Jacek Matulewski
http://www.phys.uni.torun.pl/~jacek/

Delphi 7 + OpenGL 1.1

Część I

Wersja α

Ćwiczenia

Toruń, 22 grudnia 2005

Najnowsza wersja tego dokumentu znajduje się pod adresem
http://www.phys.uni.torun.pl/~jacek/dydaktyka/rad/rad4_opengl_delphi.pdf

Źródła opisanych w tym dokumencie programów znajdują się pod adresem
http://www.phys.uni.torun.pl/~jacek/dydaktyka/rad/rad4_opengl.zip

Spis treści

Zasoby

Oficjalne strony

OpenGL: http://www.opengl.org/
GLU, GLX, DRI: http://www.opengl.org/resources/libraries/glx.html
GLUT: http://www.opengl.org/resources/libraries/glut.html

Źródła

[1] Andrzej Orłowski *Leksykon OpenGL*, Helion 2005
[2] Waldemar Pokuta *OpenGL. Ćwiczenia*, Helion 2003
[3] Alex Semichastny *OpenGL w Delphi*, http://www.borland.pl/tech/opengl.shtml
[4] Richard S. Wright Jr., Benjamin Lipchak, *OpenGL. Księga eksperta (wydanie III)*, Helion 2005

Inne strony

GLUT: http://homepages.borland.com/ccalvert/opengl/glut/OpenGLGlut.html
Delphi OpenGL Community: http://www.delphigl.com/
Delphi3D: http://www.delphi3d.net/
Przykłady projektów OpenGL dla Delphi: http://www.sulaco.co.za/opengl.htm

Rysowanie i animacja figur

Ćwiczenie 1
Inicjacja grafiki OpenGL w aplikacji Delphi

1. Tworzymy nowy projekt *Application*.
2. Przechodzimy do kodu źródłowego i do sekcji `uses` dodajemy moduł *OpenGL*.

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, OpenGL;
```

3. Deklarujemy dwie prywatne metody pomocnicze i dwa prywatne pola:

```
type
  TForm1 = class(TForm)
private
    { Private declarations }
    uchwytDC :HDC; //uchwyt do "display device context (DC)"
    uchwytRC :HGLRC; //uchwyt do "OpenGL rendering context"
    function GL_UstalFormatPikseli(uchwytDC :HDC) :Boolean;
    procedure GL_UstawienieSceny;
public
    { Public declarations }
  end;
```

Przechowanie uchwytów do kontekstu renderowania konieczne jest, aby możliwe było jego usunięcie przy zamknięciu okna.

4. Definiujemy pierwszą z nich:

```
function TForm1.GL_UstalFormatPikseli(uchwytDC :HDC) :Boolean;
```

```
var
  opisFormatuPikseli :PIXELFORMATDESCRIPTOR;
  formatPikseli :Integer;
begin
Result:=False;
with opisFormatuPikseli do
  begin
  dwFlags:=PFD_SUPPORT_OPENGL or PFD_DRAW_TO_WINDOW or PFD_DOUBLEBUFFER; //w oknie,
podwojne buforowanie
  iPixelType:=PFD_TYPE_RGBA; //typ koloru RGB
  cColorBits:=32; //jakosc kolorów 4 bajty
  cDepthBits:=16; //glebokosc bufora Z (z-buffer)
  iLayerType:=PFD_MAIN_PLANE;
  end;
formatPikseli:=ChoosePixelFormat(uchwytDC, @opisFormatuPikseli);
if (formatPikseli=0) then Exit;
if (SetPixelFormat(uchwytDC, formatPikseli, @opisFormatuPikseli) <> True) then Exit;
Result:=True;
end;
```

5. I drugą:

```
procedure TForm1.GL_UstawienieSceny;
begin
//ustawienie punktu projekcji
glMatrixMode(GL_PROJECTION); //macierz projekcji
//left,right,bottom,top,znear,zfar (clipping)
glFrustum(-0.1, 0.1, -0.1, 0.1, 0.3, 25.0); //mnozenie macierzy przez macierz
perspektywy - ustalanie piramidy frustum
glMatrixMode(GL_MODELVIEW); //powrot do macierzy widoku modelu
glEnable(GL_DEPTH_TEST); //z-buffer aktywny = ukrywanie niewidocznych trojkatow !!!
end;
```

6. Zmieniamy własność `BorderStyle` formy na `bsSingle` oraz `BorderIcons.biMaximize` na `False`.

7. Tworzymy metodę zdarzeniową do `OnCreate` formy i w niej tworzymy kontekst OpenGL związany z bieżącym oknem:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
//biezace okno staje sie oknem OpenGL
uchwytDC:=GetDC(Handle);
GL_UstalFormatPikseli(uchwytDC);
uchwytRC:=wglCreateContext(uchwytDC);
wglMakeCurrent(uchwytDC,uchwytRC);
GL_UstawienieSceny;
Caption:='OpenGL '+glGetString(GL_VERSION);
end;
```

8. Musimy pamiętać o usunięciu kontekstu OpenGL (kontekstu renderowania) przy zamknięciu okna. W tym celu tworzymy metodę zdarzeniową do `OnClose` formy:

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
```

```
wglMakeCurrent(0,0);
wglDeleteContext(uchwytRC);
ReleaseDC(Handle,uchwytDC);
PostQuitMessage(0);
end;
```

Od tej chwili główne okno aplikacji może być wykorzystane do rysowania za pomocą OpenGL

Ćwiczenie 2
Rysowanie figury płaskiej (trójkąta). Podwójne buforowanie

1. Deklarujemy metodę `Rysuj`:

```
type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormKeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
  private
    { Private declarations }
    uchwytDC :HDC; //uchwyt do "display device context (DC)"
    uchwytRC :HGLRC; //uchwyt do "OpenGL rendering context"
    function GL_UstalFormatPikseli(uchwytDC :HDC) :Boolean;
    procedure GL_UstawienieSceny;
    procedure Rysuj;
  public
    { Public declarations }
  end;
```

2. Definiujemy ją:

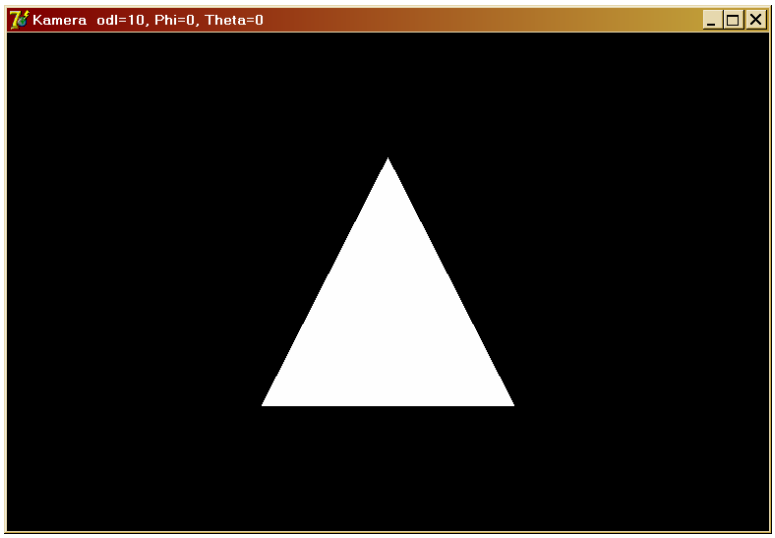
```
procedure TForm1.Rysuj;
const x0=1.0; y0=1.5; z0=1.0;
begin
//Przygotowanie bufora
glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
glLoadIdentity; //biezaca macierz = I
glTranslatef(0.0, 0.0, -10.0); //odsuniecie calosci o 10
//Rysowanie trojkata
glBegin(GL_TRIANGLES);
//ustalanie trzech wierzcholkow trojkata (werteksow (x,y,z))
//(0,0,???) jest mniej wiecej w srodku ekranu
glVertex3f( -x0, -y0, z0); //dolny lewy
glVertex3f(x0, -y0, z0); //dolny prawy
glVertex3f(0, y0, z0); //gorny
//koniec rysowania figury
glEnd;
//Z bufora na ekran
SwapBuffers(wglGetCurrentDC);
```

```
end;
```

3. Za pomocą inspektora obiektów tworzymy metodę związaną ze zdarzeniem `OnPaint` formy i umieszczamy w niej wywołanie metody `Rysuj`. To zapewnia odświeżanie rysunku.

```
procedure TForm1.FormPaint(Sender: TObject);
begin
Rysuj;
end;
```

Przyrostek nazwy wskazuje na ilość i typ przeciążonych funkcji np. `glVertex3f`. W Delphi można tych funkcji używać też bez przyrostków (argumenty są rozpoznawane automatycznie) np. `glVertex`.



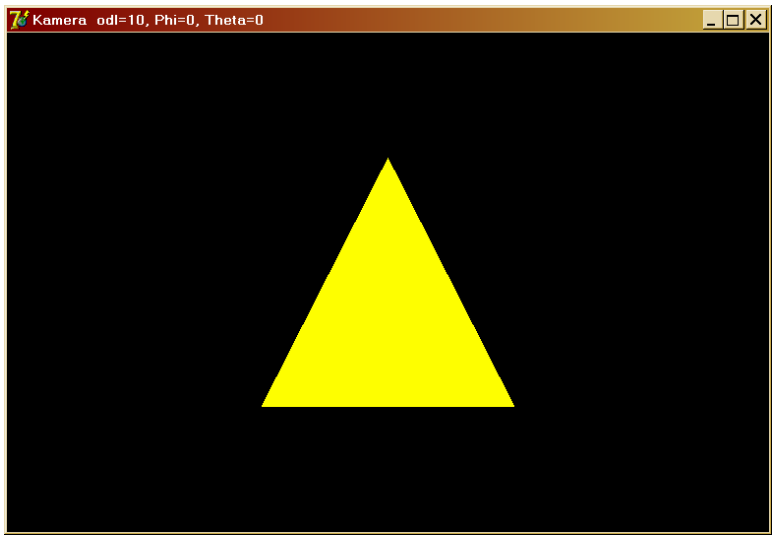
Rysunek OXY(2D)

Ćwiczenie 3
Kolor

Do powyżej funkcji dodajemy wywołanie funkcji `glColor`, a dokładnie jej wersji `glColor3ub`:

```
procedure TForm1.Rysuj;
const x0=1.0; y0=1.5; z0=1.0;
begin
//Przygotowanie bufora
glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
glLoadIdentity; //biezaca macierz = I
glTranslatef(0.0, 0.0, -10.0); //mnozenie przez macierz translacji
//Rysowanie trojkata
glBegin(GL_TRIANGLES);
//ustalanie trzech wierzcholkow trojkata (werteksow: x,y,z)
//(0,0,???) jest mniej wiecej w srodku ekranu
glColor3ub(255,255,0); //zolty
glVertex3f( -x0, -y0, z0); //dolny lewy
glVertex3f(x0, -y0, z0); //dolny prawy
```

