Grafika komputerowa

Autor:
Jacek Wieczorek (181043)

Prowadzący:
Dr inż. Tomasz Kapłon

Wydział Elektroniki
III rok
Pn TP 08.15 - 11.00

6 listopada 2011

1 Cel laboratorium

Celem laboratorium jest zaprezentowanie elementarnych możliwości biblioteki graficznej OpenGL wraz z rozszerzeniem GL Utility Toolkit (GLUT). Ćwiczenie niniejsze obejmuje inicjalizację i zamykanie trybu OpenGL oraz rysowanie tworów pierwotnych (prymitywów) w przestrzeni 2D.

2 Dywan sierpińskiego

Pierwszym zadaniem było narysowanie dywanu sierpińskeigo wykorzystując algorytm iteracyjny oraz rekurencyjny. Ponadto, użytkownik powinien zdefiniować stopień dywanu oraz poziom zniekształcenia.

2.1 Algorytm iteracyjny

Iteracyjny algorytm rysowania dywanu sierpińskeigo polega na stworzeniu najpierw pojedynczego, dużego kwadratu, a następnie zgodnie ze stopniem figury, wycinaniu mniejszych kwadratów, tworząc wzór figury. Co każdy stopień ilośc wyciannych kwadratów wzrastała 3-krotnie, a pole powierzchni malało 3-krotnie.

Procedura generująca dywan sierpińskiego iteracyjnie

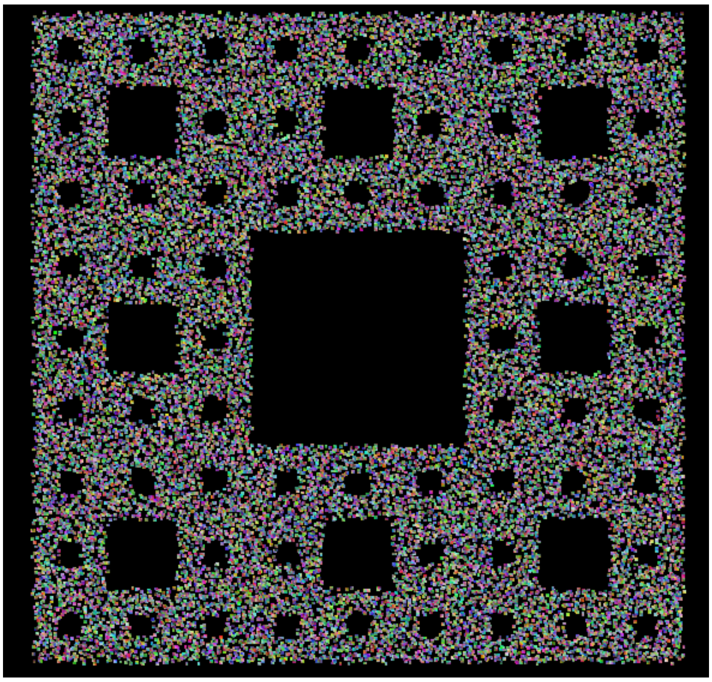
```
void DrawSierpIt(float a, float ax, float ay, int st)
{       //a - bok kwadratu,
        //(ax, ay) wspolrzędne lewego,górnego wierzchołka
        DrawRect(a,ax,ay);
        for(int i=0; i<st; i++)
        {

                int pow = powr(i);
                float x = a/(1.0*(pow));

                for(int j=0; j<pow; j++)
                {
                        for(int k=0; k<pow; k++)
                        {
                                float na = a/(3*pow); //bok kwadratu
                                    wycinanego w danym stopniu trójkąta
                                float nx = ax + (k * x) + (a/(3*pow)); //
                                    wspl. wyciannaego kwadratu
                                float ny = ay - ((j * x) + (a/(3*pow)));
                                DrawBlack(na, nx, ny); // rysowanie czarnego
                                     kwadratu
                        }
                }

        }
}
```

