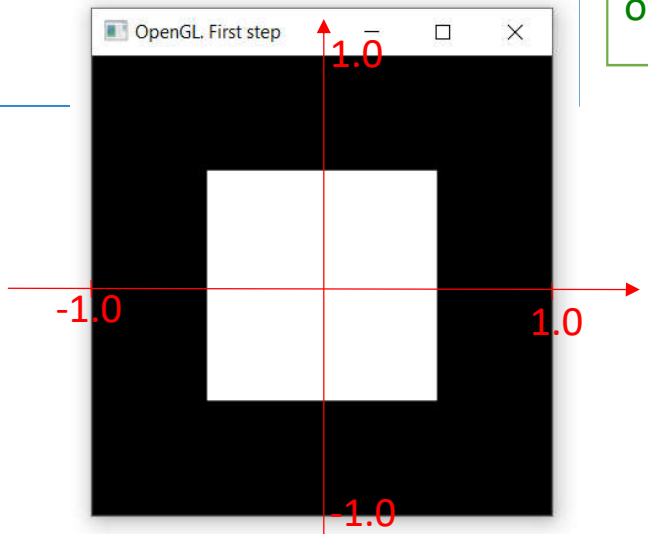
OpenGL (Open Graphics Library) — спецификация, определяющая платформонезависимый (независимый от языка программирования) программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику.

Включает более 300 функций для рисования сложных трёхмерных сцен из простых примитивов. Используется при создании компьютерных игр, САПР, виртуальной реальности, визуализации в научных исследованиях. На платформе Windows конкурирует с Direct3D.

ru.wikipedia.org/wiki/OpenGL

c:\windows\system32\opengl32.dll	основная библиотека	#include "gl.h"
c:\windows\system32\glu32.dll	дополнительная	#include "glu.h"
c:\windows\system32\glut32.dll	взаимодействие с ОС	#include "glut.h"

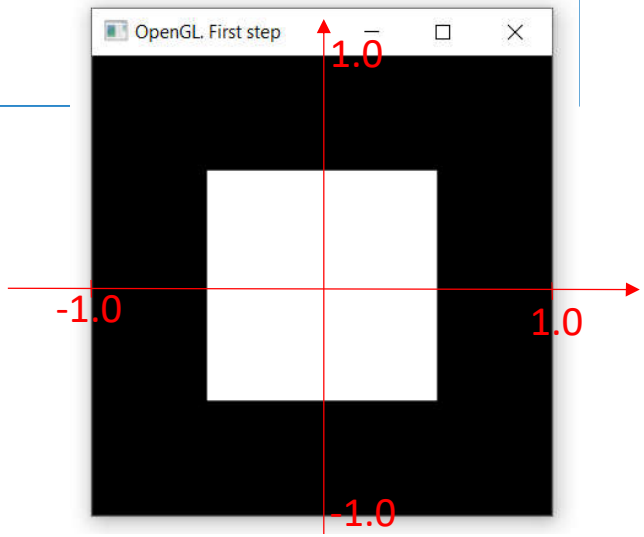
```
#include "glut.h"

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glRectf(-0.5, -0.5, 0.5, 0.5);
    glFinish();
}

void main()
{
    glutCreateWindow("OpenGL. First step");
    glutDisplayFunc(display);
    glutMainLoop();
}
```



Рисование в окне (рендер):
- очистка буфера цвета
- рисование прямоугольника
- прорисовка кадра

создание графического окна
установка функции обратного вызова
основной цикл работы приложения

```c
#include "glut.h"

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glRectf(-0.5, -0.5, 0.5, 0.5);
    glFinish();
}

void main()
{
    glutCreateWindow("OpenGL. First step");
    glutDisplayFunc(display);
    glutMainLoop();
}
```

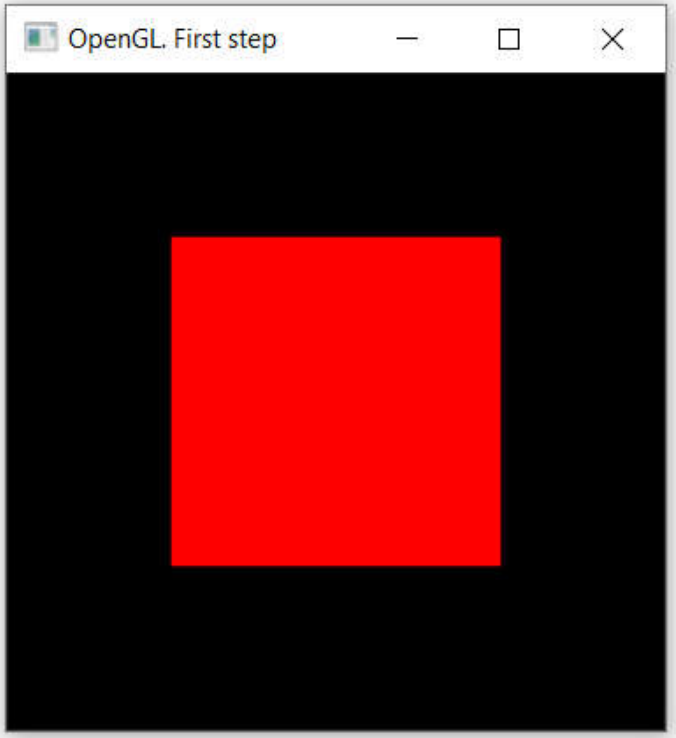



```python
from OpenGL.GL import *
from OpenGL.GLUT import *

def display():
    glClear(GL_COLOR_BUFFER_BIT)
    glRectf(0.5, 0.5, -0.5, -0.5)
    glFinish()

glutInit()
glutCreateWindow("OpenGL + Python")
glutDisplayFunc(display)
glutMainLoop()
```

```
#include "glut.h"

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
    glRectf(-0.5, -0.5, 0.5, 0.5)
    glFinish();
}

void main()
{
    glutCreateWindow("OpenGL. First step");
    glutDisplayFunc(display);
    glClearColor(0, 0, 0, 0);
    glutMainLoop();

}
```

установка текущего цвета
Красный (R=1 G=0 B=0)

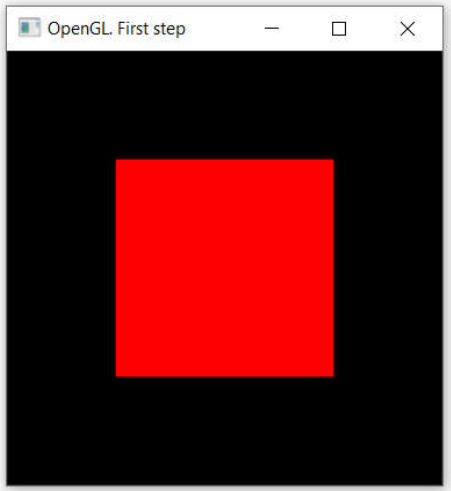
установка цвета очистки
Черный (R=0 G=0 B=0)



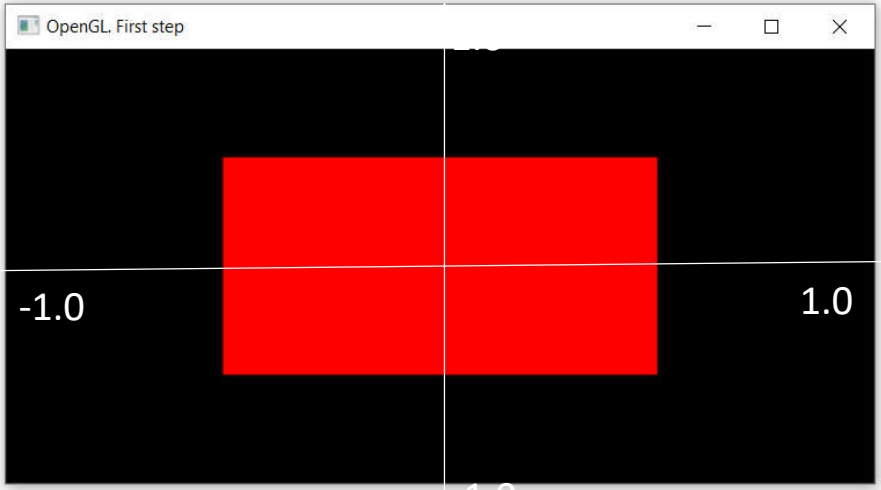
```
#include "glut.h"

void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 0.0, 0.0);
   glRectf(-0.5, -0.5, 0.5, 0.5);
   glFinish();
}

void reshape(int width, int height)
{
   glViewport(0, 0, width, height);
   gluOrtho2D(-1, 1, -1, 1);
}

void main()
{

   glutInitWindowSize(300, 300);
   glutInitWindowPosition(100, 100);

   glutInitDisplayMode(GLUT_RGB);
   glutCreateWindow("OpenGL. First step");

   glutReshapeFunc(reshape);
   glutDisplayFunc(display);
   glClearColor(0, 0, 0, 0);

   glutMainLoop();
}
```

Подключение библиот

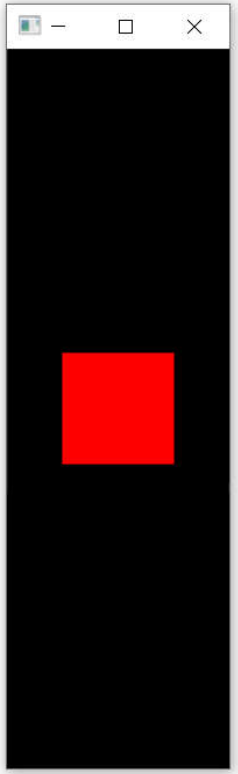
Рисование в окне (р
- очистка
- установка синего
- рисование прямоуг
- прорисовка кадра