```
glVertex3f(0, y0, z0); //gorny
//koniec rysowania figury
glEnd;
//Z bufora na ekran
SwapBuffers(wglGetCurrentDC);
end;
```

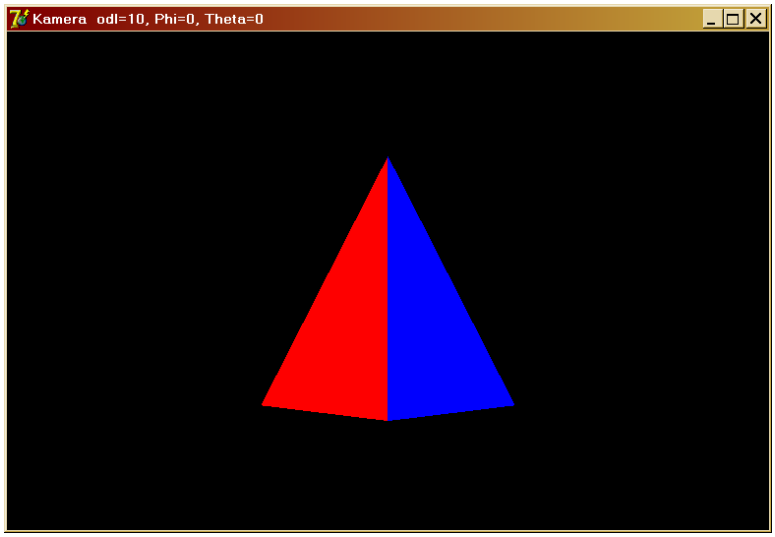


Ćwiczenie 4
Rysowanie figury przestrzennej (ostrosłupa)

Dodajemy trzy trójkąty (zob. rysunek):

```
procedure TForm1.Rysuj;
const x0=1.0; y0=1.5; z0=1.0;
begin
//Przygotowanie bufora
glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
glLoadIdentity; //biezaca macierz = I
glTranslatef(0.0, 0.0, -10.0); //mnozenie przez macierz translacji

//Rysowanie trojkata
glBegin(GL_TRIANGLES);
//ustalanie trzech wierzcholkow trojkata (werteksow: x,y,z)
//(0,0,???) jest mniej wiecej w srodku ekranu

//sciana tylnia
glColor3ub(255,255,0); //zolty
glVertex3f( -x0, -y0, z0); //dolny lewy
glVertex3f(x0, -y0, z0); //dolny prawy
glVertex3f(0, y0, z0); //gorny

//podstawa
glColor3ub(0,255,0); //zielony
glVertex3f( -x0, -y0, z0); //dolny lewy
glVertex3f(x0, -y0, z0); //dolny prawy
```

```
glVertex3f(0, -y0, 2*z0); //dolny przedni

//lewa
glColor3ub(255,0,0); //czerwony
glVertex3f( -x0, -y0, z0); //dolny lewy
glVertex3f(0, -y0, 2*z0); //dolny przedni
glVertex3f(0, y0, z0); //gorny

//prawa
glColor3ub(0,0,255); //niebieski
glVertex3f(x0, -y0, z0); //dolny prawy
glVertex3f(0, -y0, 2*z0); //dolny przedni
glVertex3f(0, y0, z0); //gorny

//koniec rysowania figury
glEnd;
//Z bufora na ekran
SwapBuffers(wglGetCurrentDC);
end;
```



Rysunek OXYZ(3D)

Ćwiczenie 5
Wyodrębnienie metody rysującej figurę

1. Definiujemy metodę `RysujOstroslup` i umieszczamy w niej polecenia od `glBegin` do `glEnd`:

```
procedure TForm1.RysujOstroslup(x0,y0,z0 :Single);
begin
//Rysowanie trojkatow
glBegin(GL_TRIANGLES);
//ustalanie trzech wierzcholkow trojkata (werteksow (x,y,z))
//(0,0,???) jest mniej wiecej w srodku ekranu
```

```
//sciana tylnia
glColor3ub(255,255,0); //zolty
glVertex3f( -x0, -y0, z0); //dolny lewy
glVertex3f(x0, -y0, z0); //dolny prawy
glVertex3f(0, y0, z0); //gorny

//podstawa
glColor3ub(0,255,0); //zielony
glVertex3f( -x0, -y0, z0); //dolny lewy
glVertex3f(x0, -y0, z0); //dolny prawy
glVertex3f(0, -y0, 2*z0); //dolny przedni

//lewa
glColor3ub(255,0,0); //czerwony
glVertex3f( -x0, -y0, z0); //dolny lewy
glVertex3f(0, -y0, 2*z0); //dolny przedni
glVertex3f(0, y0, z0); //gorny

//prawa
glColor3ub(0,0,255); //niebieski
glVertex3f(x0, -y0, z0); //dolny prawy
glVertex3f(0, -y0, 2*z0); //dolny przedni
glVertex3f(0, y0, z0); //gorny

//koniec rysowania figury
glEnd;
end;
```

2. Do definicji klasy dodajemy odpowiednią deklarację.
3. Modyfikujemy metodę `Rysuj`:

```
procedure TForm1.Rysuj;
const x0=1.0; y0=1.5; z0=1.0;
begin
//Przygotowanie bufora
glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
glLoadIdentity; //biezaca macierz = I
glTranslatef(0.0, 0.0, -10.0); //odsuniecie calosci o 10

RysujOstroslup(x0,y0,z0);

//Z bufora na ekran
SwapBuffers(wglGetCurrentDC);
end;
```

Aby rysować szkielet ostrosłupa w `glBegin` zamiast `GL_TRIANGLES` należy użyć `GL_LINE_LOOP`.

Ćwiczenie 6
Obroty obiektu

1. Definiujemy pole `Kat` (kąt) typu `Single`:

```
type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormKeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
  private
    { Private declarations }
    uchwytDC :HDC; //uchwyt do "display device context (DC)"
    uchwytRC :HGLRC; //uchwyt do "OpenGL rendering context"
    Phi, Theta :Single;
    function GL_UstalFormatPikseli(uchwytDC :HDC) :Boolean;
    procedure GL_UstawienieSceny;
  public
    { Public declarations }
  end;
```

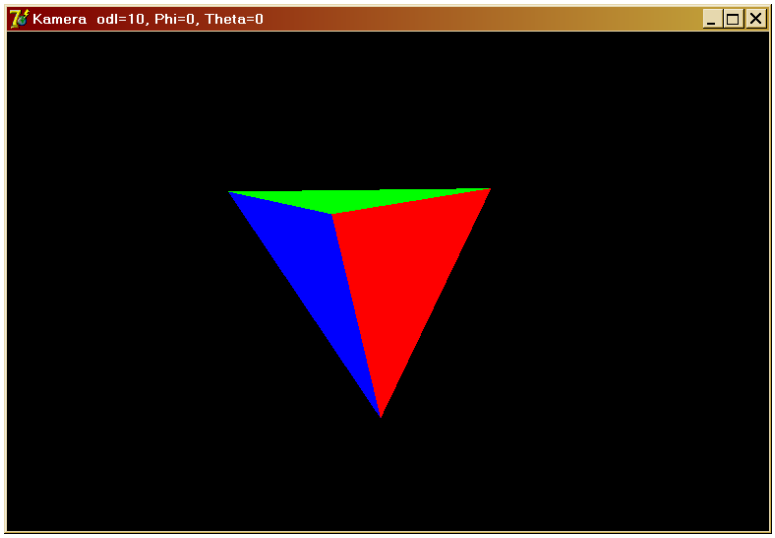
2. Do metody rysującej ostrosłup dodajemy polecenia obracające macierz widoku modelu (bieżącą macierz):

```
procedure TForm1.Rysuj;
const x0=1.0; y0=1.5; z0=1.0;
begin
//Przygotowanie bufora
glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
glLoadIdentity; //biezaca macierz = I
glTranslatef(0.0, 0.0, -10.0); //mnozenie przez macierz translacji

//obroty
glRotatef(Phi, 0.0, 1.0, 0.0); //wokol OY
glRotatef(Theta, 1.0, 0.0, 0.0); //wokol OX

RysujOstroslup(x0,y0,z0);

//Z bufora na ekran
SwapBuffers(wglGetCurrentDC);
end;
```

3. Zmieniamy własność `KeyPreview` formy na `True`.
4. Za pomocą inspektora tworzymy metodę związaną z `OnKeyDown` formy:

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
case Key of
  VK_LEFT :Phi:=Phi-3;
  VK_RIGHT :Phi:=Phi+3;
  VK_UP :Theta:=Theta-3;
```

```
  VK_DOWN :Theta:=Theta+3;
end;
Rysuj;
end;
```



Ćwiczenie 7
Przesunięcia obiektu

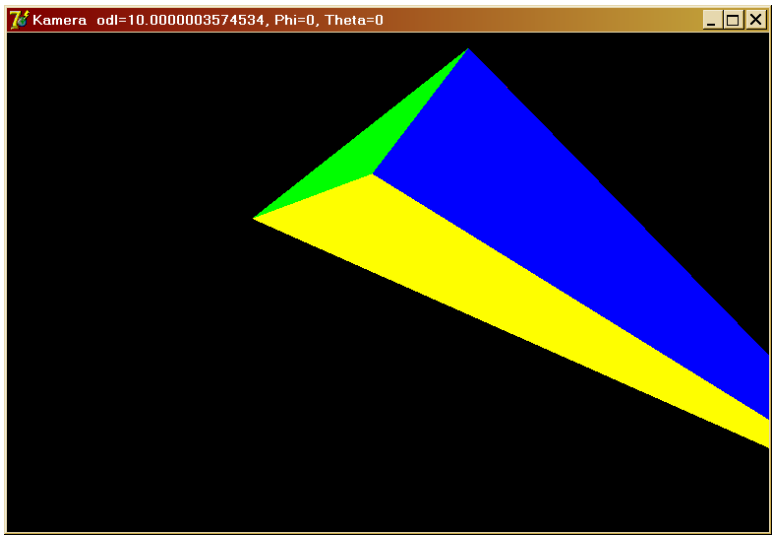
1. W klasie `TForm1` definiujemy jeszcze dwa pola prywatne typu `Single` o nazwach `PozycjaX`, `PozycjaY` i `PozycjaZ`.
2. Do metody `FormPaint` dodajemy polecenie przesuwające figurę:

```
procedure TForm1.Rysuj;
const x0=1.0; y0=1.5; z0=1.0;
begin
//Przygotowanie bufora
glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
glLoadIdentity; //biezaca macierz = I
glTranslatef(0.0, 0.0, -10.0); //mnozenie przez macierz translacji

//obroty
glRotatef(Phi, 0.0, 1.0, 0.0); //wokol OY
glRotatef(Theta, 1.0, 0.0, 0.0); //wokol OX

//przesuniecia
glTranslatef(PozycjaX,PozycjaY,PozycjaZ);

RysujOstroslup(x0,y0,z0);

//Z bufora na ekran
SwapBuffers(wglGetCurrentDC);
end;
```

3. Rozbudowujemy metodę `FormKeyDown`:

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
```

```
begin
//obroty
if Shift=[] then
case Key of
  VK_LEFT :Phi:=Phi-3;
  VK_RIGHT :Phi:=Phi+3;
  VK_UP :Theta:=Theta-3;
  VK_DOWN :Theta:=Theta+3;
end;
//przesuniecia
if Shift=[ssCtrl] then
case Key of
  VK_LEFT :PozycjaX:=PozycjaX-0.1;
  VK_RIGHT :PozycjaX:=PozycjaX+0.1;
  VK_UP :PozycjaY:=PozycjaY+0.1;
  VK_DOWN :PozycjaY:=PozycjaY-0.1;
end;
if Shift=[ssShift] then
case Key of
  VK_LEFT :PozycjaX:=PozycjaX-0.1;
  VK_RIGHT :PozycjaX:=PozycjaX+0.1;
  VK_UP :PozycjaZ:=PozycjaZ-0.1;
  VK_DOWN :PozycjaZ:=PozycjaZ+0.1;
end;
 //rysowanie
Rysuj;
end;
```



Ćwiczenie 8
Prosta animacja

1. Na formie umieszczamy komponent `TTimer` z zakładki *System*.
2. Tworzymy metodę zdarzeniową do `OnTimer`:

```
procedure TForm1.Timer1Timer(Sender: TObject);
```

```
begin
Phi:=Phi+1;
Rysuj;
end;
```

3. Zmieniamy własność `Interval` komponentu `Timer1` na 10.
4. Dodajemy możliwość wyłączenie/włączenia animacji za pomocą klawisza *q*:

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
case Key of
  Ord('Q'):  Timer1.Enabled:=not Timer1.Enabled;
  VK_ESCAPE: Close;
end;
//ciag dalszy metody
```

Ilość klatek na sekundę (własność `Timer1.Interval`) nie wpływa na obciążenie głównego procesora, a na procesora karty graficznej (jeżeli jest ona zgodna z OpenGL i nie ma emulacji).