2.2 Algorytm rekurencyjny

```
void DrawOneSierp(float a, float ax, float ay, int st){
        if(st == 0){
                DrawRect(a,ax,ay);
        } else {
                float x = a/3;
                for(int i=0; i<3; i++)
                {
                        for(int j=0; j<3; j++)
                        {
                                if(3*i + j != 4)
                                {
                                        DrawOneSierp(x, ax + j*x, ay - i*x,
                                            st-1);
                                }
                        }
                }
        }
}
```

2.3 Przykładowy rysunek

Przykładowy obraz dywanu sierpińskiego trzeciego stopnia o perturbacjach równych 2.



Rysunek 1: Dywan sierpińskiego

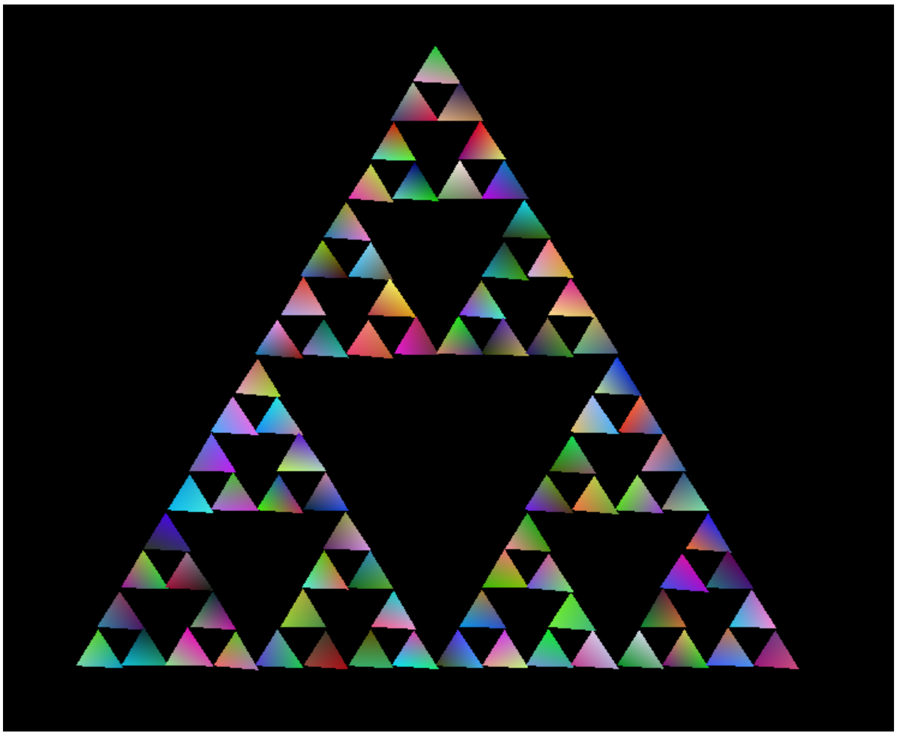
3 Trójkąt sierpińskiego

Drugie zadanie polegało na narysowaniu trójkąta sierpińskiego dwiema metodami : algorytmu rekurencyjnego oraz za pomocą "gry w chaos"

3.1 Algorytm rekurencyjny

Parametry początkowe funkcji : długość boku a, współrzędne górnego wierzchołka, stopień

```
void drawTriangle(float a, float ax, float ay, int st)
{
	//perturbacje
	float h = a * sqrt(3.0) / 2.0;
	float ax2 = rand() % 1 ? ax + (float)rand()/(float)RAND_MAX*per : ax
	    - (float)rand()/(float)RAND_MAX*per;
	float ay2 = rand() % 1 ? ay + (float)rand()/(float)RAND_MAX*per : ay
	    - (float)rand()/(float)RAND_MAX*per;
	colorTab c;

	if(st == 0)
	{

		glBegin(GL_TRIANGLES);
			randColor(c);
			glColor3fv(c);
			glVertex2f(ax2,ay2);

			randColor(c);
			glColor3fv(c);
			glVertex2f(ax2 - a/2, ay - h);

			randColor(c);
			glColor3fv(c);
			glVertex2f(ax + a/2, ay2 - h);
		glEnd();
	} else {
		drawTriangle(a/2, ax,ay, st-1);
		drawTriangle(a/2, ax-a/4, ay - h/2, st-1);
		drawTriangle(a/2, ax+a/4, ay - h/2, st-1);
	}
}
```



