Grafika komputerowa

Autor:
Tymon Tobolski (181037)

Prowadzący:
Dr inż. Tomasz Kapłon

Wydział Elektroniki
III rok
Pn TP 08.15 - 11.00

13 listopada 2011

1 Cel laboratorium

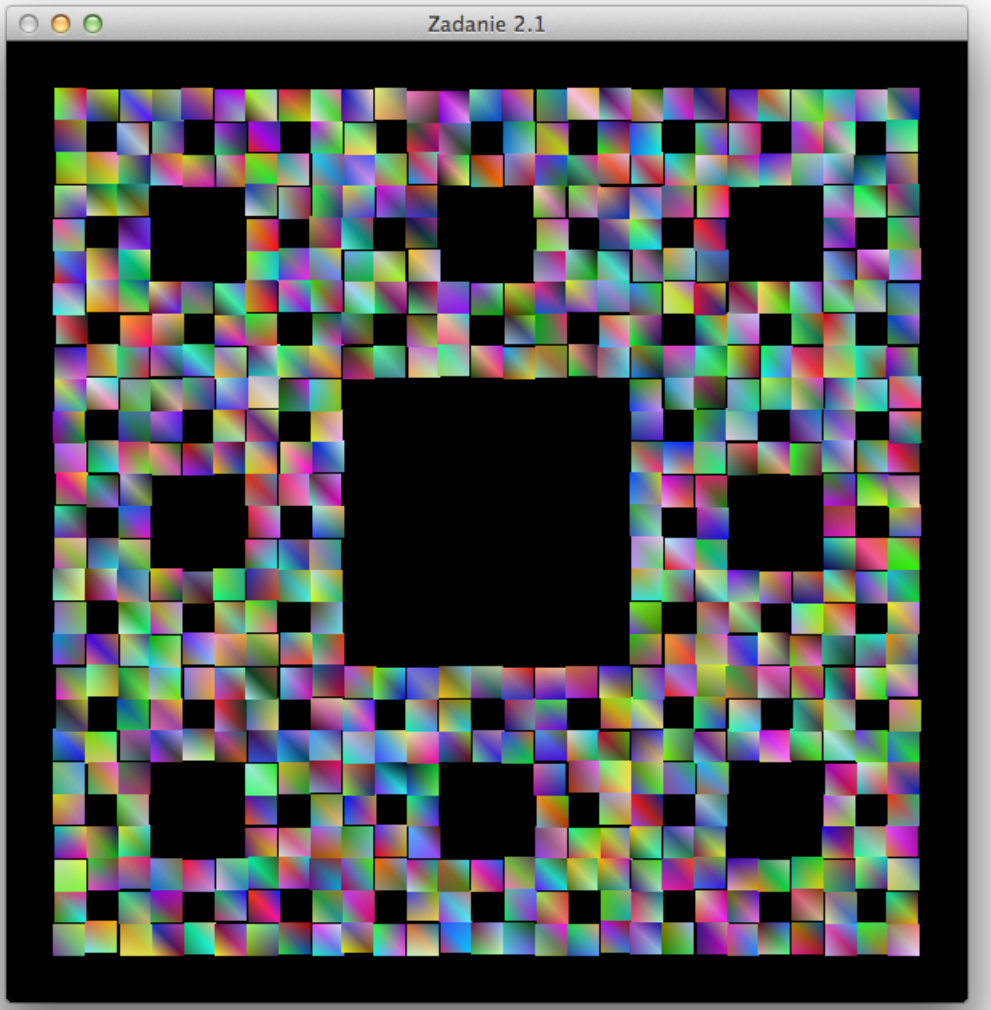
Celem laboratorium było zapoznanie sie z podstawowymi funkcjami biblioteki OpenGL oraz jej rozszerzenia GLUT. Poniższe zadania obejmuja rysowanie prostych kształtów w przestrzeni dwuwymiarowej.