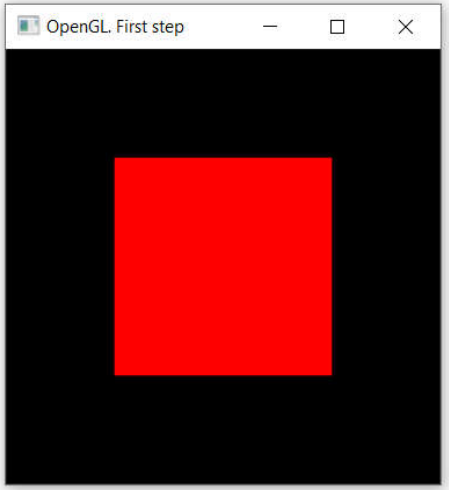


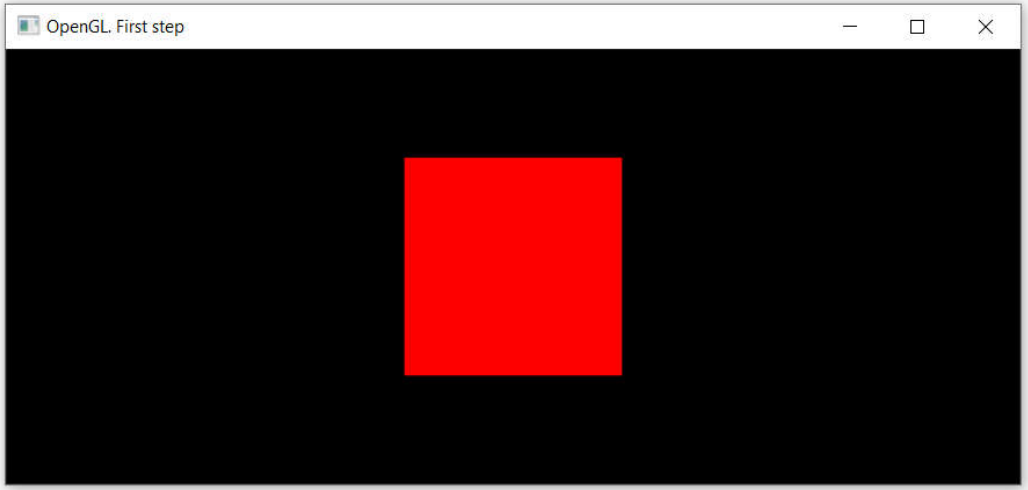
Действия при измене
- установка границ
- установка объема отсечения



```
void Reshape(int width, int height)
{
   glViewport(0, 0, width, height);
   float base = 1;
   float kw = 1;
   float kh = 1;
   if (width > height) kw = (float)width / (float)height;
   if (height > width) kh = (float)height / (float)width;
   glLoadIdentity();
   gluOrtho2D(-kw*base, kw*base, -kh*base, kh*base);
}
```







```c
void reshape(int width, int height)
{
glViewport(0, 0, width, height);
gluOrtho2D(-1, 1, -1, 1);
}
```

```c
void display(void)
{
glClear(GL_COLOR_BUFFER_BIT);

if (t == 0) glColor3f(1.0f, 0.0f, 0.0f);
if (t == 1) glColor3f(0.0f, 1.0f, 0.0f);
if (t == 2) glColor3f(0.0f, 0.0f, 1.0f);

glRectf(-0.5, -0.5, 0.5, 0.5);

glFinish();
}
```

```c
void Menu(int v)
{
t = v;
glutPostRedisplay();
}
```

```c
int t=0;

void main()
{

glutInitWindowSize(800, 600);
glutInitWindowPosition(20, 20);

glutInitDisplayMode(GLUT_RGB);
glutCreateWindow("OpenGL. First step");


glutReshapeFunc(reshape);
glutDisplayFunc(display);

glutTimerFunc(500, timer, 0);

glutKeyboardFunc(keyboard);

glutMouseFunc(mouse);

int menu=glutCreateMenu(Menu);
glutAddMenuEntry("Red",0);
glutAddMenuEntry("Green",1);
glutAddMenuEntry("Blue",2);
glutAttachMenu(GLUT_RIGHT_BUTTON);

glClearColor(0, 0, 0, 0);

glutMainLoop();
}
```

```c
void timer(int v)
{
   t += 1;
   if (t == 3) t = 0;

   glutPostRedisplay();
   glutTimerFunc(500, timer, 0);
}
```

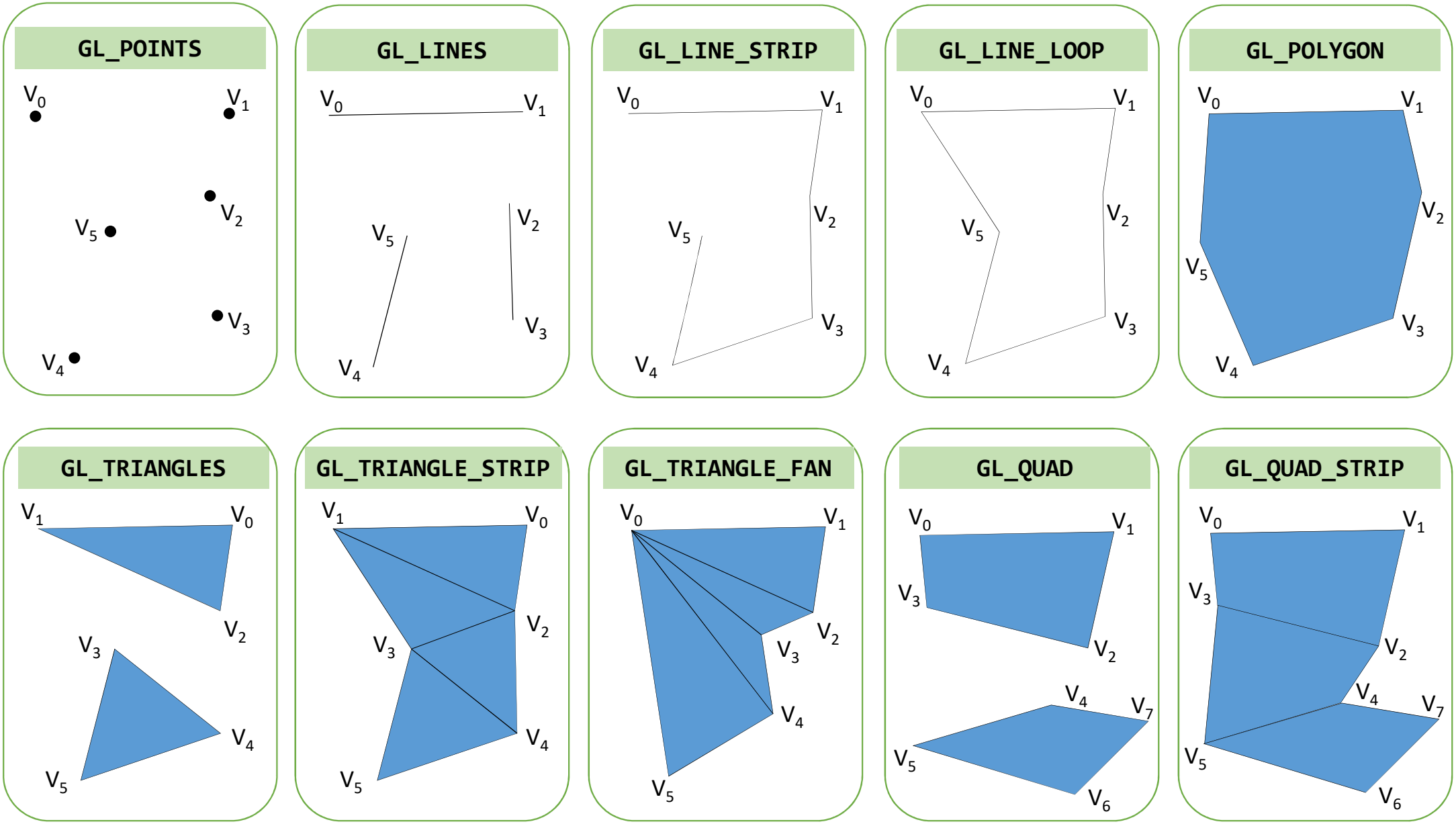
```c
void keyboard(unsigned char key, int x, int y)
{
   if (key == 32)
   {
       t += 1;
       if (t == 3) t = 0;
       glutPostRedisplay();
   }
}
```

```c
void mouse(int button, int state, int x, int y)
{
   if ((button == GLUT_LEFT_BUTTON))
   {
      if ((state == GLUT_DOWN))
      {
         t += 1;
         if (t == 3) t = 0;
      }
   }
   glutPostRedisplay();
}
```