Ćwiczenie 9
Rysowanie osi układu współrzędnych

1. Definiujemy metodę rysującą osie układu współrzędnych OXYZ:

```
procedure TForm1.RysujOsie(rozmiar :Single);
begin
glBegin(GL_LINES);
glColor3ub(255,255,255);
glVertex3f(0,0,0); glVertex3f(rozmiar,0,0); //OX, w prawo
glVertex3f(0,0,0); glVertex3f(0,rozmiar,0); //OY, do gory
glVertex3f(0,0,0); glVertex3f(0,0,rozmiar); //OZ, w glab
glEnd;
end;
```

2. Wywołujemy ją z metody `Rysuj` (przed poleceniami wykonującymi obroty i translacje).

```
procedure TForm1.Rysuj;
const x0=1.0; y0=1.5; z0=1.0;
begin
//Przygotowanie bufora
glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
glLoadIdentity; //biezaca macierz = I
glTranslatef(0.0, 0.0, -10.0); //odsuniecie calosci o 10

RysujOsie(x0);

//obroty
glRotatef(Phi, 0.0, 1.0, 0.0); //wokol OY
glRotatef(Theta, 1.0, 0.0, 0.0); //wokol OX
```

```
//przesuniecia
glTranslatef(PozycjaX,PozycjaY,PozycjaZ);

RysujOstroslup(x0,y0,z0);

//Z bufora na ekran
SwapBuffers(wglGetCurrentDC);
end;
```

Ćwiczenia 11
Zamknięcie formy

Tworzymy metodę zdarzeniową do `OnClose` formy:

??????????

Kamera

Ćwiczenie 12
Jawnie ustawić położenie kamery

W metodzie `Rysuj` zastąpić polecenie odsuwające scenę o 10 poleceniem ustalającym położenie kamery:

```
procedure TForm1.Rysuj;
const x0=1.0; y0=1.5; z0=1.0;
begin
//Przygotowanie bufora
glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
glLoadIdentity; //biezaca macierz = I
~~glTranslatef(0.0, 0.0, -10.0); //odsuniecie calosci o 10~~

gluLookAt(0,0,10,  //polozenie kamery/oka
          0,0,0,   //punkt, na ktory skierowana jest kamera/oko
          0,1,0);  //kierunek "do gory" kamery

//dalsza czesc metody
```

Funkcja `gluLookAt` jest zdefiniowana w bibliotece GLU (*OpenGL Utility Library*) – bibliotece zawierającej funkcje wyższego poziomu (krzywe, figury przestrzenne, itp.). Zob. paragraf o GLU.

Ćwiczenie 13
Ruch kamery kontrolowany położeniem kursora myszy

Zakładamy, że kamera zawsze będzie skierowana do środka układu współrzędnych.

1. Wyłączamy animację zmieniając własność `Timer1.Enabled` na `False`.
2. Do klasy `TForm1` dodajemy trzy pola prywatne określające położenie kamery oraz pole o wartości logicznej kontrolujące, czy ruch kamery jest aktywny.

```
type
```

```
TForm1 = class(TForm)
  Timer1: TTimer;
  procedure FormCreate(Sender: TObject);
  procedure FormPaint(Sender: TObject);
  procedure FormKeyDown(Sender: TObject; var Key: Word;
    Shift: TShiftState);
  procedure Timer1Timer(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure FormMouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
private
  { Private declarations }
  uchwytDC :HDC; //uchwyt do "display device context (DC)"
  uchwytRC :HGLRC; //uchwyt do "OpenGL rendering context"
  Phi, Theta :Single;
  PozycjaX, PozycjaY, PozycjaZ :Single;
  RuchKamery :Boolean;
  KameraX, KameraY, KameraZ :Single;
  function GL_UstalFormatPikseli(uchwytDC :HDC) :Boolean;
  procedure GL_UstawienieSceny;
  procedure Rysuj;
  procedure RysujOstroslup(x0,y0,z0 :Single);
  procedure RysujOsie(rozmiar :Single);
public
  { Public declarations }
end;
```

3. Modyfikujemy argumenty funkcji `gluLookAt` w metodzie Rysuj:

```
procedure TForm1.Rysuj;
const x0=1.0; y0=1.5; z0=1.0;
begin
//Przygotowanie bufora
glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
glLoadIdentity; //biezaca macierz = I

gluLookAt(KameraX,KameraY,KameraZ,  //polozenie oka
          0,0,0,  //polozenie srodka ukladu wsp.
          0,1,0); //kierunek "do gory"

//dalsza czesc metody
```

4. Do metody `FormCreate` dodajemy polecenie inicjujące składową *z* położenia kamery, aktywujące ruch kamery oraz ustawiające myszkę w środku układu współrzędnych obrazu.

```
procedure TForm1.FormCreate(Sender: TObject);
var
  polozenieMyszy :TPoint;
begin
//biezace okno staje sie oknem OpenGL
```

```
uchwytDC:=GetDC(Handle);
GL_UstalFormatPikseli(uchwytDC);
uchwytRC:=wglCreateContext(uchwytDC);
wglMakeCurrent(uchwytDC,uchwytRC);
GL_UstawienieSceny;
Caption:='OpenGL '+glGetString(GL_VERSION);

KameraZ:=10;
RuchKamery:=True;

polozenieMyszy.X:=ClientWidth div 2;
polozenieMyszy.Y:=ClientHeight div 2;
if RuchKamery then Mouse.CursorPos:=ClientToScreen(polozenieMyszy);
end;
```

5. Do sekcji `uses` dodajemy moduł `Math` (funkcja `ArcTan2`).
6. Położenie myszy ustalać będzie kąt kamery. Tworzymy metodę zdarzeniową do `OnMouseMove` formy:

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
const PI_2=PI/2.0;
var
  srodek :TPoint;
  dX,dY :Double;
  R,_Phi,_Theta :Double;
begin
srodek.X:=ClientWidth div 2;
srodek.Y:=ClientHeight div 2;

R:=Sqrt(KameraX*KameraX+KameraY*KameraY+KameraZ*KameraZ);

dX:=PI_2*(X-srodek.X)/srodek.X;
dY:=-PI_2*(Y-srodek.Y)/srodek.Y; //minus bo wsp. Y jest skierowana inaczej we wsp.
ekranu i w XYZ z OpenGL
_Theta:=Sqrt(dX*dX+dY*dY); //To jest przeksztalcenie z 2D kart. do 3D sferyczne (model)
_Phi:=ArcTan2(dY,dX);

KameraX:=R*cos(_Phi)*sin(_Theta);
KameraY:=R*sin(_Phi)*sin(_Theta);
KameraZ:=R*cos(_Theta);

Caption:='OpenGL '+glGetString(GL_VERSION)+':  Kamera  odl='+FloatToStr(R)+',
Phi='+IntToStr(Round(180*_Phi/PI))+', Theta='+IntToStr(Round(180*_Theta/PI));
//Caption:='OpenGL '+glGetString(GL_VERSION)+':  Kamera
(X,Y,Z)=('+FloatToStr(KameraX)+', '+FloatToStr(KameraY)+', '+FloatToStr(KameraZ)+')';
Rysuj;
end;
```

7. Rolką ustalać będziemy jej odległość:

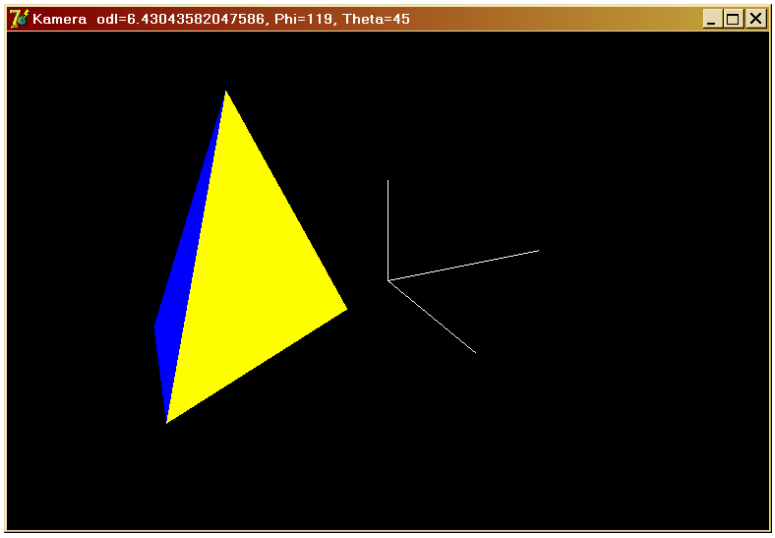
```
procedure TForm1.FormMouseWheel(Sender: TObject; Shift: TShiftState;
```

```
  WheelDelta: Integer; MousePos: TPoint; var Handled: Boolean);
const wsp=0.1;
begin
//proporcjonalna zmiana pozycji wszystkich wsp. kamery
KameraX:=KameraX*(1+Sign(WheelDelta)*wsp);
KameraY:=KameraY*(1+Sign(WheelDelta)*wsp);
KameraZ:=KameraZ*(1+Sign(WheelDelta)*wsp);
Rysuj;
end;
```

8. Klawisz *W* będzie aktywował/dezaktywował ruch kamery:

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
case Key of
  Ord('Q'):  Timer1.Enabled:=not Timer1.Enabled;
  Ord('W'):  RuchKamery:=not RuchKamery;
  VK_ESCAPE: Close;
end;

//dalsza część metody
```

Informacje o sferycznym układzie współrzędnych: http://mathworld.wolfram.com/SphericalCoordinates.html.



