Podstawy Programowania 1

Podstawy Grafiki 2D - część druga
Biblioteka Allegro

Arkadiusz Chrobot

Katedra Systemów Informatycznych

27 stycznia 2021

Plan

Wstęp

W tej części wykładu zostaną zaprezentowane przykłady programów wykorzystujących opisane w pierwszej części elementy biblioteki Allegro do tworzenia obrazów w grafice 2D. Dwa z nich będą używały mechanizmów związanych z animacją.

Fraktale

Fraktale są skomplikowanymi obiektami geometrycznymi. Istnieje wiele definicji matematycznych opisujących czym jest fraktal[1], jak również wiele metod tworzenia fraktali. Najprościej scharakteryzować je jako obiekty o powtarzalnej strukturze. W ramach tego wykładu zostaną zaprezentowane trzy przykłady fraktali, które są generowane różnymi sposobami.

[1] Więcej informacji na temat fraktali można znaleźć w książce Jacka Kudrewicza „Fraktale i chaos", WNT, Warszawa, 1993.

Fraktale - IFS

Fraktale mogą być rysowane z użyciem układu iterowanych odwzorowań (ang. *Iterated Function System* - IFS). Ta metoda polega na wyliczaniu współrzędnych kolejnych punktów należących do fraktala z użyciem *odwzorowań afinicznych*, czyli takich, które są określone następującym układem:

$$\begin{cases} x' = a \cdot x + b \cdot y + c \\ y' = d \cdot x + e \cdot y + f \end{cases}$$

W tym zapisie x' i y' są współrzędnymi nowego punktu fraktala, a x i y współrzędnymi bieżącego punktu. W metodzie wykorzystującej IFS jest dostępnych kilka takich układów, z których na drodze losowania wybierany jest jeden, który posłuży do wyliczenia kolejnego punktu. Im więcej punktów zostanie wyliczonych i zaznaczonych na płaszczyźnie, tym dokładniejszy będzie obraz fraktala.

Fraktale - IFS

W programie, który demonstruje rysowanie fraktali z użyciem IFS zostaną użyte cztery odwzorowania afiniczne, których współczynniki a, b, c, d, e i f są podane w tabeli:

	a	**b**	**c**	**d**	**e**	**f**
1	-0,67	-0,02	0	-0,18	0,81	10
2	0,4	0,4	0	-1	0,4	0
3	-0,4	-0,4	0	-0,1	0,4	0
4	-0,1	0	0	0,44	0,44	-2

Fraktale - IFS

```
#include<allegro.h>
#include<allegro/keyboard.h>
#include<stdlib.h>
#include<time.h>

#define WIDTH 1366

#define HEIGHT 768

#define SCALE 15
```

Fraktale - IFS

Oprócz plików nagłówkowych związanych z biblioteką Allegro do programu włączane są również pliki nagłówkowe potrzebne do obsługi generatora liczb pseudolosowych. Stałe `WIDTH` i `HEIGHT` określają szerokość i wysokość w pikselach ekranu, z którym program będzie współpracował. Stała `SCALE` określa wartość współczynnika, przez który obraz fraktala zostanie przeskalowany - inaczej byłby stosunkowo mały.

Fraktale - IFS

```
int initialize(int card, int width, int height)
{
        srand(time(NULL));
        if(allegro_init()) {
                allegro_message("allegro_init: %s\n",allegro_error);
                return -1;
        }
        if(install_keyboard()) {
                allegro_message("install_keyboard: %s\n",allegro_error);
                allegro_exit();
                return -1;
        }
        set_color_depth(32);
        if(set_gfx_mode(card,width,height,0,0)) {
                allegro_message("%s\n",allegro_error);
                allegro_exit();
                return -1;
        }
        return 0;
}
```

Fraktale - IFS

Funkcja `initialize()` odpowiedzialna jest za inicjację pracy biblioteki Allegro oraz generatora liczb pseudolosowych. Podobnie lub tak samo będzie zdefiniowana w pozostałych zaprezentowanych programach. Wewnątrz tej funkcji najpierw jest uruchamiany generator liczb pseudolosowych poprzez wywołanie funkcji `srand()`. Następnie przy pomocy makra `allegro_init` inicjowana jest biblioteka Allegro. Kolejną wykonywaną czynnością jest inicjacja obsługi klawiatury przy pomocy funkcji `install_keyboard()`. Po jej wykonaniu określana jest, poprzez wywołanie `set_color_depth()`, przestrzeń kolorów. W programie używany będzie model RGBA kolorów. Ostatnią wykonywaną czynnością jest włączenie odpowiedniego trybu graficznego za pomocą funkcji `set_gfx_mode()`. Jako pierwsze trzy argumenty są do tej funkcji przekazywane parametry funkcji `initialize()`. Dwa ostatnie argumenty wynoszą zero, ponieważ w programie nie będzie używany mechanizm tworzenia animacji.