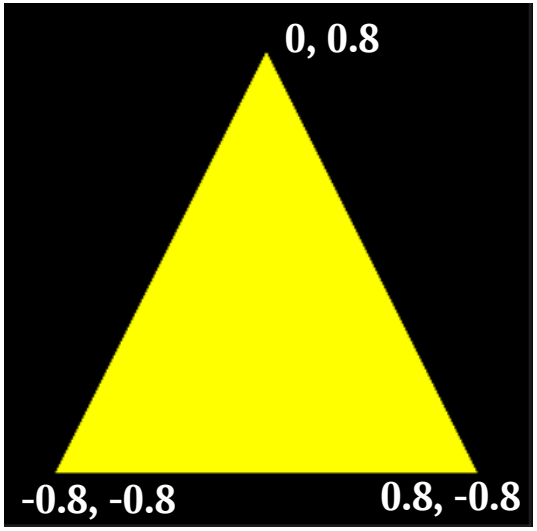
Примитивы

```
void glVertex[2 3 4][s i f d](type coords)
void glVertex[2 3 4][s i f d]v(type *coords)

glBegin(Glenum mode);
glVertex...
glVertex...
...
glEnd();
```

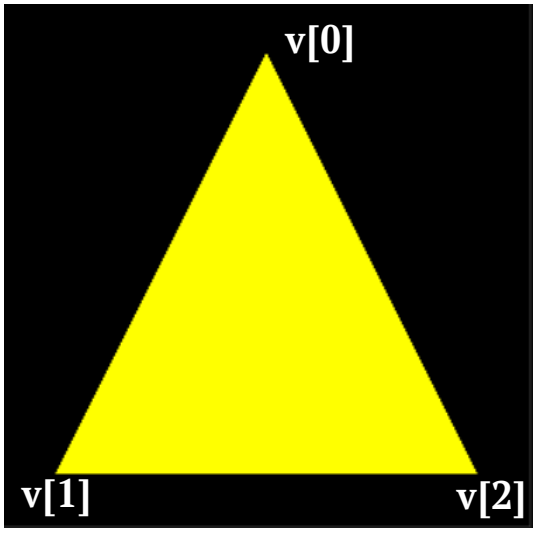
Значение mode	Соответствующие примитивы
GL_POINTS	индивидуальные точки
GL_LINES	вершины попарно интерпретируются как самостоятельные отрезки
GL_LINE_STRIP	серия соединенных отрезков (ломаная)
GL_LINE_LOOP	аналогично предыдущему, но, кроме того, автоматически добавляется отрезок, соединяющий первую и последнюю вершины (замкнутая ломаная)
GL_TRIANGLES	каждая тройка вершин интерпретируется как треугольник
GL_TRIANGLE_STRIP	цепочка соединенных треугольников
GL_TRIANGLE_FAN	веер из соединенных треугольников
GL_QUADS	каждая четверка вершин интерпретируется как четырехугольный полигон
GL_QUAD_STRIP	цепочка соединенных четырехугольников
GL_POLYGON	граница простого выпуклого полигона


GL_POINTS
V0 V1 V2 V3 V4 V5
GL_LINES
V0 V1 V2 V3 V4 V5
GL_LINE_STRIP
V0 V1 V2 V3 V4 V5
GL_LINE_LOOP
V0 V1 V2 V3 V4 V5
GL_POLYGON
V0 V1 V2 V3 V4 V5
GL_TRIANGLES
V1 V0 V2 V3 V4 V5
GL_TRIANGLE_STRIP
V1 V0 V2 V3 V4 V5
GL_TRIANGLE_FAN
V0 V1 V2 V3 V4 V5
GL_QUAD
V0 V1 V3 V2 V4 V7 V5 V6
GL_QUAD_STRIP
V0 V1 V3 V2 V4 V7 V5 V6

```
glColor3f(1.0, 1.0, 0.0);
glBegin(GL_TRIANGLES);
glVertex2f(0, 0.8);
glVertex2f(-0.8, -0.8);
glVertex2f(0.8, -0.8);
glEnd();
```

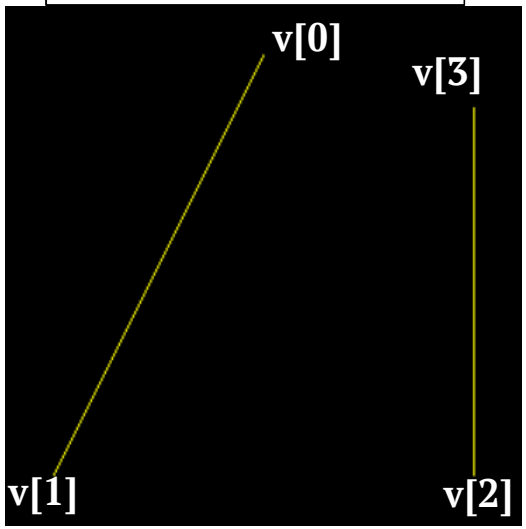



```
GLfloat yellow[3] = { 1.0, 1.0, 0.0 };
GLfloat v[3][2] = { {0, 0.8}, {-0.8, -0.8}, {0.8, -0.8} };

glColor3fv(yellow);
glBegin(GL_TRIANGLES);
glVertex2fv(v[0]);
glVertex2fv(v[1]);
glVertex2fv(v[2]);
glEnd();
```

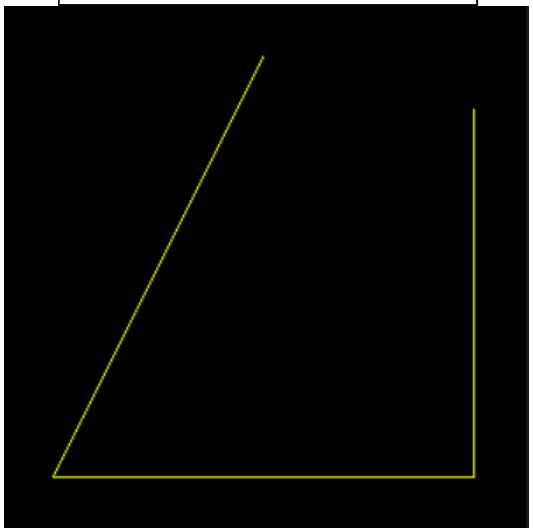


```
GLfloat yellow[3] = { 1.0, 1.0, 0.0 };
GLfloat v[4][2] = { {0, 0.8},
                    {-0.8, -0.8},
                    {0.8, -0.8},
                    {0.8, 0.6 } };

glColor3fv(yellow);
glBegin(GL_LINES);
//glBegin(GL_LINE_STRIP);
//glBegin(GL_LINE_LOOP);
//glBegin(GL_POLYGON);
glVertex2fv(v[0]);
glVertex2fv(v[1]);
glVertex2fv(v[2]);
glVertex2fv(v[3]);
glEnd();
```