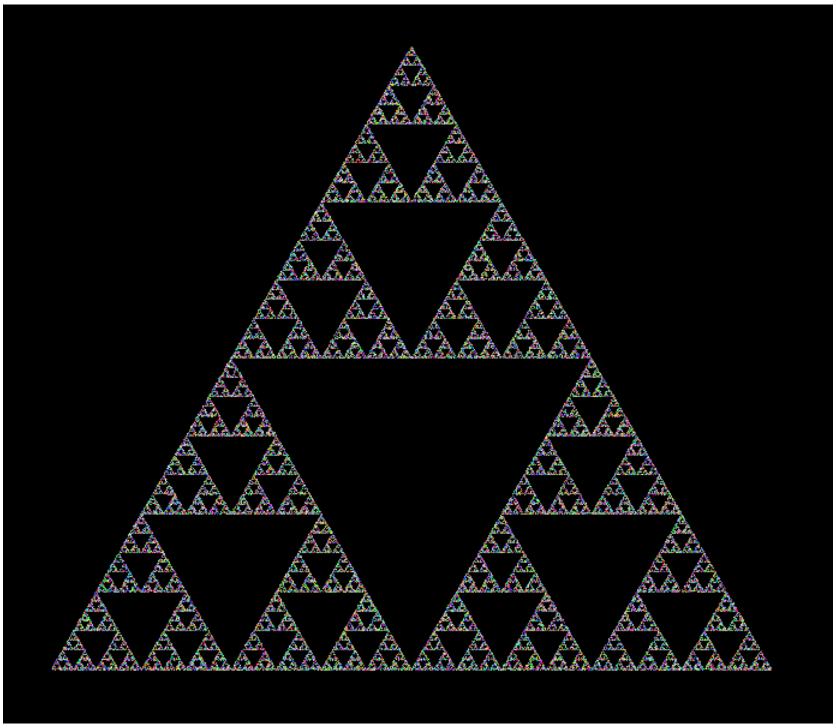
Rysunek 2: Trójkąt sierpińskiego perturbacje = 1, stopień = 4

3.2 Gra w chaos

Algorytm generowania trójkąta sierpińskeigo poprzez grę w chaos polega na zdefiniowaniu trzech niewspółliniowych punktów, które będą tworzyły trójkąt. Następnie, dla dowolnie wylosowanego punktu D, należącego do środka okręgu losujemy jeden z wierzchołków i obliczamy środek odcinka i rysujemy punkt w wyliczonych współrzędnych. Następnie dla nowo powstałego punktu, znowu losujemy któryś z wierzchołków trójkąta, obliczamy środek i rysujemy punkt. Powtarzając tę sekwencję określoną ilość razy otrzymujemy trójką sierpińskiego.

```
void ChaosTriangle(){
        float x = 0.0, y = 0.0;
        int t;
        colorTab c;
        x_1 = 0.0f;
        y_1 = at * sqrt(3.0f) / 3.0f;
        x_2 = -at/2;
        y_2 = -at * sqrt(3.0f) / 6.0f;
```

```
        x_3 = at/2;
        y_3 = -at * sqrt(3.0f) / 6.0f;
        do{
                x = (((float)rand()/(float)RAND_MAX) * at) - at * 0.5;
                y = (((float)rand()/(float)RAND_MAX) * at*sqrt(3.0)* 0.5) -
                    (at*sqrt(3.0)/6);
        }while(!isInside(x,y));
        for(int i=0; i<100000; i++){
                t = rand() % 3;
                if(t==0){
                        x = (x_1 + x) * 0.5;
                        y = (y_1 + y) * 0.5;
                }else if(t==1){
                        x = (x_2 + x) * 0.5;
                        y = (y_2 + y) * 0.5;
                } else {
                        x = (x_3 + x) * 0.5;
                        y = (y_3 + y) * 0.5;
                }
                glBegin(GL_POINTS);
                        randColor(c);
                        glColor3fv(c);
                        glVertex2f(x, y);
                glEnd();
        }
}
```



