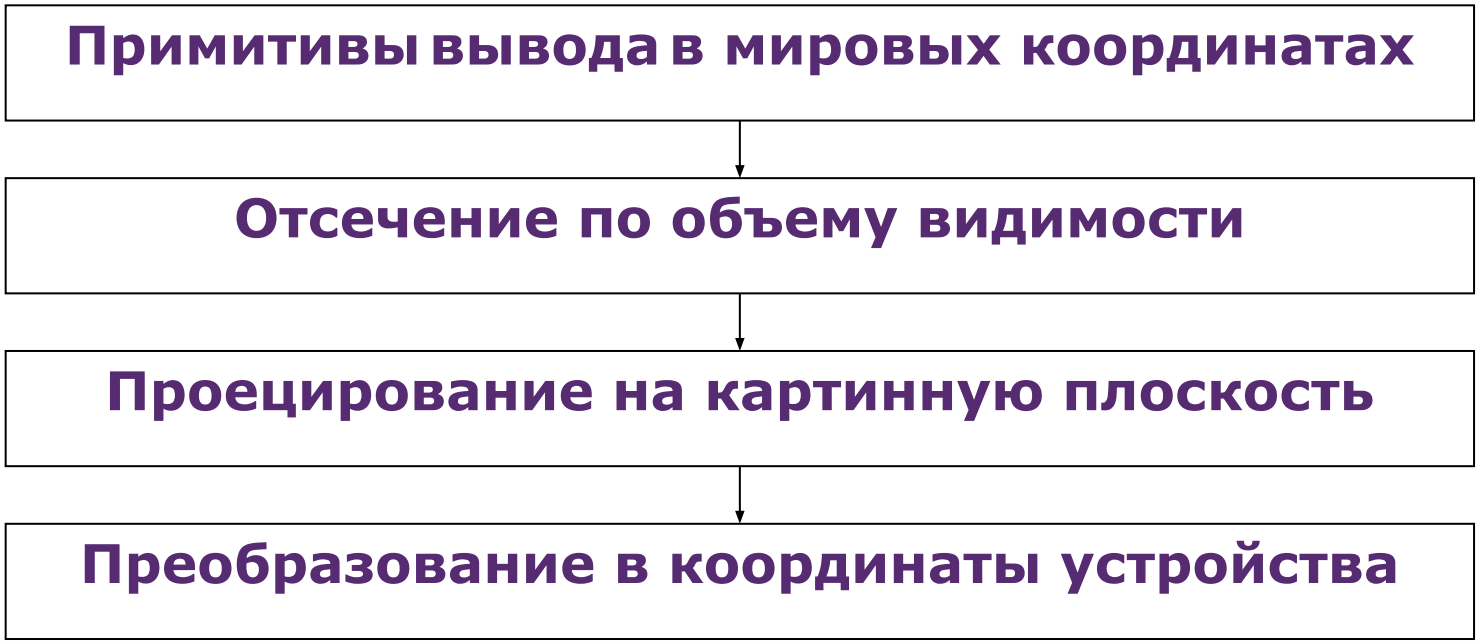
Моделирование 3D сцен с применением OpenGL

Моделирование 3D сцен с применением графической библиотеки OpenGl

1. **Виды проектирования.**
2. **Алгоритмы удаления невидимых линий и поверхностей.**
3. **Преобразования координат, проектирование и алгоритм Z-буфера в OpenGL.**
4. **Визуализация 3D объектов и подключение библиотеки Glut.**

МЕТОДЫ И АЛГОРИТМЫ ТРЕХМЕРНОЙ ГРАФИКИ

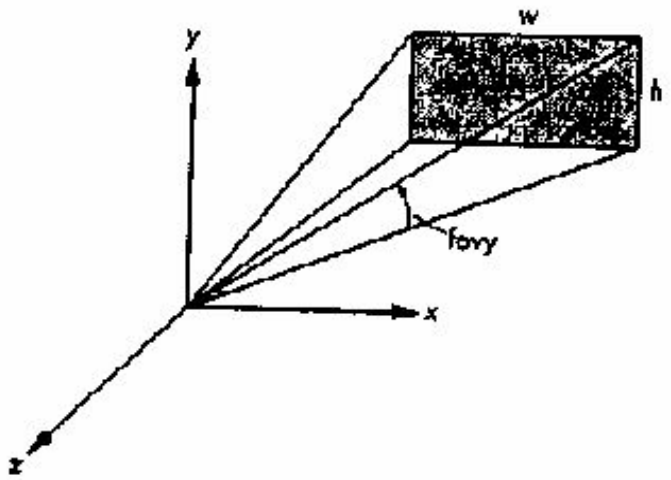


Проекции в OpenGL

glOrtho(left, right, bottom, top, near, far: GLdouble)

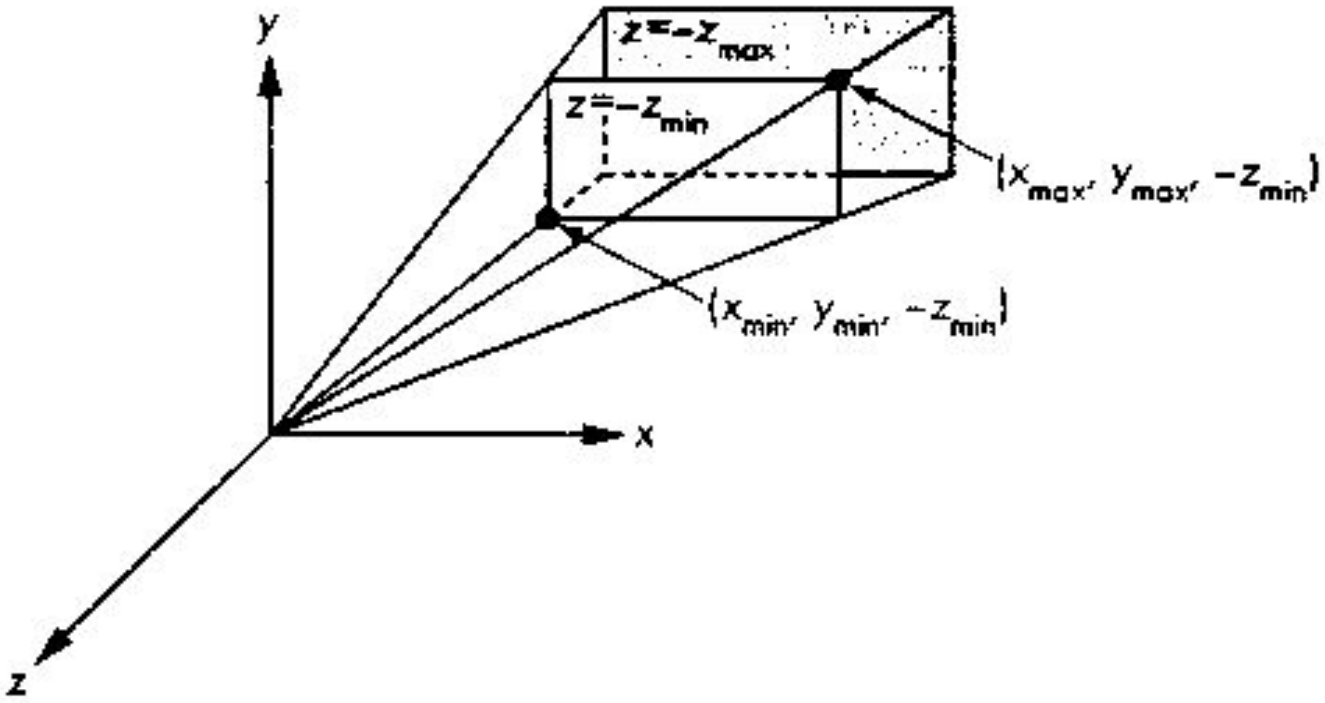
gluOrtho2D(left, right, bottom, top: GLdouble)

gluPerspective(fovy,aspect, near, far)