2 Dywan sierpińskiego

Program rysuje reprezentacje dywanu sierpińskiego o zadanym rozmiarze, stopniu przybliżenia oraz poziomie zniekształcenia. Wykorzystany algortym rekurencyjnie oblicza pozycje kolejnych kwadratów aż do osiągnięcia określonego poziomu. Na ostatnim poziomie rysowa są zniekształcone kwadraty, które układają się w przedstawiony poniżej wzór.

```
1  void carpetRec(GLfloat x, GLfloat y, GLfloat size, int n, int p){
       GLfloat x1 = x-size/2;
       GLfloat y1 = y+size/2;
       _carpetRec(x1, y1, size, n, p);
   }

   void _carpetRec(GLfloat x, GLfloat y, GLfloat size, int n, int p){
       if(n == 0) return;

       GLfloat a = size/3;
11
       glBegin(GL_QUADS);
       glColor3f(0.0, 0.0, 0.0);
       glVertex2f(x+a,   y-a);
       glVertex2f(x+a+a, y-a);
       glVertex2f(x+a+a, y-a-a);
       glVertex2f(x+a,   y-a-a);
       glEnd();


21     if(n == 1){
           for(int i=-1; i<=1; i++){
               for(int j=-1; j<=1; j++){
                   if(!(i == 0 && j == 0)){
                       GLfloat xm = x+i*a + p*randp();
                       GLfloat ym = y+j*a + p*randp();

                       glBegin(GL_POLYGON);
                       randomColor();
                       glVertex2f(xm+a,   ym-a);
31                     randomColor();
                       glVertex2f(xm+a+a, ym-a);
                       randomColor();
                       glVertex2f(xm+a+a, ym-a-a);
                       randomColor();
                       glVertex2f(xm+a,   ym-a-a);
                       glEnd();
                   }
               }
```

```
            }
        }


        n -= 1;
        _carpetRec(x,     y,     a, n, p);
        _carpetRec(x+a,   y,     a, n, p);
        _carpetRec(x+a+a, y,     a, n, p);
        _carpetRec(x,     y-a,   a, n, p);
        _carpetRec(x+a+a, y-a,   a, n, p);
        _carpetRec(x,     y-a-a, a, n, p);
        _carpetRec(x+a,   y-a-a, a, n, p);
        _carpetRec(x+a+a, y-a-a, a, n, p);
    }
```