Efekty: kolor, materiał, oświetlenie

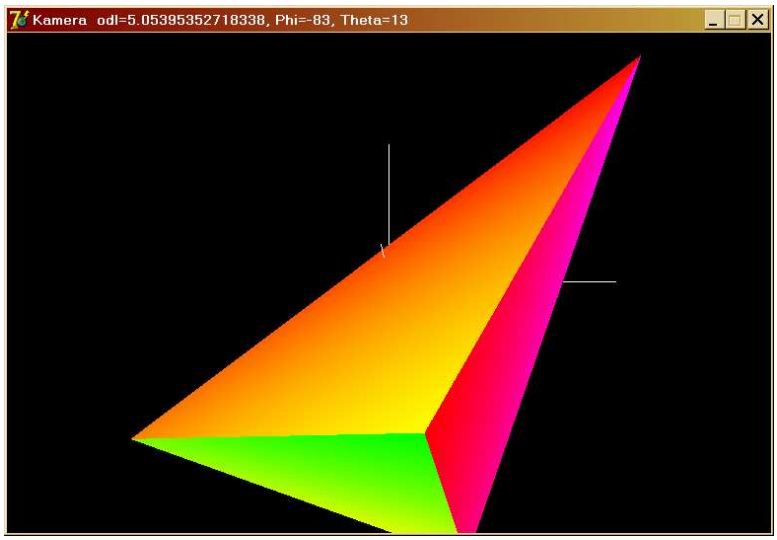
Ćwiczenie 14
Aby do kolorowania ścian użyć cieniowania

1. Modyfikujemy metodę `RysujOstroslup`:

```
procedure TForm1.RysujOstroslup(x0,y0,z0 :Single);
```

```
begin
//Rysowanie trojkatow
glBegin(GL_TRIANGLES);
//glBegin(GL_LINE_LOOP);
//ustalanie trzech wierzcholkow trojkata (werteksow (x,y,z))
//(0,0,???) jest mniej wiecej w srodku ekranu

//sciana przednia
glColor3ub(255,255,0); //zolty
glVertex3f( -x0, -y0, z0); //dolny lewy
glColor3ub(255,128,0);
glVertex3f(x0, -y0, z0); //dolny prawy
glColor3ub(255,0,0);
glVertex3f(0, y0, z0); //gorny

//podstawa
glColor3ub(0,255,0); //zielony
glVertex3f( -x0, -y0, z0); //dolny lewy
glColor3ub(128,255,0);
glVertex3f(x0, -y0, z0); //dolny prawy
glColor3ub(255,255,0);
glVertex3f(0, -y0, 2*z0); //dolny tylny

//tylna lewa
glColor3ub(255,0,0); //czerwony
glVertex3f( -x0, -y0, z0); //dolny lewy
glColor3ub(255,0,128);
glVertex3f(0, -y0, 2*z0); //dolny tylny
glColor3ub(255,0,255);
glVertex3f(0, y0, z0); //gorny

//tylna lewa
glColor3ub(0,0,255); //niebieski
glVertex3f(x0, -y0, z0); //dolny prawy
glColor3ub(0,128,255);
glVertex3f(0, -y0, 2*z0); //dolny tylny
glColor3ub(0,255,255);
glVertex3f(0, y0, z0); //gorny

//koniec rysowania figury
glEnd;
end;
```

2. Oglądamy wynik (zob. rysunek), podziwiamy.
3. Następnie usuwamy lub komentujemy dodatkowe linie (żeby cieniowanie nie interferowało z oświetleniem, który dodany za chwilę).



Cieniowanie można też wyłączyć poleceniem `glShadeModel(GL_FLAT);`. Wówczas do kolorowania trójkąta używany jest kolor ostatniego wierzchołka. Domyślna wartość modelu cieniowania to `GL_SMOOTH`.

Ćwiczenie 15
Aby umożliwić wyłączanie kolorów ścian

1. Definiujemy pole `Kolory`:

```
type
  TForm1 = class(TForm)
    Timer1: TTimer;
    procedure FormCreate(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure FormKeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
    procedure Timer1Timer(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormMouseMove(Sender: TObject; Shift: TShiftState; X,
      Y: Integer);
    procedure FormMouseWheel(Sender: TObject; Shift: TShiftState;
      WheelDelta: Integer; MousePos: TPoint; var Handled: Boolean);
  private
    { Private declarations }
    uchwytDC :HDC; //uchwyt do "display device context (DC)"
    uchwytRC :HGLRC; //uchwyt do "OpenGL rendering context"
    Phi, Theta :Single;
    PozycjaX, PozycjaY, PozycjaZ :Single;
    RuchKamery :Boolean;
    KameraX, KameraY, KameraZ :Single;
    Kolory :Boolean;
    function GL_UstalFormatPikseli(uchwytDC :HDC) :Boolean;
    procedure GL_UstawienieSceny;
    procedure Rysuj;
    procedure RysujOstroslup(x0,y0,z0 :Single);
```

```
    procedure RysujOsie(rozmiar :Single);
    procedure Oswietlenie;
  public
    { Public declarations }
  end;
```

2. W `FormCreate` ustawiamy jego wartość na `True`:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  polozenieMyszy :TPoint;
begin
//biezace okno staje sie oknem OpenGL
uchwytDC:=GetDC(Handle);
GL_UstalFormatPikseli(uchwytDC);
uchwytRC:=wglCreateContext(uchwytDC);
wglMakeCurrent(uchwytDC,uchwytRC);
GL_UstawienieSceny;
Caption:='OpenGL '+glGetString(GL_VERSION);

KameraZ:=10;
RuchKamery:=True;
polozenieMyszy.X:=ClientWidth div 2;
polozenieMyszy.Y:=ClientHeight div 2;
if RuchKamery then Mouse.CursorPos:=ClientToScreen(polozenieMyszy);

Kolory:=True;
end;
```

3. Aby od stanu pola `Kolory` uzależnić stosowanie kolorów do rysowania ścian ostrosłupa:

```
procedure TForm1.RysujOstroslup(x0,y0,z0 :Single);
begin
//Rysowanie trojkatow
glBegin(GL_TRIANGLES);
//glBegin(GL_LINE_LOOP);
//ustalanie trzech wierzcholkow trojkata (werteksow (x,y,z))
//(0,0,???) jest mniej wiecej w srodku ekranu

//sciana przednia
if Kolory then glColor3ub(255,255,0); //zolty
glVertex3f( -x0, -y0, z0); //dolny lewy
glVertex3f(x0, -y0, z0); //dolny prawy
glVertex3f(0, y0, z0); //gorny

//podstawa
if Kolory then glColor3ub(0,255,0); //zielony
glVertex3f( -x0, -y0, z0); //dolny lewy
glVertex3f(x0, -y0, z0); //dolny prawy
glVertex3f(0, -y0, 2*z0); //dolny tylny
```

```
//tylna lewa
if Kolory then glColor3ub(255,0,0); //czerwony
glVertex3f( -x0, -y0, z0); //dolny lewy
glVertex3f(0, -y0, 2*z0); //dolny tylny
glVertex3f(0, y0, z0); //gorny

//tylna lewa
if Kolory then glColor3ub(0,0,255); //niebieski
glVertex3f(x0, -y0, z0); //dolny prawy
glVertex3f(0, -y0, 2*z0); //dolny tylny
glVertex3f(0, y0, z0); //gorny

//koniec rysowania figury
glEnd;
end;
```

4. Aby umożliwiać przełączanie stanu nowego pola przez naciskanie litery *E*:

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
case Key of
  Ord('Q'):  Timer1.Enabled:=not Timer1.Enabled;
  Ord('W'):  RuchKamery:=not RuchKamery;
  Ord('E'):  Kolory:=not Kolory;
  VK_ESCAPE: Close;
end;

//dalsza czesc metody
```

Nie można poprawnie oświetlić przedmiotu, jeżeli nie ustawi się wcześniej odpowiednich własności materiału, z którego jest on wykonany. Niestety kolor, który ustawialiśmy korzystając z funkcji `clColor` nie jest kolorem w sensie fizycznym (nie wynika z własności materiału, a dokładnie z jego własności odbijania i pochłaniania światła). Zobaczymy to po włączeniu silnego oświetlenia tła, które w zamierzeniu ma odtworzyć sytuację sprzed włączenia systemu oświetlenia.

Ćwiczenie 16
Włączenie systemu oświetlenia i ustawianie rozproszonego światła tła (bez określonego źródła i rozświetlającego całą scenę)

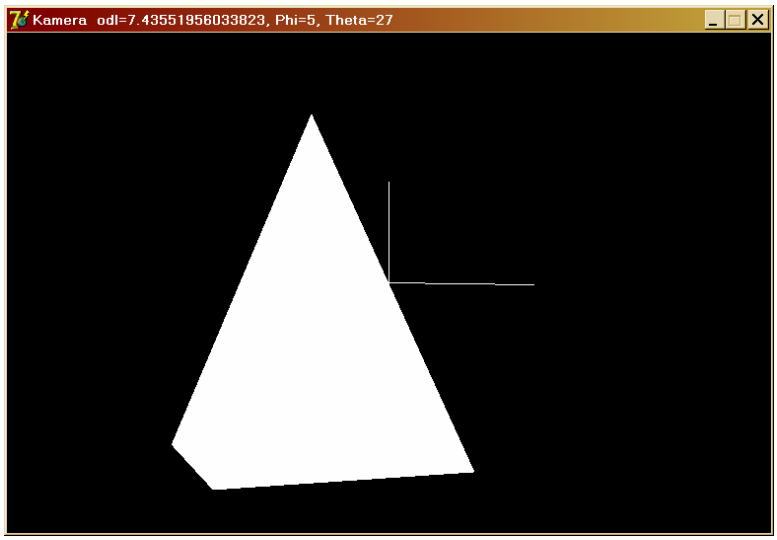
1. Definiujemy prywatną metodę `GL_Oswietlenie` (pamiętajmy o jej deklaracji w definicji klasy `TForm1`). W niej włączamy system oświetlenia i uruchamiamy światło otoczenia (ang. *ambient*):

```
procedure TForm1.GL_Oswietlenie;
const
  kolor_otoczenie :TGLArrayf3 =(255,255,255); //biel
begin
glEnable(GL_LIGHTING); //wlaczenie systemu oswietlania

glLightModelfv(GL_LIGHT_MODEL_AMBIENT,@kolor_otoczenie); //swiatlo tla
```

```
end;
```

2. Metodę tą wywołujemy z metody `GL_UstawienieSceny`. Może być wywołana później, ale gdy zaczniemy rozstawiać światła o określonym kierunku i pozycji należy pamiętać, aby przekształcenia te znalazły się za poleceniami konfigurującymi źródło światła, które chcemy pozostawić nieruchome.

```
procedure TForm1.GL_UstawienieSceny;
begin
//ustawienie punktu projekcji
glMatrixMode(GL_PROJECTION); //macierz projekcji
//left,right,bottom,top,znear,zfar
glFrustum(-0.1, 0.1, -0.1, 0.1, 0.3, 25.0); //mnozenie macierzy przez macierz
perspektywy – ustalanie piramidy frustum
glMatrixMode(GL_MODELVIEW); //powrot do macierzy widoku
glEnable(GL_DEPTH_TEST); //z-buffer aktywny = ukrywanie niewidocznych trojkatow !!!
GL_Oswietlenie;
end;
```



Włączenie systemu oświetlenia `glEnable(GL_LIGHTING);` bez ustalenia źródła światła powoduje, że cała scena staje się ciemna – widzimy jedynie czerń.

