

## Лабораторная работа № 7

**Тема:** Моделирование 3D объектов с использованием OpenGL.

**Цель:** Научиться строить трехмерные объекты средствами OpenGL.

### Краткая теория

#### *Трехмерные фигуры*

Функции для построения сплошных 3D фигур:

- auxSolidSphere(R) // сфера
- auxSolidCube(width) // куб
- auxSolidBox(width, height, depth) // коробка
- auxSolidTorus(r,R) // тор
- auxSolidCylinder(r,height) // цилиндр
- auxSolidCone(r,height) // конус
- auxSolidIcosahedron(width) // многогранники
- auxSolidOctahedron(width)
- auxSolidTetrahedron(width)
- auxSolidDodecahedron(width)
- auxSolidTeapot(width) // рисует чайник

Для построения каркасных фигур вместо Solid необходимо использовать Wire. Пример: auxWireCube(1) // рисует каркасную модель куба.

#### *Преобразование объектов в пространстве*

В процессе построения изображения координаты вершин подвергаются определенным преобразованиям. Подобным преобразованиям подвергаются заданные векторы нормали.

Изначально камера находится в начале координат и направлена вдоль отрицательного направления оси Oz.

В OpenGL существуют две матрицы, последовательно применяющиеся в преобразовании координат. Одна из них – матрица моделирования (modelview matrix), а другая – матрица проецирования (projection matrix). Первая служит для задания положения объекта и его ориентации, вторая отвечает за выбранный способ проецирования. OpenGL поддерживает два типа проецирования – параллельное и перспективное.

Существует набор различных процедур, умножающих текущую матрицу (моделирования или проецирования) на матрицу выбранного геометрического преобразования.

Текущая матрица задается при помощи процедуры glMatrixMode(GLenum mode). Параметр mode может принимать значения GL\_MODELVIEW, GL\_TEXTURE или GL\_PROJECTION, позволяя выбирать в качестве текущей матрицы матрицу моделирования (видовую матрицу), матрицу проецирования или матрицу преобразования текстуры.

Процедура `glLoadIdentity()` устанавливает единичную текущую матрицу.

Обычно задание соответствующей матрицы начинается с установки единичной матрицы и последовательного применения матриц геометрических преобразований.

Преобразование переноса задается процедурой `glTranslate{f d}(TYPE x, TYPE y, TYPE z)`, обеспечивающей перенос объекта на величину  $(x, y, z)$ .

Преобразование поворота задаётся процедурой `glRotate{f d}(TYPE angle, TYPE x, TYPE y, TYPE z)`, обеспечивающей поворот на угол `angle` в направлении против часовой стрелки вокруг прямой с направляющим вектором  $(x, y, z)$ .

Преобразование масштабирования задаётся процедурой `glScale{f d}(TYPE x, TYPE y, TYPE z)`.

Если указано несколько преобразований, то текущая матрица в результате будет последовательно умножена на соответствующие матрицы.

### ***Получение проекций***

Видимым объемом при перспективном преобразовании в OpenGL является усеченная пирамида.

Для задания перспективного преобразования в OpenGL служит процедура:

`glFrustum(GLdouble teft, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)`.

Параметры определяют плоскости, по которым проводится отсечение.

Величины `near` и `far` должны быть неотрицательными.

Иногда для задания перспективного преобразования удобнее воспользоваться следующей процедурой из библиотеки утилит OpenGL:

`gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear, GLdouble zFar)`.

Эта процедура создает матрицу для задания симметричного поля зрения и умножает текущую матрицу на неё. Здесь `fovy` – угол зрения камеры в плоскости  $Oxz$ , лежащий в диапазоне  $[0, 180]$ . Параметр `aspect` – отношение ширины области к её высоте, `zNear` и `zFar` – расстояния вдоль отрицательного направления оси  $Oz$ , определяющие ближнюю и дальнюю плоскости отсечения.

Существует ещё одна удобная функция для задания перспективного проецирования:

`gluLookAt (GLdouble eyeX, GLdouble eyeY, GLdouble eyeZ, GLdouble centerX, GLdouble centerY, GLdouble centerZ, GLdouble upX, GLdouble upY, GLdouble upZ)`.

Вектор  $(eyeX, eyeY, eyeZ)$  задаёт положение наблюдателя, вектор  $(centerX, centerY, centerZ)$  – направление на центр сцены, а вектор  $(upX, upY, upZ)$  – направление вверх.