Rysunek 1: Dywan sierpińskiego

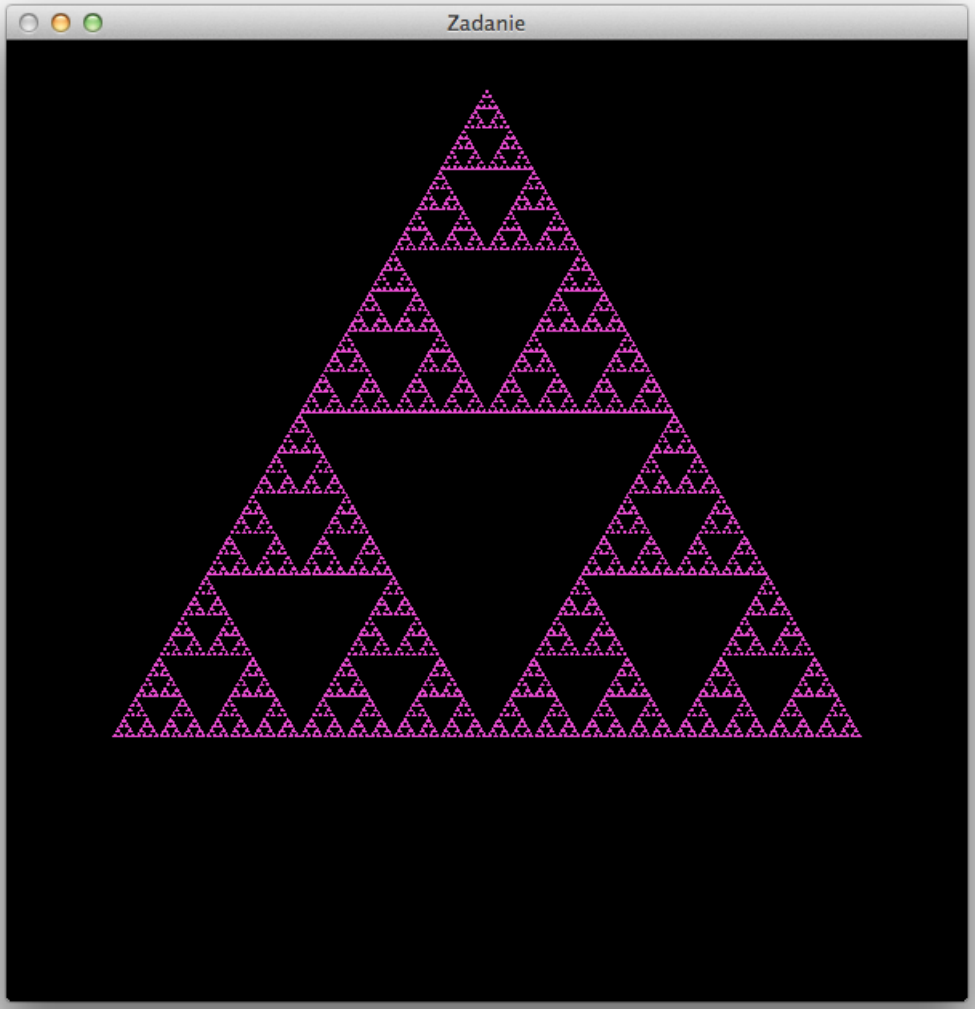
3 Trójkąt sierpińskiego

Pierwsza metoda rysowania trójkąta sierpińskiego polega na narysowaniu dużego trójkąta, a następnie "wycinaniu" mniejszego trójkąta ze środka poprzedniego. Czynność tę należy powtarzać, aż do otrzymania porządanego rezultatu.

Druga metoda rysowania trójkąta spierpińskiego polega na wybraniu losowego punktu w środku trójkąta, a następnie losowe przesuwanie sie w jednym z trzech kierunków, za każdym razem stawiając punkt.

```
void triangleRec(GLfloat x, GLfloat y, GLfloat size, int n){
    GLfloat sq = sqrtf(3);
    GLfloat x1 = -sq/2;
    GLfloat y1 = -1.0f/2;
    GLfloat x2 = sq/2;
    GLfloat y2 = -1.0f/2;
    GLfloat x3 = 0;
    GLfloat y3 = 1;

    randomColor();
    glBegin(GL_TRIANGLES);
    glVertex2f(x+x1*size, y+y1*size);
    glVertex2f(x+x2*size, y+y2*size);
    glVertex2f(x+x3*size, y+y3*size);
    glEnd();

    _triangleRec(x+x1*size, y+y1*size, x+x2*size, y+y2*size, x+x3*size, y+y3
        *size, n);
}

void _triangleRec(GLfloat x1, GLfloat y1, GLfloat x2, GLfloat y2, GLfloat x3
    , GLfloat y3, int n){
    if(n == 0) return;

    GLfloat x12 = center(x1, x2);
    GLfloat y12 = center(y1, y2);
    GLfloat x23 = center(x2, x3);
    GLfloat y23 = center(y2, y3);
    GLfloat x31 = center(x3, x1);
    GLfloat y31 = center(y3, y1);

    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_TRIANGLES);
    glVertex2f(x12, y12);
    glVertex2f(x23, y23);
    glVertex2f(x31, y31);
    glEnd();

    n -= 1;
    _triangleRec(x1, y1, x12, y12, x31, y31, n);
    _triangleRec(x12, y12, x2, y2, x23, y23, n);
    _triangleRec(x31, y31, x23, y23, x3, y3, n);
}
```

```
void triangleRand(GLfloat x, GLfloat y, GLfloat size, int n){
    GLfloat sq = sqrtf(3);

47      randomColor();
        glBegin(GL_POINTS);

        GLfloat xp = 0, yp = 0;

        for(int i=0; i<n; i++){
            int r = rand() % 3;
            switch(r){
                case 0:
                    xp = xp/2;
57                  yp = yp/2 + 0.5f;
                    break;
                case 1:
                    xp = xp/2 - sq/4;
                    yp = yp/2 - 0.25f;
                    break;
                case 2:
                    xp = xp/2 + sq/4;
                    yp = yp/2 - 0.25f;
                    break;
67          }

            glVertex2f(x+xp*size, y+yp*size);
        }

        glEnd();
}
```