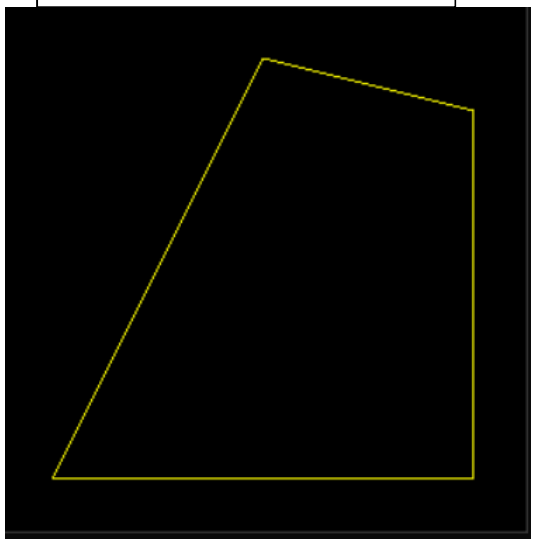
GL_LINES



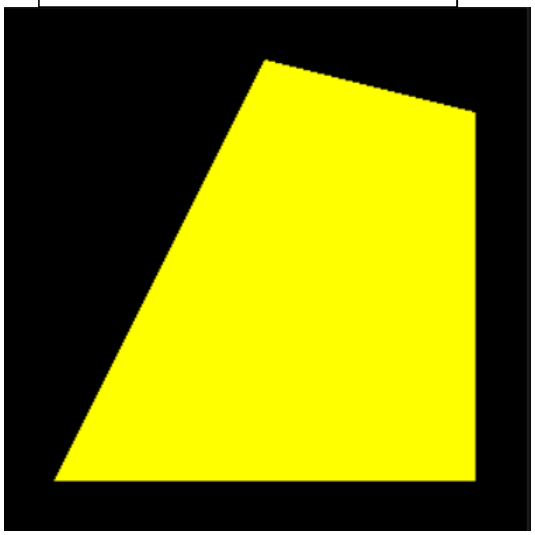
GL_LINE_STRIP



GL_LINE_LOOP

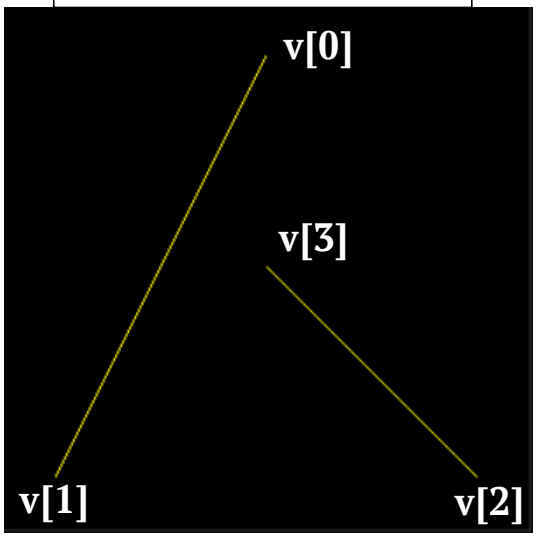


GL_POLYGON

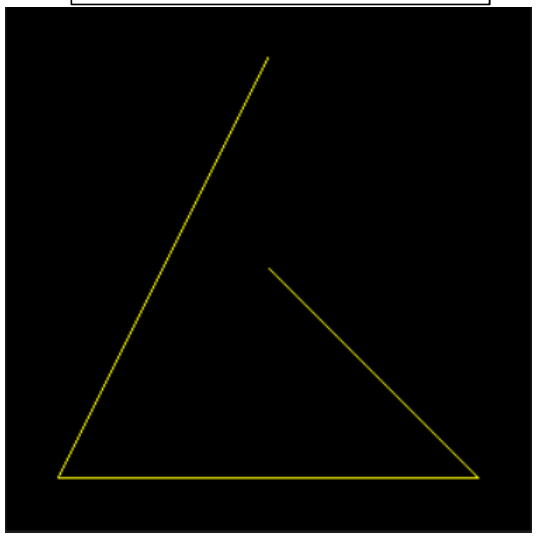


```
GLfloat yellow[3] = { 1.0, 1.0, 0.0 };
GLfloat v[4][2] = { {0, 0.8},
                    {-0.8, -0.8},
                    {0.8, -0.8},
                    {0.0, 0.0 } };

glColor3fv(yellow);
glBegin(GL_LINES);
//glBegin(GL_LINE_STRIP);
//glBegin(GL_LINE_LOOP);
//glBegin(GL_POLYGON);
glVertex2fv(v[0]);
glVertex2fv(v[1]);
glVertex2fv(v[2]);
glVertex2fv(v[3]);
glEnd();
```

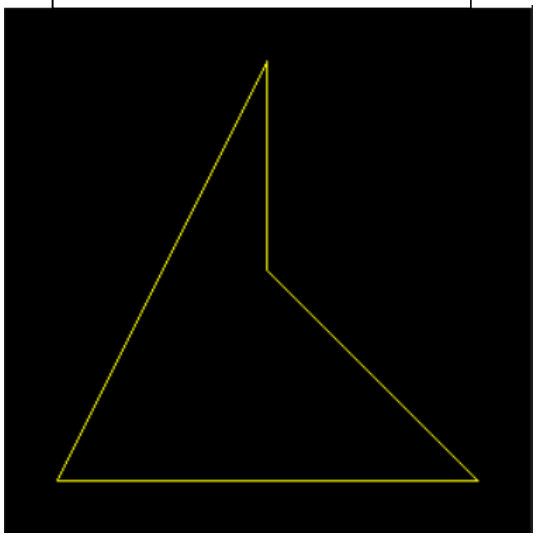
GL_LINES



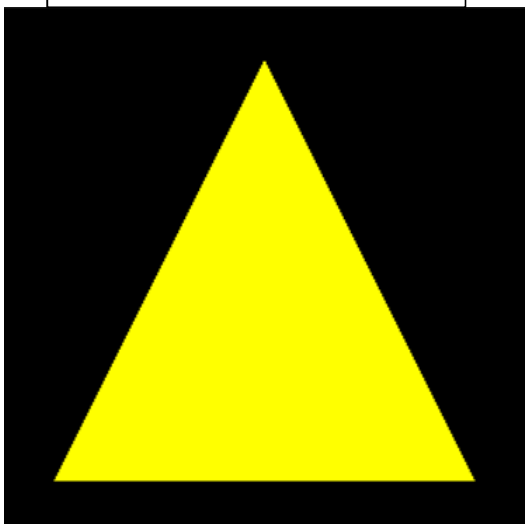
GL_LINE_STRIP



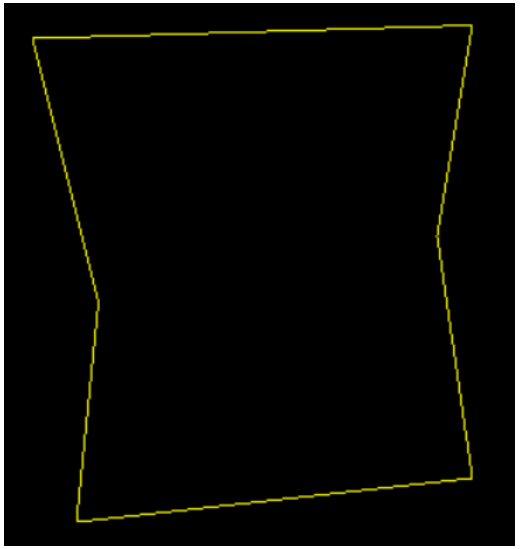
GL_LINE_LOOP



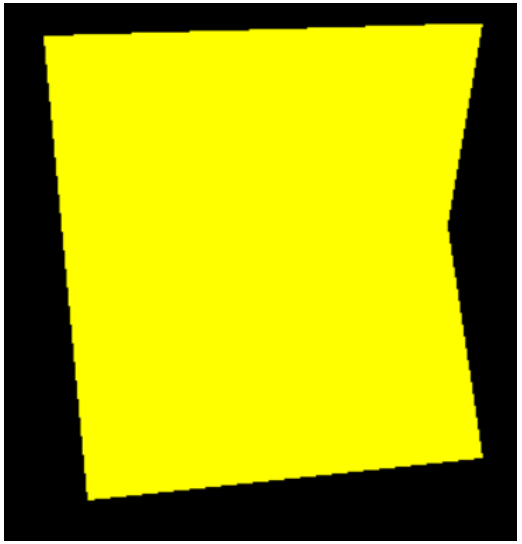
GL_POLYGON



```
glBegin(GL_LINE_LOOP);

glVertex2f(-5, 5);
glVertex2f(5, 5.3);
glVertex2f(4.2, 0.5);
glVertex2f(5, -5);
glVertex2f(-4, -6);
glVertex2f(-3.5, -1);

glEnd();
```



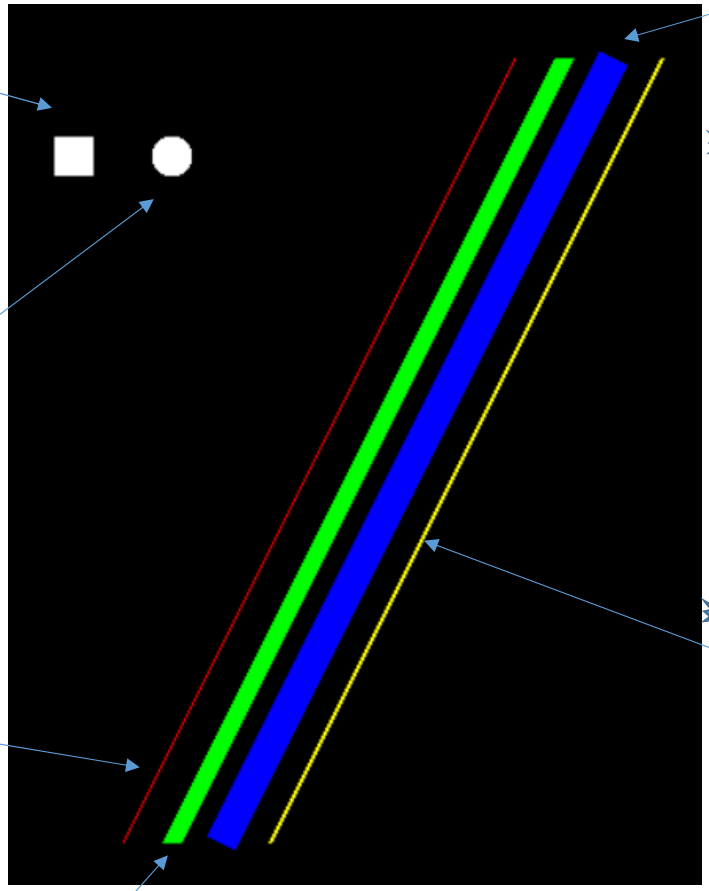
```
glBegin(GL_POLYGON);

glVertex2f(-5, 5);
glVertex2f(5, 5.3);
glVertex2f(4.2, 0.5);
glVertex2f(5, -5);
glVertex2f(-4, -6);
glVertex2f(-3.5, -1);

glEnd();
```



```
glPointSize(20);
glBegin(GL_POINTS);
glVertex2f(-0.6, 0.6);
glEnd();
```

```
glEnable(GL_POINT_SMOOTH);
glPointSize(20);
glBegin(GL_POINTS);
glVertex2f(-0.4, 0.6);
glEnd();
```

```
glColor3f(1,0,0);
glBegin(GL_LINES);
glVertex2f(-0.5, -0.8);
glVertex2f(0.3, 0.8);
glEnd();
```



```
glLineWidth(10);
glColor3f(0, 1, 0);
glBegin(GL_LINES);
glVertex2f(-0.4, -0.8);
glVertex2f(0.4, 0.8);
glEnd();
```

```
glLineWidth(15);
glColor3f(0, 0, 1);
glEnable(GL_LINE_SMOOTH);
glBegin(GL_LINES);
glVertex2f(-0.3, -0.8);
glVertex2f(0.5, 0.8);
glEnd();
```