Перспективные преобразования в OpenGL

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(xmin, xmax, ymin, ymax, near, far)
```

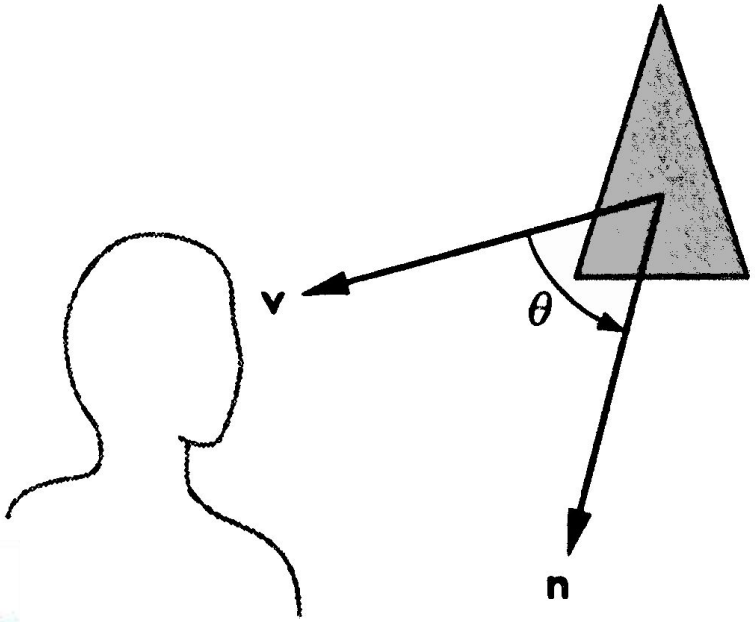


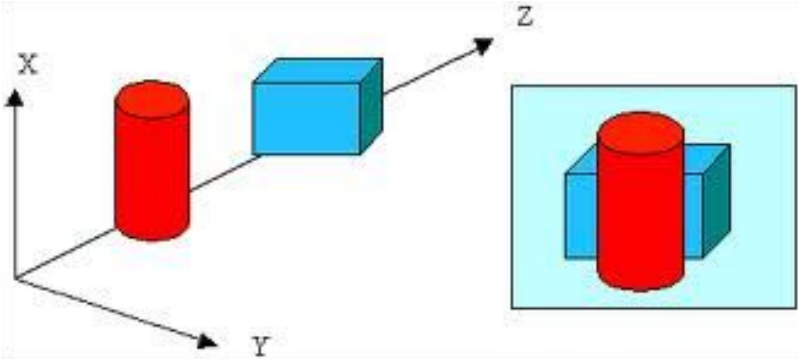
glFrustum (*Xmin, Xmax, Ymin, Ymax, near, far*)

УДАЛЕНИЕ НЕВИДИМЫХ ЛИНИЙ И ПОВЕРХНОСТЕЙ

$$-90^{\circ} \le \theta \le 90^{\circ}.$$

$$\cos\theta \ge 0.$$





Рисуем единичный куб

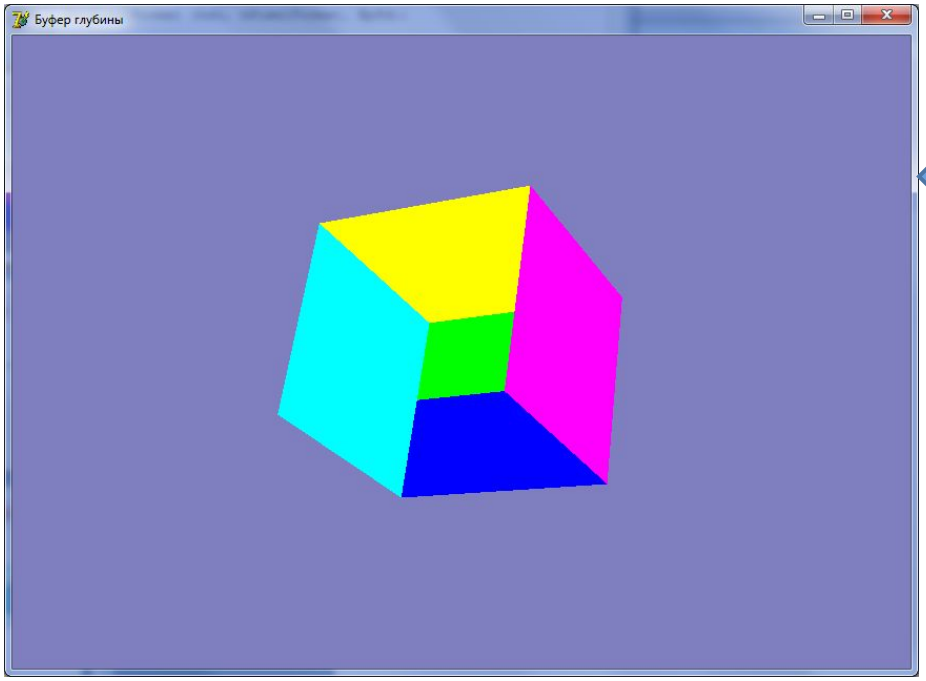
```
procedure TForm1.FormResize(Sender: TObject);
begin
glViewport(0, 0, ClientWidth, ClientHeight);
 glMatrixMode (GL_PROJECTION);
 glLoadIdentity;
 glFrustum (-1, 1, -1, 1, 3,10);
 glMatrixMode (GL_MODELVIEW);
 glLoadIdentity;
 glTranslatef(0.0, 0.0, -8.0);
 glScaled(0.7,0.9,0.7);
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
begin
 DC := GetDC(Handle);
  SetDCPixelFormat(DC);
  hrc := wglCreateContext(DC);
  wglMakeCurrent(DC, hrc);
  glEnable(GL_DEPTH_TEST);
   a:=0;
 Form1.Timer1.Enabled:=true;
end;
```

Рисуем единичный куб

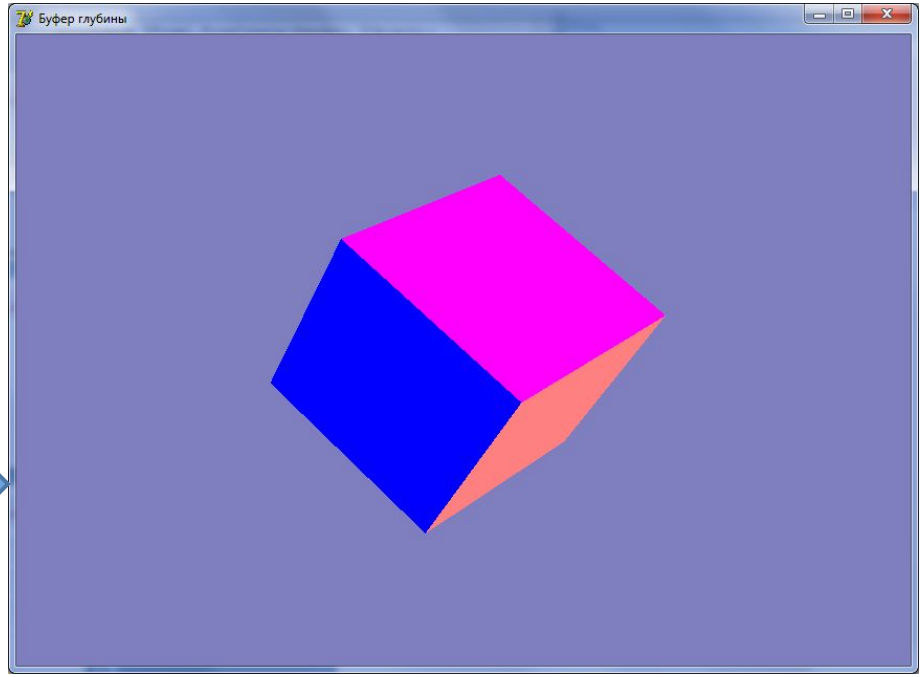
```
procedure TForm1.FormPaint(Sender: TObject);
begin
 wglMakeCurrent(Canvas.Handle, hrc);
 glViewPort (0, 0, ClientWidth, ClientHeight);
 glClearColor (0.5, 0.5, 0.75, 1.0);
 glClear (GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
 glRotated(a,1,1,0);
 glColor3f (1.0, 0.5, 0.5);
 glBegin(GL_QUADS);
    glVertex3f(1.0, 1.0, 1.0);
    glVertex3f(-1.0, 1.0, 1.0);
    glVertex3f(-1.0, -1.0, 1.0);
    glVertex3f(1.0, -1.0, 1.0);
  glEnd;
  glColor3f (0.0, 1.0, 0.0);
  glBegin(GL_QUADS);
    glVertex3f(1.0, 1.0, -1.0);
    glVertex3f(1.0, -1.0, -1.0);
    glVertex3f(-1.0, -1.0, -1.0);
    glVertex3f(-1.0, 1.0, -1.0);
  glEnd;
  glColor3f (0.0, 0.0, 1.0);
  glBegin(GL_QUADS);
    glVertex3f(-1.0, 1.0, 1.0);
    glVertex3f(-1.0, 1.0, -1.0);
    glVertex3f(-1.0, -1.0, -1.0);
    glVertex3f(-1.0, -1.0, 1.0);
  glEnd;
  glColor3f (1.0, 1.0, 0.0);
  glBegin(GL_QUADS);
    glVertex3f(1.0, 1.0, 1.0);
    glVertex3f(1.0, -1.0, 1.0);
    glVertex3f(1.0, -1.0, -1.0);
    glVertex3f(1.0, 1.0, -1.0);
  glEnd;
  glColor3f (1.0, 0.0, 1.0);
  glBegin(GL_QUADS);
    glVertex3f(-1.0, 1.0, -1.0);
    glVertex3f(-1.0, 1.0, 1.0);
    glVertex3f(1.0, 1.0, 1.0);
    glVertex3f(1.0, 1.0, -1.0);
  glEnd;
  glColor3f (0.0, 1.0, 1.0);
 glBegin(GL_QUADS);
    glVertex3f(-1.0, -1.0, -1.0);
    glVertex3f(1.0, -1.0, -1.0);
    glVertex3f(1.0, -1.0, 1.0);
    glVertex3f(-1.0, -1.0, 1.0);
  glEnd;
 SwapBuffers(DC);
 wglMakeCurrent(0, 0);
end;
```

Алгоритм Z-буфера



Без применения буфера глубины

С применением буфера глубины



Методы отображения

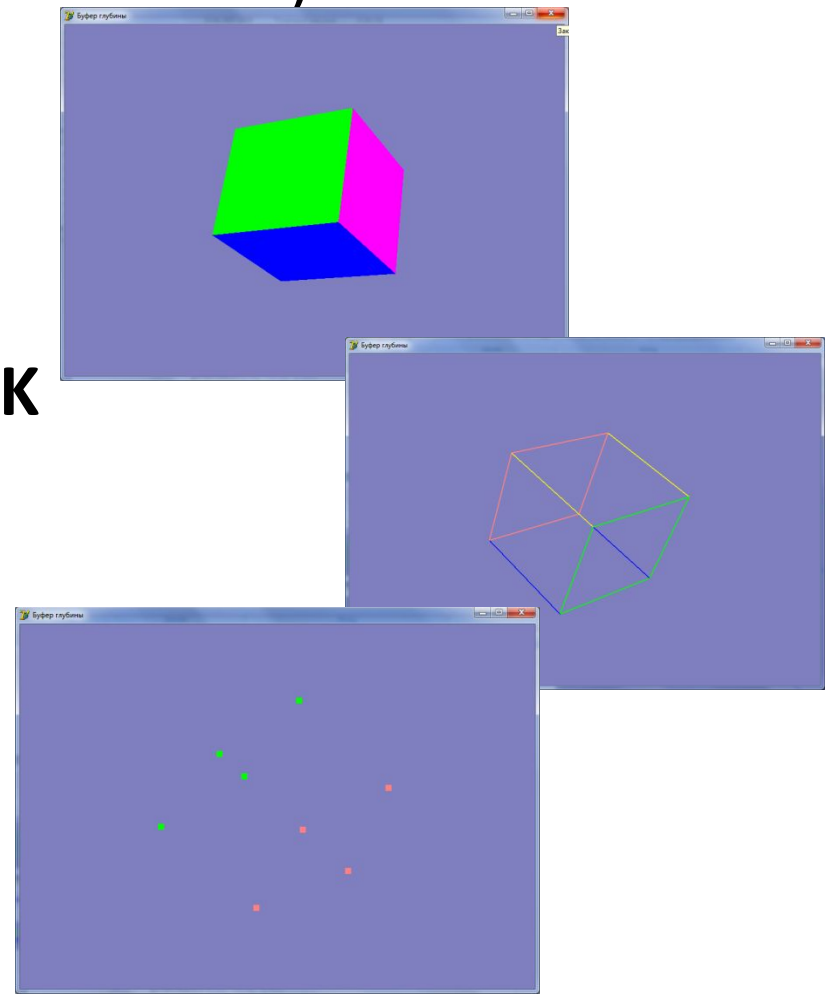
glPolygonMode(face, mode :GLenum)

face:

- **GL_FRONT**
- **GL_BACK**
- **GL_FRONT_AND_BACK**

Mode:

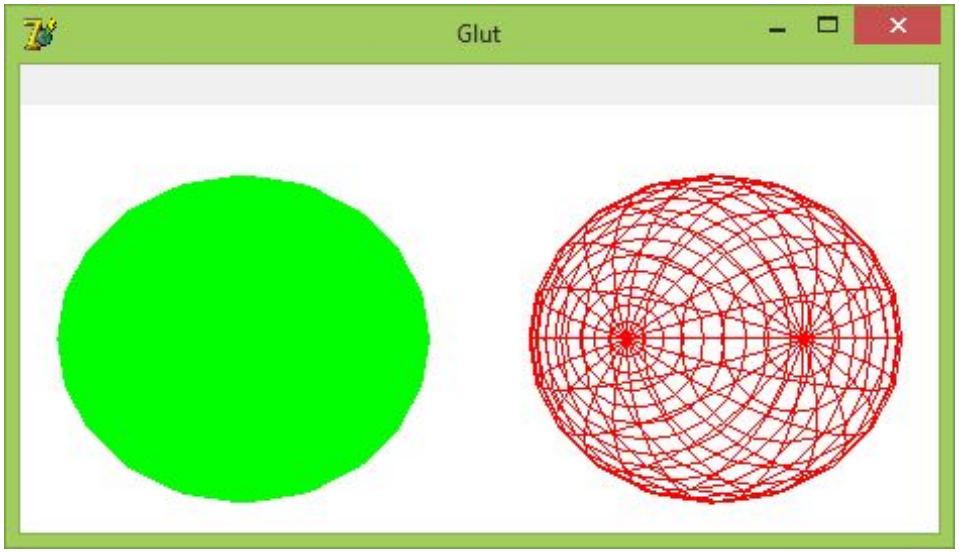
- **GL_POINT**
- **GL_LINE**
- **GL_FILL**



Сфера Glut

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, ExtCtrls, StdCtrls, OpenGl, GLUT;
```

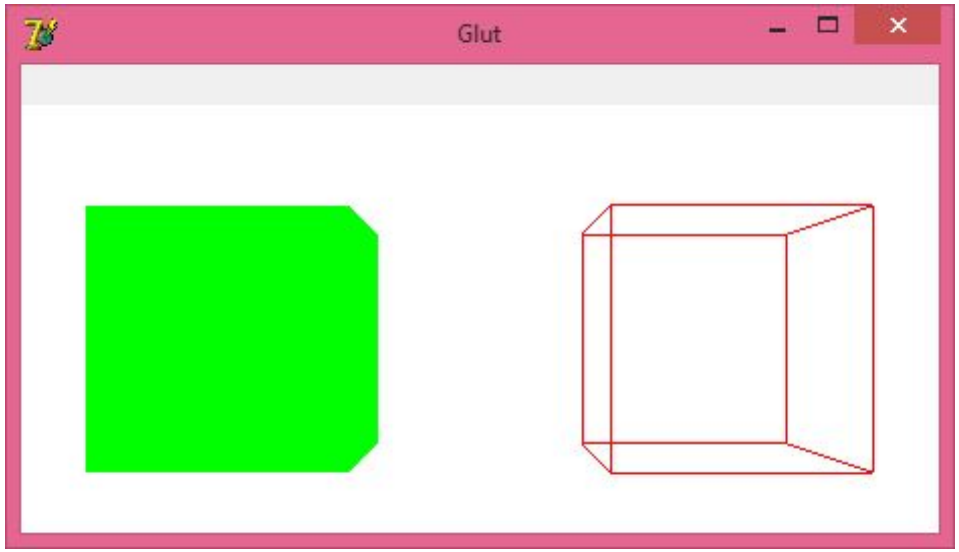
- **glutSolidSphere**(radius, slices, stacks)
- **glutWireSphere**(radius, slices, stacks)




```
procedure TForm1.FormPaint(Sender: TObject);
begin
wglMakeCurrent(Canvas.Handle, hrc);
glClearColor (1, 1, 1, 1.0);
glClear (GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
glPushMatrix;
glTranslated(1,0,0);
glColor3d(1,0,0);
glutWireSphere(1,20,20);
glPopMatrix;

glPushMatrix;
glTranslated(-1,0,0);
glColor3d(0,1,0);
glutSolidSphere(1,20,20);
glPopMatrix;
SwapBuffers(dc);
wglMakeCurrent(0, 0);
end;
```

Куб Glut

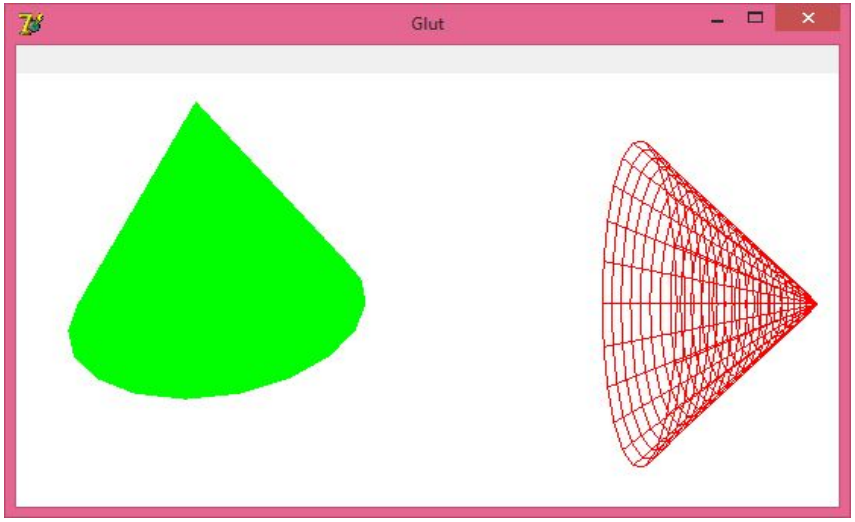
- glutSolidCube(size)
- glutWireCube(size)




```
procedure TForm1.FormPaint(Sender: TObject);
begin
wglMakeCurrent(Canvas.Handle, hrc);
glClearColor (1, 1, 1, 1.0);
glClear (GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);

glPushMatrix;
glTranslated(1,0,0);
glColor3d(1,0,0);
glutWireCube(1);
glPopMatrix;

glPushMatrix;
glTranslated(-1,0,0);
glColor3d(0,1,0);
glutSolidCube(1);
glPopMatrix;
SwapBuffers(dc);
wglMakeCurrent(0, 0);
end;
```