В случае параллельного проецирования видимым объемом является прямоугольный параллелепипед. Для задания параллельного проецирования служит процедура:

`glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far).`

Параметры `left` и `right` определяют координаты левой и правой вертикальных плоскостей отсечения, а `bottom` и `top` – нижней и верхней горизонтальных плоскостей.

Следующим шагом в задании проецирования (после выбора параллельного или перспективного преобразования) является задание области в окне, в которую будет помещено получаемое изображение. Для этого служит процедура:

`glViewport(GLint x, GLint y, GLsizei width, GLsizei height).`

Здесь `(x, y)` задаёт нижний левый угол прямоугольной области в окне, а `width` и `height` являются её шириной и высотой.

OpenGL содержит стек матриц для каждого из трёх типов преобразований. При этом текущую матрицу можно поместить в стек или извлечь матрицу из стека и сделать её текущей.

Для помещения текущей матрицы в стек служит процедура `glPushMatrix()`, для извлечения матрицы из стека – процедура `glPopMatrix()`.

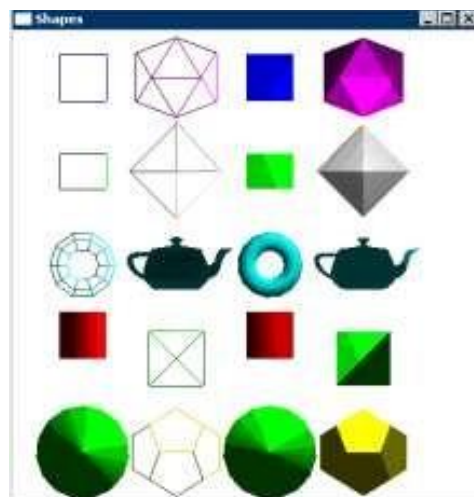
### ***Задание моделей закрашивания***

Линия или заполненная грань могут быть нарисованы одним цветом (плоское закрашивание, `GL_FLAT`) или путём интерполяции цветов в вершинах (закрашивание Гуро, `GL_SMOOTH`).

Для задания режима закрашивания служит процедура `glShadeModel(GLenum mode)`, где параметр `mode` принимает значение `GL_SMOOTH` или `GL_FLAT`.

### ***Пример построения 3D объектов.***

#### **Результат выполнения программы**



## Текст программы

```
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/GLaux.h>
void CALLBACK resize(int width,int height)
{
    glViewport(0,0,width,height);
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    glOrtho(-5,5, -5,5, 2,12);
    gluLookAt( 0,0,5, 0,0,0, 0,1,0 );
    glMatrixMode( GL_MODELVIEW );
}
void CALLBACK display(void)
{
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
glPushMatrix();
    glTranslated(0.5,4,0);
    glColor3d(0,0,1);
    auxSolidCube(1);        // куб
    glTranslated(0,-2,0);
    glColor3d(0,1,0);
    auxSolidBox(1,0.75,0.5); // коробка
    glTranslated(0,-2,0);
    glColor3d(0,1,1);
    auxSolidTorus(0.2,0.5); // тор
    glTranslated(0,-2,0);
    glColor3d(1,0,0);
    auxSolidCylinder(0.5,1); // цилиндр
    glTranslated(0,-2,0);
    glColor3d(0,1,0);
    auxSolidCone(1,1);      // конус
    glTranslated(2,8,0);
    glColor3d(1,0,1);
    auxSolidIcosahedron(1); // многогранники
    glTranslated(0,-2,0);
    glColor3d(1,1,1);
    auxSolidOctahedron(1);
    glTranslated(0,-2,0);
    glColor3d(0,1,1);
    auxSolidTeapot(0.7);    // чайник
    glTranslated(0,-2,0);
    glColor3d(0,1,0);
```

```

    auxSolidTetrahedron(1);
    glTranslated(0,-2,0);
    glColor3d(1,1,0);
    auxSolidDodecahedron(1);
    glTranslated(-6,8,0);
    glColor3d(0,0,1);
    auxWireCube(1);      // каркасная модель куба
    glTranslated(0,-2,0);
    glColor3d(0,1,0);
    auxWireBox(1,0.75,0.5); // каркасная модель параллелограмма
    glTranslated(0,-2,0);
    glColor3d(0,1,1);
    auxWireTorus(0.2,0.5); // каркасная модель тора
    glTranslated(0,-2,0);
    glColor3d(1,0,0);
    auxWireCylinder(0.5,1); // каркасная модель цилиндра
    glTranslated(0,-2,0);
    glColor3d(0,1,0);
    auxWireCone(1,1);    // каркасная модель конуса
    glTranslated(2,8,0);
    glColor3d(1,0,1);
    auxWireIcosahedron(1); // каркасные модели многогранников
    glTranslated(0,-2,0);
    glColor3d(1,1,1);
    auxWireOctahedron(1);
    glTranslated(0,-2,0);
    glColor3d(0,1,1);
    auxWireTeapot(0.7);  // каркасная модель чайника
    glTranslated(0,-2,0);
    glColor3d(0,1,0);
    auxWireTetrahedron(1);
    glTranslated(0,-2,0);
    glColor3d(1,1,0);
    auxWireDodecahedron(1);
    glPopMatrix();
    auxSwapBuffers();
} void main()
{
    float pos[4] = { 3,3,3,1 };
    float dir[3] = { -1,-1,-1 };
    auxInitPosition( 50, 10, 400, 400);
    auxInitDisplayMode( AUX_RGB | AUX_DEPTH | AUX_DOUBLE );
    auxInitWindow( "Shapes" );
    auxIdleFunc(display);
    auxReshapeFunc(resize);
}

