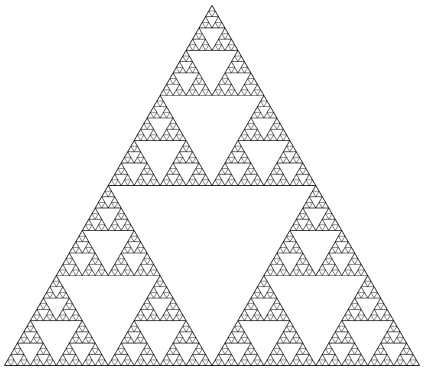
INFORMATYKA I: INSTRUKCJA 10
Fraktale

Na dzisiejszych zajęciach będziemy się zajmować fraktalami. Fraktale to zbiory matematyczne charakteryzujące się tzw. samopodobieństwem - w każdym powiększeniu wyglądają identycznie[1].

1 Trójkąt Sierpińskiego

Struktura trójkąta Sierpińskiego polega na tym, że każdy trójkąt zawiera w sobie trzy takie same trójkąty zmniejszone dwukrotnie (patrz Rys. 1).



Rysunek 1: Trójkąt Sierpińskiego.

Trójkąt Sierpińskiego można stworzyć na różne sposoby. My zajmiemy się rekurencją.

Rekurencyjne wywoływanie funkcji

Rekurencyjne wywoływanie funkcji polega na tym, że dana funkcja we własnym ciele wywołuje samą siebie. Mechanizm rekurencji jest bardzo prosty, ilustruje to poniższy przykład:

[1] Z tego powodu wydają się bardzo skomplikowane. Okazuje się jednak, że fraktale stanowią całkiem niezły sposób opisu przyrody - wystarczy spojrzeć na kawałek wybrzeża Wielkiej Brytanii albo na gałąź choinki, żeby stwierdzić, że przypominają całość, tylko, że w pomniejszeniu.

```
void recursiveDraw(double x, double y, double R){

     circle(x,y,R);
          if(x<500){
               recursiveDraw(x+5, y+5, 0.88*R);
               }
     }
```

Funkcja `recursiveDraw` rysuje okrąg o zadanych parametrach oraz wywołuje samą siebie z inymi wartościami argumentów, dopóki wartość x nie przekroczy 500.
W wypadku trójkąta Sierpińskiego napiszemy funkcję, która generuje fraktal do pewnej „głębokości", gdzie głębokość 1 oznacza zwykły trójkąt, 2 — trzy trójkąty, 3 — dziewięć, itd. Funkcję taką można napisać zauważając, że trójkąt o głębokości n składa się z trzech trójkątów o głębokości $n-1$.

Ćwiczenia

Napisz funkcje `sierpinski(x,y,d,n)`, która:

1. Dla $n = 1$ stworzy trójkąt równoboczny o wysokości d o lewym dolnym wierzchołku w punkcie (x, y)
2. Dla $n > 1$ wywoła trzy razy funkcję `sierpinski` z odpowiednimi przesunięciami, dla dwa razy mniejszego d i $n-1$.
3. Umieść na początku funkcji wywołanie `animate` i zobacz w jakiej kolejności rysowane są najmniejsze trójkąty.

2 Zbiór Julii, zbiór Mandelbrota

Wyobraźmy sobie prosty ciąg liczb zespolonych:

$$z_{i+1} = z_i^2 + c$$

Taki ciąg ma bardzo nietypowe własności. Między innymi, jego rozbieżność zależy w bardzo złożony sposób od warunku początkowego z_0 i stałej c. Zbiór takich z_0, dla których ciąg ten nie jest rozbieżny nazwany został zbiorem Julii (zależy on od c). Zaś zbiór takich c (dla $z_0 = 0$) — to zbiór Mandelbrota.

2.1 Mapa kolorów

Do zobrazowania tych zbiorów potrzebna jest możliwość kolorowania pojedynczych pikseli. Wyobraźmy sobie, że nasze okno grafiki to układ dwuwymiarowy, z x w przedziale $[-4, 4]$ i y w przedziale $[-3, 3]$. Używając funkcji `setcolor`, możemy narysować wykres funkcji $\sin(x^2 + y^2)$. Funkcja `setcolor` jako argument przyjmuje liczbę z przedziału od 0 do 1 i ustawia kolor rysowania. Przeanalizuj następujący program:

```
double fun(double x, double y) {
    return (sin((x*x+y*y)*50.0)+1.0)/2.0;
}

void main()
{
    double r;
    int i, j;
    graphics(810,610);
    for (i=0; i<800; i++)
        for (j=0; j<600; j++)
        {
            r = fun(i/200.-2., j/200. - 1.5);
            setcolor(r);
            point(i,j);
        }
    wait();
}
```

2.2 Rozbieżność

Stwórz funkcję `divergence(zR,zC, cR, cC)`, która liczy ciąg $z_{i+1} = z_i^2 + c$. Gdzie $z_0 =$ `zR` $+$ `zC`i, zaś $c =$ `cR` $+$ `cC`i. Jeśli ciag jest rozbieżny, niech funkcja zwraca ilość iteracji, po której $|z_n| > 2$ podzieloną przez 600. Za maksymalną liczbę iteracji przyjmij 600. Jeżeli ciąg jest zbieżny (tzn. po 600 iteracjach nadal $|z_n| < 2$), niech funkcja zwraca 0.

Przypomnienie: Działania na liczbach zespolonych wykonuj jak zwykłe mnożenie dwumianów, pamiętając jedynie, że $i^2 = -1$.