Конус Glut

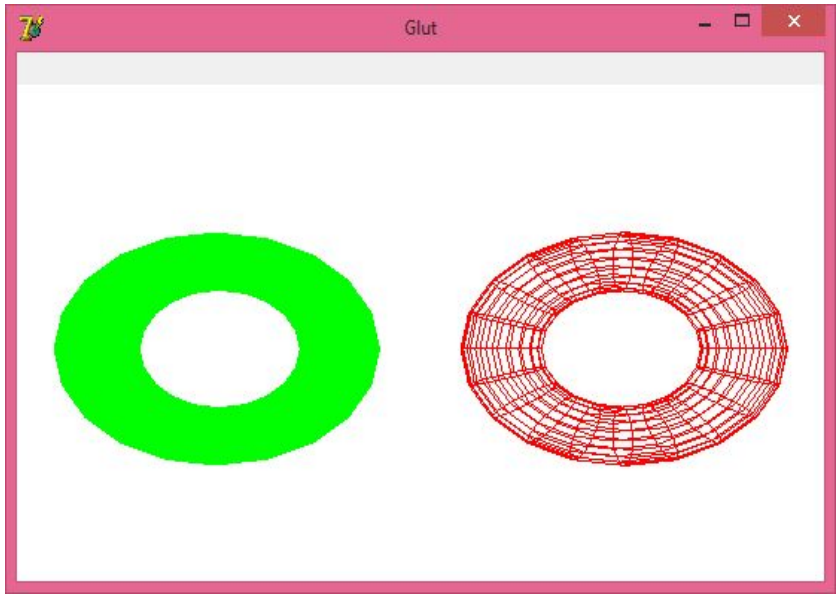
- **glutSolidCone(base, height, slices, stacks)**
- **glutWireCone(base**, height, slices, stacks)




```
procedure TForm1.FormPaint(Sender: TObject);
begin
wglMakeCurrent(Canvas.Handle, hrc);
glClearColor (1, 1, 1, 1.0);
glClear (GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);

glPushMatrix;
glTranslated(1,0,0);
glColor3d(1,0,0);
glRotated(90,0,1,0);
glutWireCone(0.7, 0.9, 20, 20);
glPopMatrix;

glPushMatrix;
glTranslated(-1,0,0);
glColor3d(0,1,0);
glRotated(-60,1,0,0);
glutSolidCone(0.7, 0.9, 20, 20);
glPopMatrix;
SwapBuffers(dc);
wglMakeCurrent(0, 0);
end;
```

Top Glut

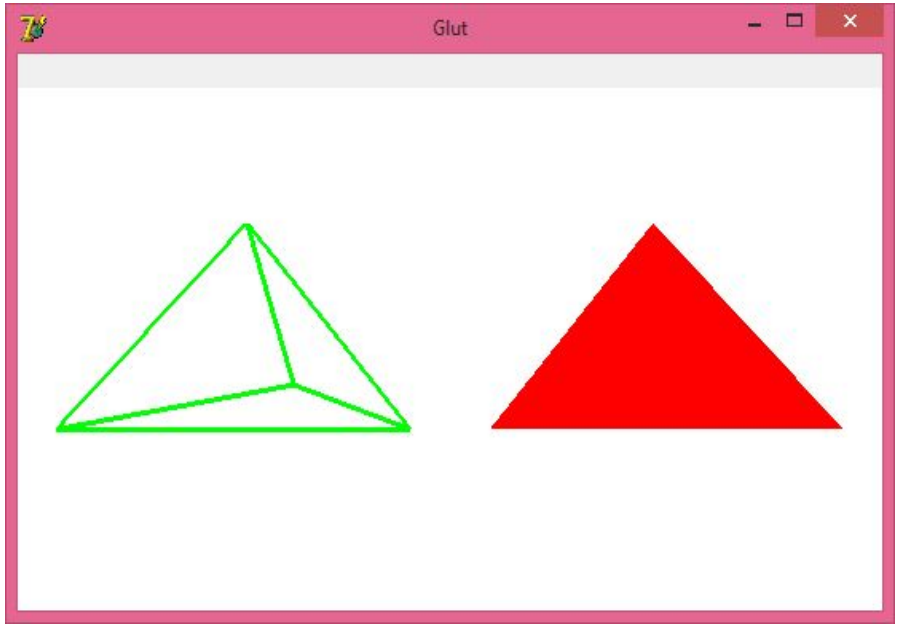
- glutSolidTorus(innerRadius, outerRadius, nsides, rings)
- glutWireTorus(innerRadius, outerRadius, nsides, rings)




```
procedure TForm1.FormPaint(Sender: TObject);
begin
wglMakeCurrent(Canvas.Handle, hrc);
glClearColor (1, 1, 1, 1.0);
glClear (GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);

glPushMatrix;
glTranslated(1,0,0);
glColor3d(1,0,0);
glutWireTorus(0.2, 0.6, 20, 20);
glPopMatrix;

glPushMatrix;
glTranslated(-1,0,0);
glColor3d(0,1,0);
glutSolidTorus(0.2, 0.6, 20, 20);
glPopMatrix;
SwapBuffers(dc);
wglMakeCurrent(0, 0);
end;
```

Тетраэдр Glut

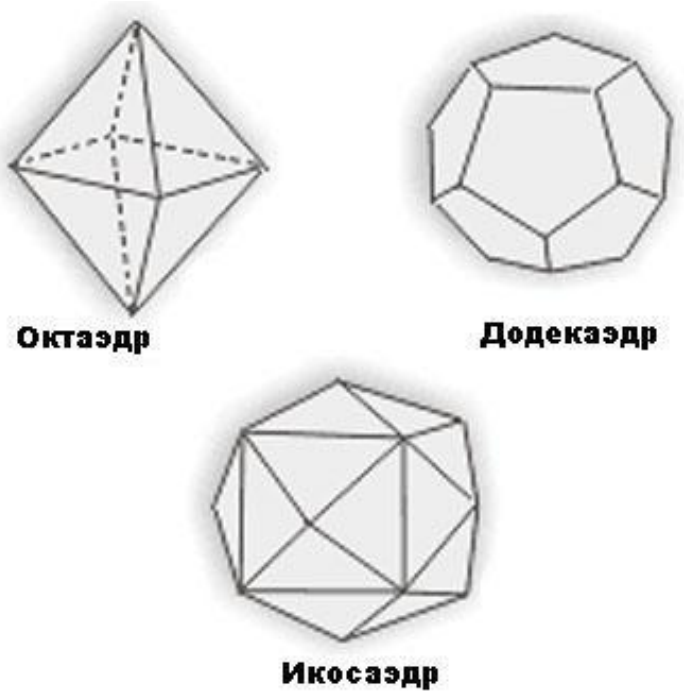
- glutSolidTetrahedron
- glutWireTetrahedron




```
procedure TForm1.FormPaint(Sender: TObject);
begin
wglMakeCurrent(Canvas.Handle, hrc);
glClearColor (1, 1, 1, 1.0);
glClear (GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);

glPushMatrix;
glTranslated(1,0,0);
glColor3d(1,0,0);
glRotated(90,0,1,1);
glutSolidTetrahedron;
glPopMatrix;

glPushMatrix;
glTranslated(-1,0,0);
glColor3d(0,1,0);
glLineWidth(3);
glRotated(90,0,1,1);
glutWireTetrahedron;
glPopMatrix;
SwapBuffers(dc);
wglMakeCurrent(0, 0);
end;
```

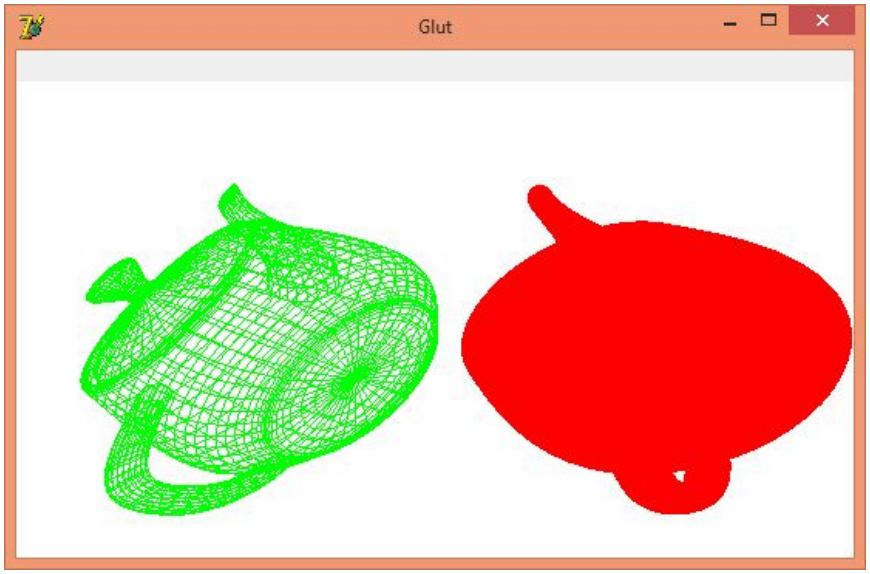
Примитивы библиотеки GLUT

- glutSolidOctahedron
- glutWireOctahedron
- glutSolidDodecahedron
- glutWireDodecahedron
- glutSolidIcosahedron
- glutWireIcosahedron



Чайник Glut

- glutSolidTeapot(size)
- glutWireTeapot (size)




```
procedure TForm1.FormPaint(Sender: TObject);
begin
wglMakeCurrent(Canvas.Handle, hrc);
glClearColor (1, 1, 1, 1.0);
glClear (GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);

glPushMatrix;
glTranslated(1,0,0);
glColor3d(1,0,0);
glRotated(90,0,1,1);
glutSolidTeapot(1);
glPopMatrix;

glPushMatrix;
glTranslated(-1,0,0);
glColor3d(0,1,0);
glRotated(90,0,1,1);
glutWireTeapot(1);
glPopMatrix;
SwapBuffers(dc);
wglMakeCurrent(0, 0);
end;
```

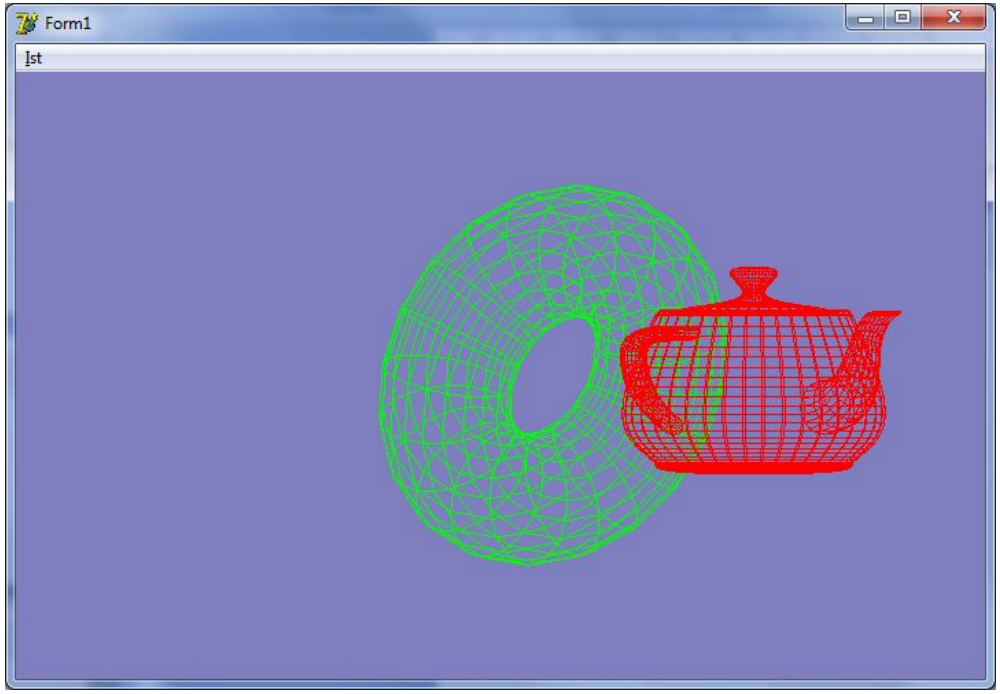
Примитивы библиотеки GLUT

Unit1

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, ExtCtrls, StdCtrls, OpenGl, GLUT;
```

Unit1

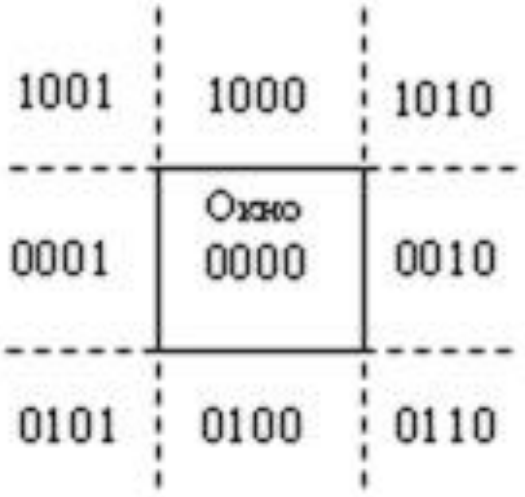
```
glPushMatrix;
glRotated (a,0,1,0);
glTranslated(6,0,0);
glRotated (b,0,1,0);
glColor3d(1,0,0);
glutWireTeapot(2);
glPopMatrix;