```

```

glEnable(GL_DEPTH_TEST);
glEnable(GL_COLOR_MATERIAL);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glLightfv(GL_LIGHT0, GL_POSITION, pos);
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, dir);
auxMainLoop(display);
}

```

***Пример функции для построения 3D буквы.***

**Текст программы**

```

int per = 20;
void paintGL()
{
    Gl.glRotated(xrot, 1.0, 0.0, 0.0);
    Gl.glRotated(yrot, 0.0, 1.0, 0.0);

    Gl.glColor3f(1.0f, 0.0f, 0.0f);
    Gl.glBegin(Gl.GL_LINES);
    Gl.glVertex3i(0, 0, 0);
    Gl.glVertex3i(Width, 0, 0);
    Gl.glColor3f(0.0f, 0.0f, 1.0f);
    Gl.glVertex3i(0, 0, 0);
    Gl.glVertex3i(0, Width, 0);
    Gl.glColor3f(0.0f, 1.0f, 0.0f);
    Gl.glVertex3i(0, 0, 0);
    Gl.glVertex3i(0, 0, Width);
    Gl.glColor3f(0.6f, 0.3f, 0.5f);
    Gl.glEnd();
    Gl.glFlush();

    // Gl.glEnable(Gl.GL_DEPTH_TEST);
    Gl.glBegin(Gl.GL_QUADS);

    Gl.glColor3d(1.0, 0.0, 0.0);
    //передняя грань
    Gl.glNormal3f(0.0f, 0.0f, 1.0f);
    Gl.glVertex3d(-0.4 * per, 0 * per, 0.1 * per);
    Gl.glVertex3d(-0.2 * per, 0 * per, 0.1 * per);
    Gl.glVertex3d(-0.2 * per, 1 * per, 0.1 * per);
    Gl.glVertex3d(-0.4 * per, 1 * per, 0.1 * per);

    Gl.glVertex3d(-0.2 * per, 0.8 * per, 0.1 * per);
    Gl.glVertex3d( 0.2 * per, 0.8 * per, 0.1 * per);