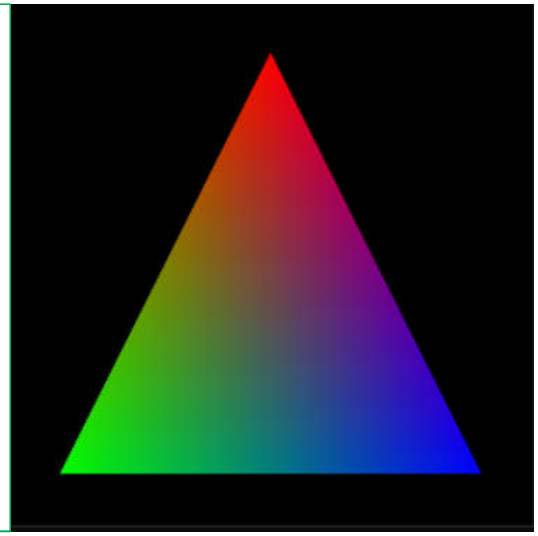
```
glLineWidth(1);
glEnable(GL_LINE_SMOOTH);
glColor3f(1, 1, 0);
glBegin(GL_LINES);
glVertex2f(-0.2, -0.8);
glVertex2f(0.6, 0.8);
glEnd();
```

Шаблонирование линий

```
glClear(GL_COLOR_BUFFER_BIT);
glLineWidth(5);
glColor3f(1, 1, 0);
glEnable(GL_LINE_STIPPLE);

glLineStipple(1, 0x3F07); //0011111100000111
glBegin(GL_LINES);
glVertex2f(-0.7, 0.5);
glVertex2f(0.7, 0.5);
glEnd();

glLineStipple(2, 0x3F07);
glBegin(GL_LINES);
glVertex2f(-0.7, 0.4);
glVertex2f(0.7, 0.4);
glEnd();

glLineStipple(3, 0x3F07);
glBegin(GL_LINES);
glVertex2f(-0.7, 0.3);
glVertex2f(0.7, 0.3);
glEnd();
glFinish();
```

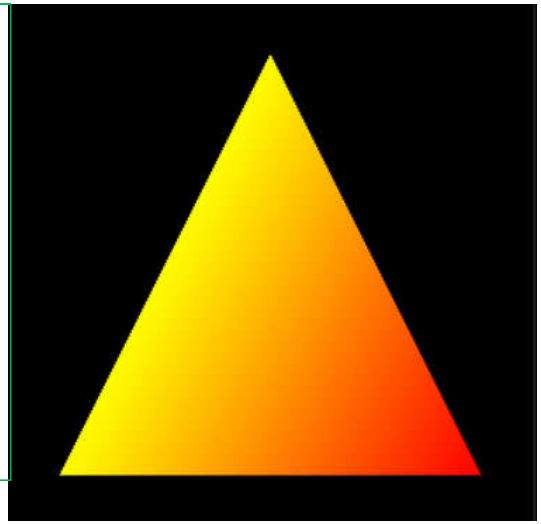
```
GLfloat red[3] =    { 1.0, 0.0, 0.0 };
GLfloat green[3] =  { 0.0, 1.0, 0.0 };
GLfloat blue[3] =   { 0.0, 0.0, 1.0 };
GLfloat yellow[3] = { 1.0, 1.0, 0.0 };
GLfloat white[3] =  { 1.0, 1.0, 1.0 };
GLfloat v[3][2] =   { 0.0, 0.8,
                     -0.8, -0.8,
                      0.8, -0.8};
```

```
glBegin(GL_TRIANGLES);
glColor3fv(red);
glVertex2fv(v[0]);

glColor3fv(green);
glVertex2fv(v[1]);

glColor3fv(blue);
glVertex2fv(v[2]);
glEnd();
```

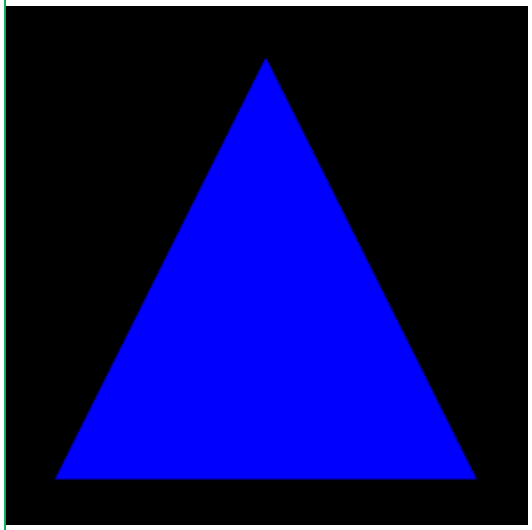


```
glShadeModel(GL_FLAT)
glShadeModel(GL_SMOOTH)
```

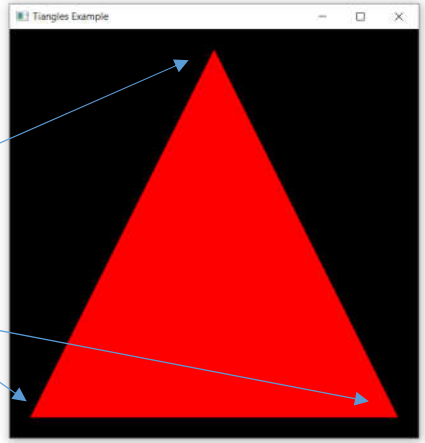
```
glColor3fv(yellow);
glBegin(GL_TRIANGLES);
glVertex2fv(v[0]);

glVertex2fv(v[1]);

glColor3fv(red);
glVertex2fv(v[2]);
glEnd();
```



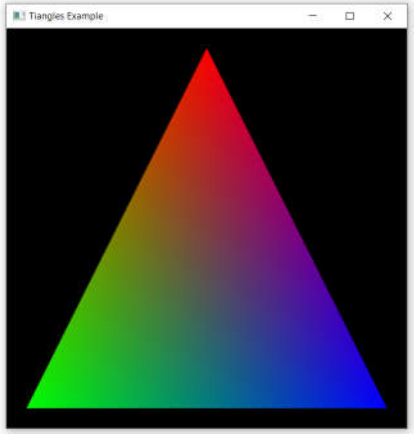
```
glShadeModel(GL_FLAT);

glBegin(GL_TRIANGLES);
glColor3fv(red);
glVertex2fv(v[0]);

glColor3fv(green);
glVertex2fv(v[1]);

glColor3fv(blue);
glVertex2fv(v[2]);
glEnd();
```



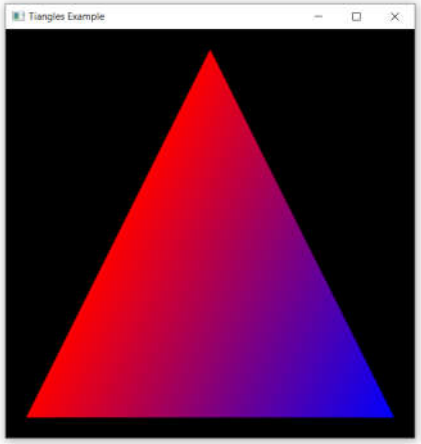
```
void RenderScene(void)
{
glClear(GL_COLOR_BUFFER_BIT);

glColor3f(1, 0, 0);
glBegin(GL_TRIANGLES);
glVertex2f(0, 0.9);
glVertex2f(-0.9, -0.9);
glVertex2f(0.9, -0.9);

glEnd();
glFinish();
}
```



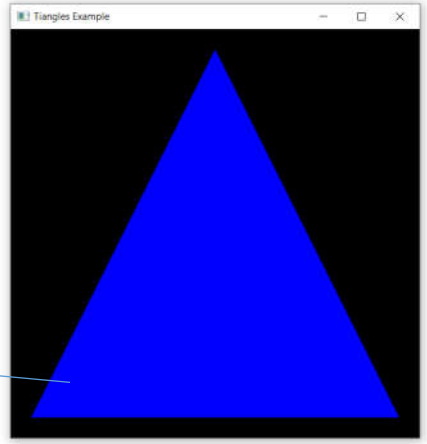

```
void RenderScene(void)
{
glClear(GL_COLOR_BUFFER_BIT);

glColor3f(1, 0, 0);
glBegin(GL_TRIANGLES);
glVertex2f(0, 0.9);
glColor3f(0, 1, 0);
glVertex2f(-0.9, -0.9);
glColor3f(0, 0, 1);

glVertex2f(0.9, -0.9);

glEnd();
glFinish();
}
```




```
void RenderScene(void)
{
glClear(GL_COLOR_BUFFER_BIT);

glColor3f(1, 0, 0);
glBegin(GL_TRIANGLES);
glVertex2f(0, 0.9);
glVertex2f(-0.9, -0.9);
glColor3f(0, 0, 1);

glVertex2f(0.9, -0.9);

glEnd();
glFinish();
}
```