glPushMatrix;
glTranslated(0.8,0,0);
glRotated (a,1,1,1);
glColor3d(0,1,0);
glutWireTorus(1,2,20,20);
glPopMatrix;
```

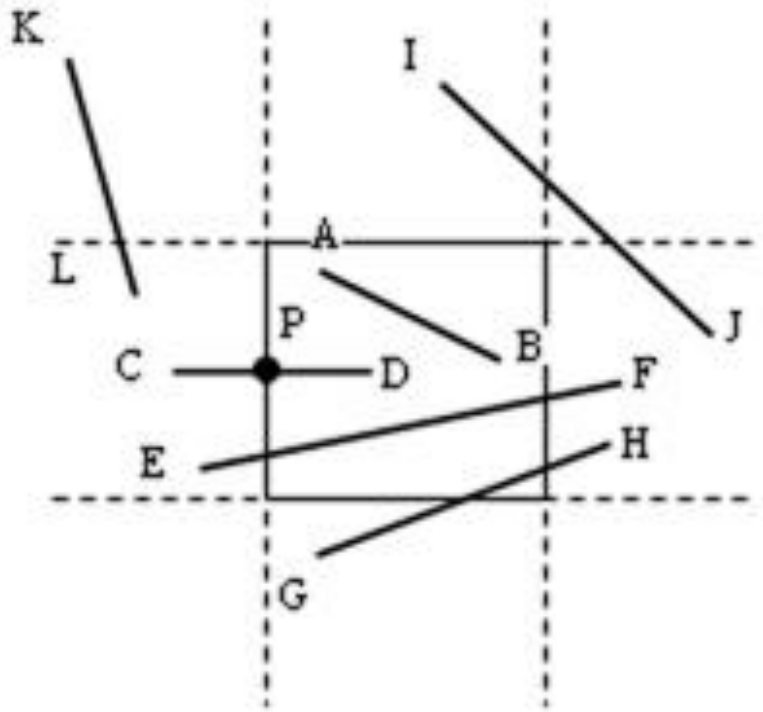


Отсечение

- **алгоритмы, использующие кодирование концов отрезка или всего отрезка (Коэна-Сазерленда);**
- **алгоритмы, использующие параметрическое представление отсекаемых отрезков и окна отсечения (Лианга-Барского).**

Алгоритм Коэна-Сазерленда

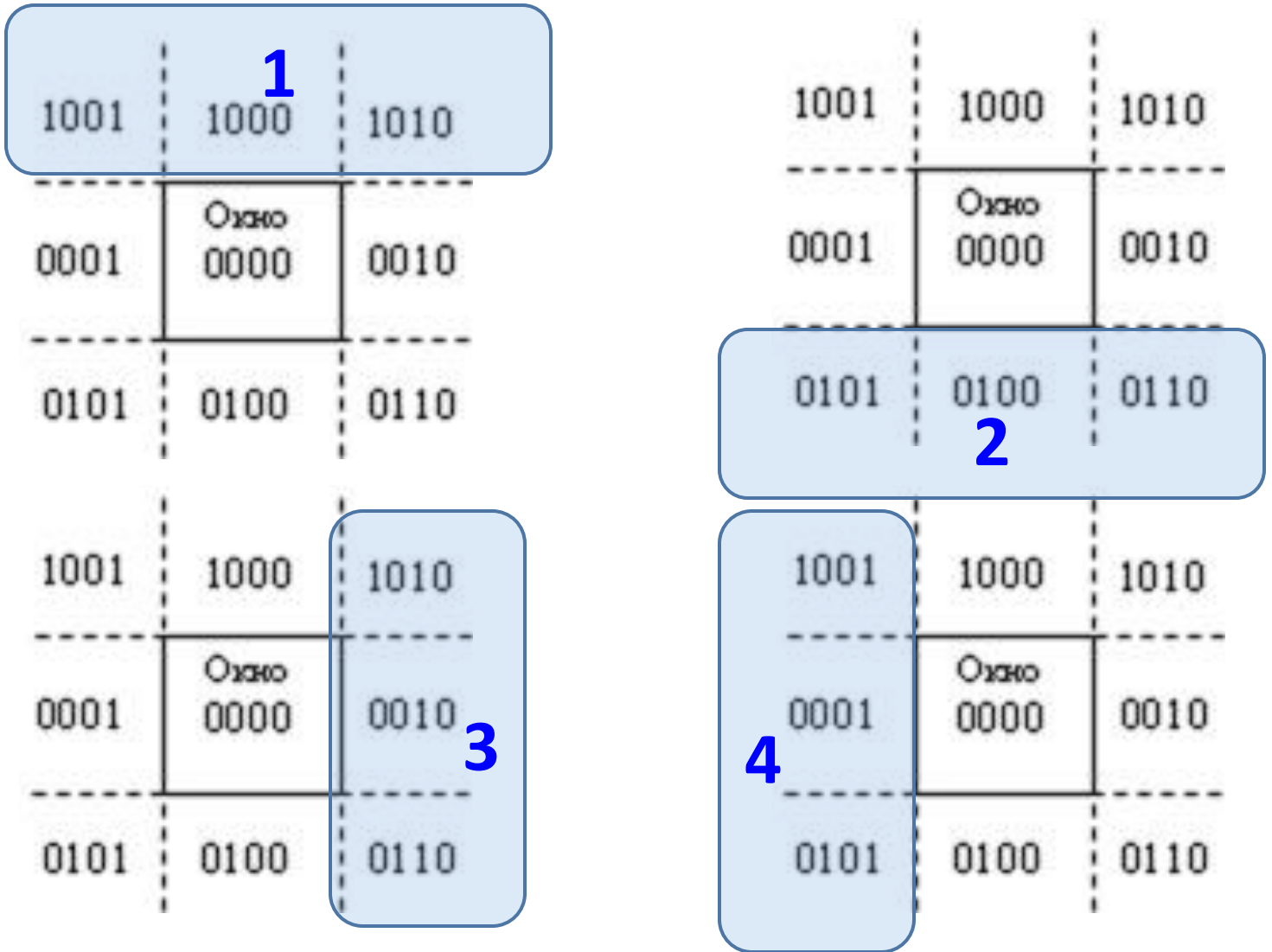




Алгоритм Коэна-Сазерленда

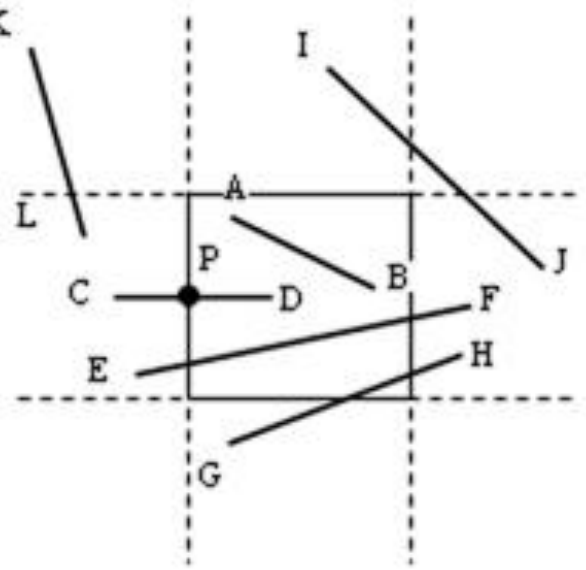
- **Две конечные точки отрезка получают 4-х разрядные коды, соответствующие областям, в которые они попали:**
- **1 рр = 1 - точка над верхним краем окна;**
- **2 рр = 1 - точка под нижним краем окна;**
- **3 рр = 1 - точка справа от правого края окна;**
- **4 рр = 1 - точка слева от левого края окна.**

Битовый код

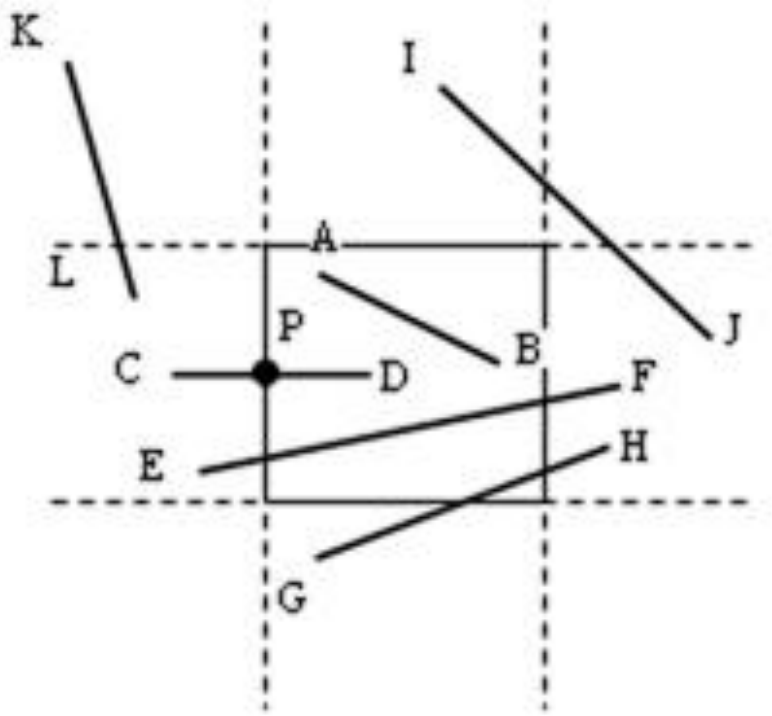


Битовый код

- A=0000;
- E=0001;
- I=1000;
- B=0000;
- F=0010;
- J=0010;
- C=0001;
- G=0101;
- K=1001
- D=0000;
- H=0010;
- L=0001.



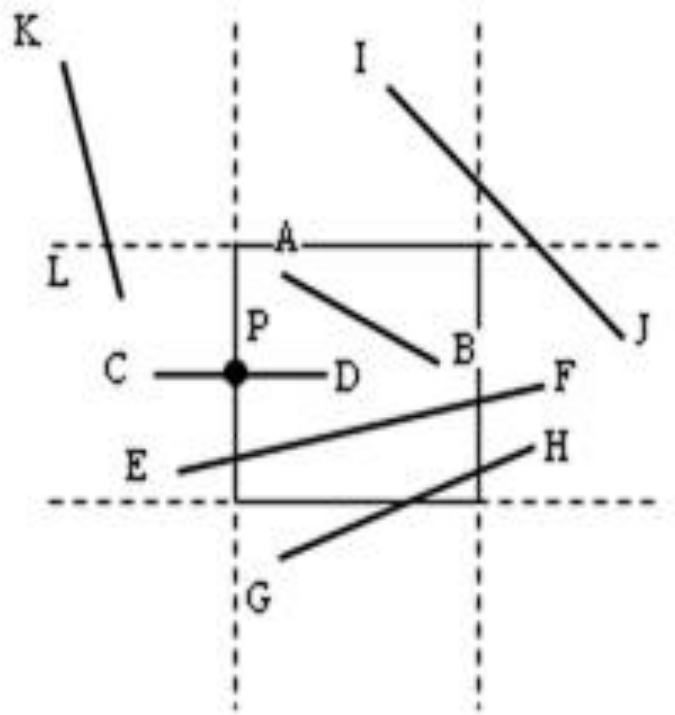
Алгоритм Коэна-Сазерленда



Пусть X - код точки-начала отрезка, Y - код точки-конца отрезка, тогда возможны три случая:

1. X = Y = 0000. Этот случай означает, что обе точки лежат внутри прямоугольника (т. е. отсечение не требуется).
2. X and Y ≠ 0. В этом случае точки лежат по одну сторону от какой-либо отсекающей линии (с внешней ее стороны). Следовательно, отрезок полностью лежит вне окна.
3. Если не выполнены условия 1 или 2, то необходимо находить точки пересечения с некоторыми из отсекающих прямых. Для этого разбивают отрезок найденными точками пересечения и затем применяют тот же анализ кодов концов для полученных новых отрезков.

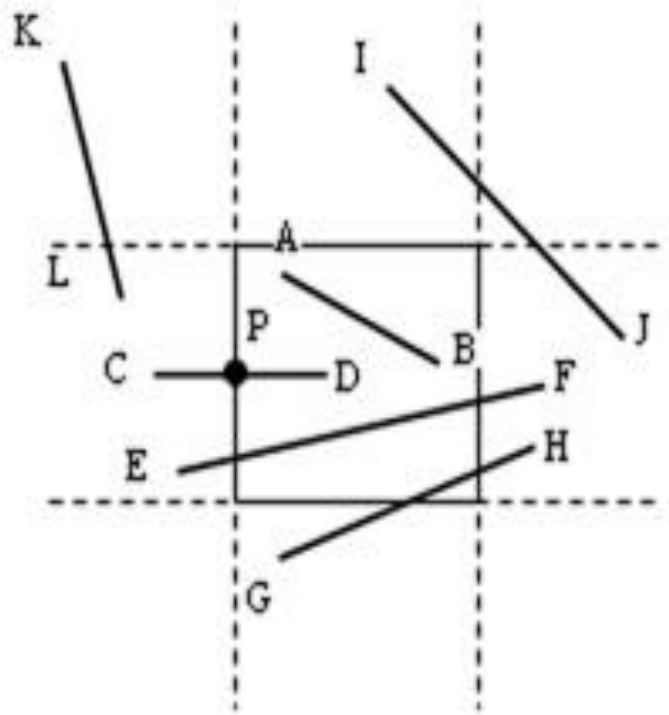
Алгоритм Коэна-Сазерленда



- X = Y=0:
- Отрезок AB

(0000)and(0000)

Алгоритм Коэна-Сазерленда



- X and Y≠0:
- Отрезок KL

(1001)and(0001)

Алгоритм Коэна-Сазерленда

Отрезки	Коды концов	Результаты логического умножения	Примечание
AB	0000 0000	0000	Целиком видим
CD	0001 0000	0000	
EF	0001 0010	0000	
GH	0101 0010	0000	
IJ	1000 0010	0000	
KL	1001 0001	0001	Целиком невидим

Задание

- Используя алгоритм Коэна-Сазерленда, выявить видимые, невидимые и отсекаемые отрезки AB, CD, EF, GH, IJ, KL.
- Если известны: A(0000), B(0010), C(0001), D(1001), E (0110), F(0010), G(0000), H(0000), I(1000), J(0010), K(1001), L(1010).
- Отобразите расположение отрезков относительно окна отсечения.

Алгоритм Коэна-Сазерленда

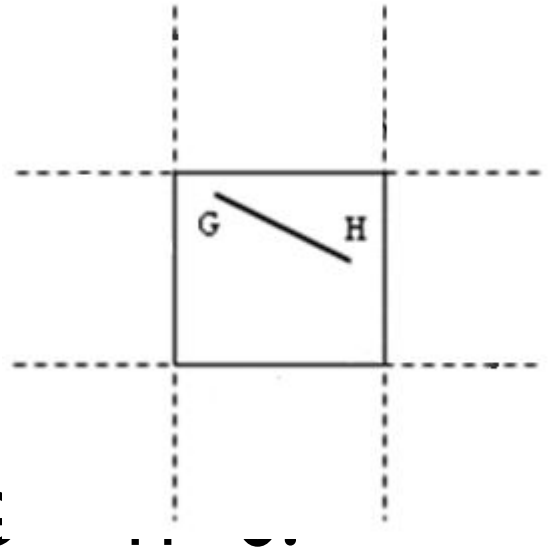
Отрезки	Коды концов
AB	0000 0010
CD	0001 1001
EF	0110 0010
GH	0000 0000
IJ	1000 0010
KL	1001 1010

Пусть X - код точки-начала отрезка, Y - код точки-конца отрезка, тогда возможны три случая:

1. X = Y = 0000. Этот случай означает, что обе точки лежат внутри прямоугольника (т. е. отсечение не требуется).
2. X and Y ≠ 0. В этом случае точки лежат по одну сторону от какой-либо отсекающей линии (с внешней ее стороны). Следовательно, отрезок полностью лежит вне окна.
3. Если не выполнены условия 1 или 2, то необходимо находить точки пересечения с некоторыми из отсекающих прямых. Для этого разбивают отрезок найденными точками пересечения и затем применяют тот же анализ кодов концов для полученных новых отрезков.

Алгоритм Коэна-Сазерленда

- Случай 1



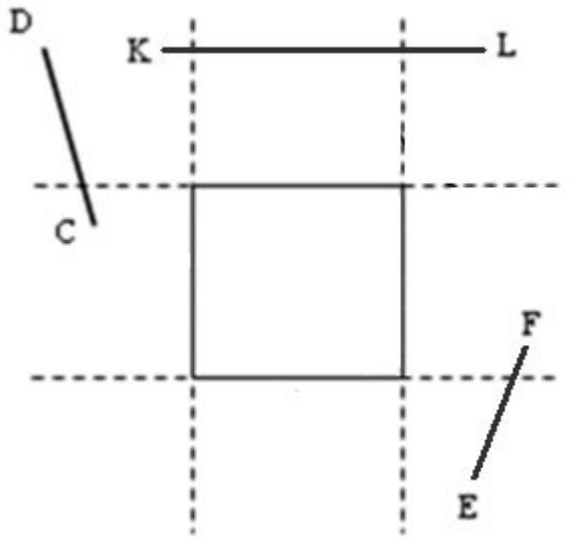
- G

- Отрезок GH

Отрезки	Коды концов
AB	0000 0010
CD	0001 1001
EF	0110 0010
GH	0000 0000
IJ	1000 0010
KL	1001 1010

Алгоритм Коэна-Сазерленда

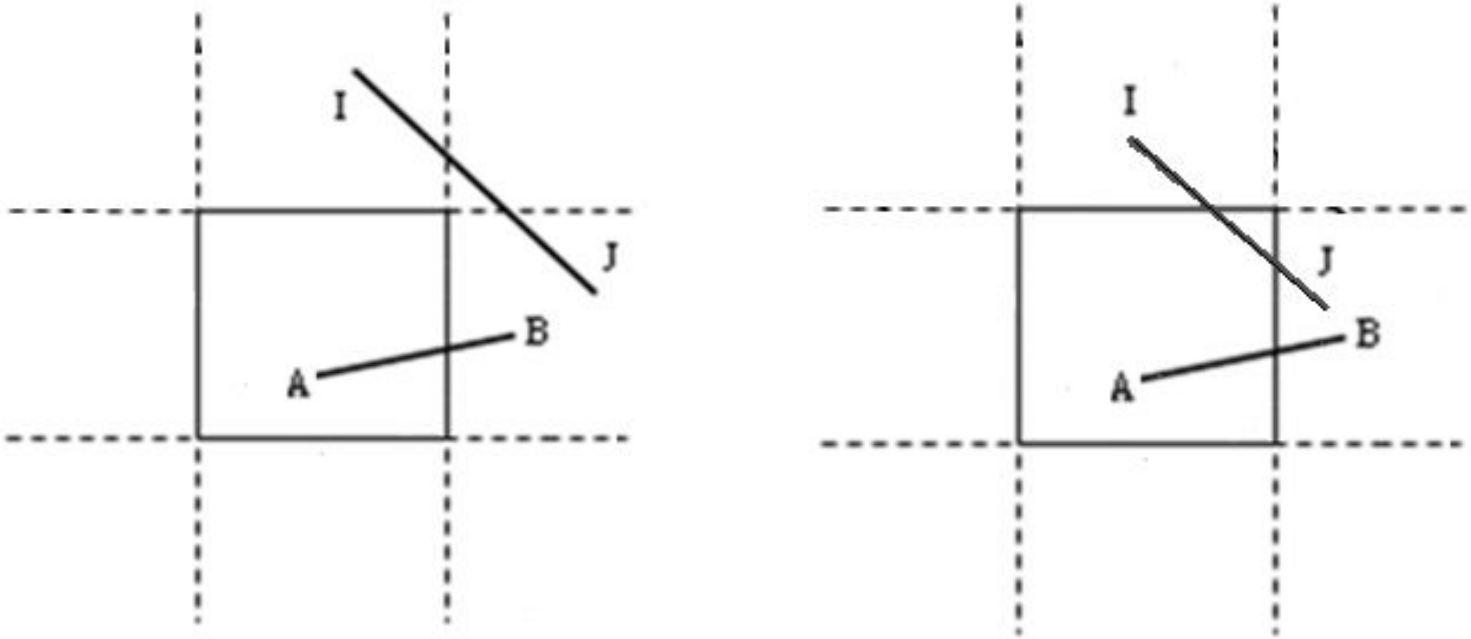
- Случай 2



- X and Y≠0:
- Отрезок KL
- Отрезок DC
- Отрезок EF

Алгоритм Коэна-Сазерленда

- Случай 3

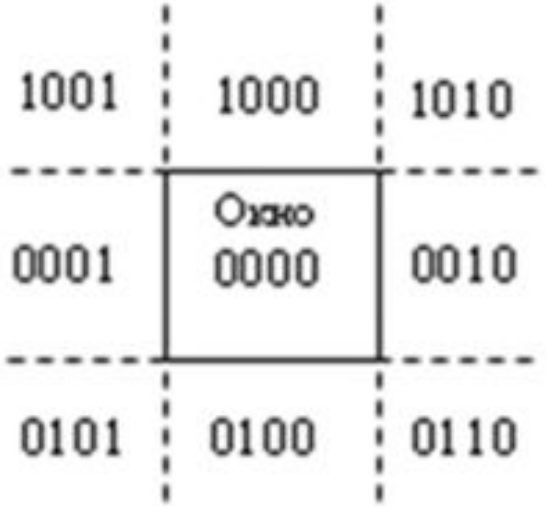


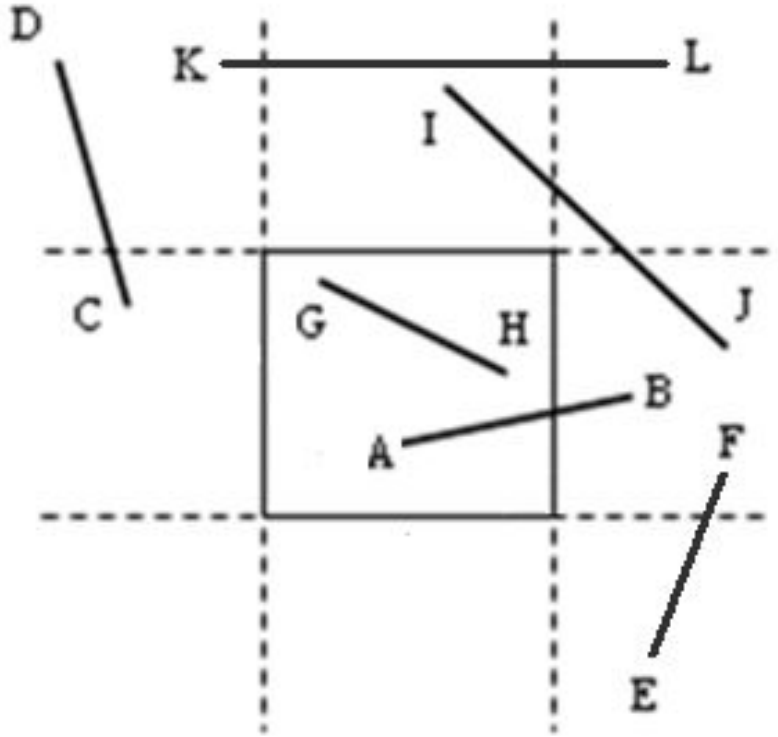
- Отсекаемые отрезки: Отрезок AB; Отрезок IJ.

Алгоритм Коэна-Сазерленда

Отрезки	Коды концов	Результаты логического умножения	Примечание
AB	0000 0010	0000	
CD	0001 1001	0001	Целиком невидим
EF	0110 0010	0010	Целиком невидим
GH	0000 0000	0000	Целиком видим
IJ	1000 0010	0000	
KL	1001 1010	1000	Целиком невидим

Алгоритм Коэна-Сазерленда





Визуальные эффекты OpenGL

Визуальные эффекты
OpenGL

- Материалы и освещение
  - Свойства материалов
  - Источники света
  - Модели освещения
- Туман
- Прозрачность
- Трафарет – *самостоятельно*
- Текстура – *самостоятельно*

Материалы и освещение

glEnable(GL_LIGHTING),
glEnable(GL_LIGHT*0*).