Rysunek 2: Trójkąt sierpińskiego

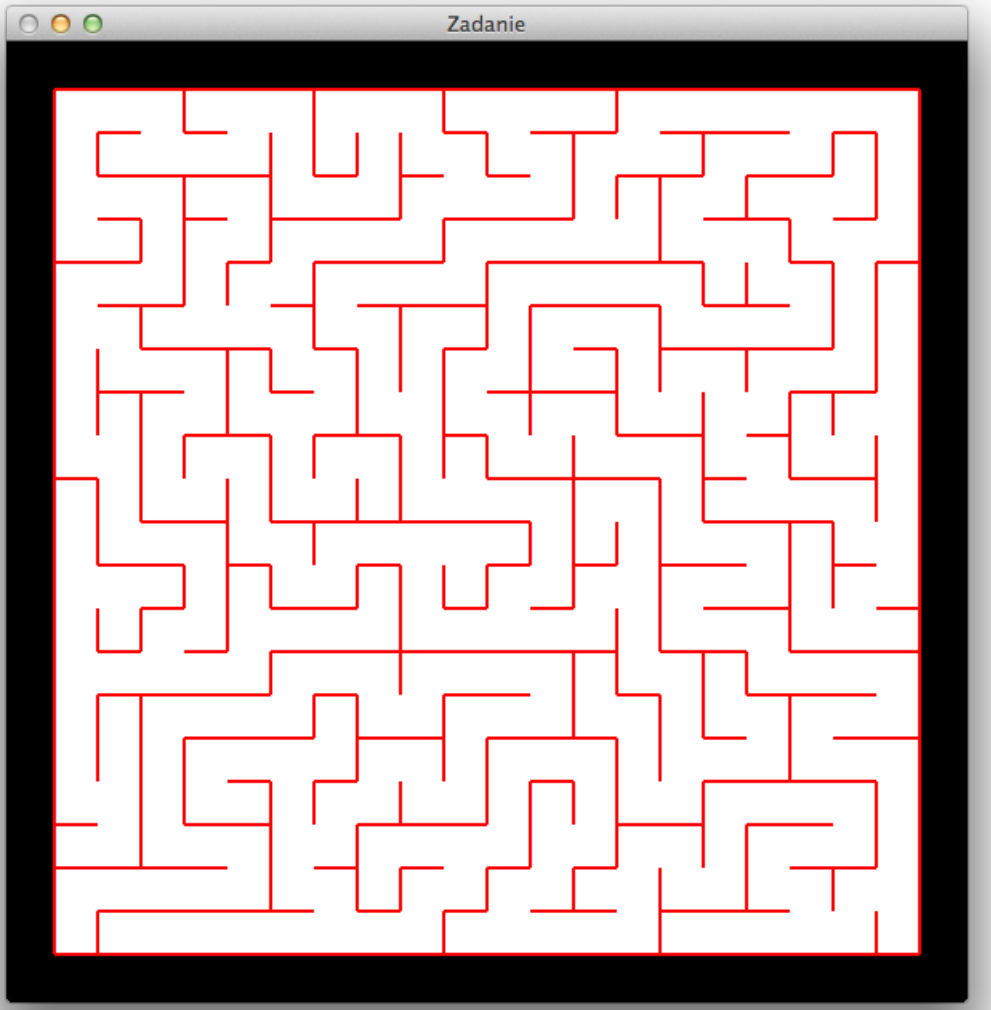
4 Labirynt

Algortym generujący labirynt zaczyna się od wypełnienia powierzchni labiryntu siatką komnat o rozmiarach 1x1. Początek ustala się w dowolnym punkcie przy krawędzi labirytnu. Z każdym krokiem losowo wybierana jest sąsiadująca komnata, która posiada wszystkie ściany, ściana między wybraną komnata a obecna zostaje usunięta a algorytm zaczyna ponownie tę samą operacje od nowo otwartej komnaty. W przypadku braku sąsiadującej komnaty, która miała by wszystkie ściany program cofa się o jedno pole i ponawia poszukiwania. Algorytm kończy się kiedy wszystkie komnaty zostaną odwiedzone.

```
class Cell {
public:
    int x, y;
    Cell(int x, int y):x(x),y(y){}
    Cell():x(0),y(0){}
};

int maze_moves[4][2] = {
    {-1, 0},
    {0, -1},
    {0, 1},
    {1, 0}
};


void maze(GLfloat x, GLfloat y, GLfloat size){
    int s = size/MAZE_SIZE;
    x -= size/2;
    y -= size/2;

    for(int i=0; i<MAZE_SIZE; i++){
        for(int j=0; j<MAZE_SIZE; j++){
            grid[i][j] = N | S | E | W;
        }
    }


    stack<Cell *> stk;
    Cell found[4];
    int foundc = 0;
    int total = MAZE_SIZE*MAZE_SIZE;
    int visited = 1;
    Cell * current = new Cell(0,0);

    while(visited < total){
        foundc = 0;
        for(int i=0; i<4; i++){
            int x1 = current->x + maze_moves[i][0];
            int y1 = current->y + maze_moves[i][1];

            if(x1 >= 0 && y1 >= 0 && x1 < MAZE_SIZE && y1 < MAZE_SIZE){
                if(grid[x1][y1] == ALL){
                    found[foundc].x = x1;
```

```
                    found[foundc].y = y1;
                    foundc++;
                }
            }
        }

        if(foundc > 0){
            Cell * c = &found[rand() % foundc];
            Cell * one = new Cell(c->x, c->y);
            if(one->x == current->x){
                if(one->y > current->y){
                    grid[one->x][one->y] &= ~N;