Rysunek 3: Trójkąt sierpińskiego "gra w chaos"

4 Fraktal plazmowy

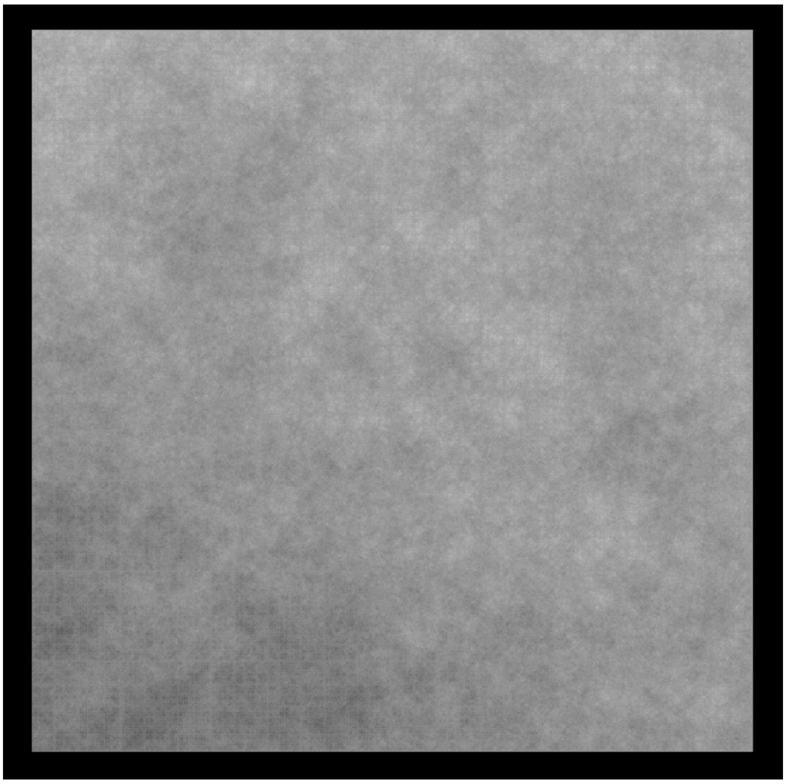
Algorytm generujący fraktal plazmowy został dokładnie opisany w pdfie dostępnym na stronie zakładu. Jedyną własną innwencją było opracowanie wykresów funkcji W(x) i Wc(x). Zdecydowałem się na użycie następującej funkcji dla obu równań :