Fraktale - IFS

W przypadku niepowodzenia wykonania któregokolwiek z podprogramów użytych w `initialize()` wypisywany jest komunikat z użyciem funkcji `allegro_message()`, finalizowana jest praca biblioteki Allegro poprzez wywołanie `allegro_exit()` oraz zwracany jest odpowiedni kod wyjątku.

Fraktale - IFS

```
void draw_with_ifs(double x, double y)
{
        const int GREEN_COLOUR = makecol(0,255,0);
        while(!(key[KEY_Q]||key[KEY_ESC])) {
                switch(rand()%4) {
                        case 0:
                                x=-0.678*x-0.02*y;
                                y=-0.18*x+0.81*y+10;
                                break;
                        case 1:
                                x=0.4*x+0.4*y;
                                y=-0.1*x+0.4*y;
                                break;
                        case 2:
                                x=-0.4*x-0.4*y;
                                y=-0.1*x+0.4*y;
                                break;
                        case 3:
                                x=-0.1*x;
                                y=0.44*x+0.44*y-2;
                                break;
                }
                putpixel(screen,(SCREEN_W>>1)-(SCALE*x),(SCREEN_H-35)-(SCALE*y),GREEN_COLOUR);
        }
}
```

Fraktale - IFS

Funkcja `draw_with_ifs()` jest odpowiedzialna za narysowanie fraktala na ekranie. Poprzez jej dwa parametry przekazywane są współrzędne punktu początkowego (nie jest on rysowany), od których rozpocznie się wyliczanie kolejnych punktów. Na początku funkcji definiowana jest stała `GREEN_COLOUR`, której jest przypisywana wartość kodu koloru zwrócona przez wywołanie funkcji `makecol()`. Jest to kod koloru zielonego. Obliczenia współrzędnych kolejnych punktów fraktala, wraz z jego rysowaniem są wykonywane wewnątrz pętli `while`, która powtarza te czynności do momentu, kiedy użytkownik wciśnie klawisz `Esc` lub `q`. Najpierw jest losowane jedno z przekształceń afinicznych. Następnie jest ono użyte do wyznaczenia współrzędnych nowego punktu fraktala. Potem te współrzędne są przekształcane na współrzędne piksela, który będzie należał do obrazu fraktala. Te liczby przekazywane są jako drugi i trzeci argument wywołania funkcji `putpixel()`, która zmienia kolor piksela o takich współrzędnych. Jako pierwszy argument przekazywana jest zmienna `screen` będąca wskaźnikiem na bitmapę bezpośrednio związaną z ekranem komputera, a jako ostatni przekazywany jest kod nowego koloru piksela.

Fraktale - IFS

Ponieważ współrzędne punktu są liczbami zmiennoprzecinkowymi, a dodatkowo liczone są względem punktu $(0, 0)$ w „normalnym" układzie kartezjańskim, to muszą one być przeliczone na współrzędne odpowiadającego mu pikselowi na ekranie. Współrzędna pozioma piksela jest obliczana poprzez wyznaczenie połowy szerokości ekranu (zastosowano tu operator >>), a następnie odjęcie przeskalowanej wartości współrzędnej poziomej punktu. Otrzymana liczba jest niejawnie rzutowana na typ `int`. Podobnie przekształcana jest składowa pionowa. Najpierw jest wyznaczana liczba o 35 (wartość dobrana metodą prób i błędów) mniejsza od wysokości ekranu określonej w pikselach. Następnie od tej wielkości odejmowana jest przeskalowana składowa współrzędnej pionowej punktu. Dzięki tym translacjom obraz fraktala jest wyświetlony na środku ekranu i nie jest odwrócony.

Fraktale - IFS

```
int main(void) {
        if(initialize(GFX_AUTODETECT_FULLSCREEN,WIDTH,HEIGHT)<0)
                return -1;
        draw_with_ifs(0.0,0.0);
        allegro_exit();
        return 0;
}
END_OF_MAIN()
```

Fraktale - IFS

W funkcji `main()` najpierw wywoływana jest funkcja `initialize()`. Jej pierwszym argumentem wywołania jest `GFX_AUTODETECT_FULLSCREEN`, co oznacza, że będzie użyty w programie tryb pełnoekranowy, jeśli uda się go zainicjować. Dwa pozostałe argumenty to odpowiednio szerokość i wysokość ekranu w pikselach, wyrażona poprzez odpowiednie stałe. Po wykonaniu inicjacji pracy biblioteki Allegro wywoływana jest funkcja `draw_with_ifs()`. Jej argumenty to współrzędne punktu początkowego. Po jej zakończeniu uruchamiana jest funkcja `allegro_exit()`, która finalizuje pracę biblioteki.