```
glBegin(GL_POLYGON);
   glNormal3f(0.0, 0.0, -1.0);
   glVertex3f(1.0, 1.0, -1.0);
   glVertex3f(1.0, -1.0, -1.0);
   glVertex3f(-1.0, -1.0, -1.0);
   glVertex3f(-1.0, 1.0, -1.0);
 glEnd;
```

Свойства материала

glMaterial[i f](face, pname, param)

Какой именно параметр будет определяться значением param, зависит от значения pname:

- **GL_AMBIENT ,** значение по умолчанию:

  (0.2, 0.2, 0.2, 1.0).

- **GL_DIFFUSE,** значение по умолчанию:

  (0.8, 0.8, 0.8, 1.0).

Свойства материала

- **GL_SPECULAR,** значение по умолчанию: (0.0, 0.0, 0.0, 1.0).
- **GL_SHININESS,** [0..128]. Значение по умолчанию: 0.
- **GL_EMISSION,** значение по умолчанию: (0.0, 0.0, 0.0, 1.0).

Свойства материала

```
{Свойство материала}
glMaterialfv(GL_FRONT,
        GL_AMBIENT,
        @MaterialColor);
   {Прорисовка одной грани куба}
  glBegin(GL_POLYGON);
    glNormal3f(0.0, 0.0, -1.0);
    glVertex3f(1.0, 1.0, -1.0);
    glVertex3f(1.0, -1.0, -1.0);
    glVertex3f(-1.0, -1.0, -1.0);
    glVertex3f(-1.0, 1.0, -1.0);
 glEnd;
```

Материалы и освещение

frmMain.pas

frmMain

```
procedure TfrmGL.WMPaint(var Msg: TWMPaint);
var
  ps : TPaintStruct;
const
  low_ambient : Array [0..3] of GLfloat = ( 0.1, 0.1, 0.1, 1.0 );
  more_ambient : Array [0..3] of GLfloat = ( 0.4, 0.4, 0.4, 1.0 );
  most_ambient : Array [0..3] of GLfloat = ( 1.0, 1.0, 1.0, 1.0 );
begin
  BeginPaint(Handle, ps);

  glClear( GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT );

  glMaterialfv(GL_FRONT, GL_AMBIENT, @low_ambient);
  glMaterialf(GL_FRONT, GL_SHININESS, 40.0);
  glPushMatrix;
  glTranslatef (0.0, 2.0, 0.0);
  glutSolidTeapot(1.0);
  glPopMatrix;

  glMaterialfv(GL_FRONT, GL_AMBIENT, @more_ambient);
  glPushMatrix;
  glTranslatef (0.0, 0.0, 0.0);
  glutSolidTeapot(1.0);
  glPopMatrix;

  glMaterialfv(GL_FRONT, GL_AMBIENT, @most_ambient);
  glPushMatrix;
  glTranslatef (0.0, -2.0, 0.0);
  glutSolidTeapot(1.0);
  glPopMatrix;