                    grid[current->x][current->y] &= ~S;
                } else {
                    grid[one->x][one->y] &= ~S;
                    grid[current->x][current->y] &= ~N;
                }
            } else {
                if(one->x > current->x){
                    grid[one->x][one->y] &= ~W;
                    grid[current->x][current->y] &= ~E;
                } else {
                    grid[one->x][one->y] &= ~E;
                    grid[current->x][current->y] &= ~W;
                }
            }

            stk.push(current);
            current = one;
            visited++;
        } else {
            current = stk.top();
            stk.pop();
        }
    }



    glColor3f(1.0f, 1.0f, 1.0f);
    for(int i=0; i<MAZE_SIZE; i++){
        for(int j=0; j<MAZE_SIZE; j++){
            glRectf(x+i*s, y+j*s, x+(i+1)*s, y+(j+1)*s);
        }
    }

    glColor3f(1.0f, 0.0f, 0.0f);
    glLineWidth(2.0f);

    for(int i=0; i<MAZE_SIZE; i++){
        for(int j=0; j<MAZE_SIZE; j++){

            if(grid[i][j] & N){
                glBegin(GL_LINES);
                glVertex2f(x+i*s, y+j*s);
                glVertex2f(x+(i+1)*s, y+j*s);
                glEnd();
            }

            if(grid[i][j] & S){
                glBegin(GL_LINES);
                glVertex2f(x+i*s, y+(j+1)*s);
```

```
                    glVertex2f(x+(i+1)*s, y+(j+1)*s);
107                 glEnd();
                }

                if(grid[i][j] & E){
                    glBegin(GL_LINES);
                    glVertex2f(x+(i+1)*s, y+j*s);
                    glVertex2f(x+(i+1)*s, y+(j+1)*s);
                    glEnd();
                }

117             if(grid[i][j] & W){
                    glBegin(GL_LINES);
                    glVertex2f(x+i*s, y+j*s);
                    glVertex2f(x+i*s, y+(j+1)*s);
                    glEnd();
                }
            }
        }
    }
```