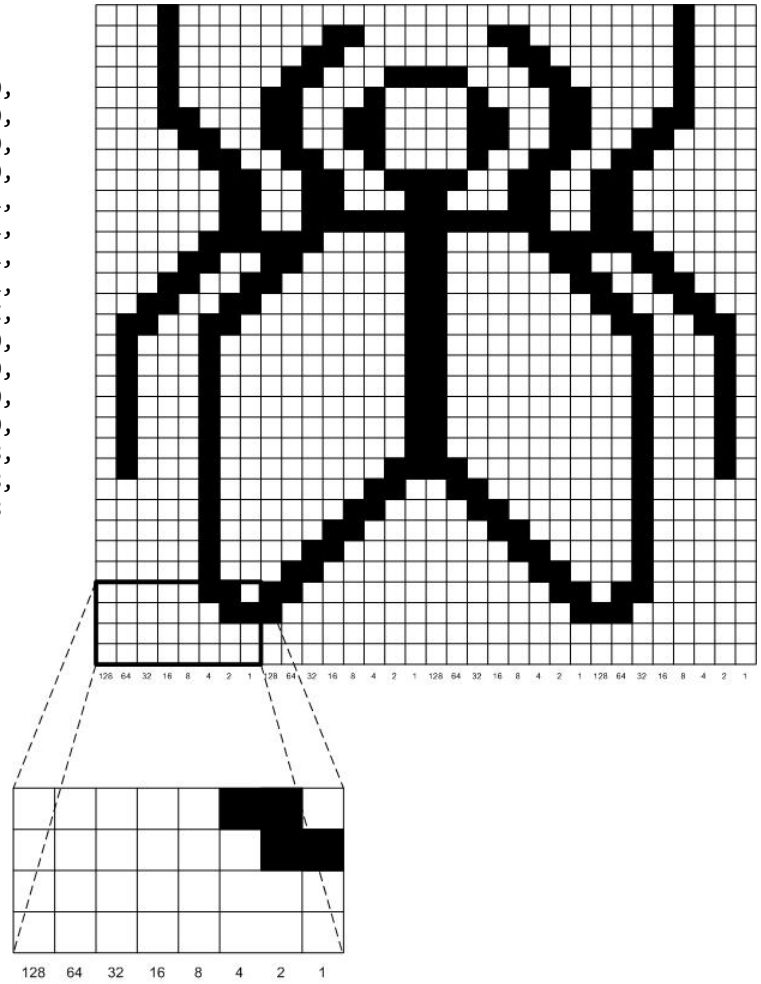

```
void RenderScene(void)
{
glClear(GL_COLOR_BUFFER_BIT);

glShadeModel(GL_FLAT);
glColor3f(1, 0, 0);
glBegin(GL_TRIANGLES);
glVertex2f(0, 0.9);
glColor3f(0, 1, 0);
glVertex2f(-0.9, -0.9);
glColor3f(0, 0, 1);

glVertex2f(0.9, -0.9);

glEnd();
glFinish();
}
```



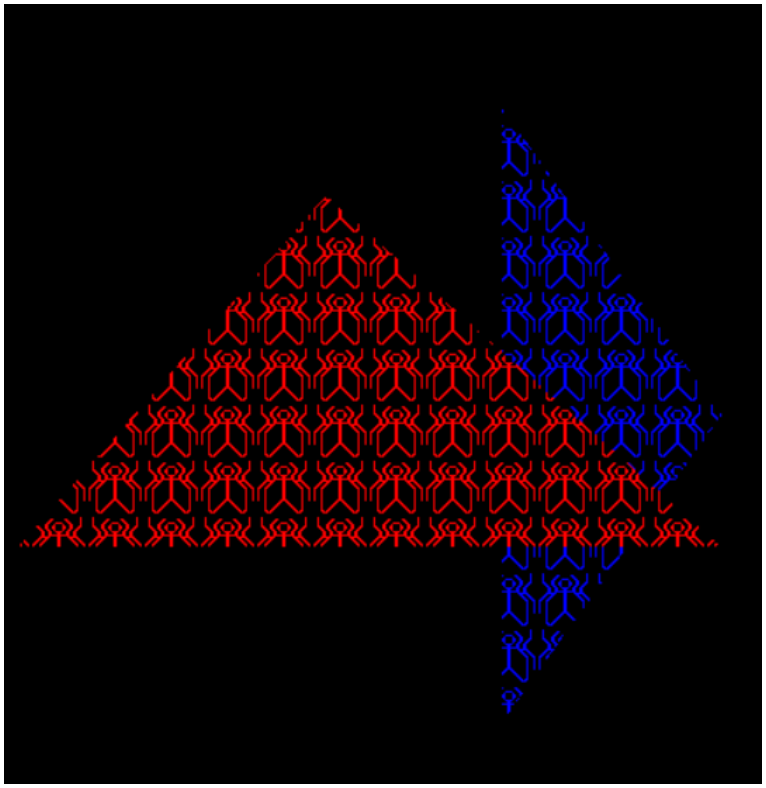

```
glShadeModel(GL_FLAT)
glShadeModel(GL_SMOOTH)
```

Шаблонирование полигонов

```
void Draw(void)
{
GLubyte stipple[] = {
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x03,0x80,0x01,0xC0,0x06,0xC0,0x03,0x60,
   0x04,0x60,0x06,0x20,0x04,0x30,0x0C,0x20,
   0x04,0x18,0x18,0x20,0x04,0x0C,0x30,0x20,
   0x04,0x06,0x60,0x20,0x44,0x03,0xC0,0x22,
   0x44,0x01,0x80,0x22,0x44,0x01,0x80,0x22,
   0x44,0x01,0x80,0x22,0x44,0x01,0x80,0x22,
   0x44,0x01,0x80,0x22,0x44,0x01,0x80,0x22,
   0x66,0x01,0x80,0x66,0x33,0x01,0x80,0xCC,
   0x19,0x81,0x81,0x98,0x0C,0xC1,0x83,0x30,
   0x07,0xE1,0x87,0xE0,0x03,0x3F,0xFC,0xC0,
   0x03,0x31,0x8C,0xC0,0x03,0x33,0xCC,0xC0,
   0x06,0x64,0x26,0x60,0x0C,0xCC,0x33,0x30,
   0x18,0xCC,0x33,0x18,0x10,0xC4,0x23,0x08,
   0x10,0x63,0xC6,0x08,0x10,0x30,0x0C,0x08,
   0x10,0x18,0x18,0x08,0x10,0x00,0x00,0x08
};
glClear(GL_COLOR_BUFFER_BIT);

glEnable(GL_POLYGON_STIPPLE);
glPolygonStipple(stipple);

glColor3f(0, 0, 1);
glBegin(GL_TRIANGLES);
glVertex3f(0.2, 0.7, 0.1);
glVertex3f(0.2, -0.7, 0.1);
glVertex3f(0.7, 0.0, 0.1);

glColor3f(1, 0, 0);
glVertex3f(-0.2, 0.5, -0.1);
glVertex3f(-0.9, -0.3, -0.1);
glVertex3f(0.7, -0.3, -0.1);

glEnd();
glFinish();
}
```

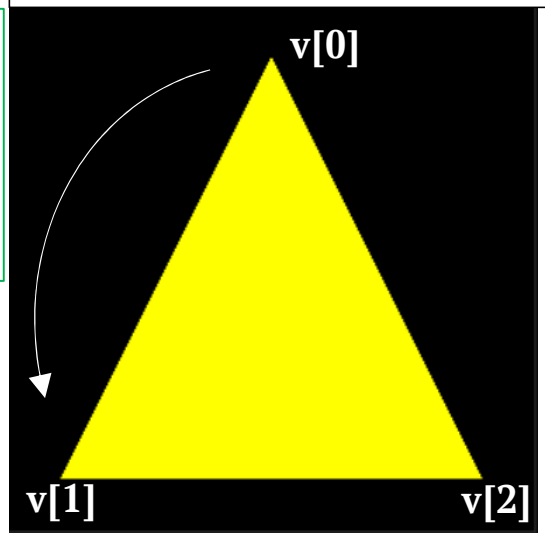


По умолчанию для каждого байта наиболее значимым битом считается младший. Порядок битов может быть изменен вызовом **glPixelStore*()**.



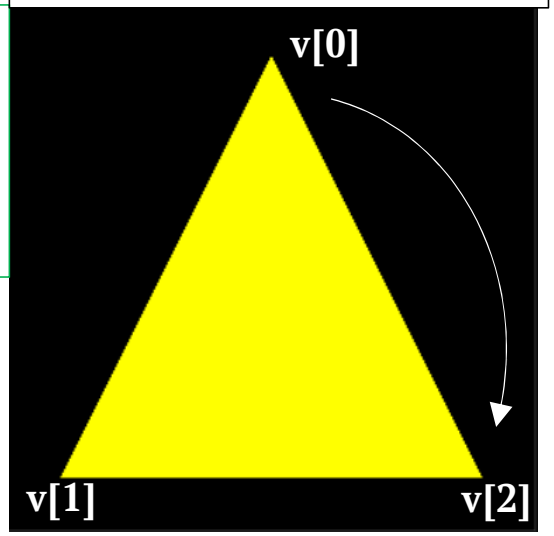
```
glBegin(GL_TRIANGLES);
glVertex2fv(v[0]);
glVertex2fv(v[1]);
glVertex2fv(v[2]);
glEnd();
```

FRONT //CCW – Contra ClockWise)