```

```
Gl.glVertex3d( 0.2 * per, 1 * per, 0.1 * per);  
Gl.glVertex3d(-0.2 * per, 1 * per, 0.1 * per);
```

```
Gl.glVertex3d(0.2 * per, 0 * per, 0.1 * per);  
Gl.glVertex3d(0.4 * per, 0 * per, 0.1 * per);  
Gl.glVertex3d(0.4 * per, 1 * per, 0.1 * per);  
Gl.glVertex3d(0.2 * per, 1 * per, 0.1 * per);
```

```
Gl.glVertex3d(-0.6 * per, -0.2 * per, 0.1 * per);  
Gl.glVertex3d(0.6 * per, -0.2 * per, 0.1 * per);  
Gl.glVertex3d(0.6 * per, 0 * per, 0.1 * per);  
Gl.glVertex3d(-0.6 * per, 0 * per, 0.1 * per);
```

```
Gl.glVertex3d(-0.6 * per, -0.5 * per, 0.1 * per);  
Gl.glVertex3d(-0.4 * per, -0.5 * per, 0.1 * per);  
Gl.glVertex3d(-0.4 * per, -0.2 * per, 0.1 * per);  
Gl.glVertex3d(-0.6 * per, -0.2 * per, 0.1 * per);
```

```
Gl.glVertex3d(0.4 * per, -0.5 * per, 0.1 * per);  
Gl.glVertex3d(0.6 * per, -0.5 * per, 0.1 * per);  
Gl.glVertex3d(0.6 * per, -0.2 * per, 0.1 * per);  
Gl.glVertex3d(0.4 * per, -0.2 * per, 0.1 * per);
```

```
Gl.glColor3d(1.0, 0.0, 0.0);
```

```
// ВЕРХНЯЯ
```

```
Gl.glNormal3f(0.0f, 1.0f, 0.0f);
```

```
Gl.glVertex3d(-0.4 * per, 0 * per, 0.1 * per);  
Gl.glVertex3d(-0.6 * per, 0 * per, 0.1 * per);  
Gl.glVertex3d(-0.6 * per, 0 * per, -0.1 * per);  
Gl.glVertex3d(-0.4 * per, 0 * per, -0.1 * per);
```

```
Gl.glVertex3d(0.6 * per, 0 * per, 0.1 * per);  
Gl.glVertex3d(0.4 * per, 0 * per, 0.1 * per);  
Gl.glVertex3d(0.4 * per, 0 * per, -0.1 * per);  
Gl.glVertex3d(0.6 * per, 0 * per, -0.1 * per);
```

```
Gl.glColor3d(1.0, 0.0, 0.0);  
Gl.glVertex3d(-0.4 * per, 1 * per, 0.1 * per);  
Gl.glVertex3d( 0.4 * per, 1 * per, 0.1 * per);  
Gl.glVertex3d( 0.4 * per, 1 * per, -0.1 * per);  
Gl.glVertex3d(-0.4 * per, 1 * per, -0.1 * per);
```

```
Gl.glColor3d(1.0, 0.0, 0.0);
```

// НИЖНЯЯ

Gl.glNormal3f(0.0f, -1.0f, 0.0f);

Gl.glVertex3d(-0.6 \* per, -0.5 \* per, -0.1 \* per);

Gl.glVertex3d(-0.6 \* per, -0.5 \* per, 0.1 \* per);

Gl.glVertex3d(-0.4 \* per, -0.5 \* per, 0.1 \* per);

Gl.glVertex3d(-0.4 \* per, -0.5 \* per, -0.1 \* per);

Gl.glVertex3d(0.4 \* per, -0.5 \* per, -0.1 \* per);

Gl.glVertex3d(0.4 \* per, -0.5 \* per, 0.1 \* per);

Gl.glVertex3d(0.6 \* per, -0.5 \* per, 0.1 \* per);

Gl.glVertex3d(0.6 \* per, -0.5 \* per, -0.1 \* per);

Gl.glColor3d(1.0, 0.0, 0.0);

Gl.glVertex3d(-0.4 \* per, -0.2 \* per, -0.1 \* per);

Gl.glVertex3d(-0.4 \* per, -0.2 \* per, 0.1 \* per);

Gl.glVertex3d(0.4 \* per, -0.2 \* per, 0.1 \* per);

Gl.glVertex3d(0.4 \* per, -0.2 \* per, -0.1 \* per);

// БОКОВЫЕ

Gl.glNormal3f(-1.0f, 0.0f, 0.0f);

Gl.glColor3d(1.0, 0.0, 0.0);

Gl.glVertex3d(-0.6 \* per, -0.5 \* per, -0.1 \* per);

Gl.glVertex3d(-0.6 \* per, -0.5 \* per, 0.1 \* per);

Gl.glVertex3d(-0.6 \* per, 0 \* per, 0.1 \* per);

Gl.glVertex3d(-0.6 \* per, 0 \* per, -0.1 \* per);

Gl.glVertex3d(-0.4 \* per, -0.5 \* per, -0.1 \* per);

Gl.glVertex3d(-0.4 \* per, -0.5 \* per, 0.1 \* per);

Gl.glVertex3d(-0.4 \* per, 0 \* per, 0.1 \* per);

Gl.glVertex3d(-0.4 \* per, 0 \* per, -0.1 \* per);

Gl.glColor3d(1.0, 0.0, 0.0);

Gl.glVertex3d(-0.4 \* per, 0 \* per, -0.1 \* per);