Niestety po uruchomieniu programu zobaczymy, że nasz ostrosłup jest cały biały. Dokładnie jest w kolorze zdefiniowanego funkcją `glLightModelfv` światła tła. Jeżeli macierz `kolor` definiowałaby kolor czerwony – ostrosłup byłby czerwony. Aby przywrócić stan sprzed włączeniem oświetlenia musimy mieć dwie rzeczy – białe światło otoczenia i materiał, którego własności rozpraszające są zdefiniowane w sposób odpowiadający wybranym wcześniej kolorom. Domyślne ustawienia materiału są takie, że jednakowo rozpraszają wszystkie składowe światła (tzn. trzy: R, G i B) i w efekcie przyjmuje on kolor światła. Można to jednak zmienić. I to na dwa sposoby. Po pierwsze definiować własności poszczególnych ścian funkcją `glMaterial`. A po drugie nakazać, aby własność materiału była zgodna z barwą nadaną funkcją `glColor`. I to właśnie zrobimy.

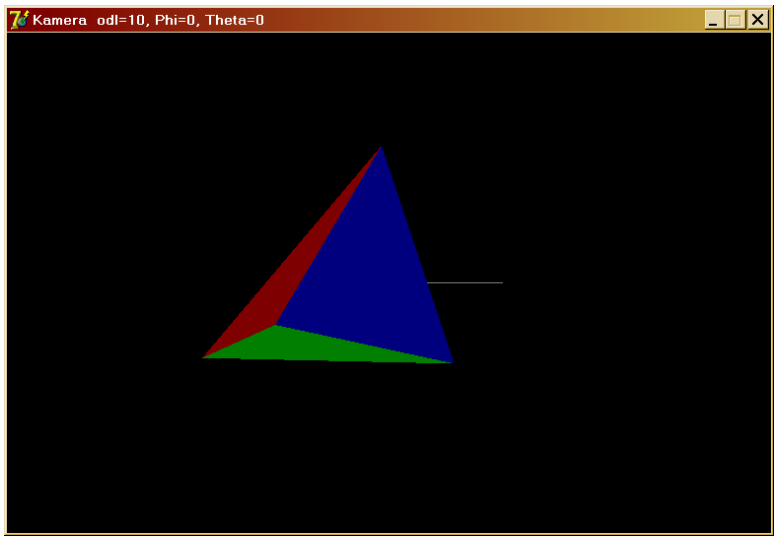
Ćwiczenie 17
Aby uzgodnić kolor „fizyczny” z kolorem ustawianym funkcją glColor

Do metody `GL_Oswietlenia` dodajemy dwa polecenia. Możemy też osłabić źródło światła – po włączeniu systemu `GL_COLOR_MATERIAL` sens mają składowe o wartościach od 0.0 do 1.0.

```
procedure TForm1.Oswietlenie;
const
  kolor_otoczenie :TGLArrayf3 =(0.5,0.5,0.5); //biel
```

```
begin
glEnable(GL_LIGHTING); //wlaczenie systemu oswietlania

glLightModelfv(GL_LIGHT_MODEL_AMBIENT,@kolor_otoczenie); //swiatlo tla

glEnable(GL_COLOR_MATERIAL);
glColorMaterial(GL_FRONT,GL_AMBIENT_AND_DIFFUSE);
end;
```

Przyciemnione światło tła, którego efekt widać na rysunku pozwoli zobaczyć efekt rozproszonego światła, które zaraz włączymy. Gdybyśmy światło otoczenia ustawili na maksimum (wszystkie składowe równe 1.0 lub więcej) – trudniej byłoby to zobaczyć. Podobnie jak żarówka, której światło jest słabo widoczne w ciągu jasnego letniego dnia.



Polecenia te włączają system, który uzgadnia kolor materiału i bieżący kolor ustalany funkcją `glColor`. Zwróćmy uwagę na argumenty polecenia `glColorMaterial`, a szczególnie na drugi. Określa on jakie własności materiału (reakcja na jakie typy światła) mają być identyczne jak bieżący kolor. Mamy do wyboru światło otoczenia (`GL_AMBIENT`, takie właśnie zdefiniowaliśmy), światło rozproszone (`GL_DIFFUSE`, np. mleczna żarówka), światło ukierunkowane (`GL_SPECULAR`, np. reflektor lub silne słońce). W naszym przypadku zażądaliśmy, aby ostrosłup zgodnie ze użytym do jego namalowania kolorem odbijał światło rozproszone i tła. Pierwszy argument w przypadku zdefiniowanego jedynie światła otoczenia nie ma wielkiego znaczenia. Nabierze go, gdy zdefiniujemy światło rozproszone i kierunkowe.

Ćwiczenie 18
Ilu programistów OpenGL potrzeba, aby wkręcić mleczną żarówkę

1. Definiujemy metodę `Swiatlo0` aktywujące źródło światła rozproszonego identyfikowane przez stałą `GL_LIGHT0` (jedno z ośmiu możliwych źródeł światła):

```
procedure TForm1.Swiatlo0;
const
  kolor0_rozproszone :TGLArrayf4 =(0.5,0.5,0.5,1.0);
begin
glLightfv(GL_LIGHT0, GL_DIFFUSE, @kolor0_rozproszone);
glENABLE(GL_LIGHT0);
end;
```

2. Wywołanie tej funkcji umieszczamy w metodzie `GL_Oswietlenie`:

```
procedure TForm1.GL_Oswietlenie;
const
```

```
  kolor_otoczenie :TGLArrayf3 =(0.5,0.5,0.5); //biel
begin
glEnable(GL_LIGHTING); //wlaczenie systemu oswietlania

glLightModelfv(GL_LIGHT_MODEL_AMBIENT,@kolor_otoczenie); //swiatlo tla

glEnable(GL_COLOR_MATERIAL);
glColorMaterial(GL_FRONT,GL_AMBIENT_AND_DIFFUSE);

Swiatlo0;
end;
```

Nie wszystko działa jak należy. Ściany niebieska i czerwona są oświetlone, gdy są skierowane do kamery, ale podstawa staje się właśnie wtedy ciemna. Innymi słowy – nie wszystkie ściany w poprawny sposób identyfikują kierunek do źródła światła.

Ćwiczenie 19
Definiowanie wektorów normalnych do wierzchołków (sic!)

1. Zdefiniowanie wektorów normalnych do żółtej ściany i do podstawy jest proste. Pierwszy ma kierunek –*z*, drugi –*y*:

```
procedure TForm1.RysujOstroslup(x0,y0,z0 :Single);
begin
//Rysowanie trojkatow
glBegin(GL_TRIANGLES);
//glBegin(GL_LINE_LOOP);
//ustalanie trzech wierzcholkow trojkata (werteksow (x,y,z))

//sciana tylnia
if Kolory then glColor3ub(255,255,0); //zolty
glNormal(0,0,-1.0); //w glab
glVertex3f( -x0, -y0, z0); //dolny lewy
glVertex3f(x0, -y0, z0); //dolny prawy
glVertex3f(0, y0, z0); //gorny

//podstawa
if Kolory then glColor3ub(0,255,0); //zielony
glNormal(0,-1.0,0); //do dolu
glVertex3f( -x0, -y0, z0); //dolny lewy
glVertex3f(x0, -y0, z0); //dolny prawy
glVertex3f(0, -y0, 2*z0); //dolny przedni

//dalsza czesc metody
```

Z pozostałymi niestety nie jest tak łatwo. Możemy się jednak posłużyć wiedzą o wektorach i korzystając z iloczynu wektorowego dwóch wektorów napinających trójkąt obliczyć wektor prostopadły do niego. W którą stronę ma być skierowany?

2. Aby wyznaczyć wektory normalne do pozostałych ścian potrzebujemy „aparat matematyczny”, który składać się będzie z kilku funkcji. Umieścimy je w osobnym module. Powinny być w miarę zrozumiałe dla osób, które wiedzą czym jest iloczyn wektorowy:

   a) Z menu *File*, podmenu *New* wybieramy *Unit*.

   b) Naciskamy klawisze *Ctrl+S*, aby zachować nowy moduł do pliku. Wybierzmy dla niego nazwę *GLWektory.pas* (zmieniona zostanie również nazwa modułu widoczna za słowem kluczowym `unit`).

   c) Do modułu dodajemy zbiór funkcji, z których eksportujemy (na razie) tylko jedną:

```
unit GLWektory;

interface

uses OpenGL;

function JednostkowyWektorNormalny(punkt1,punkt2,punkt3 :TGLArrayf3) :TGLArrayf3;

implementation

function Roznica3fv(punkt1,punkt2 :TGLArrayf3) :TGLArrayf3;
var i :Integer;
begin
for i:=0 to 2 do Result[i]:=punkt2[i]-punkt1[i];
end;

function IloczynWektorowy3fv(a,b :TGLArrayf3) :TGLArrayf3;
const x=0;y=1;z=2;
begin
//zmienic na wskazniki
Result[x]:=a[y]*b[z]-a[z]*b[y];
Result[y]:=-(a[x]*b[z]-a[z]*b[x]);
Result[z]:=a[x]*b[y]-a[y]*b[x];
end;

function NormalizujWektor3fv(wektor :TGLArrayf3) :TGLArrayf3;
var
  wsp :Single;
  i :Integer;
begin
wsp:=Sqr(wektor[0])+Sqr(wektor[1])+Sqr(wektor[2]);
wsp:=Sqrt(wsp);
for i:=0 to 2 do Result[i]:=wektor[i]/wsp;
end;

function JednostkowyWektorNormalny(punkt1,punkt2,punkt3 :TGLArrayf3) :TGLArrayf3;
begin
Result:=NormalizujWektor3fv(IloczynWektorowy3fv(Roznica3fv(punkt1,punkt2),Roznica3fv(pun
kt1,punkt3)));
end;
```

```
end.
```

3. Wracamy do pierwotnego modułu *Unit1*.
4. Do jego sekcji `uses` dodajemy nowy moduł *GLWektory*:

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, OpenGL, StdCtrls, ExtCtrls, Math, GLWektory;
```

5. Wreszcie modyfikujemy metodę `RysujOstroslup` zgodnie z poniższym wzorem:

```
procedure TForm1.RysujOstroslup(x0,y0,z0 :Single);
var
  punkt1,punkt2,punkt3 :TGLArrayf3;
  wektor :TGLArrayf3;
begin
//Rysowanie trojkatow
glBegin(GL_TRIANGLES);
//glBegin(GL_LINE_LOOP);
//ustalanie trzech wierzcholkow trojkata (werteksow (x,y,z))

//sciana tylnia
if Kolory then glColor3ub(255,255,0); //zolty
glNormal(0,0,-1.0); //w glab
glVertex3f( -x0, -y0, z0); //dolny lewy
glVertex3f(x0, -y0, z0); //dolny prawy
glVertex3f(0, y0, z0); //gorny

