Programowanie Grafiki Komputerowej
Laboratorium 1

OpenGL – rysowanie obiektów 2D

Opracowanie:
Krzysztof Skabek
Politechnika Krakowska, 2012

Opracowano na podstawie podręcznika:
Angel: Interactive Computer Graphics, Addison-Wesley 2009

Cel ćwiczenia:

Zapoznanie się z podstawowymi strukturami oraz metodami biblioteki OpenGL oraz mechanizmami biblioteki Glut.

Zagadnienia szczegółowe:

1. Konfiguracja bibliotek OpenGL i Glut w środowisku Visual Studio 2010
2. Przykład 1: Budowa prostego okna aplikacji
3. Przykład 2: Rysowanie prostych figur geometrycznych
4. Przykład 3: Rysowanie figur geometrycznych w przestrzeni
5. Zadanie 1: Zapoznanie się z parametrami inicjalizacyjnymi biblioteki OpenGL oraz metodami rysowania prymitywów
6. Zadanie 2: Składanie prostych figur geometrycznych
7. Zadanie 3: Rysowanie figur geometrycznych w przestrzeni

Konfiguracja bibliotek OpenGL i Glut w środowisku Visual Studio 2010

Do konfiguracji bibliotek OpenGL i Glut w środowisku C++ Visual Studio 2010 należy posłużyć się plikami dostępnymi w załączonym repozytorium:

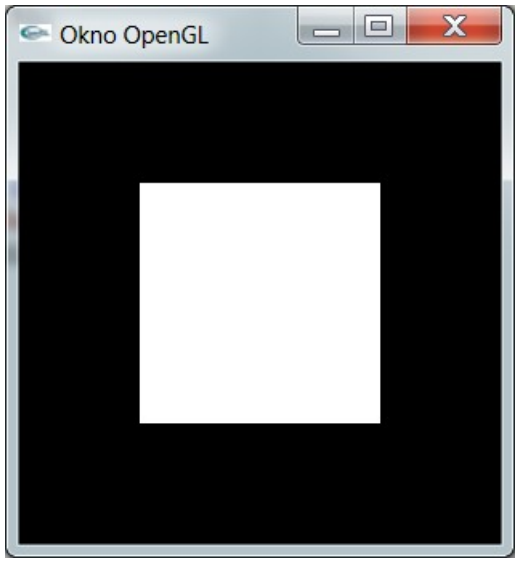
- **glut.h** - kopiujemy do katalogu includów (np. `C:\Program Files\Microsoft Visual Studio 10.0\VC\include\GL`)
- **glut32.lib** - kopiujemy do katalogu libów (np. `C:\Program Files\Microsoft Visual Studio 10.0\VC\lib`)
- **glut32.dll** - kopiujemy do katalogu systemowego (`C:\Windows\System32`)

Po rozpakowaniu, kopiujemy plik – glut32.dll do

W środowisku przy tworzeniu nowego projektu należy wybrać opcję aplikacja konsolowa.

Przykład 1: Budowa prostego okna aplikacji

Generowanie najprostszego okna aplikacji z wykorzystaniem bibliotek OpenGL i Glut.

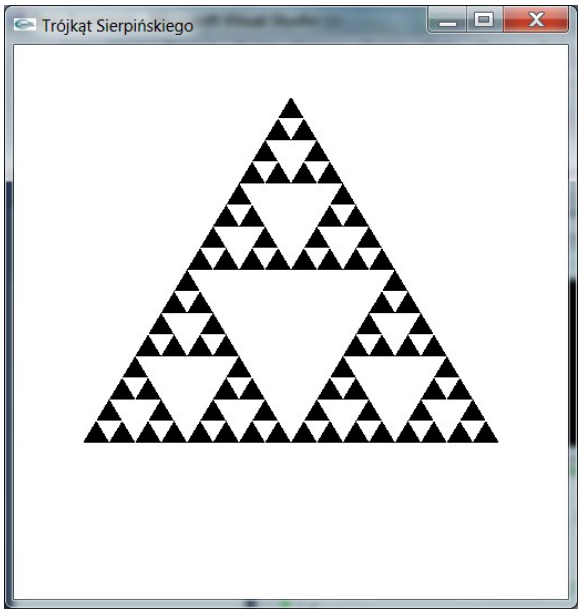


Kod źródłowy programu:

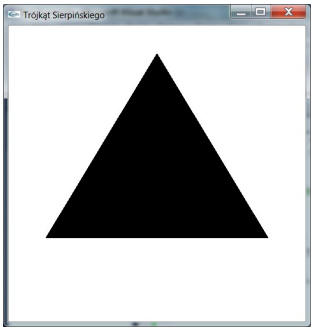
```
#include <GL/glut.h>

void mydisplay(){
glClear(GL_COLOR_BUFFER_BIT); //czyszczenie bufora obrazu
glBegin(GL_POLYGON);
glVertex2f(-0.5, -0.5);
glVertex2f(-0.5, 0.5);
glVertex2f(0.5, 0.5);
glVertex2f(0.5, -0.5);
glEnd();
glFlush(); //wyświetlenie bufora obrazu
}

int main(int argc, char** argv){
glutCreateWindow("Okno OpenGL"); //tworzenie okna aplikacji
glutDisplayFunc(mydisplay); //definiowanie funkcji callback do wyświetlania zawartości okna
glutMainLoop(); //główna pętla aplikacji
}
```

Przykład 2: Składanie prostych figur geometrycznych (trójkąt sierpińskiego)

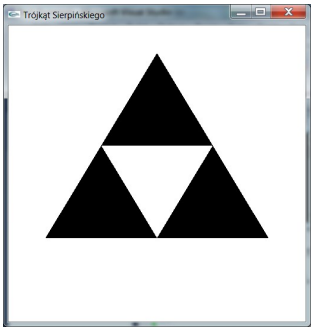
Generowanie fraktalu trójkąt Sierpińskiego za pomocą prostych figur geometrycznych GL_TRIANGLES.



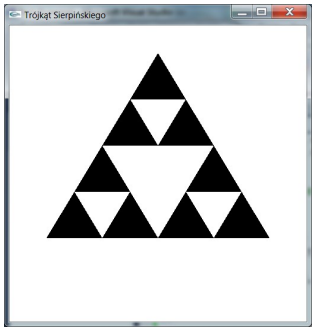
Konstrukcja fraktalu trójkąt Sierpińskiego (procedura rekurencyjna):



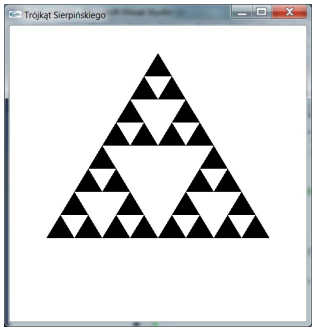
n=0
trójkąt równoboczny



n=1
wycięcie trójkąta wyznaczonego środkami



n=2
wycięcie trójkątów z pozostałych trójkątów



n=3

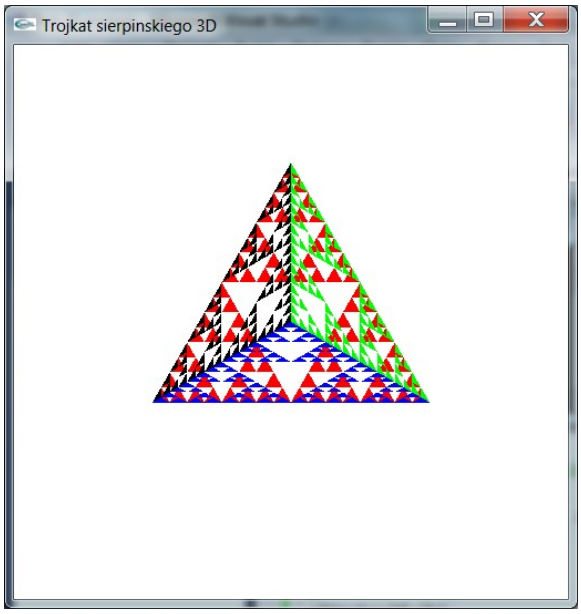
Kod źródłowy programu:

```
#include <GL/glut.h>

/* dane inicjalizacyjne trójkąta */
GLfloat v[3][2]={{-1.5, -0.87},
{1.5, -0.87}, {0.0, 1.625}};

int n; /* liczba kroków rekursywnych */

void triangle( GLfloat *a, GLfloat *b,
GLfloat *c)

/* wyświetlenie trójkąta */
{
glVertex2fv(a);
glVertex2fv(b);
glVertex2fv(c);
}
```

```
void divide_triangle(GLfloat *a, GLfloat *b, GLfloat *c,
int m)
{
/* podział trójkąta */
GLfloat v0[2], v1[2], v2[2];
int j;
if(m>0)
{
for(j=0; j<2; j++) v0[j]=(a[j]+b[j])/2;
for(j=0; j<2; j++) v1[j]=(a[j]+c[j])/2;
for(j=0; j<2; j++) v2[j]=(b[j]+c[j])/2;
divide_triangle(a, v0, v1, m-1);
divide_triangle(c, v1, v2, m-1);
divide_triangle(b, v2, v0, m-1);
}
else(triangle(a,b,c));
/* wyświetlenie pojedyńczego trójkąta */
}

void display()
{ /*funkcja wyświetlania*/
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_TRIANGLES);
divide_triangle(v[0], v[1], v[2], n);
glEnd();
glFlush();
}

void myinit()
{ /*funkcja inicjująca mechanizmy OpenGL */
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(-2.0, 2.0, -2.0, 2.0);
glMatrixMode(GL_MODELVIEW);
glClearColor (1.0, 1.0, 1.0,1.0);
glColor3f(0.0,0.0,0.0);
}

int main(int argc, char **argv)
{
n=4;
glutInit(&argc, argv); //konfuguracja mechanizmów bibl. Glut
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB); //inicjowanie trybu wyświetlania
glutInitWindowSize(500, 500); //inicjowanie rozmiaru okna
glutCreateWindow("Trójkąt Sierpińskiego"); //tworzenie okna aplikacji
glutDisplayFunc(display); //wskazanie funkcji wyświetlania
myinit(); //inicjowanie mechanizmów OpenGL
glutMainLoop();
}
```