Gl.glVertex3d(-0.4 \* per, 0 \* per, 0.1 \* per);

Gl.glVertex3d(-0.4 \* per, 1 \* per, 0.1 \* per);

Gl.glVertex3d(-0.4 \* per, 1 \* per, -0.1 \* per);

Gl.glColor3d(1.0, 0.0, 0.0);

Gl.glVertex3d(0.4 \* per, 0 \* per, -0.1 \* per);

Gl.glVertex3d(0.4 \* per, 0 \* per, 0.1 \* per);

Gl.glVertex3d(0.4 \* per, 1 \* per, 0.1 \* per);

Gl.glVertex3d(0.4 \* per, 1 \* per, -0.1 \* per);



```
Gl.glColor3d(1.0, 0.0, 0.0);
Gl.glVertex3d(0.4 * per, -0.5 * per, -0.1 * per);
Gl.glVertex3d(0.4 * per, -0.5 * per, 0.1 * per);
Gl.glVertex3d(0.4 * per, 0 * per, 0.1 * per);
Gl.glVertex3d(0.4 * per, 0 * per, -0.1 * per);
```

```
Gl.glColor3d(1.0, 0.0, 0.0);
Gl.glVertex3d(0.6 * per, -0.5 * per, -0.1 * per);
Gl.glVertex3d(0.6 * per, -0.5 * per, 0.1 * per);
Gl.glVertex3d(0.6 * per, 0 * per, 0.1 * per);
Gl.glVertex3d(0.6 * per, 0 * per, -0.1 * per);
```

```
Gl.glColor3d(1.0, 0.0, 0.0);
// внутренняя
Gl.glNormal3f(1.0f, 0.0f, 0.0f);
```

```
Gl.glVertex3d(-0.2 * per, 0 * per, 0.1 * per);
Gl.glVertex3d(-0.2 * per, 0 * per, -0.1 * per);
Gl.glVertex3d(-0.2 * per, 0.8 * per, -0.1 * per);
Gl.glVertex3d(-0.2 * per, 0.8 * per, 0.1 * per);
```

```
Gl.glVertex3d(0.2 * per, 0 * per, 0.1 * per);
Gl.glVertex3d(0.2 * per, 0 * per, -0.1 * per);
Gl.glVertex3d(0.2 * per, 0.8 * per, -0.1 * per);
Gl.glVertex3d(0.2 * per, 0.8 * per, 0.1 * per);
```

```
Gl.glNormal3f(0.0f, -1.0f, 0.0f);
Gl.glColor3d(1.0, 0.0, 0.0);
```

```
Gl.glVertex3d(-0.2 * per, 0 * per, 0.1 * per);
Gl.glVertex3d(0.2 * per, 0 * per, 0.1 * per);
Gl.glVertex3d(0.2 * per, 0 * per, -0.1 * per);
Gl.glVertex3d(-0.2 * per, 0 * per, -0.1 * per);
```

```
Gl.glNormal3f(0.0f, 1.0f, 0.0f);
Gl.glVertex3d(-0.2 * per, 0.8 * per, 0.1 * per);
Gl.glVertex3d(0.2 * per, 0.8 * per, 0.1 * per);
Gl.glVertex3d(0.2 * per, 0.8 * per, -0.1 * per);
Gl.glVertex3d(-0.2 * per, 0.8 * per, -0.1 * per);
```

```
Gl.glEnd();
```

```
Gl.glFrontFace(Gl.GL_CW);
Gl.glBegin(Gl.GL_QUADS);
// задняя
```

```

    Gl.glColor3d(1.0, 0.0, 0.0);
