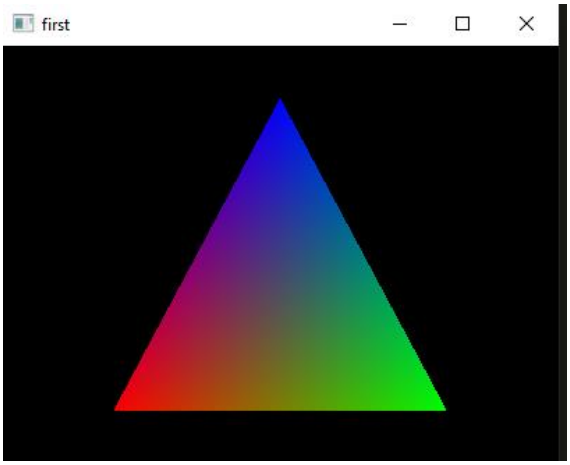
تمرین دوم:

پنج تابع `renderFunction(1-5)` برای نمایش پنج شکل تعریف میکنیم. و سپس در تابع `main` برای اینکه هر پنج شکل را در یک فایل داشته باشیم می توانیم پنج بار `glutCreateWindow` را و سپس پنج بار `glutDisplayFunction` را فراخوانی می کنیم.

در ادامه هر کدام از توابع را تشریح می کنیم.

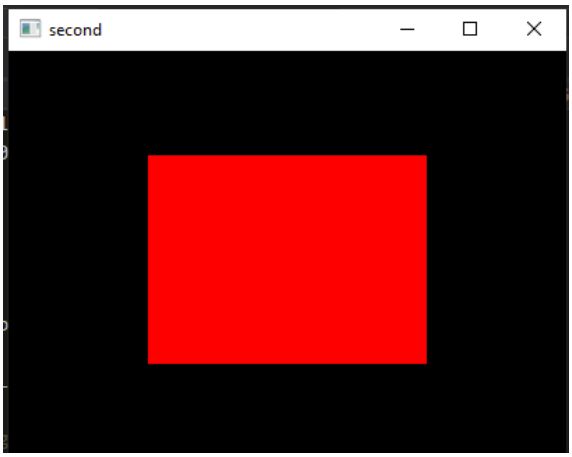
renderFunction1

این تابع یک مثلث با رنگ های متفاوت در هر گره تولید میکند. در ShadeModel این تابع GL_SMOOTH است در نتیجه رنگ ها در یکدیگر پخش می شوند.



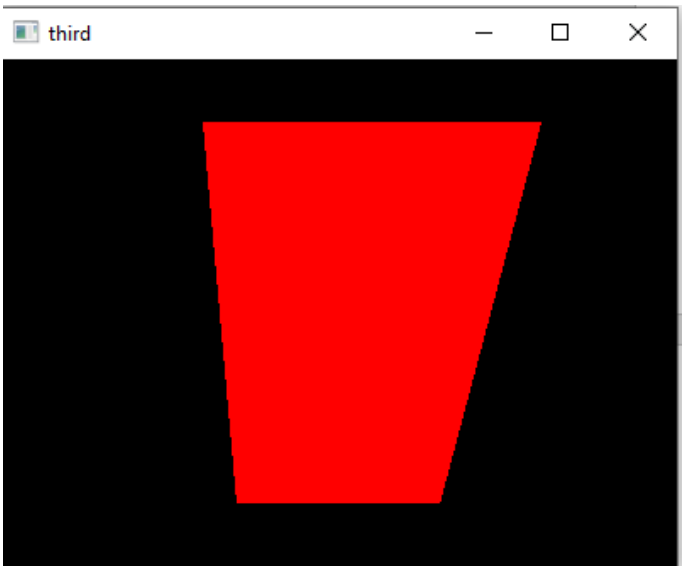
renderFunction2

دومین تابع با استفاده از دستور glRect یک مستطیل به رنگ قرمز رسم میکند.