//podstawa
if Kolory then glColor3ub(0,255,0); //zielony
glNormal(0,-1.0,0); //do dolu
glVertex3f( -x0, -y0, z0); //dolny lewy
glVertex3f(x0, -y0, z0); //dolny prawy
glVertex3f(0, -y0, 2*z0); //dolny przedni

//lewa
if Kolory then glColor3ub(255,0,0); //czerwony
punkt1[0]:=-x0; punkt1[1]:=-y0; punkt1[2]:=z0;
punkt2[0]:=0;   punkt2[1]:=-y0; punkt2[2]:=2*z0;
punkt3[0]:=0;   punkt3[1]:=y0;  punkt3[2]:=z0;
wektor:=JednostkowyWektorNormalny(punkt1,punkt2,punkt3);
glNormal3fv(@wektor);
glVertex3fv(@punkt1); //dolny lewy
glVertex3fv(@punkt2); //dolny przedni
glVertex3fv(@punkt3); //gorny

//prawa
if Kolory then glColor3ub(0,0,255); //niebieski
punkt1[0]:=x0;  punkt1[1]:=-y0; punkt1[2]:=z0;
```

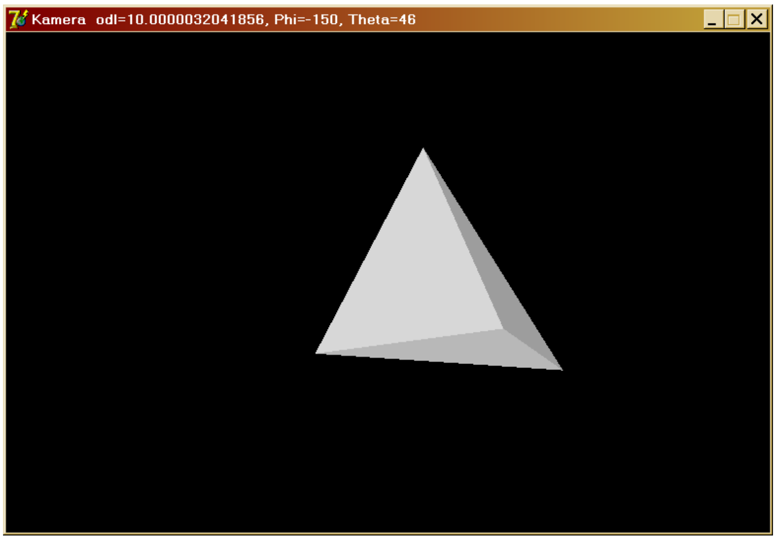
```
punkt2[0]:=0;   punkt2[1]:=-y0; punkt2[2]:=2*z0;
punkt3[0]:=0;   punkt3[1]:=y0;  punkt3[2]:=z0;
//odwrocone - aby nawijanie bylo prawidlowe (przy def. trojkata tez zmiana kolejnosci)
wektor:=JednostkowyWektorNormalny(punkt1,punkt3,punkt2);
glNormal3fv(@wektor);
glVertex3fv(@punkt1); //dolny prawy
glVertex3fv(@punkt3); //gorny
glVertex3fv(@punkt2); //dolny przedni

//koniec rysowania figury
glEnd;
end;
```

Korzystając ze zdefiniowanych w tablicach punktów użyliśmy do określenia worteksów przeciążoną wersję metody, która za argument pobiera macierz trójelementową.

Teraz gdy ostrosłup jest tak obrócony, że płaszczyzna jest prostopadła do kamery, czyli gdy wektor normalny wskazuje źródło światła – płaszczyzna jest najjaśniejsza. Gdy kąt między tym wektorem, a kierunkiem, z którego pada wirtualne promieniowania rośnie – płaszczyzna jest coraz ciemniejsza. To daje wrażenie reakcji na promieniowanie rozproszone. Pamiętajmy, że wymagało to ustawienia odpowiedniej własności materiału, z którego powierzchnia jest wirtualnie wykonana.

Od tego, w którym miejscu umieścimy wywołanie metody `Swiatlo0` zależy, czy obracanie kamerą będzie oznaczało równoczesną zmianę pozycji źródeł światła, czy nie (będą obracane razem z osiami, jeżeli znajdą się za poleceniami obrotu). ?????????????????????? !!

Teraz nawet wyłączenie koloru nie spowoduje, że figura staje się płaska (zob. rysunek).



Wspomnieć o technice uśredniania normalnych!

Dodatkową zaletą zdefiniowania wektorów normalnych jest możliwość zdefiniowania odbłysków imitujących rozbłyski, jakie towarzyszą odbijaniu od powierzchni silnego źródła światła (np. „zajączek”). Własność ta nie musi dotyczyć wszystkich powierzchni. W naszym przypadku będzie zastosowana jedynie do żółtej powierzchni tylnej ściany.

Ćwiczenie 20
Żółta ściana będzie zachowywała się jak tafla szkła pod względem odbijania światła (rozbłysk).

```
procedure TForm1.RysujOstroslup(x0,y0,z0 :Single);
const
  wsp_odbicia_szklo :TGLArrayf4 =(1.0,1.0,1.0,1.0);
  wsp_odbicia_matowy :TGLArrayf4 =(0.0,0.0,0.0,1.0);
var
  punkt1,punkt2,punkt3 :TGLArrayf3;
  wektor :TGLArrayf3;
begin
//Rysowanie trojkatow
glBegin(GL_TRIANGLES);
//glBegin(GL_LINE_LOOP);
//ustalanie trzech wierzcholkow trojkata (werteksow (x,y,z))

//sciana tylnia
if Kolory then glColor3ub(255,255,0); //zolty
//lsniacy material
glMaterialfv(GL_FRONT,GL_SPECULAR,@wsp_odbicia_szklo);
glMateriali(GL_FRONT,GL_SHININESS,100);
glNormal(0,0,-1.0); //w glab
glVertex3f( -x0, -y0, z0); //dolny lewy
glVertex3f(x0, -y0, z0); //dolny prawy
glVertex3f(0, y0, z0); //gorny
//powrot do oryginalnych
glMaterialfv(GL_FRONT,GL_SPECULAR,@wsp_odbicia_matowy);
glMateriali(GL_FRONT,GL_SHININESS,0);

//dalsza czesc metody
```

W efekcie ściana, która ustawiona jest tak, że promienie biegnące ze źródła światła odbite są dokładnie w kierunku kamery, robi się coraz bielsza (w zależności od niewielkiego odchylenia).

Ustawienia te można dodatkowo uatrakcyjnić (lub zepsuć) zmieniając własność GL_SPECULAR źródła światła tzn. nadając mu charakter reflektora.

Ćwiczenie 21
Ustawianie reflektora i montowanie włączników światła

1. Zdefiniujmy metodę `Swiatlo1`, w której zdefiniujemy drugie źródło światła (pamiętajmy o deklaracji tej metody w sekcji `private` klasy `TForm1`).

```
procedure TForm1.Swiatlo1;
const
  kolor1_rozproszone :TGLArrayf4 =(0.3,0.3,0.3,1.0);
  kolor1_reflektora :TGLArrayf4 =(1.0,1.0,1.0,1.0);
  pozycja :TGLArrayf4 =(0.0,-10.0,0.0,1.0);
  szerokosc_wiazki = 60.0; //w stopniach
begin
glLightfv(GL_LIGHT1, GL_POSITION, @pozycja);
```

```
glLightfv(GL_LIGHT1, GL_DIFFUSE, @kolor1_rozproszone);
glLightfv(GL_LIGHT1, GL_SPECULAR, @kolor1_reflektora);
glLightf(GL_LIGHT1, GL_SPOT_CUTOFF, szerokosc_wiazki);
glENABLE(GL_LIGHT1);
end;
```

2. Do metody FormKeyDown dodajemy:

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
case Key of
  Ord('Q'):  Timer1.Enabled:=not Timer1.Enabled;
  Ord('W'):  RuchKamery:=not RuchKamery;
  Ord('E'):  Kolory:=not Kolory;
  Ord('1'):  if glIsEnabled(GL_LIGHT0) then glDisable(GL_LIGHT0) else
glEnable(GL_LIGHT0);
  Ord('2'):  if glIsEnabled(GL_LIGHT1) then glDisable(GL_LIGHT1) else
glEnable(GL_LIGHT1);
  VK_ESCAPE: Close;
end;
//obroty
```

3. Kompilujemy i uruchamiamy aplikację.

Nowe źródło światła składa się z silnej części generującej światło odbite na obiektach i trzykrotnie słabszej części generującej światło rozproszone. Reflektor ustawiliśmy „na dole” tj. na ujemnej połowie osi Y w pozycji (0,-10,0) i skierowaliśmy do góry. Szerokość smugi światła to aż 120 stopni (maksymalnie odchylony promień biegnie pod kątem 60 stopni od kierunku głównego).

Ponieważ podstawowe źródło światła (`GL_LIGHT1`) jest dość silne, zgaśmy je (klawisz *1*) pozostawiając jedynie reflektor i oświetlenie tła. Sprawdźmy czy i przy jakim ustawieniu uda nam się zrobić „zajączka” żółtą powierzchnią. Następnie wyłączmy kolory (klawisz *E*) i spróbujmy włączać i wyłączać źródła światła (klawisze *1* i *2*).

W projekcie dodanym dostępnym w źródłach dodane zostały także kolejne różnobarwne źródła światła (klawisze *3* i *4*).

Realizm obrazu podniosłyby cienie figury (oczywiście niezbędne byłoby dodanie jakiegoś podłoża), ale póki co OpenGL nie obejmuje takich możliwości. Należałoby samodzielnie zdefiniować odpowiednie funkcje (zob. [4]).