  SwapBuffers(DC);
  EndPaint(Handle, ps);
end;
```

90: 1 Modified Insert Code Diagram

Материалы и освещение

Шаг 1. Создаем новый проект и рисуем фигуру согласно варианту задания.

Шаг 2. Добавляем режим работы с расчетом освещения `Gl.glEnable(Gl.GL_LIGHTING).`

Шаг 3. Добавляем первый источник света `Gl.glEnable(Gl.GL_LIGHT0).`

Шаг 4. Настраиваем нормализацию векторов нормалей для корректной работы с источниками света `Gl.glEnable(Gl.GL_NORMALIZE).`

Шаг 5. Указываем `Gl.glLightModelf(Gl.GL_LIGHT_MODEL_TWO_SIDE, Gl.GL_TRUE).`

Шаг 6. Добавляем `Gl.glLightfv(Gl.GL_LIGHT0, Gl.GL_DIFFUSE, light0_dif);`

где `float []light0_dif = { 0.7f, 0.2f, 0.2f }.`

Шаг 7. Дополняем `Gl.glLightfv(Gl.GL_LIGHT0, Gl.GL_POSITION, light0_pos);`

Где `float[] light0_pos = { 1.0f, 0.0f, 0.0f, 0.0f }`

Материалы и освещение

Шаг 8. Прописываем нормали для каждой грани фигуры

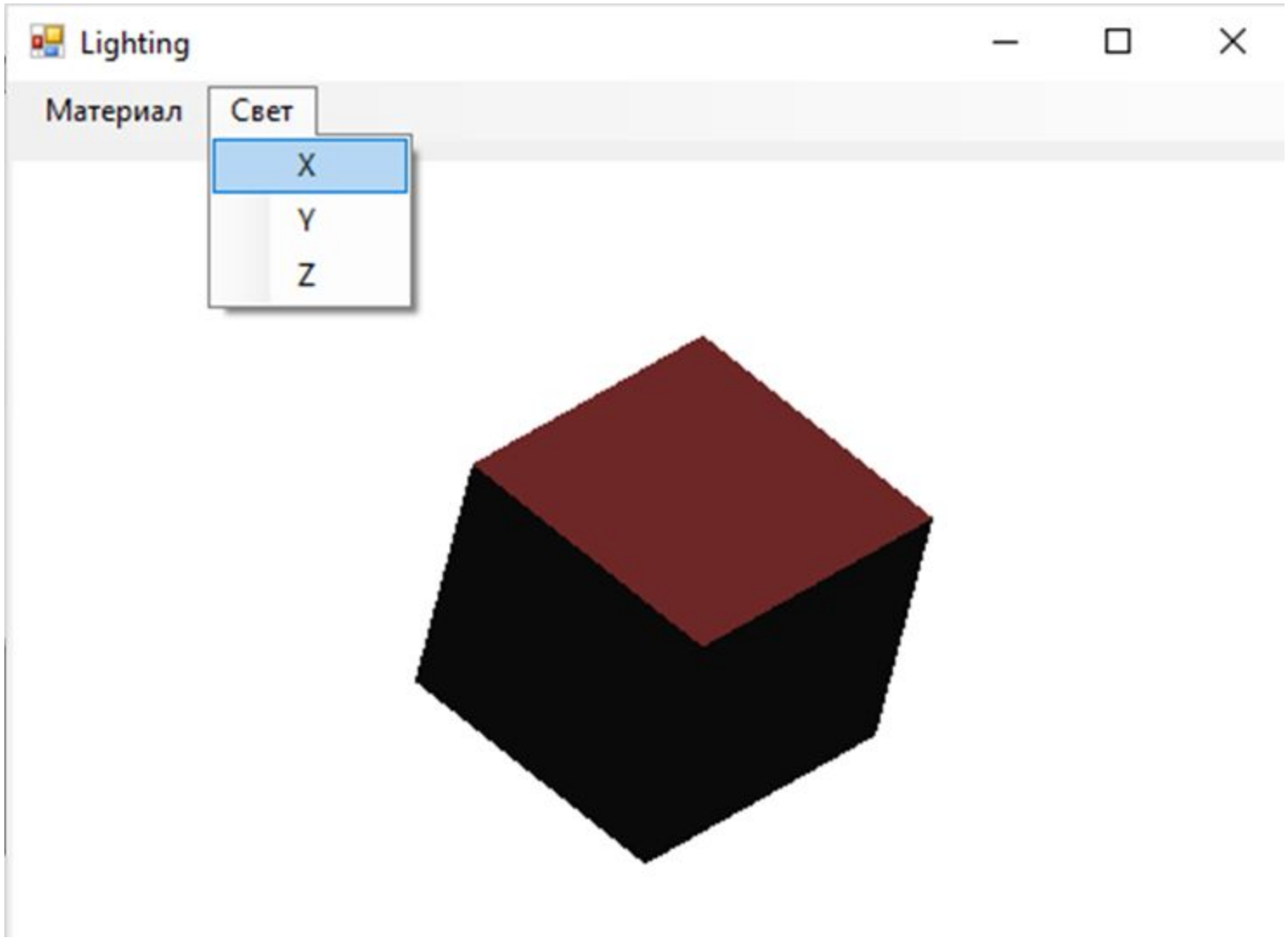
```
Gl.glBegin(Gl.GL_QUADS);
            Gl.glNormal3f(0.0f, 0.0f, -1.0f);
            Gl.glVertex3f(1.0f, 1.0f, 1.0f);
            Gl.glVertex3f(-1.0f, 1.0f, 1.0f);
            Gl.glVertex3f(-1.0f, -1.0f, 1.0f);
            Gl.glVertex3f(1.0f, -1.0f, 1.0f);
            Gl.glEnd();
```

Шаг 9. Указываем свойство материала фигуры

```
  Gl.glMaterialfv(Gl.GL_FRONT_AND_BACK, Gl.GL_EMISSION, color_am),
```

где `color_am = { 0f, 0f, 1f }` задает цвет фигуры.

Материалы и освещение



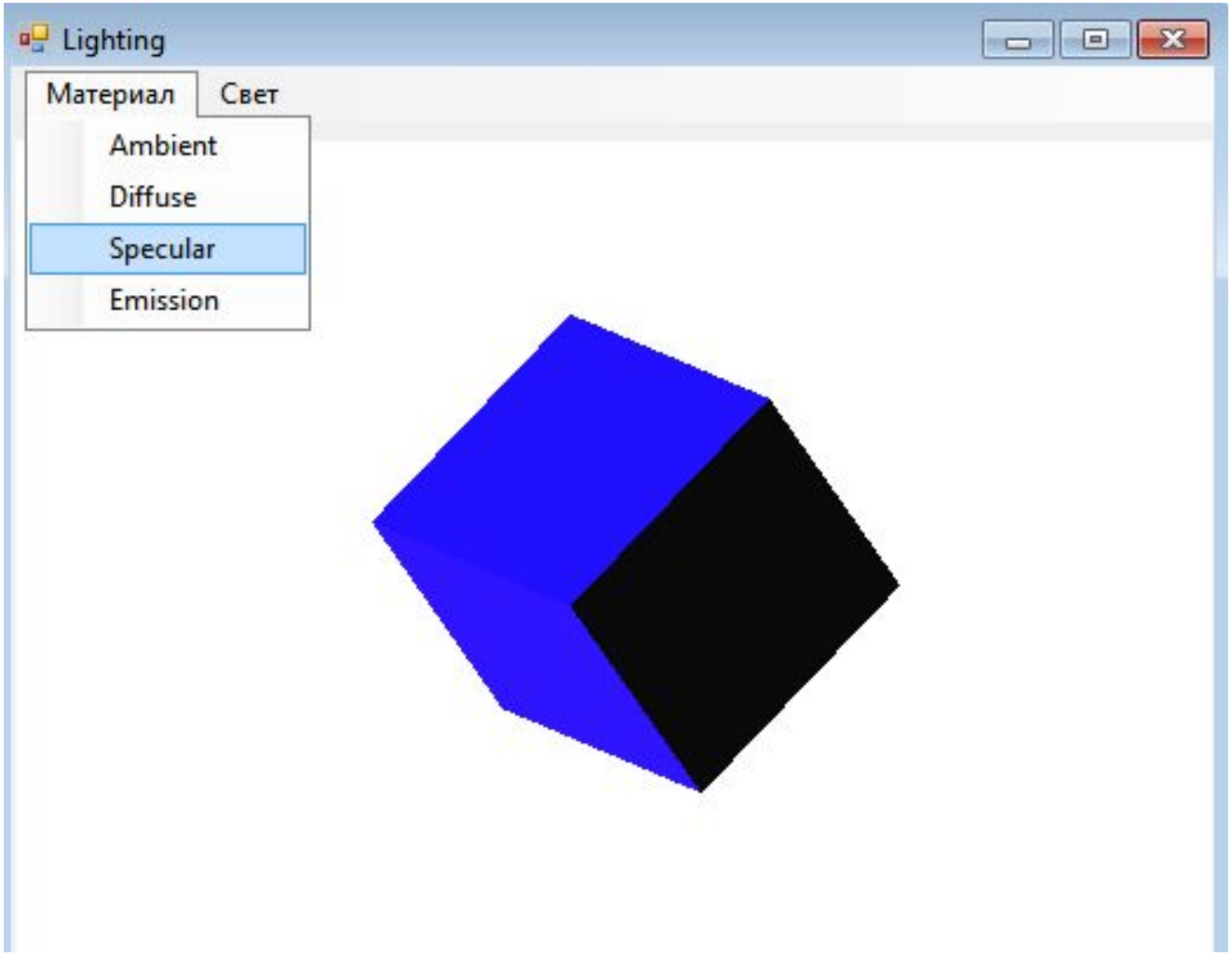
Материалы и освещение

```
private void светToolStripMenuItem_Click(object sender, EventArgs e)
{
    Gl.glEnable(Gl.GL_LIGHTING);
    Gl.glEnable(Gl.GL_LIGHT0);
    Gl.glLightModelf(Gl.GL_LIGHT_MODEL_TWO_SIDE, Gl.GL_TRUE);
    Gl.glEnable(Gl.GL_NORMALIZE);
    float[] light0_dif = { 0.7f, 0.2f, 0.2f };

    Gl.glLightfv(Gl.GL_LIGHT0, Gl.GL_DIFFUSE, light0_dif);

}
```

Материалы и освещение

```
private void specularToolStripMenuItem_Click(object sender, EventArgs e)
{
    Gl.glMaterialfv(Gl.GL_FRONT_AND_BACK, Gl.GL_SPECULAR, color_am);
}