    Gl.glNormal3f(0.0f, 0.0f, -1.0f);

    Gl.glVertex3d(-0.4 * per, 0 * per, -0.1 * per);
    Gl.glVertex3d(-0.2 * per, 0 * per, -0.1 * per);
    Gl.glVertex3d(-0.2 * per, 1 * per, -0.1 * per);
    Gl.glVertex3d(-0.4 * per, 1 * per, -0.1 * per);

    Gl.glVertex3d(-0.2 * per, 0.8 * per, -0.1 * per);
    Gl.glVertex3d(0.2 * per, 0.8 * per, -0.1 * per);
    Gl.glVertex3d(0.2 * per, 1 * per, -0.1 * per);
    Gl.glVertex3d(-0.2 * per, 1 * per, -0.1 * per);

    Gl.glVertex3d(0.2 * per, 0 * per, -0.1 * per);
    Gl.glVertex3d(0.4 * per, 0 * per, -0.1 * per);
    Gl.glVertex3d(0.4 * per, 1 * per, -0.1 * per);
    Gl.glVertex3d(0.2 * per, 1 * per, -0.1 * per);

    Gl.glVertex3d(-0.6 * per, -0.2 * per, -0.1 * per);
    Gl.glVertex3d(0.6 * per, -0.2 * per, -0.1 * per);
    Gl.glVertex3d(0.6 * per, 0 * per, -0.1 * per);
    Gl.glVertex3d(-0.6 * per, 0 * per, -0.1 * per);

    Gl.glVertex3d(-0.6 * per, -0.5 * per, -0.1 * per);
    Gl.glVertex3d(-0.4 * per, -0.5 * per, -0.1 * per);
    Gl.glVertex3d(-0.4 * per, -0.2 * per, -0.1 * per);
    Gl.glVertex3d(-0.6 * per, -0.2 * per, -0.1 * per);

    Gl.glVertex3d(0.4 * per, -0.5 * per, -0.1 * per);
    Gl.glVertex3d(0.6 * per, -0.5 * per, -0.1 * per);
    Gl.glVertex3d(0.6 * per, -0.2 * per, -0.1 * per);
    Gl.glVertex3d(0.4 * per, -0.2 * per, -0.1 * per);

    Gl.glEnd();

    Gl.glEnd();

    // Возвращаем обход сторон к значению по умолчанию
    Gl.glFrontFace(Gl.GL_CCW);

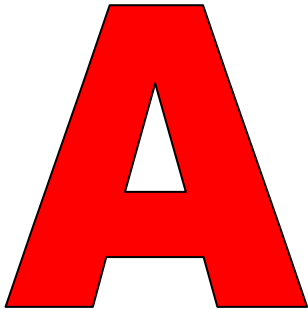


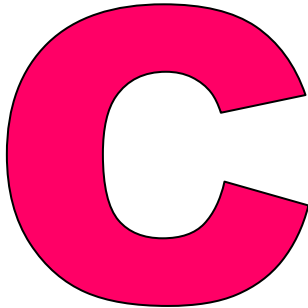
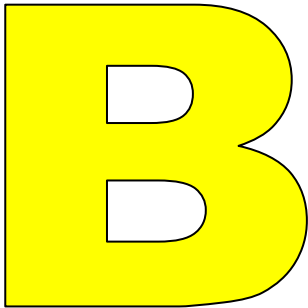
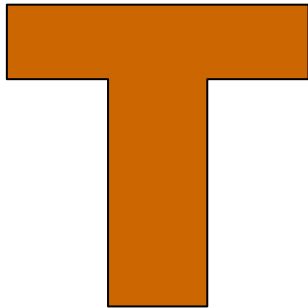
    // Переключаем буфер кадра
    Glut.glutSwapBuffers();
}

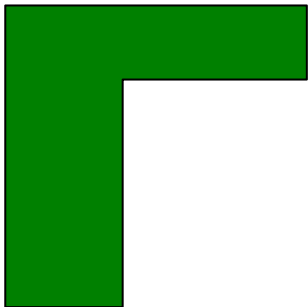
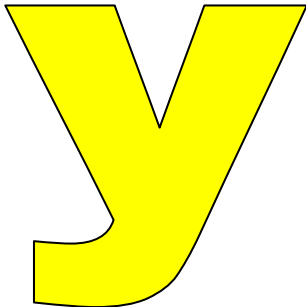
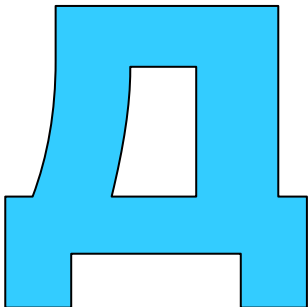
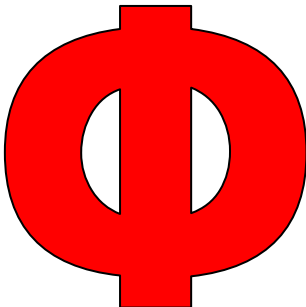
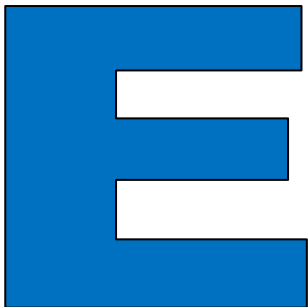
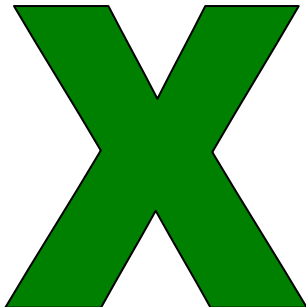
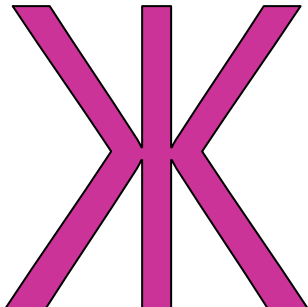
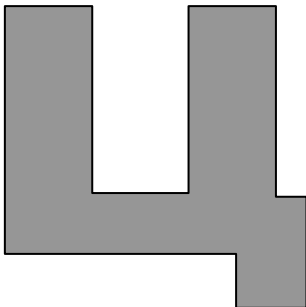
```