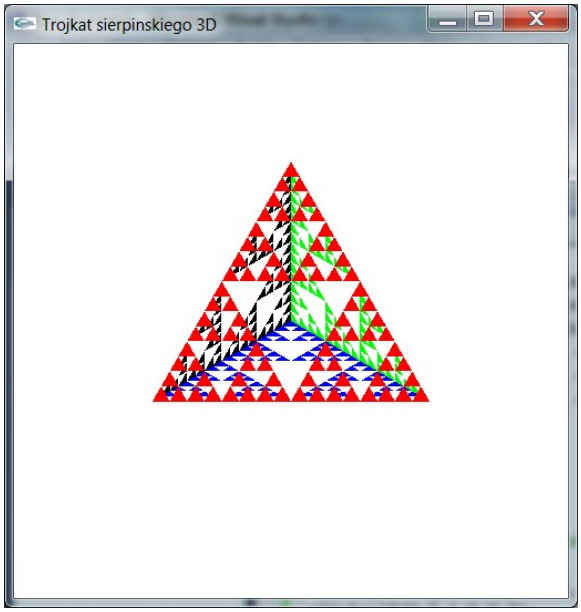
Przykład 3: Rysowanie prostych figur geometrycznych w przestrzeni

Generowanie przestrzennego trójkąta Sierpińskiego, gdzie na trzech ścianach czworościanu foremnego naniesiono trójkąty Sierpińskiego.



Generowanie figur z wykorzystaniem mechanizmu przesłonięć można uzyskać dodając następujące polecenia:

- `glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH)` - dodane do funkcji main, inicjalizacja trybu bufora głębokości
- `glEnable(GL_DEPTH_TEST);` - dodane do fnkcji inicjacji OpenGL, inicjalizacja bufora głębokości
- `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)` - dodane do funkcji wyświetlania, czyszczenie buforów koloru i głębokości przy wyświetlaniu



Kod źródłowy programu:

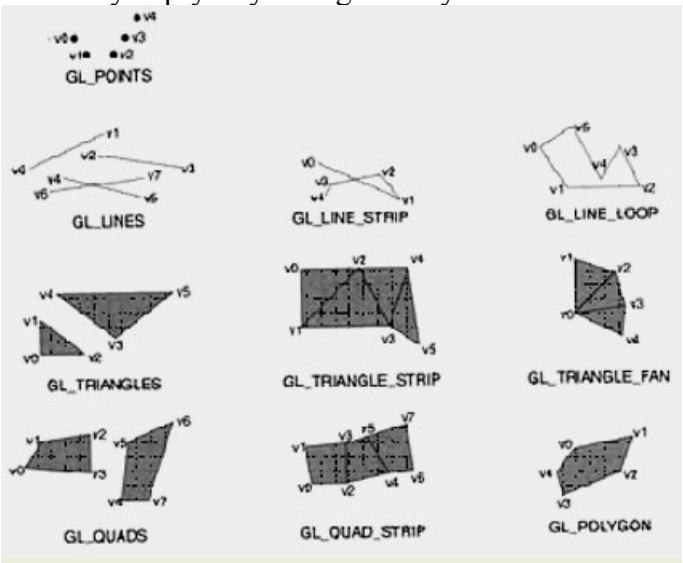
```
#include <GL/glut.h>

/* inicjalizacja trójkąta */
GLfloat v[3][2]={{-1.0, -0.58},
{1.0, -0.58}, {0.0, 1.15}};
int n; /* liczba kroków rekursywnych */

void triangle( GLfloat *a, GLfloat *b,
GLfloat *c){
glVertex3fv(a);
glVertex3fv(b);
glVertex3fv(c);}
```

```
void divide_triangle(GLfloat *a, GLfloat *b,
GLfloat *c, int m)
{
GLfloat v1[3], v2[3], v3[3]; //punkty pomocnicze w 3D
int j;
if(m>0)
{
for(j=0; j<3; j++) v1[j]=(a[j]+b[j])/2;
for(j=0; j<3; j++) v2[j]=(a[j]+c[j])/2;
for(j=0; j<3; j++) v3[j]=(b[j]+c[j])/2;
divide_triangle(a, v1, v2, m-1);
divide_triangle(c, v2, v3, m-1);
divide_triangle(b, v3, v1, m-1);
}
else(triangle(a,b,c));
}

void tetrahedron( int m)
{ /* generowanie czworościanu */
glColor3f(1.0,0.0,0.0);
divide_triangle(v[0], v[1], v[2], m);
glColor3f(0.0,1.0,0.0);
divide_triangle(v[3], v[2], v[1], m);
glColor3f(0.0,0.0,1.0);
divide_triangle(v[0], v[3], v[1], m);
glColor3f(0.0,0.0,0.0);
divide_triangle(v[0], v[2], v[3], m);
}

void display()
{
glClear(GL_COLOR_BUFFER_BIT);
//glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glBegin(GL_TRIANGLES);
tetrahedron(n); //generowanie czworościanu
glEnd();
glFlush();
}

void myinit()
{
//glEnable(GL_DEPTH_TEST); //generowanie przesloniec
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(-2.0, 2.0, -2.0, 2.0);
glMatrixMode(GL_MODELVIEW);
glClearColor (1.0, 1.0, 1.0,1.0);
glColor3f(0.0,0.0,0.0);
}

int main(int argc, char **argv)
{
n=4;
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
//glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH); //przydatne do generowania
przesloniec
glutInitWindowSize(500, 500);
glutCreateWindow("Trojkat sierpinskiego 3D");
glutDisplayFunc(display);
myinit();
glutMainLoop();
}
```

Zadania do wykonania:

Zadanie 1
Na podstawie przykładu 1 zaobserwuj wpływ następujących funkcji na wygląd okna ekranu i okna obserwacji:

- `glutInitWindowSize(x,y)` - zastosowanej w funkcji main(); określa wielkość okna aplikacji
- `gluOrtho2D(x,y,dx,dy)` - zastosowanej w funkcji incjalizacji OpenGL; określa położenie i rozmiar okna obserwacji
- `glViewport(x,y,dx,dy)` - zastosowanej w funkcji incjalizacji OpenGL; określa położenie i rozmiar okna wyświetlania
- Przetestuj rysowanie innych prymitywów graficznych:



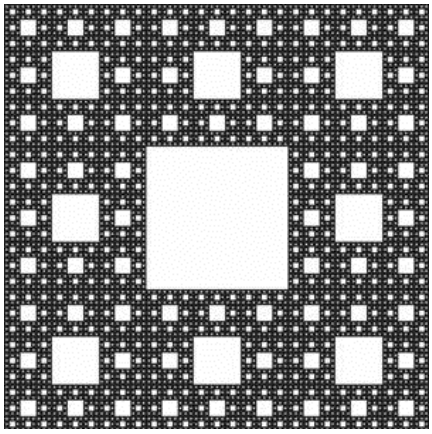
- Nadaj inny kolor prymitywom graficznym przy pomocy polecenia glColor3f(r,g,b), gdzie r,g,b są liczbami z przedziału [0,1]

Zadanie 2: Rysowanie fraktalu dywan Sierpińskiego

W oparciu o funkcję z przykładu 2 napisz program do rysowania dywanu Sierpińskiego.

Konstrukcja dywanu Sierpińskiego zaczyna się od kwadratu. Kawdrat ten jest następnie dzielony na 9 równych kwadratów, jako tablica 3 x 3, z czego kwadrat środkowy jest usuwany. Ta sama procedura jest powtarzana rekursywnie dla pozostałych 8 kwadratów, aż do osiągnięcia zadanej liczby iteracji *n*.

Zastosuj widok z przesłonięciami.



Zadanie 3: Rysowanie fraktalu kostka Sierpińskiego
Na podstawie przykładu 3 napisz program do generowania fraktalu kostka Sierpińskiego będącego rozwinięciem na 6 ścian kostki fraktalu dywan Sierpińskiego z zadania 2