private void xToolStripMenuItem_Click(object sender, EventArgs e)
{
    float[] light0_pos = { -1.0f, 0.0f,1.0f, 0.0f };
    Gl.glLightfv(Gl.GL_LIGHT0, Gl.GL_POSITION, light0_pos);
}
```

Материалы и освещение



Визуальные эффекты. Туман

- Туман является самым простым в использовании спецэффектом, предназначенным для передачи глубины пространства. Он позволяет имитировать атмосферные эффекты дымки и тумана.
  При его использовании объекты сцены перестают быть яркими и становятся более реалистичными, естественными для восприятия.

Туман имеет несколько характеристик:

- Цвет (GL_FOG_COLOR);
- Плотность (GL_FOG_DENSITY);
- Закон распространения (GL_FOG_MODE).

Упаравление наложением тумана

- Для того, чтобы включить туман используют команду с параметром **glEnable(GL_FOG),**
- для выключения **glDisable(GL_FOG);**
- Для вычисления интенсивности тумана используют команду **glFog[f i][v](pname,param[s]);**

pname:

GL_FOG_MODE – задает закон распространения тумана:

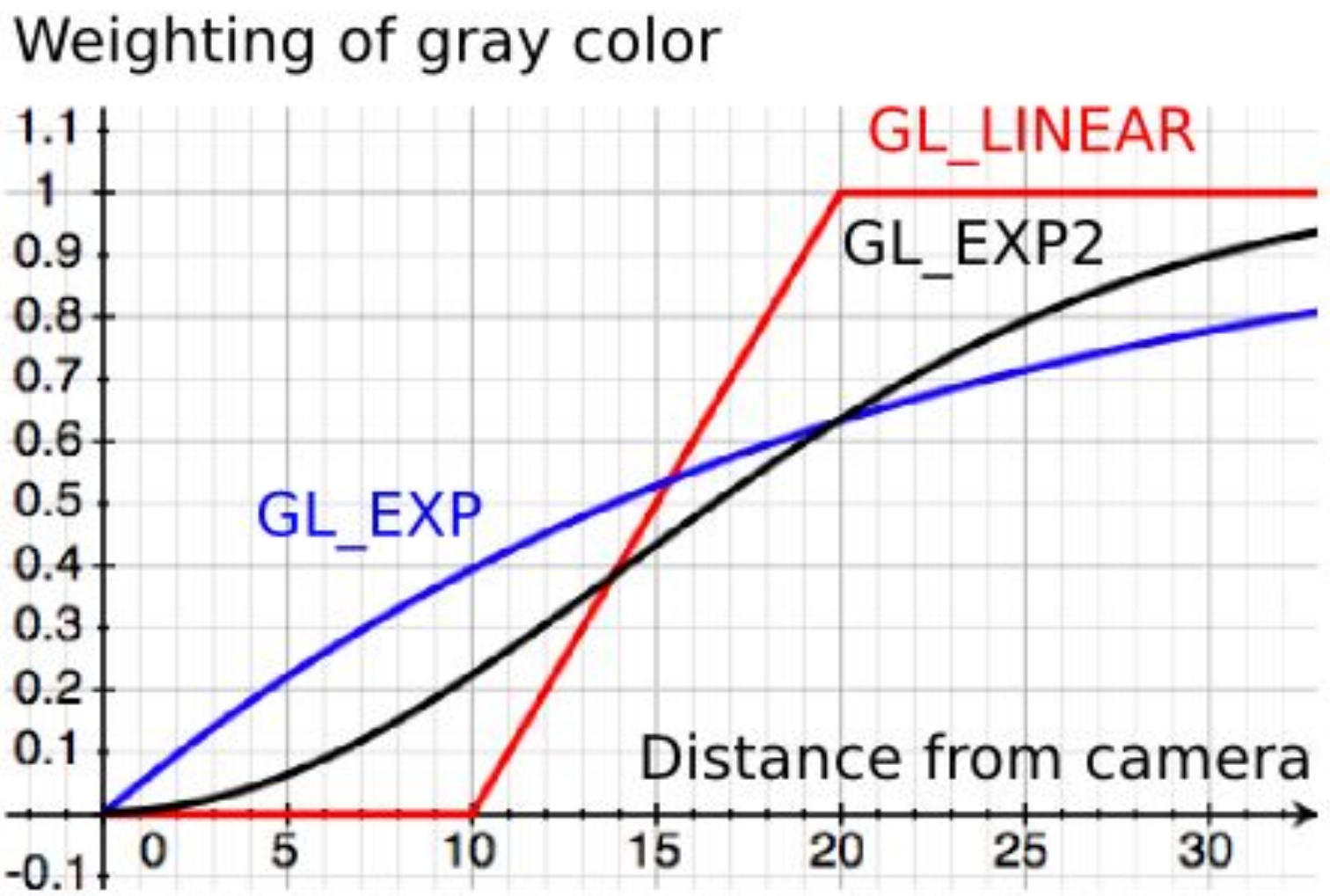
- *GL_LINEAR* – говорит о том, что плотность тумана распространяется по линейному закону, т.е. чем дальше туман, тем он плотнее;

  f=e-z/e-s

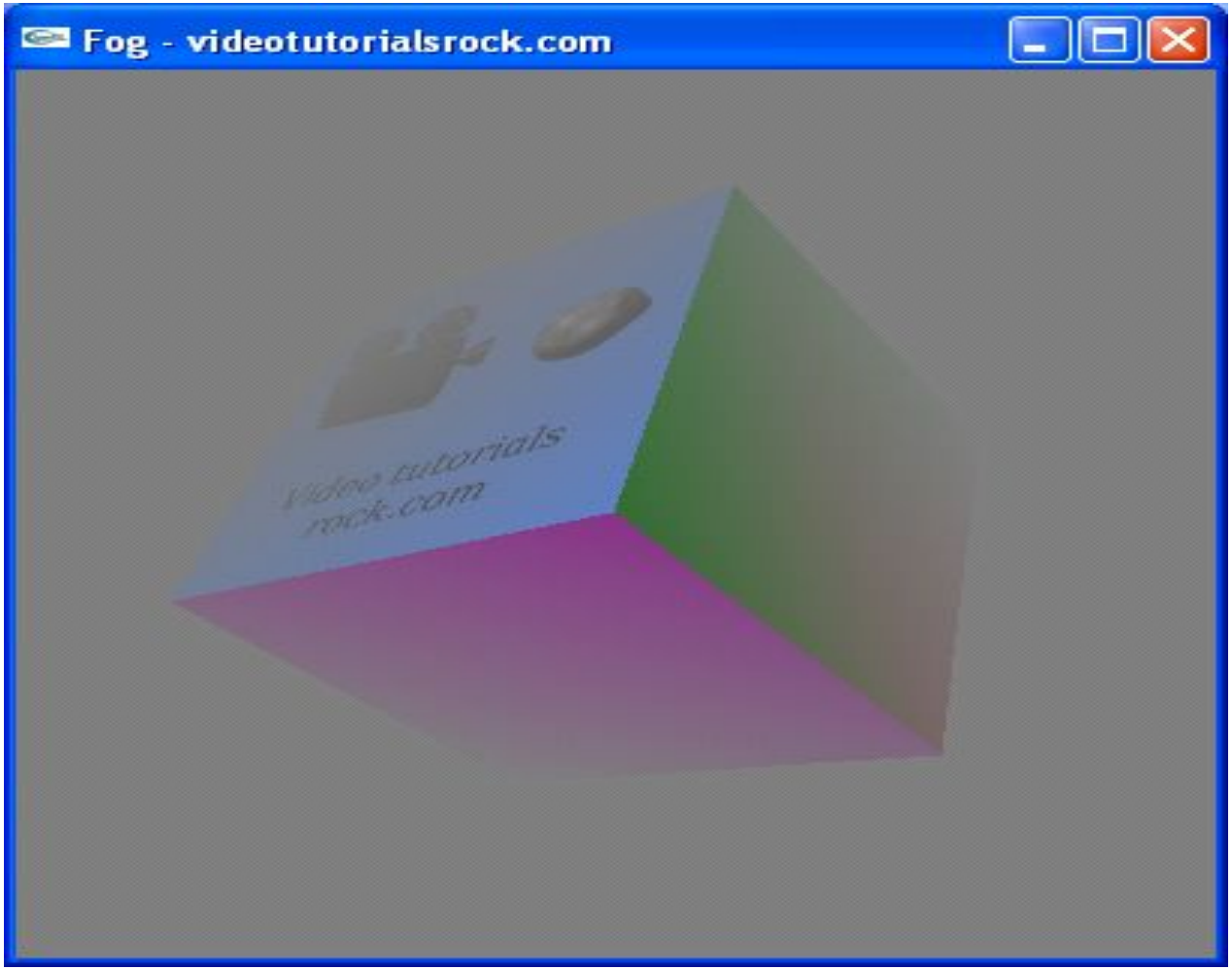
- *GL_EXP* - обычный туман, заполняющий весь экран. $f=exp(-d*z)$
- *GL_EXP2* - это следующий шаг после GL_EXP. Затуманит весь экран, за то придает больше глубины всей сцене. $f=exp(-(d*z)^2)$

Упаравление наложением тумана

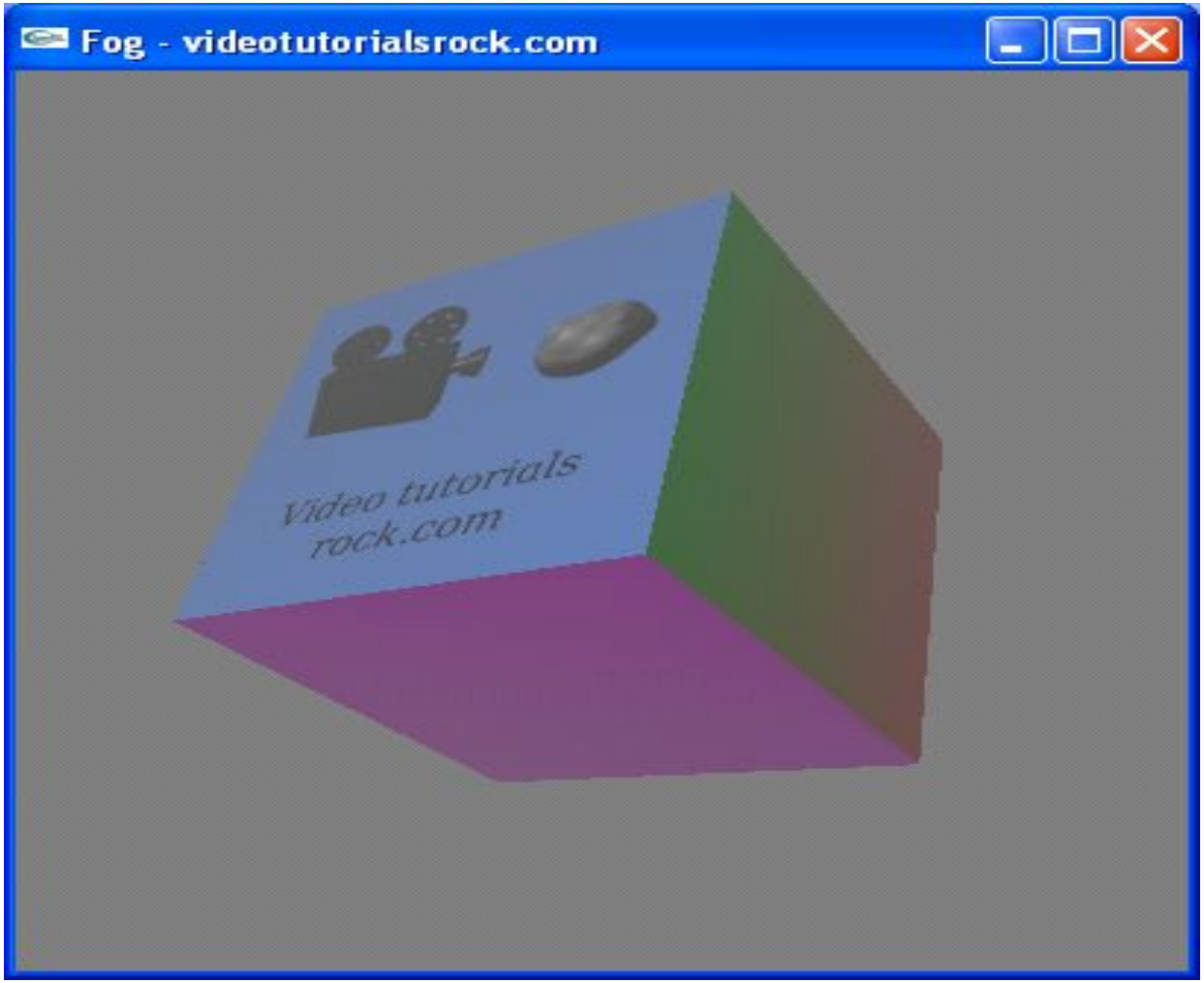
- **GL_FOG_DENSITY** – плотность тумана (по умолчанию принимает значение равное единице);
- **GL_FOG_START** – задает начальную границу тумана по координате z(по умолчанию принимает значение равное нулю);
- **GL_FOG_END** – задает конечную границу тумана по координате z(по умолчанию принимает значение равное единице);
- ***GL_FOG_INDEX*** - Индекс цвета тумана, только если вы используете палитру.
- **GL_FOG_COLOR –** массив RGBA задает цвет тумана.


Weighting of gray color
GL_LINEAR
GL_EXP2
GL_EXP
Distance from camera
1.1
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
-0.1
0
5
10
15
20
25
30

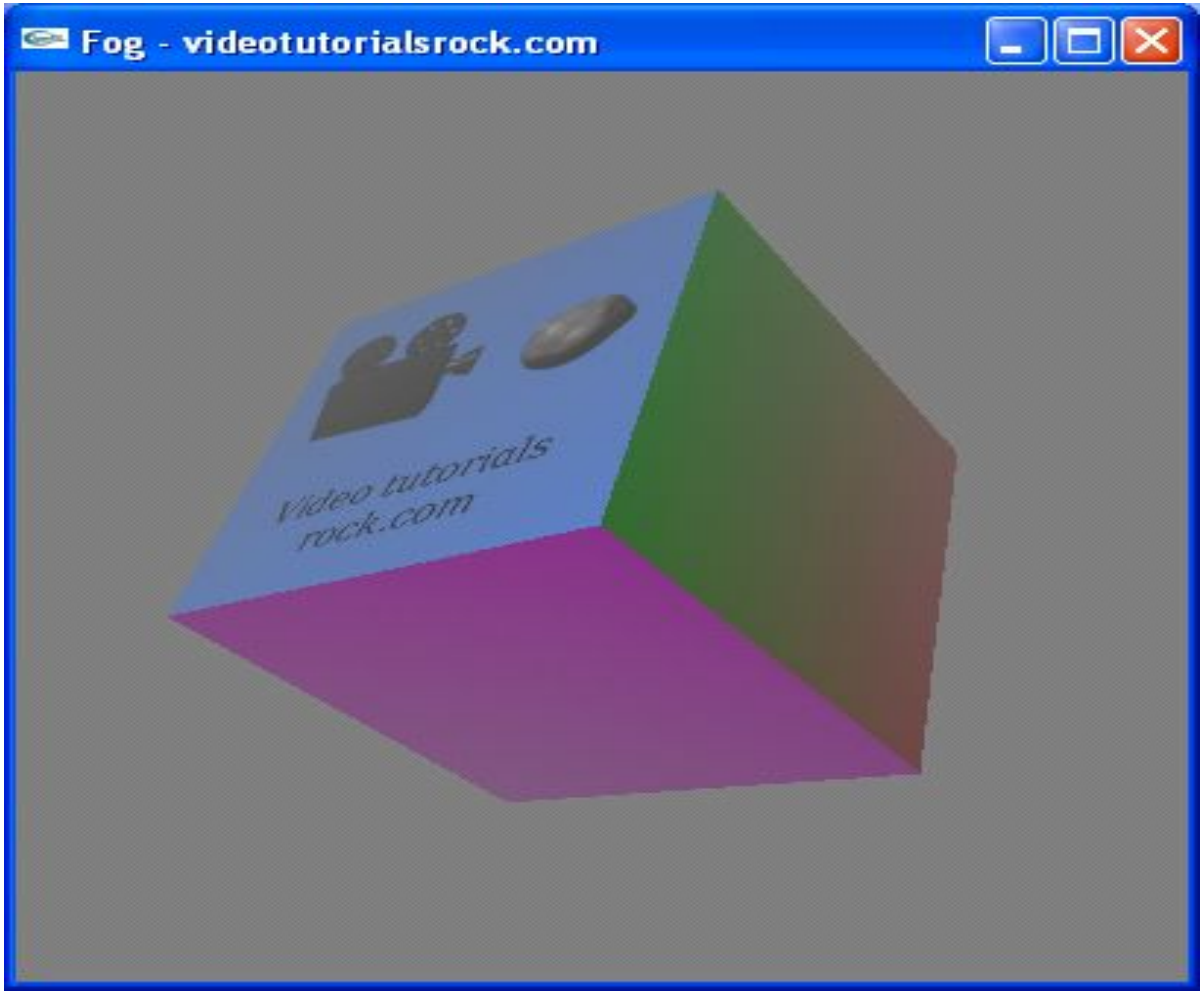
GL_LINEAR



GL_EXP

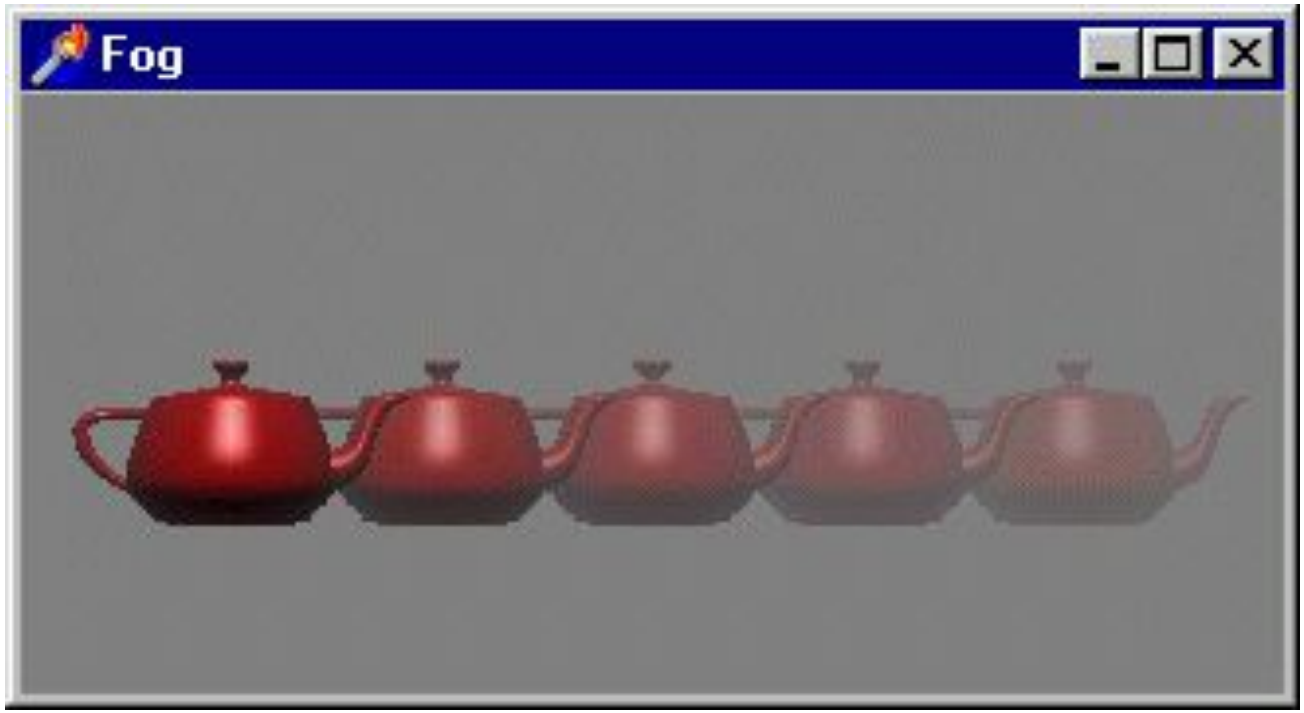


GL_EXP2



Пример на использование тумана

Пять чайников, располагающихся от наблюдателя на различном удалении



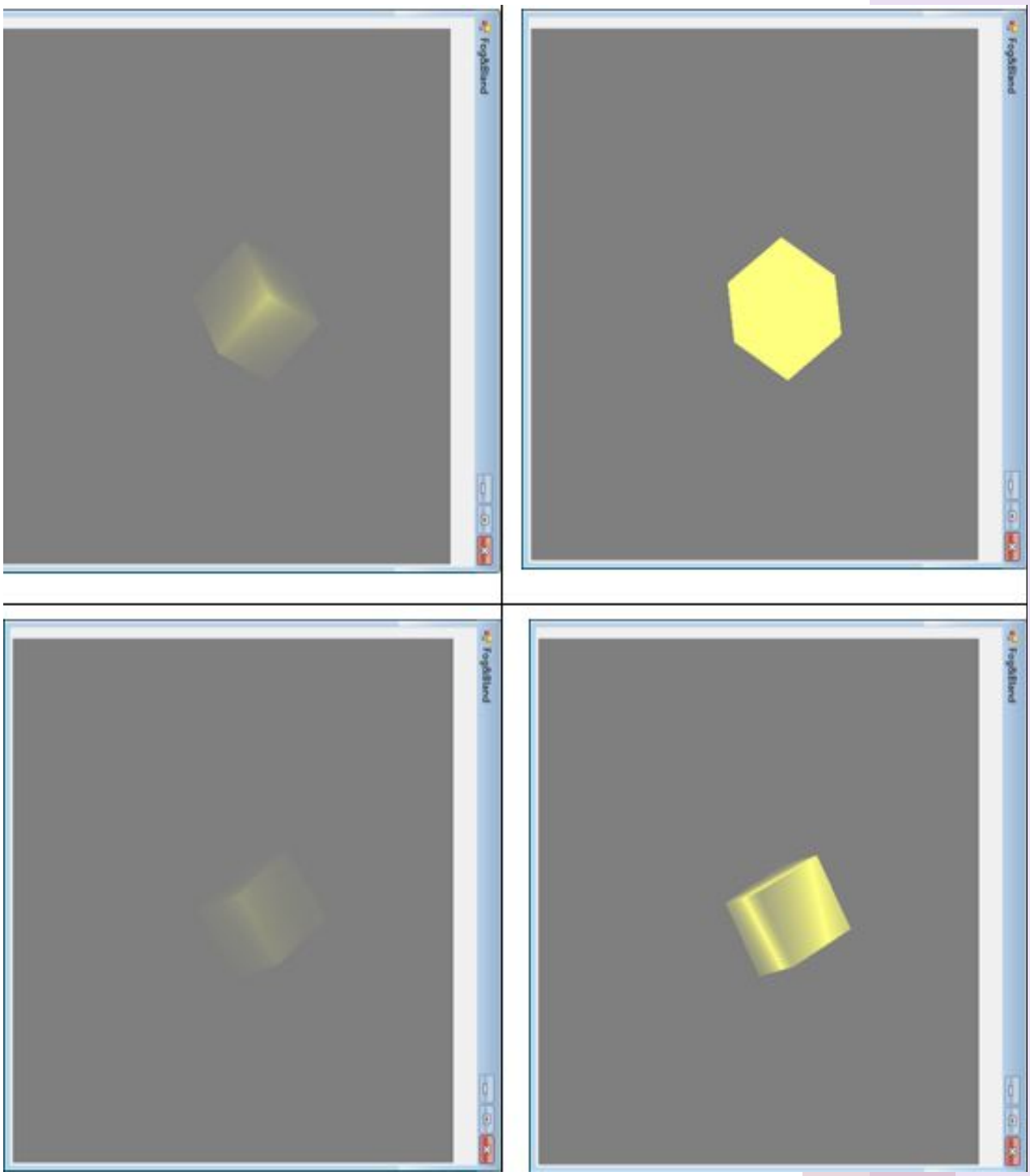
Наложение тумана