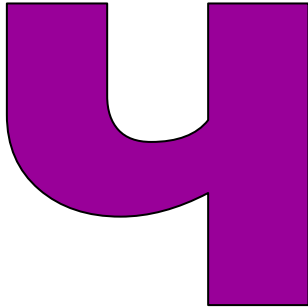
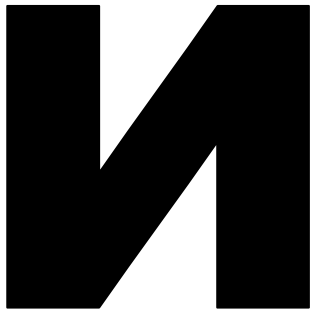
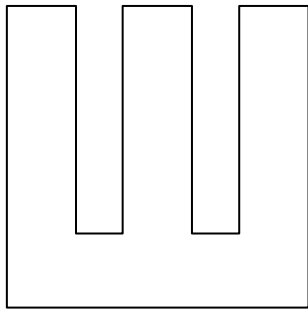
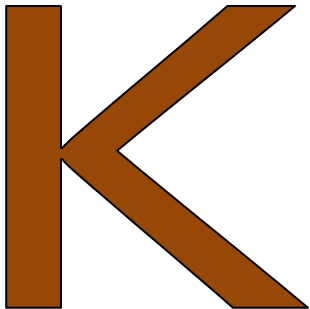
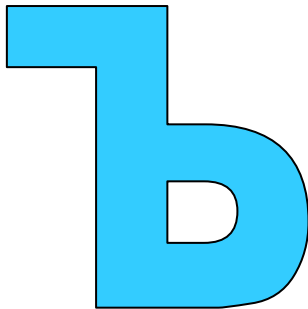
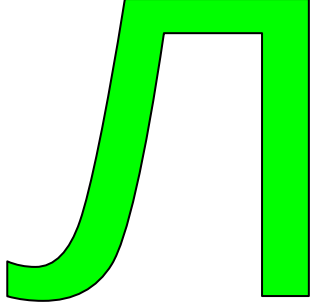
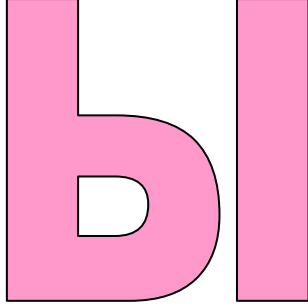
## Ход работы