$$F(x) = 1000 * \frac{p * sin(x)}{10 * x} \tag{1}$$

gdzie p jest zadawanym parametrem najlepiej z przedziału (0.001 - 0.01). Program pozwala na rysowanie zarówno kolorowych fraktali, jak i monochromatycznych. Poniżej przykład algorytmu generującego fraktal monochromatyczny

```
void drawPoints(float x, float y, float a, int corner, float c1, float c2,
    float c3, float c4 ){

        float c12 = randColor(), c13 = randColor(), c24 = randColor(), c34 =
             randColor(), cc = randColor();
        float col;


        c12 = (1-2*getHalf(a/2))*c12 + getHalf(a/2) * (c1 + c2);
        c13= (1-2*getHalf(a/2))*c13 + getHalf(a/2) * (c1 + c3);
        c24 = (1-2*getHalf(a/2))*c24 + getHalf(a/2) * (c4 + c2);
        c34 = (1-2*getHalf(a/2))*c34 + getHalf(a/2) * (c3 + c4);
        cc = (1-4*getCenter(a/2))*cc + getCenter(a/2) * (c1 + c2 + c3 + c4);

        if(corner == 1){
                drawPoint(x + a/2, y, c12);
                drawPoint(x, y-a/2, c13);
                drawPoint(x + a/2, y - a/2, cc);
                drawPoint(x + a, y - a/2, c24);
        } else if(corner == 2){
                drawPoint(x + a/2, y, c12);
                drawPoint(x + a/2, y - a/2, cc);
                drawPoint(x + a, y - a/2, c24);
                drawPoint(x + a/2, y - a, c34);
        } else if(corner == 3){
                drawPoint(x + a/2, y, c12);
                drawPoint(x, y-a/2, c13);
                drawPoint(x + a/2, y - a/2, cc);
                drawPoint(x + a/2, y - a, c34);
        } else if(corner == 4){
                drawPoint(x, y-a/2, c13);
                drawPoint(x + a/2, y - a/2, cc);
                drawPoint(x + a, y - a/2, c24);
                drawPoint(x + a/2, y - a, c34);
        } else {
                drawPoint(x + a/2, y, c12);
                drawPoint(x, y-a/2, c13);
                drawPoint(x + a/2, y - a/2, cc);
                drawPoint(x + a, y - a/2, c24);
                drawPoint(x + a/2, y - a, c34);
        }
```

```
        if(a > roz){
                drawPoints(x,y,a/2, 1, c1, c12, c13, cc);
                drawPoints(x+a/2,y,a/2, 2, c12, c2, cc, c24);
                drawPoints(x, y-a/2, a/2, 3, c13, cc, c3, c34);
                drawPoints(x+a/2, y - a/2,  a/2,4, cc, c24, c34, c4);
        }
}
```



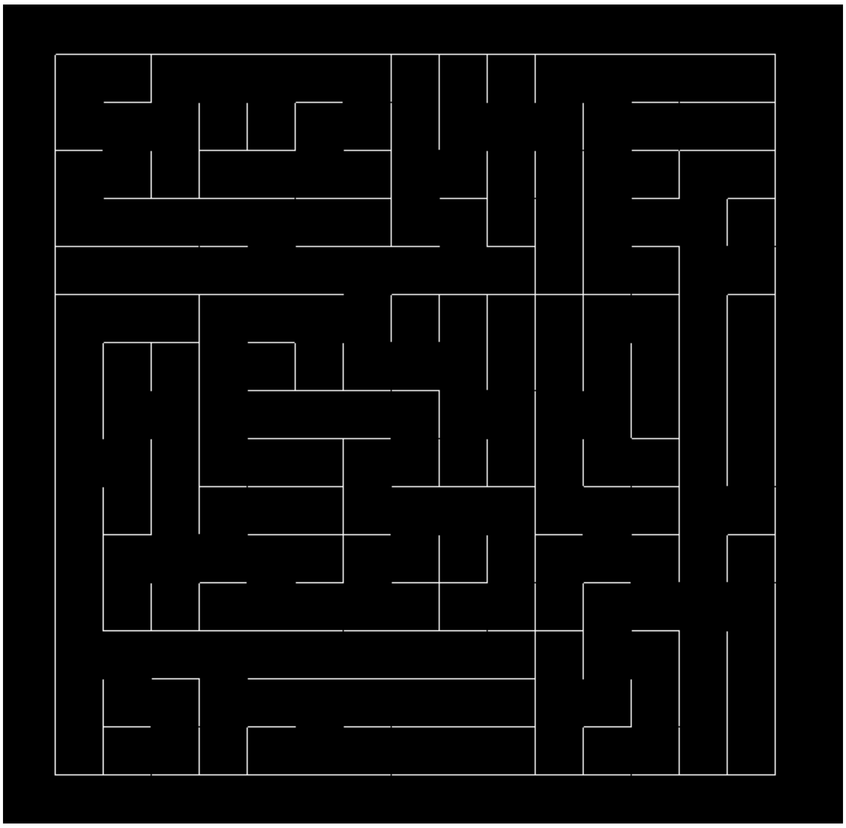
Rysunek 4: a = 150, n = 1500, p dla W(x)=0.005, p dla Wc(x) = 0.003