Mieszanie kolorów

Ćwiczenie 22
Przeźroczystość

1. Definiujemy prywatne pole typu `Boolean` klasy `TForm1` o nazwie `CzyPrzezroczysty`, którego zadaniem będzie kontrola korzystania z efektu przezroczystości.
2. Włączamy system mieszania kolorów i ustalamy równanie mieszania na:
   kolor docelowy * kanał alpha docelowego + kolor źródłowy * (1 – alpha koloru źródłowego)

```
procedure TForm1.GL_UstawienieSceny;
begin
//ustawienie punktu projekcji
glMatrixMode(GL_PROJECTION); //macierz projekcji
```

```
//left,right,bottom,top,znear,zfar
glFrustum(-0.1, 0.1, -0.1, 0.1, 0.3, 25.0); //mnozenie macierzy przez macierz
perspektywy
glMatrixMode(GL_MODELVIEW); //powrot do macierzy widoku
glEnable(GL_DEPTH_TEST); //z-buffer aktywny = ukrywanie niewidocznych trojkatow !!!
GL_Oswietlenie;
//przezroczystosc
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
end;
```

3. Następnie modyfikujemy ustawienia materiału czerwonej ściany ostrosłupa, a rysowanie jej samej przenosimy na koniec:

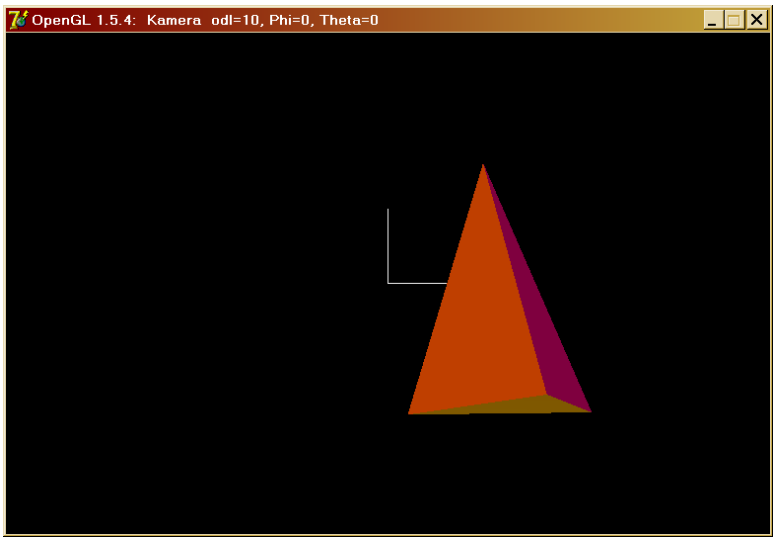
```
procedure TForm1.RysujOstroslup(x0,y0,z0 :Single);
const
  wsp_odbicia_szklo :TGLArrayf4 =(1.0,1.0,1.0,1.0);
  wsp_odbicia_matowy :TGLArrayf4 =(0.0,0.0,0.0,1.0);
var
  punkt1,punkt2,punkt3 :TGLArrayf3;
  wektor :TGLArrayf3;
begin
//Rysowanie trojkatow
glBegin(GL_TRIANGLES);
//glBegin(GL_LINE_LOOP);
//ustalanie trzech wierzcholkow trojkata (werteksow (x,y,z))

//sciana tylnia
if Kolory then glColor3ub(255,255,0); //zolty polprzezroczysty
//lsniacy material
glMaterialfv(GL_FRONT,GL_SPECULAR,@wsp_odbicia_szklo);
glMateriali(GL_FRONT,GL_SHININESS,100);
glNormal(0,0,-1.0); //w glab
glVertex3f( -x0, -y0, z0); //dolny lewy
glVertex3f(x0, -y0, z0); //dolny prawy
glVertex3f(0, y0, z0); //gorny
//powrot do oryginalnych
glMaterialfv(GL_FRONT,GL_SPECULAR,@wsp_odbicia_matowy);
glMateriali(GL_FRONT,GL_SHININESS,0);

//podstawa
if Kolory then glColor3ub(0,255,0); //zielony
glNormal(0,-1.0,0); //do dolu
glVertex3f( -x0, -y0, z0); //dolny lewy
glVertex3f(x0, -y0, z0); //dolny prawy
glVertex3f(0, -y0, 2*z0); //dolny przedni

//lewa
if Kolory then glColor3ub(255,0,0); //czerwony
```

```
punkt1[0]:=-x0; punkt1[1]:=-y0; punkt1[2]:=z0;
punkt2[0]:=0;   punkt2[1]:=-y0; punkt2[2]:=2*z0;
punkt3[0]:=0;   punkt3[1]:=y0;  punkt3[2]:=z0;
wektor:=JednostkowyWektorNormalny(punkt1,punkt2,punkt3);
glNormal3fv(@wektor);
glVertex3fv(@punkt1); //glVertex3f(-x0, -y0, z0); //dolny lewy
glVertex3fv(@punkt2); //glVertex3f(0, -y0, 2*z0); //dolny przedni
glVertex3fv(@punkt3); //glVertex3f(0, y0, z0);    //gorny

//prawa
if Kolory then glColor3ub(0,0,255); //niebieski
punkt1[0]:=x0;  punkt1[1]:=-y0; punkt1[2]:=z0;
punkt2[0]:=0;   punkt2[1]:=-y0; punkt2[2]:=2*z0;
punkt3[0]:=0;   punkt3[1]:=y0;  punkt3[2]:=z0;
//odwrocone - aby nawijanie bylo prawidlowe (przy def. trojkata tez zmiana kolejnosci)
wektor:=JednostkowyWektorNormalny(punkt1,punkt3,punkt2);
glNormal3fv(@wektor);
glVertex3fv(@punkt1); //glVertex3f(x0, -y0, z0); //dolny prawy
glVertex3fv(@punkt3); //glVertex3f(0, y0, z0); //gorny
glVertex3fv(@punkt2); //glVertex3f(0, -y0, 2*z0); //dolny przedni

//lewa
if Kolory then glColor4ub(255,0,0,128) //czerwony polprzezroczysty
          else glColor4ub(255,255,255,128);
punkt1[0]:=-x0; punkt1[1]:=-y0; punkt1[2]:=z0;
punkt2[0]:=0;   punkt2[1]:=-y0; punkt2[2]:=2*z0;
punkt3[0]:=0;   punkt3[1]:=y0;  punkt3[2]:=z0;
wektor:=JednostkowyWektorNormalny(punkt1,punkt2,punkt3);
glNormal3fv(@wektor);
glVertex3fv(@punkt1); //glVertex3f(-x0, -y0, z0); //dolny lewy
glVertex3fv(@punkt2); //glVertex3f(0, -y0, 2*z0); //dolny przedni
glVertex3fv(@punkt3); //glVertex3f(0, y0, z0);    //gorny

//koniec rysowania figury
glEnd;
end;
```

9. Do kontroli stosowania efektu przezroczystości wykorzystamy klawisz *R*:

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
case Key of
  Ord('Q'):  Timer1.Enabled:=not Timer1.Enabled;
  Ord('W'):  RuchKamery:=not RuchKamery;
  Ord('E'):  Kolory:=not Kolory;
  Ord('R'):  if glIsEnabled(GL_BLEND) then glDisable(GL_BLEND) else glEnable(GL_BLEND);
  Ord('1'):  if glIsEnabled(GL_LIGHT0) then glDisable(GL_LIGHT0) else
glEnable(GL_LIGHT0);
```

```
  Ord('2'):  if glIsEnabled(GL_LIGHT1) then glDisable(GL_LIGHT1) else
glEnable(GL_LIGHT1);
  VK_ESCAPE: Close;
end;

//dalsza czesc metody
```



Teoretycznie rzecz biorąc zamiast definiować pole `CzyPrzezroczysty` moglibyśmy użyć metody `glIsEnabled(GL_BLEND)` zwracającą wartość typu `TGLBoolean`, ale ta funkcja (podobnie jak `glGetBoolean(GL_BLEND,@zmienna)`) nie zwracają poprawnej wartości.

Przeniesienie rysowania elementów przezroczystych na koniec jest bardzo ważne. Bez tego podczas rysowania ściany żółtej do jej koloru dodany zostałby kolor tła (oraz narysowane wcześniej osie). Natomiast rysowane później ściany mogłyby nie być narysowane ze względu na testowanie głębi i unikania malowania niewidocznych płaszczyzn.

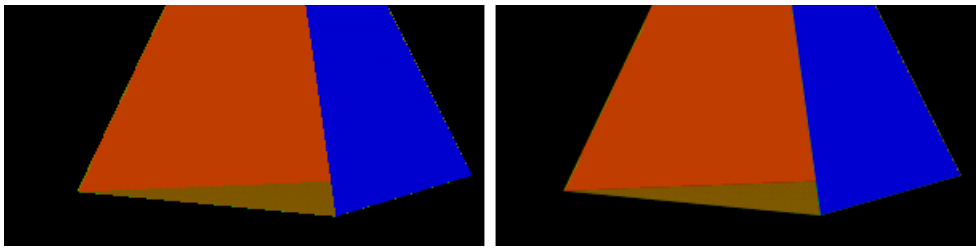
Przezroczystość powoduje osłabienie niektórych efektów związanych z oświetleniem np. rozbłysków światła odbitego.

Ćwiczenie 23
Antialiasing

Do metody `FormKeyDown` dodajmy polecenia inicjujące mechanizm antialiasingu (wygładzanie linii = unikanie schodków na krawędziach):

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
case Key of
  Ord('Q'):  Timer1.Enabled:=not Timer1.Enabled;
  Ord('W'):  RuchKamery:=not RuchKamery;
  Ord('E'):  Kolory:=not Kolory;
  Ord('R'):  if glIsEnabled(GL_BLEND) then glDisable(GL_BLEND) else glEnable(GL_BLEND);
  Ord('T'):
    if glIsEnabled(GL_POINT_SMOOTH) then
      begin
      glDisable(GL_POINT_SMOOTH);
```

```
        glDisable(GL_LINE_SMOOTH);
        glDisable(GL_POLYGON_SMOOTH);
        end
        else
        begin
        glEnable(GL_POINT_SMOOTH);
        glEnable(GL_LINE_SMOOTH);
        glEnable(GL_POLYGON_SMOOTH);
        end;
  Ord('1'):  if glIsEnabled(GL_LIGHT0) then glDisable(GL_LIGHT0) else
glEnable(GL_LIGHT0);
  Ord('2'):  if glIsEnabled(GL_LIGHT1) then glDisable(GL_LIGHT1) else
glEnable(GL_LIGHT1);
  VK_ESCAPE: Close;
end;

//dalsza czesc metody
```



Problem widoczny jest najbardziej w starszych wersjach OpenGL (np. 1.0 i 1.1). Antialiasing realizowany jest on różnie w zależności od wersji OpenGL i mocy komputera. Można zasugerować czy użyty algorytm antyaliasingu ma być zoptymalizowany ze względu na końcowy efekt (wówczas powinniśmy użyć polecenia `glHint(GL_LINE_SMOOTH_HINT,GL_NICEST);`), czy na szybkość działania (wówczas drugim argumentem powinno być `GL_FASTEST`). Aby przywrócić wartość domyślną należy użyć stałej `GL_DONT_CARE`.

Kolejnym popularnym efektem wykorzystującym mieszanie kolorów jest **mgła**.

Biblioteka GLU

Biblioteka GLU zawiera zbiór funkcji wyższego poziomu.

Kwadryka = zbiór, który można zmapować równaniem kwadratowym
(zob. http://mathworld.wolfram.com/Quadric.html).

Zadajemy powierzchnie nie za pomocą zbioru werteksów, ale (prawie) równaniem.

Wcześniej użyliśmy funkcji GLU: `gluLookAt`.