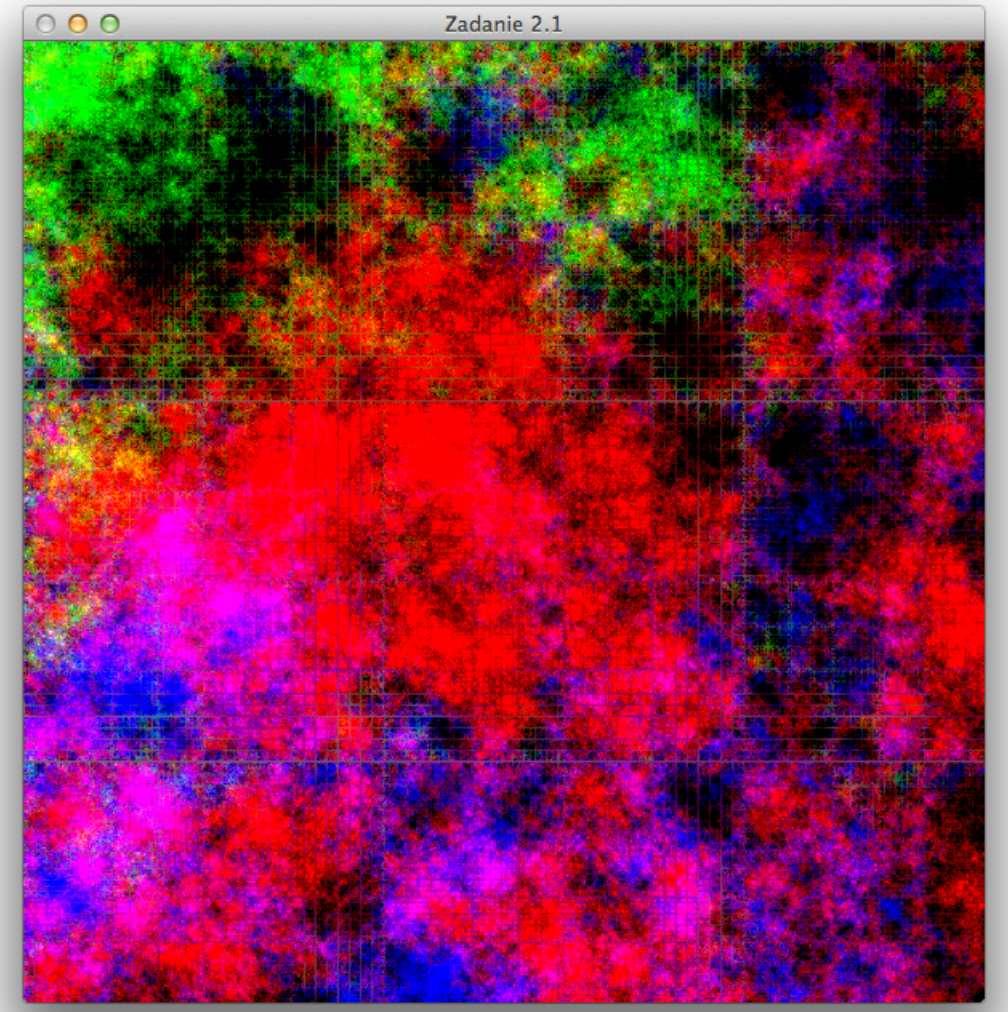
Rysunek 3: Labirynt

5 Fraktal plazmowy

Program pozwala na rysowanie monochromatycznych jak i kolorowych fraktali plazmowych. Jedyną trudnością przy implementacji opisanego w dokumentacji algorytmu było dobranie odpowiednich funckji W(x) i Wc(x).

```
Color newColor3fv(Color c1, Color c2, GLfloat x){
    Color nc = new GLfloat[3];
    Color rc = randomColor3fv();

    for(int i=0; i<3; i++){
        nc[i] = ((1 - 2*W(x)) * rc[i]) + (W(x) * c1[i]) + (W(x) * c2[i]);
    }
    delete[] rc;
    return nc;
}

Color newCenterColor3fv(Color c1, Color c2, Color c3, Color c4, GLfloat x){
    Color nc = new GLfloat[3];
    Color rc = randomColor3fv();

    for(int i=0; i<3; i++){
        nc[i] = (1 - 4*Wc(x)) * rc[i] + (Wc(x)*c1[i] + Wc(x)*c2[i] + Wc(x)*
            c3[i] + Wc(x)*c4[i]);
    }

    delete[] rc;
    return nc;
}


void drawSquare(GLfloat x, GLfloat y, GLfloat size, Color c1, Color c2,
    Color c3, Color c4){
    if(size < .1f) return;

    GLfloat a = size / 2;

    Color c12 = newColor3fv(c1, c2, a);
    Color c13 = newColor3fv(c1, c3, a);
    Color c24 = newColor3fv(c2, c4, a);
    Color c34 = newColor3fv(c3, c4, a);
    Color cc = newCenterColor3fv(c1, c2, c3, c4, a);

    glColor3fv(c12);
    glVertex2f(x+a, y);

    glColor3fv(c13);
    glVertex2f(x, y-a);

    glColor3fv(cc);
    glVertex2f(x+a, y-a);

    glColor3fv(c24);
    glVertex2f(x+size, y-a);

    glColor3fv(c34);
    glVertex2f(x+a, y-size);
```

```
        drawSquare(x,   y,   a, c1, c12, c13, cc);
        drawSquare(x+a, y,   a, c12, c2, cc, c24);
        drawSquare(x,   y-a, a, c13, cc, c3, c34);
        drawSquare(x+a, y-a, a, cc, c24, c34, c4);
55
    }
```



Rysunek 4: Fraktal plazmowy

6 Funkcje pomocnicze

Poniżej znajdują się funkcje pomocnicze wykorzystywane w powyższych programach.

```
void randomColor(){
    glColor3f(((rand() % 1001) / 1000.0), ((rand() % 1001) / 1000.0), ((rand
        () % 1001) / 1000.0));
}

GLfloat randp(){
    return ((rand() % 1000) - 500)/ 3000.0;
}
```