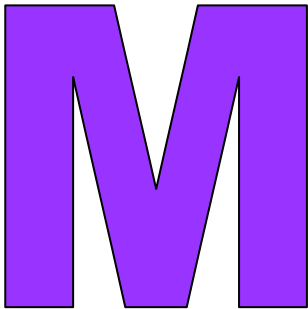
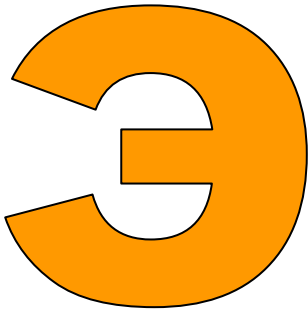
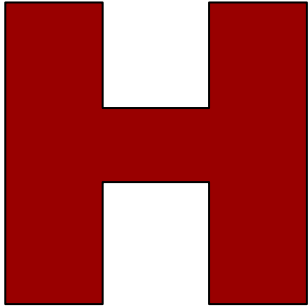
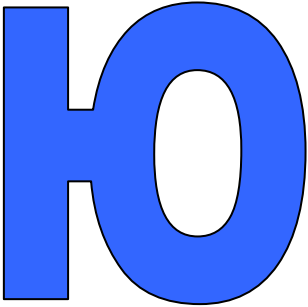
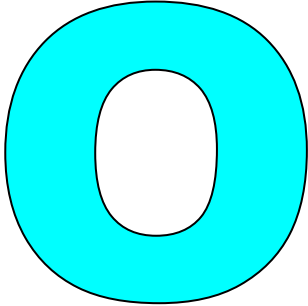
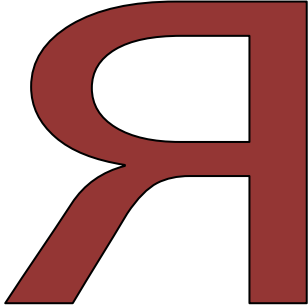
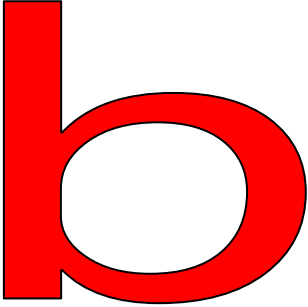
В рамках данной лабораторной работы необходимо, используя C++ и OpenGL, разработать программу построения трехмерного объектов (буква в виде многогранника) в соответствии со своим вариантом задания. Грани объекта должны быть окрашены в определенный цвет. Ребра – черного цвета. Отобразить фигуру под углом, чтобы были видны грани и можно понять, что это 3D объект.

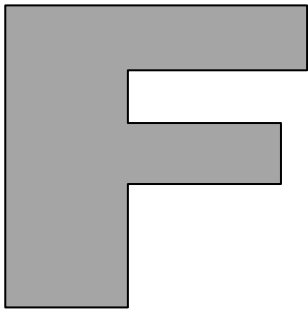
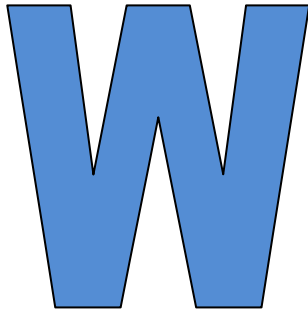
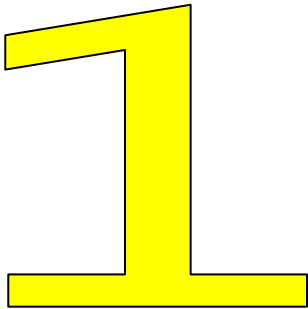


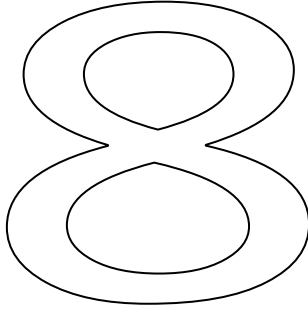
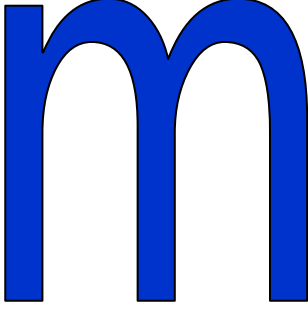
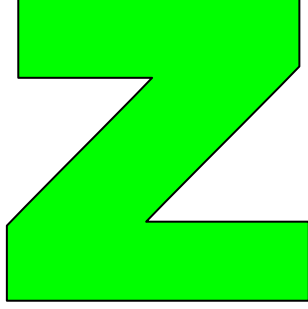
Варианты заданий:

Номер вариан- та	Задание	Номер вариан- та	Задание
1		20	
2		21	
3		22	

Номер вариан- та	Задание	Номер вариан- та	Задание
4		23	
5		24	
6		25	
7		26	

Номер вариан- та	Задание	Номер вариан- та	Задание
8		27	
9		28	
10		29	
11		30	

Номер вариан- та	Задание	Номер вариан- та	Задание
12		31	
13		32	
14		33	
15		34	

Номер варианта	Задание	Номер варианта	Задание
16		35	
17		36	
18		37	
19		38	

### Контрольные вопросы

1. В каком положении находится камера в начале работы с OpenGL?
2. Перечислите матрицы, последовательно применяющиеся в преобразовании координат OpenGL.
3. Какая процедура служит для задания режима закрашивания в OpenGL?