Ćwiczenia

- Pokoloruj piksele jak w poprzednim podpunkcie, zgodnie z `divergence(x, y, -0.3, 0.63)` (możesz umieścić to wywołanie wewnątrz `fun(x,y)` albo bezpośrednio wpisać jego wynik do zmiennej `r`).
- Zobacz, jak wygląda zbiór dla innych c, np $c = -0.1 + 0.65i$, $c = 0$, $c = 1$, zmieniając liczbę iteracji jeśli potrzeba.
- Pokoloruj piksele zgodnie z `divergence(0, 0, x, y)`. Powinieneś otrzymać zbiór Mandelbrota.
- Zmodyfikuj funkcję tak, by zwracała logarytm z liczby iteracji.
- Zmodyfikuj kod tak, aby zobaczyć obszar o środku w $(-0.345, 0.635)$.
- Powiększ obszar 200 razy.

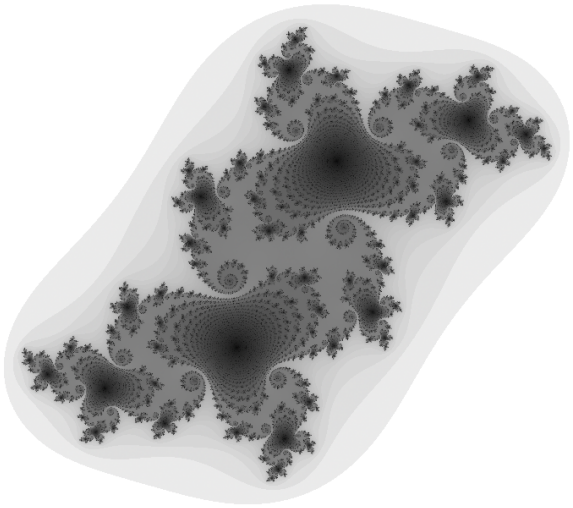
2.3 $*$ Antyaliasing - dla dociekliwych

Antyaliasing ma na celu usunięcie efektów reprezentacji ze skończoną rozdzielczością. Dla przykładu, jeśli mamy literę, to jej krawędź jest gładką krzywą, która przechodzi tylko częściowo przez piksel. Jeśli zaczernimy tylko piksele wewnątrz krzywej, uzyskamy bardzo nienaturalny efekt. Jeśli jednak użyjemy odcienia szarości, proporcjonalnego do „pola" przykrytego tą literą, uzyskamy ładną, gładką czcionkę.

Wyobraźmy sobie, że mamy obraz składający się z dużej liczby gęsto ułożonych czarnych linii na białym tle. Chcąc taki obraz zmniejszyć, możemy wziąć np. co trzeci piksel w każdą stronę. Jednak takie podejście spowoduje, że raz trafimy na w pełni czarny, a raz na w pełni biały piksel. Ostatecznie uzyskamy biało-czarną kaszę. Drugim podejściem byłoby policzenie średniej z każdej kostki 3×3. Takie podejście da nam pożądany efekt zamazania zbyt małych struktur i kolor szary w miejscu losowej kaszy. Taki rodzaj antyaliasingu nazywany jest super-samplingiem. W wypadku obrazów takich jak zbiór Mandelbrota czy zbiór Julii, możemy dla każdego piksela obliczyć $k \times k$ wartości funkcji `fun` i uśrednić wynik. Taki zabieg wygładzi obraz i usunie „odstające" piksele.

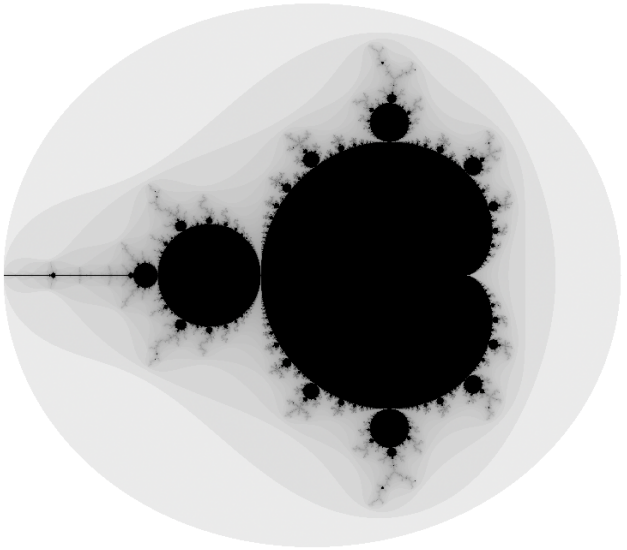
Ćwiczenia

- Stwórz funkcję `fun2(x,y)` która liczy średnią wartość funkcji `fun` w czterech punktach: (x, y), $(x + s, y)$, $(x, y + s)$ i $(x + s, y + s)$. Gdzie $s = \frac{1}{800}$. Pokoloruj piksele za jej pomocą.



Rysunek 2: Zbiór Julii.

- Pokoloruj połowę obrazu z antyaliasingiem, a połowę bez. (Np. zbiór Julii dla $c = -0.1 + 0.65i$) Czy widać różnicę?
- *Spróbuj uogólnić tę technikę z $k = 2$ na dowolne k.
- *Zamiast średniej oblicz maksimum lub minimum.
- Zmień funkcje `setcolor` na `setgray`.



Rysunek 3: Zbiór Mandelbrota.