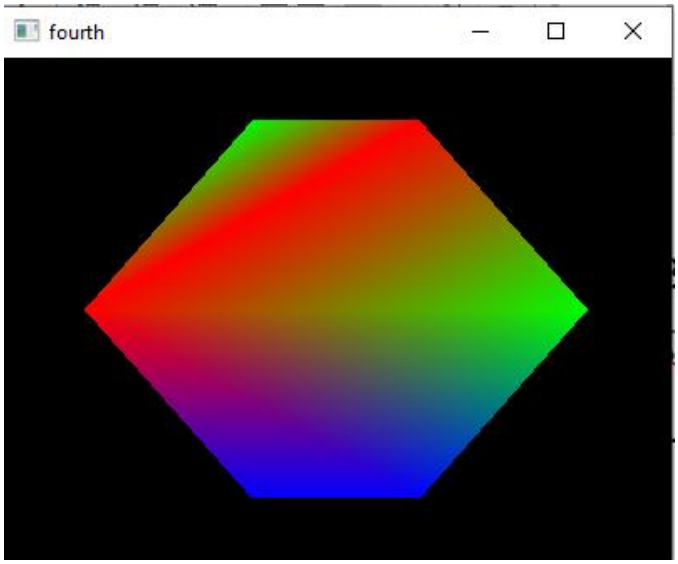
renderFunction3

تابع سوم یک ذوزنقه با `glShademodel(GL_FLAT)` و با دستور `glBegin(GL_POLYGON)` رسم می کند و چون `GL_FLAT` انتخاب شده رنگ گره اول (قرمز) را بر روی کل شکل اعمال میکند.



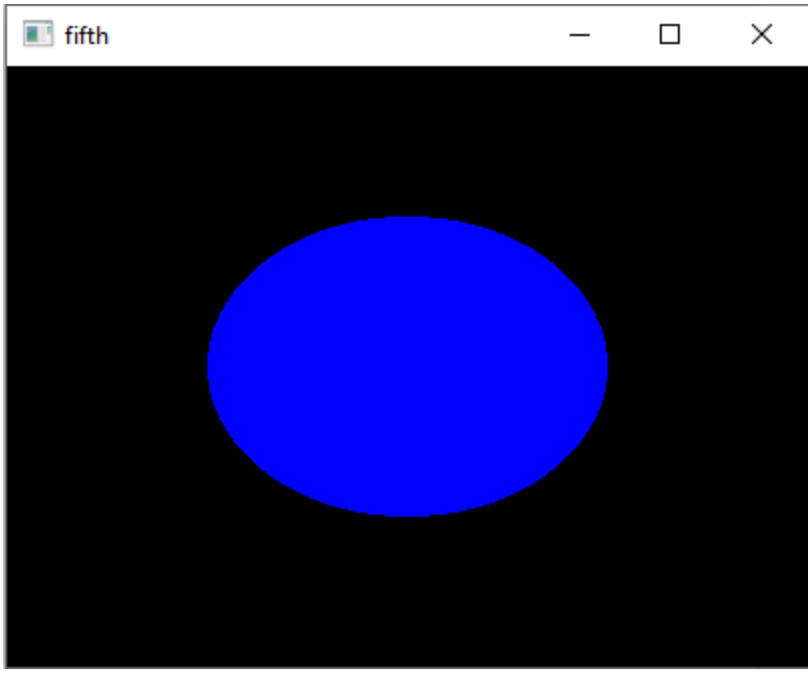
renderFunction4

با همان دستور `glBegin(GL_POLYGON)` و شش گره یک شش ضلعی رسم میکنیم.



renderFunction5

با دستور glBegin(GL_TRIANGL_FAN) یک دایره ترسیم میکنیم. به این صورت که در حلقه ی فور در هر مرحله 0.2 درجه اضافه میکنیم و گره بعدی را ترسیم میکنیم. (گره جدید در هر مرحله با اضافه کردن sin(angle)*radius و cos(angle)*radius به مبدا x1, y1 بدست می آید)



- به طور پیشفرض glShadeModel مقدار GL_SMOOTH را دارد که یعنی رنگ های داده شده(اگر بیشتر از یکی باشد) را در هم پخش می کند ولی GL_FLAT اولین رنگ داده شده را استفاده می کند.