5 Labirynt

W tym zadaniu wykorzystałem rekurencyjny algorytm generujący labirynt. Polega on na podzieleniu początkowej komory dwiema liniami : poziomą i pionową, na 4 nowe komory. Następnie losujemy 3 linie spośród nowo powstałych (punkt przecięcia rozdziela je na 2 pionowe i 2 poziome) i w losowym miejscu na nich wycinamy bramę. Algorytm powtarzamy rekurencyjnie dla każdej z komór dopóki jej długość, lub szerokość, nie osiągnie zakładanej szerokości tunelu.

```
//x,y - współrzędne lewego, górnego punktu komory
//ver, hor - opzycja wylosowanych lini poziomych i pionowych liczona od
    wierzcholka (x,y)
//ch, cv ilosc korytarzy mogacych zmiescic sie w danej komorze w pionie i
    poziomie
void drawMaze(float x, float y, int ver, int hor, int ch, int cv){

        int which = rand() % 4;

        glBegin(GL_LINES);
        glVertex2f(x , y - ver *w);
        glVertex2f(x + hor * w,y - ver *w);
        glEnd();

        glBegin(GL_LINES);
        glVertex2f(x + hor * w , y - ver *w);
        glVertex2f(x + ch  * w,  y - ver *w);
        glEnd();

        glBegin(GL_LINES);
        glVertex2f(x + hor*w, y);
        glVertex2f(x + hor*w, y- ver*w);
        glEnd();

        glBegin(GL_LINES);
        glVertex2f(x + hor*w, y- ver*w);
        glVertex2f(x + hor*w, y- cv*w);
        glEnd();

        if(which == 0){
                removeGate(x+hor*w, y, ver, true);
                removeGate(x+hor*w, y-ver*w, ch - hor, false);
                removeGate(x+hor*w, y-ver*w, cv-ver, true);
        }else if(which == 1){
                removeGate(x+hor*w, y-ver*w, ch - hor, false);
                removeGate(x+hor*w, y-ver*w, cv-ver, true);
                removeGate(x, y-ver*w, hor, false);
        }else if(which == 2){
                removeGate(x+hor*w, y-ver*w, cv-ver, true);
                removeGate(x, y-ver*w, hor, false);
                removeGate(x+hor*w, y, ver, true);
        } else {
                removeGate(x, y-ver*w, hor, false);
                removeGate(x+hor*w, y, ver, true);
                removeGate(x+hor*w, y-ver*w, ch - hor, false);
        }

        int nh, nv;
        if(hor > 1 && ver > 1){
                nh = randFun(hor);
                nv = randFun(ver);
                drawMaze(x,y,nv,nh,hor,ver);
        }

        if(hor > 1 && (cv - ver) > 1){
                nh = randFun(hor);
                nv = randFun((cv - ver));
```

```
            drawMaze(x,y-ver*w,nv,nh,hor,(cv - ver));
        }

        if((ch-hor) > 1 && ver > 1){
            nh = randFun((ch-hor));
            nv = randFun(ver);
            drawMaze(x + hor*w,y,nv,nh,(ch-hor),ver);
        }

        if((ch-hor) > 1 && (cv - ver) > 1){
            nh = randFun((ch-hor));
            nv = randFun((cv - ver));
            drawMaze(x + hor*w,y-ver*w,nv,nh,(ch-hor),(cv - ver));
        }

    }
```



Rysunek 5: Przykładowy labirynt

6 Wnioski

Laboratorium pozwoliło mi na zapoznanie się z rysowaniem obiektów 2D za pomoca biblioteki OpenGL, a także pozania algorytmów tworzących wiele podstawowych figur i fraktali.