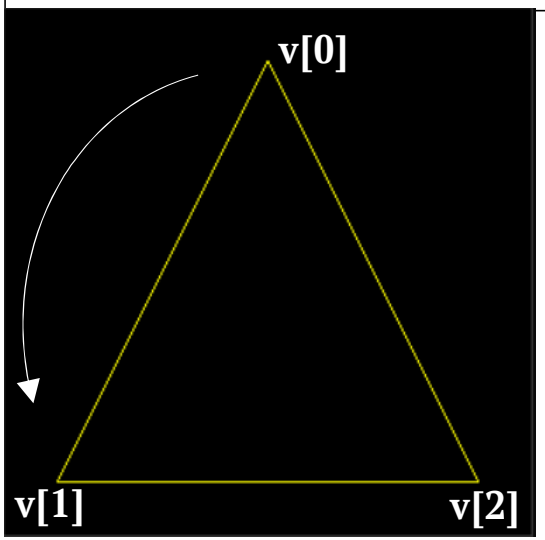

```
glBegin(GL_TRIANGLES);
glVertex2fv(v[0]);
glVertex2fv(v[2]);
glVertex2fv(v[1]);
glEnd();
```

BACK //CW – ClockWise)



```
glPolygonMode(GL_FRONT, GL_LINE);
glPolygonMode(GL_BACK, GL_POINT);
glBegin(GL_TRIANGLES);
glVertex2fv(v[0]);
glVertex2fv(v[1]);
glVertex2fv(v[2]);
glEnd();
```

FRONT //CCW – Contra ClockWise)



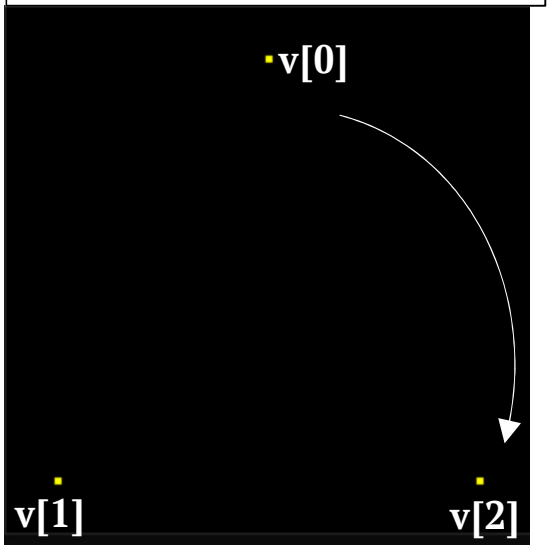

```
glPolygonMode(GL_FRONT_AND_BACK,
              GL_FILL);
```

```
glFrontFace(GL_CW);
```

```
glEnable(GL_CULL_FACE);
glCullFace(GL_BACK);
```

```
glPolygonMode(GL_FRONT, GL_LINE);
glPolygonMode(GL_BACK, GL_POINT);
glBegin(GL_TRIANGLES);
glVertex2fv(v[0]);
glVertex2fv(v[2]);
glVertex2fv(v[1]);
glEnd();
```

BACK //CW – ClockWise)

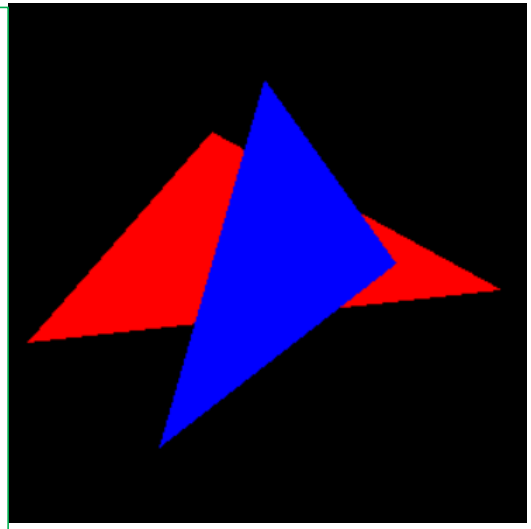


Проверка глубины

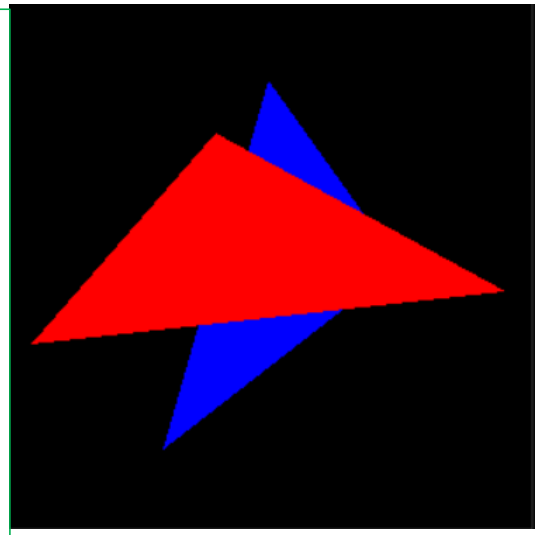
```
glEnable(GL_DEPTH_TEST);
```

```
GLfloat red[3] = { 1.0, 0.0, 0.0 };
GLfloat blue[3] = { 0.0, 0.0, 1.0 };
GLfloat vr[3][2] = { -0.2, 0.5, -0.9, -0.3, 0.9, -0.1 };
GLfloat vb[3][2] = { 0.0, 0.7, -0.4, -0.7, 0.5, 0.0 };
```

```
glBegin(GL_TRIANGLES);

glColor3fv(red);
glVertex2fv(vr[0]);
glVertex2fv(vr[1]);
glVertex2fv(vr[2]);

glColor3fv(blue);
glVertex2fv(vb[0]);
glVertex2fv(vb[1]);
glVertex2fv(vb[2]);

glEnd();
```

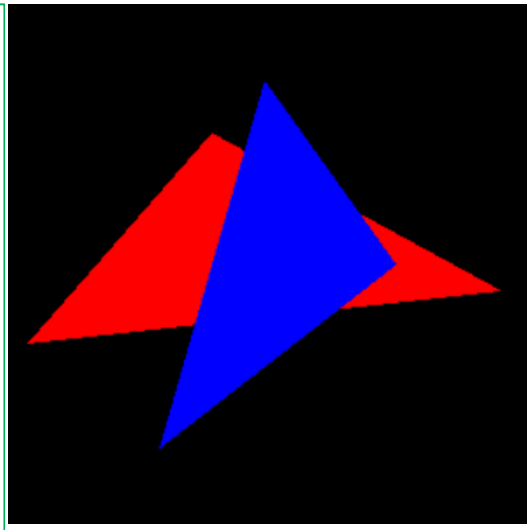


```
glBegin(GL_TRIANGLES);

glColor3fv(blue);
glVertex2fv(vb[0]);
glVertex2fv(vb[1]);
glVertex2fv(vb[2]);

glColor3fv(red);
glVertex2fv(vr[0]);
glVertex2fv(vr[1]);
glVertex2fv(vr[2]);

glEnd();
```

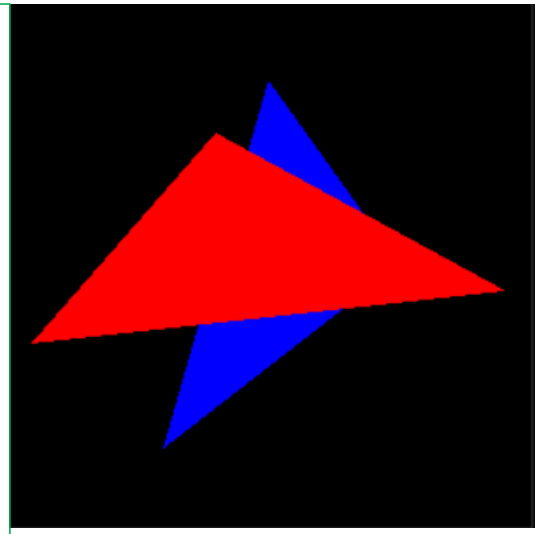


```
GLfloat red[3] = { 1.0, 0.0, 0.0 };
GLfloat blue[3] = { 0.0, 0.0, 1.0 };
GLfloat vr[3][3] = { -0.2, 0.5,  0.1,   -0.9, -0.3,  0.1,   0.9, -0.1,  0.1 };
GLfloat vb[3][3] = {  0.0, 0.7, -0.1,   -0.4, -0.7, -0.1,   0.5,  0.0, -0.1 };
```

```
glBegin(GL_TRIANGLES);

glColor3fv(red);
glVertex3fv(vr[0]);
glVertex3fv(vr[1]);
glVertex3fv(vr[2]);

glColor3fv(blue);
glVertex3fv(vb[0]);
glVertex3fv(vb[1]);
glVertex3fv(vb[2]);

glEnd();
```



```
glEnable(GL_DEPTH_TEST);
glBegin(GL_TRIANGLES);

glColor3fv(red);
glVertex3fv(vr[0]);
glVertex3fv(vr[1]);
glVertex3fv(vr[2]);

glColor3fv(blue);
glVertex3fv(vb[0]);
glVertex3fv(vb[1]);
glVertex3fv(vb[2]);

glEnd();
```



```
glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH);
```

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Двойная буферизация

```
#include "glut.h"

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glRectf(-0.5, -0.5, 0.5, 0.5);
    glFinish();
}

void main()
{
    glutInitDisplayMode(GLUT_SINGLE);
    glutCreateWindow("OpenGL. First step");
    glutDisplayFunc(display);
    glutMainLoop();
}
```