Ćwiczenie 24
Definiujemy kwadrykę i rysujemy sferę

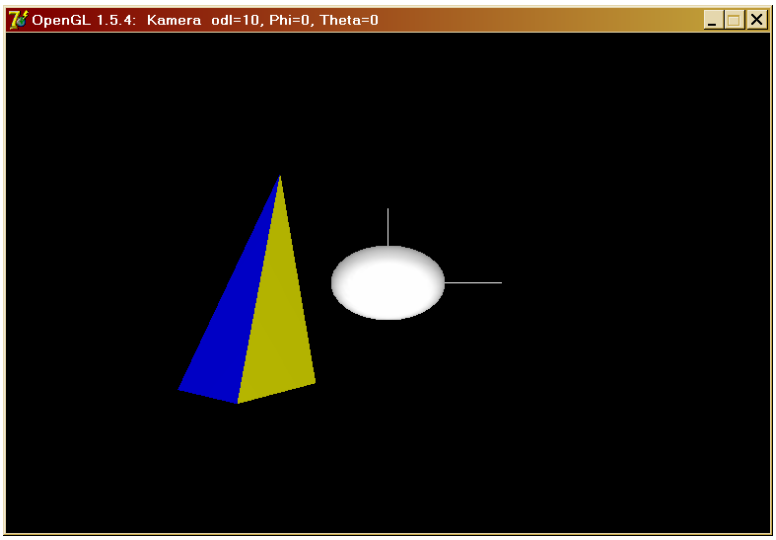
1. Tworzymy nową metodę o nazwie `RysujGLU` (jej deklarację umieszczany w sekcji `private` klasy `TForm1`) zgodnie ze wzorem:

```
procedure TForm1.RysujGLU(rozmiar :Single);
var kwadryka :GLUQuadricObj;
begin
glColor4ub(255,255,255,255);
kwadryka:=gluNewQuadric; //tworzenie obiektu kwadryki
```

```
gluQuadricDrawStyle(kwadryka,GLU_FILL); //GLU_LINE, GLU_POINT, GLU_SILHOUETTE, GLU_FILL
(domyslne)
gluSphere(kwadryka,rozmiar,30,30); //rysowanie
gluDeleteQuadric(kwadryka); //usuwanie obiektu
end;
```

2. Wywołanie metody `RysujGLU` dodajemy do metody `Rysuj`.

```
procedure TForm1.Rysuj;
const x0=1.0; y0=1.5; z0=1.0;
begin
//Przygotowanie bufora
glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
glLoadIdentity; //biezaca macierz = I
//glTranslatef(0.0, 0.0, -10.0); //odsuniecie calosci o 10

gluLookAt(KameraX,KameraY,KameraZ,  //polozenie oka
          0,0,0,  //polozenie srodka ukladu wsp.
          0,1,0); //kierunek "do gory"

RysujOsie(x0);

RysujGLU(x0/2);

//obroty
glRotatef(Phi, 0.0, 1.0, 0.0); //wokol OY
glRotatef(Theta, 1.0, 0.0, 0.0); //wokol OX

//przesuniecia
glTranslatef(PozycjaX,PozycjaY,PozycjaZ);

RysujOstroslup(x0,y0,z0);

//Z bufora na ekran
SwapBuffers(wglGetCurrentDC);
end;
```

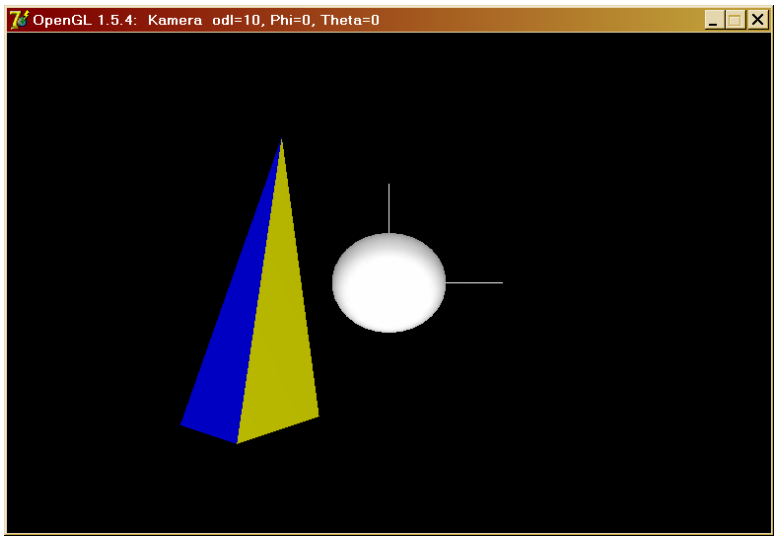


Niemiłym zaskoczeniem może być fakt, że sfera okaże się nie być sferą, a elipsoidą. To świadczy o niezachowaniu proporcji względem okna (lub view port) przy definiowaniu frustum. Możemy to łatwo poprawić.

Ćwiczenie 25
Poprawienie geometrii frustum

Modyfikujemy metodę `GL_UstawienieSceny`:

```
procedure TForm1.GL_UstawienieSceny;
const bialaMgla :TGLArrayf4 = (0.5, 0.5, 0.5, 0.5);
begin
//ustawienie punktu projekcji
glMatrixMode(GL_PROJECTION); //macierz projekcji
//left,right,bottom,top,znear,zfar
glFrustum(-0.1, 0.1, -0.075, 0.075, 0.3, 25.0); //mnozenie macierzy przez macierz
perspektywy
glMatrixMode(GL_MODELVIEW); //powrot do macierzy widoku
glEnable(GL_DEPTH_TEST); //z-buffer aktywny = ukrywanie niewidocznych trojkatow !!!
GL_Oswietlenie;
//przezroczystosc
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
//antialiasing
{glEnable(GL_POINT_SMOOTH);
glEnable(GL_LINE_SMOOTH);
glEnable(GL_POLYGON_SMOOTH);}
end;
```



Ćwiczenie 26
Teksturowanie kwadryki

1. Do sekcji `uses` dodajemy moduł *Jpeg*.
2. Do klasy `TForm1` dodajemy prywatne pola i deklarację metody:

```
private
    { Private declarations }
    uchwytDC :HDC; //uchwyt do "display device context (DC)"
    uchwytRC :HGLRC; //uchwyt do "OpenGL rendering context"
    Phi, Theta :Single;
    PozycjaX, PozycjaY, PozycjaZ :Single;
    RuchKamery :Boolean;
    KameraX, KameraY, KameraZ :Single;
    Kolory :Boolean;
    Sfera, SferaTekstura :Boolean;
    Tekstura :array of Longword;
    TeksturaSzer,TeksturaWys :Integer;
    function GL_UstalFormatPikseli(uchwytDC :HDC) :Boolean;
    procedure GL_UstawienieSceny;
    procedure Rysuj;
    procedure RysujOstroslup(x0,y0,z0 :Single);
    procedure RysujOsie(rozmiar :Single);
    procedure GL_Oswietlenie;
    procedure Swiatlo0;
    procedure Swiatlo1;
    procedure Swiatlo2;
    procedure Swiatlo3;
    procedure RysujGLU(rozmiar :Single);
    procedure PrzygotujTeksture;
```

3. Definiujemy metodę `PrzygotujTeksture`:

```
procedure TForm1.PrzygotujTeksture;
```

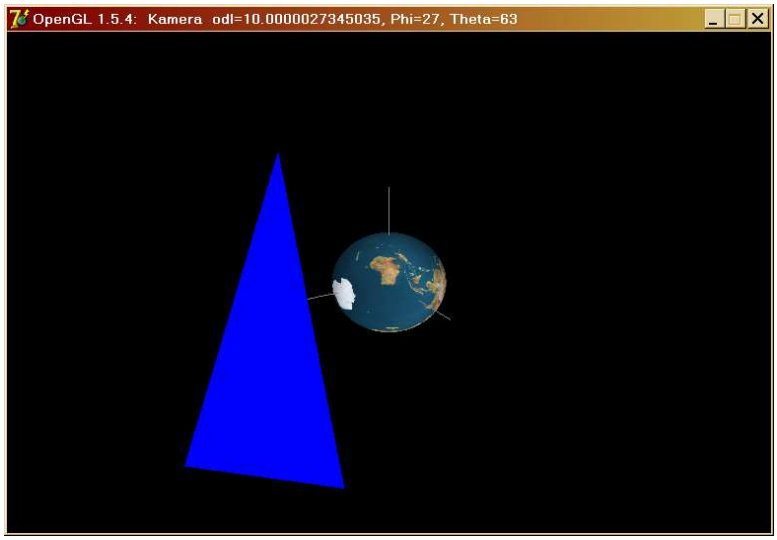
```
const PlikTekstury='tekstura.jpg';
var
  obrazJPEG :TJPEGImage;
  obrazBMP :TBitmap;
  ih,iw :Integer;
  linia :^Longword;
  c :Longword;
begin
//ladowanie tekstury z pliku
obrazJPEG:=TJPEGImage.Create;
try
  obrazJPEG.LoadFromFile(PlikTekstury);
except
  ShowMessage('Brak pliku tekstury');
  SferaTekstura:=False;
  Exit;
end;
obrazBMP:=TBitmap.Create;
obrazBMP.PixelFormat:=pf32bit;
obrazBMP.Width:=obrazJPEG.Width;
obrazBMP.Height:=obrazJPEG.Height;
obrazBMP.Canvas.Draw(0,0,obrazJPEG);
obrazJPEG.Free;

//kopiowanie do dynamicznej tablicy RGBA
TeksturaSzer:=obrazBMP.Width;
TeksturaWys:=obrazBMP.Height;
SetLength(Tekstura,TeksturaSzer*TeksturaWys);
for ih:=0 to TeksturaWys-1 do
  begin
  linia:=obrazBMP.ScanLine[ih];
  for iw:=0 to TeksturaSzer-1 do
    begin
    c:=linia^ and $FFFFFF;
    Tekstura[iw+(ih*TeksturaSzer)]:=(((c and $FF) shl 16)+(c shr 16)+(c and $FF00)) or
$FF000000;  //podzial na 4 kanaly: RGBA
    Inc(linia);
    end;
  end;
obrazBMP.Free;
end;
```

4. Modyfikujemy metodę `RysujGLU`:

```
procedure TForm1.RysujGLU(rozmiar :Single);
var
  kwadryka :GLUQuadricObj;
begin
glColor4ub(255,255,255,255);
```

```
kwadryka:=gluNewQuadric; //tworzenie obiektu kwadryki
gluQuadricDrawStyle(kwadryka,GLU_FILL); //GLU_LINE, GLU_POINT, GLU_SILHOUETTE, GLU_FILL
(domyslne)


if SferaTekstura then
  begin
  gluQuadricTexture(kwadryka,GL_TRUE);
  glEnable(GL_TEXTURE_2D);
  if Length(Tekstura)=0 then PrzygotujTeksture; //wywolywane tylko raz
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  gluBuild2DMipmaps(GL_TEXTURE_2D, 3, TeksturaSzer, TeksturaWys, GL_RGBA,
GL_UNSIGNED_BYTE, Addr(Tekstura[0]));
  end;


gluSphere(kwadryka,rozmiar,30,30); //rysowanie
gluDeleteQuadric(kwadryka); //usuwanie obiektu


if SferaTekstura then glDisable(GL_TEXTURE_2D);
end;
```

5. Ze strony http://www.oera.net/How2/TextureMaps2.htm pobieramy obraz JPEG tekstury i zapisujemy go w katalogu projektu pod nazwą *tekstura.jpg*.
6. Skompilować i uruchomić projekt.



Zagadnienia, które omówione zostaną w części II

- efekt rozmycia w trakcie ruchu i inne efekty
- uśrednianie normalnych
- biblioteka GLUT
- napisy (bez i z GLUT)
- krzywe i powierzchnie
- wczytywanie obiektów z plików