```
float[] fogColor = { 0.5f, 0.5f, 0.5f, 1.0f };

 Gl.glEnable(Gl.GL_FOG);

 Gl.glFogi(Gl.GL_FOG_MODE, Gl.GL_EXP);

Gl.glFogfv(Gl.GL_FOG_COLOR, fogColor);
```

Наложение тумана

```
Gl.glFogf(Gl.GL_FOG_DENSITY, 10f);

Gl.glFogf(Gl.GL_FOG_START, 0.5f);

Gl.glFogf(Gl.GL_FOG_END, 8.0f);
```

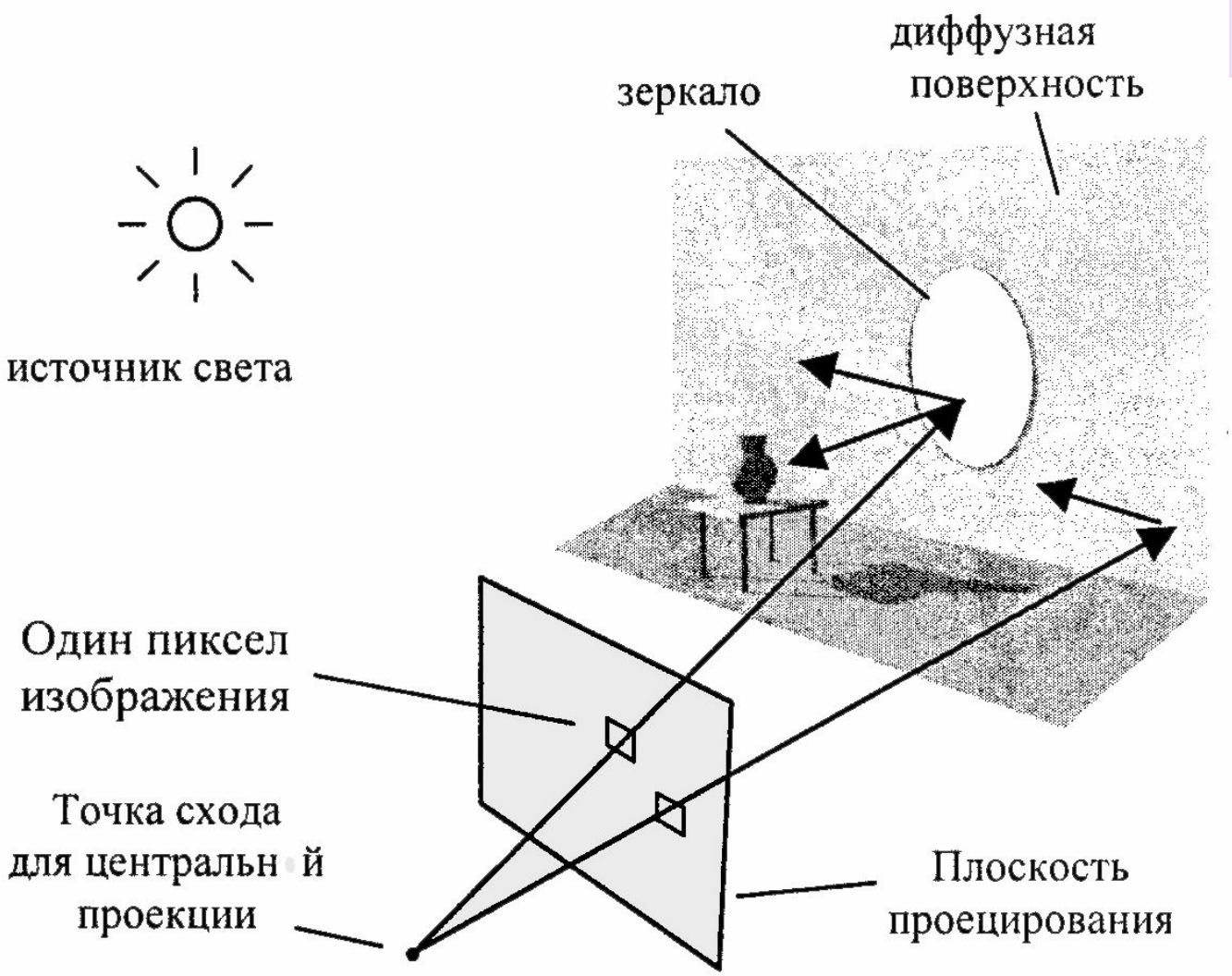
Наложение тумана



Режимы обработки прозрачности

```
Gl.glEnable(Gl.GL_BLEND);

Gl.glBlendFunc(Gl.GL_SRC_ALPHA,
               Gl.GL_ONE_MINUS_SRC_ALPHA);

Gl.glColor4f(1.0f, 1f, 0.5f, 0.5f);
```

Трассировка лучей



Трассировка лучей

Вторичные лучи разделяются на следующие:

- **лучи тени/освещения;**
- **лучи отражения;**
- **лучи преломления.**

Why ray tracing?

Environment map

Ray-traced reflections

PIXAR

Спасибо за внимание!