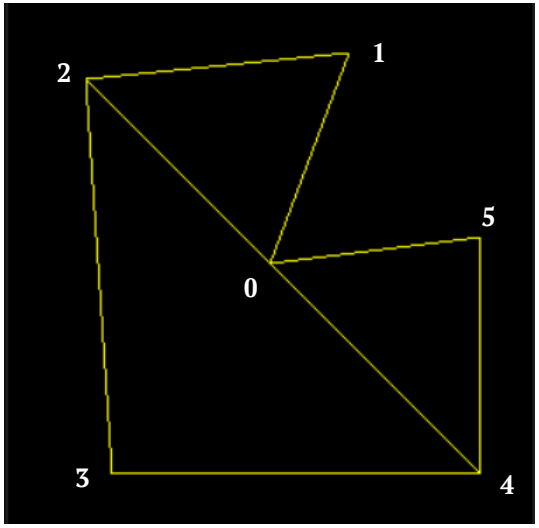
```
#include "glut.h"

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glRectf(-0.5, -0.5, 0.5, 0.5);
    glutSwapBuffers();
}

void main()
{
    glutInitDisplayMode(GLUT_DOUBLE);
    glutCreateWindow("OpenGL. First step");
    glutDisplayFunc(display);
    glutMainLoop();
}
```

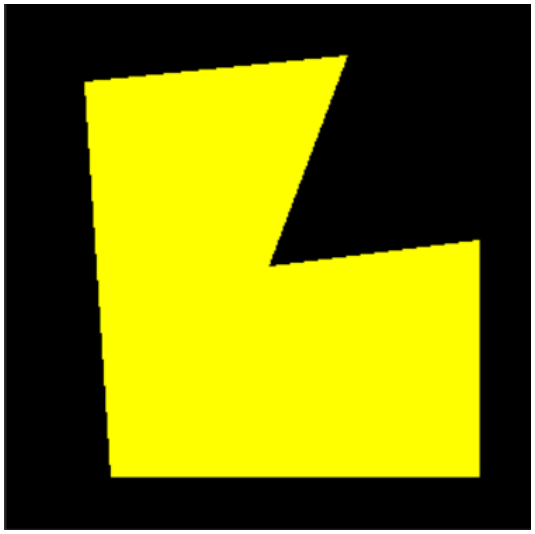
```
GLfloat v[6][2] = { {0.0, 0.0}, {0.3, 0.8}, {-0.7, 0.7}, {-0.6, -0.8}, {0.8, -0.8}, {0.8, 0.1} };
```

```
glPolygonMode(GL_FRONT,
              GL_LINE);
```

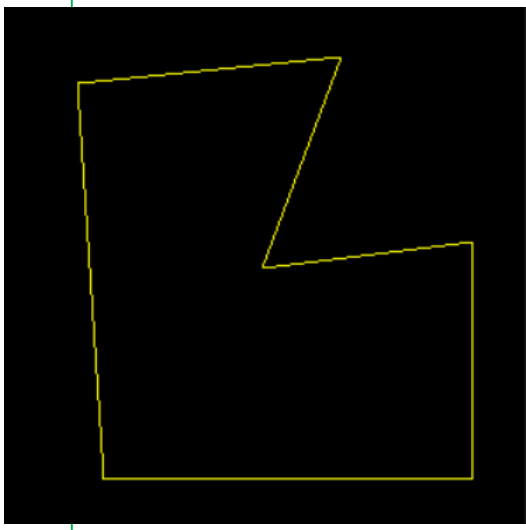
```
glBegin(GL_TRIANGLES);

glVertex2fv(v[0]);
glVertex2fv(v[1]);
glVertex2fv(v[2]);

glVertex2fv(v[2]);
glVertex2fv(v[3]);
glVertex2fv(v[4]);

glVertex2fv(v[4]);
glVertex2fv(v[5]);
glVertex2fv(v[0]);
glEnd();
```

```
glPolygonMode(GL_FRONT,
              GL_FILL);
```

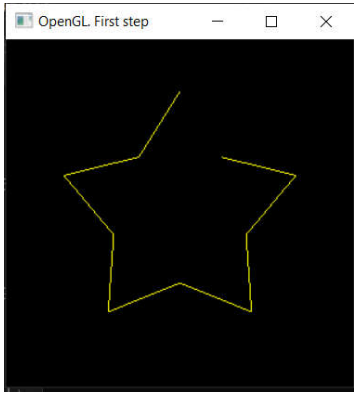


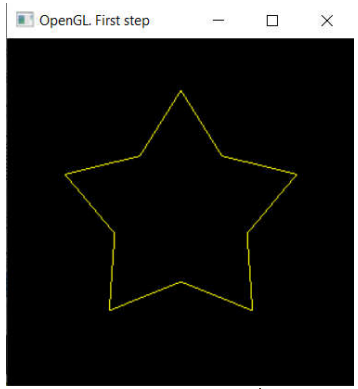


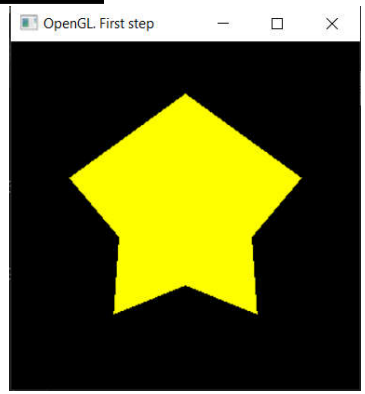
```
glPolygonMode(GL_FRONT,
              GL_LINE);
glBegin(GL_TRIANGLES);

glVertex2fv(v[0]);
glVertex2fv(v[1]);
glEdgeFlag(false);
glVertex2fv(v[2]);
glEdgeFlag(true);

glVertex2fv(v[2]);
glVertex2fv(v[3]);
glEdgeFlag(false);
glVertex2fv(v[4]);
glEdgeFlag(true);

glVertex2fv(v[4]);
glVertex2fv(v[5]);
glEdgeFlag(false);
glVertex2fv(v[0]);
glEdgeFlag(true);
glEnd();
```



```c
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 0.0);
    //glBegin(GL_LINE_STRIP);
    //glBegin(GL_LINE_LOOP);
    glBegin(GL_POLYGON);
    float r1 = 0.7;
    float r2 = 0.4;
    int k = 5;
    float a = M_PI/2;
    float da = 2 * M_PI / k;
    for (int i = 0; i < k; i++)
    {
        glVertex2f(r1 * cos(a), r1 * sin(a));
        glVertex2f(r2 * cos(a+da/2), r2 * sin(a+da/2));
        a += da;
    }
    glEnd();
    glFinish();
}
```







Векторная форма функции glVertex*v

```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 0.0);
    const int k = 5;
    GLfloat v[4 * k];

    MakeVertexes(5, 0.7, 0.4, M_PI / 2, v);

    glBegin(GL_LINE_LOOP);
    for (int i = 0; i < k; i++)
    {
        glVertex2fv(&v[i * 4]);
        glVertex2fv(&v[i * 4 + 2]);
    }
    glEnd();

    glFinish();
}
```

```
void MakeVertexes(int k, float r1, float r2,
                  float a, float v[] )
{
    GLfloat da = 2 * M_PI / k;
    for (int i = 0; i < k; i++)
        {
        v[4 * i] = r1 * cos(a);
        v[4 * i + 1] = r1 * sin(a);
        v[4 * i + 2] = r2 * cos(a + da / 2);
        v[4 * i + 3] = r2 * sin(a + da / 2);
        a += da;
        printf("%d   {%.2f, %.2f}\n",
                i, v[2 * i], v[2 * i + 1]);
        }
}
```

Вершинные массивы

```
void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 1.0, 0.0);
   const int k = 5;
   GLfloat v[4 * k];

   MakeVertexes(5, 0.7, 0.4, M_PI / 2, v);

   glVertexPointer(2, GL_FLOAT, 0, &v);
   glEnableClientState(GL_VERTEX_ARRAY);
   glDrawArrays(GL_LINE_LOOP, 0, 2 * k);


   glFinish();
}
```

```
void MakeVertexes(int k, float r1, float r2,
                  float a, float v[] )
{
    GLfloat da = 2 * M_PI / k;
    for (int i = 0; i < k; i++)
       {
       v[4 * i] = r1 * cos(a);
       v[4 * i + 1] = r1 * sin(a);
       v[4 * i + 2] = r2 * cos(a + da / 2);
       v[4 * i + 3] = r2 * sin(a + da / 2);
       a += da;
       printf("%d   {%.2f, %.2f}\n",
              i, v[2 * i], v[2 * i + 1]);
       }
}
```

Вершинные массивы

```
void glVertexPointer (Glint size, GLenum type, GLsizei stride, const GLvoid *pointer);
void glColorPointer (Glint size, GLenum type, GLsizei stride, const GLvoid *pointer);
void glIndexPointer (GLenum type, GLsizei stride, const GLvoid *pointer);
void glNormalPointer (GLenum type, GLsizei stride, const GLvoid *pointer);
void glTexCoordPointer (Glint size, GLenum type, GLsizei stride, const GLvoid *pointer);
void glEdgeFlagPointer (GLsizei stride, const GLvoid *pointer);
```

glVertexPointer	вершины
glColorPointer	**цвета**
glIndexPointer	индексированные цвета
glNormalPointer	нормали
glTexCoordPointer	текстурные координаты
glEdgeFlagPointer	флаги ребер

```
glEnableClientState(GL_VERTEX_ARRAY)
glEnableClientState(GL_COLOR_ARRAY)
glEnableClientState(GL_INDEX_ARRAY);
glEnableClientState(GL_NORMAL_ARRAY);
glEnableClientState(GL_TEXCOORD_ARRAY)
glEnableClientState(GL_EDGEFLAG_ARRAY);
```