Fraktale - IFS

Przedstawiony program tworzy na ekranie komputera fraktal nazywany choinką Za pomocą układu iterowanych funkcji możliwe jest tworzenie również innych obiektów tego typu, takich jak np. paproć Barnsley'a. W tym przypadku potrzebny jest nie tylko inny zestaw współczynników dla układów odwzorowań afinicznych, ale również trochę odmienny sposób losowego ich wybierania. Fraktale przypominające rośliny można uzyskać także za pomocą języków formalnych, które nazywają się L-Systemami od nazwiska ich twórcy, węgierskiego biologa i botanika, Aristida Lidenmayera. Ich zastosowanie w grafice komputerowej zaproponował i spopularyzował polski informatyk pracujący w Kanadzie, Przemysław Prusinkiewicz. Fraktale wykorzystywane są np. w grach komputerowych do generowania tła przypominającego rzeczywiste krajobrazy.

Fraktale - Zbiór Mandelbrota

Zbiór Mandelbrota jest zbiorem punktów płaszczyzny, dla których ciąg dany następującym równaniem rekurencyjnym („samopowtarzalnym”) jest zbieżny:

$$\begin{cases} z_0 = 0 \\ z_{n+1} = z_n^2 + c \end{cases}$$

W tym równaniu z jest zmienną zespoloną, a c stałą zespoloną. Zbiór ten został odkryty przez francuskiego matematyka Benoita Mandelbrota urodzonego w Warszawie w 1924 roku. Obraz tego fraktala jest tworzony poprzez przeskalowanie współrzędnych odpowiednio dużej liczby punktów płaszczyzny zespolonej, tak aby ich część rzeczywista i urojona (składowa pozioma i pionowa) miały wartości należące odpowiednio do przedziałów $(-2,5;1)$ i $(-1;1)$, a następnie podstawieniu ich za stałą c i wyznaczeniu pewnej liczby początkowych wyrazów opisanego wcześniej ciągu. Jeśli wszystkie te wyrazy mają moduł mniejszy od 2 ($|z_n| \leqslant 2$), to punkt ten należy do zbioru.

Fraktale - Zbiór Mandelbrota

```
#include<allegro.h>
#include<allegro/keyboard.h>

#define WIDTH 1366

#define HEIGHT 768

#define MAXITER 8000
```

Fraktale - Zbiór Mandelbrota

Początek programu generującego na ekranie zbiór Mandelbrota jest podobny do początku programu tworzącego fraktal za pomocą IFS. Jednak zamiast stałej skalującej jest zdefiniowana stała MAXITER określająca ile początkowych punktów ciągu, którego definicja została podana na poprzednim slajdzie będzie obliczonych przez program. Nie są również włączone pliki nagłówkowe związane z obsługą generatora liczb pseudolosowych.

Fraktale - Zbiór Mandelbrota

```
int initialize(int card, int width, int height)
{
        if(allegro_init()) {
                allegro_message("allegro_init: %s\n",allegro_error);
                return -1;
        }
        if(install_keyboard()) {
                allegro_message("install_keyboard: %s\n",allegro_error);
                allegro_exit();
                return -1;
        }
        set_color_depth(32);
        if(set_gfx_mode(card,width,height,0,0)) {
                allegro_message("%s\n",allegro_error);
                allegro_exit();
                return -1;
        }
        return 0;
}
```

Fraktale - Zbiór Mandelbrota

Funkcja inicjująca jest podobna do tej zaprezentowanej w poprzednim programie. Jedyna różnica polega na tym, że tym razem nie jest inicjowany generator liczb pseudolosowych.

Fraktale - Zbiór Mandelbrota

```
double scale_x0(int x0)
{
        return 3.5*(((double)x0)/(SCREEN_W-1))-2.5;
}
```

Fraktale - Zbiór Mandelbrota

```
double scale_y0(int y0)
{
        return 2.0*(((double)y0)/(SCREEN_H-1))-1.0;
}
```

Fraktale - Zbiór Mandelbrota

Funkcje `scale_x0()` i `scale_y0()` przeskalowują przekazane im przez parametry składowe współrzędnych piksela na współrzędne punktu leżącego w płaszczyźnie $(-2,5;1) \times (-1;1)$.

Fraktale - Zbiór Mandelbrota

```
unsigned int calculate_mandelbrot(int xp, int yp)
{
        double x,y,x2,y2;
        unsigned int iteration = 0;
        double x0 = scale_x0(xp);
        double y0 = scale_y0(yp);

        x=y=x2=y2=0.0;

        while(x2+y2<=4.0 && iteration<MAXITER) {
                double tmp = x2-y2+x0;
                y=2.0*x*y+y0;
                x=tmp;
                iteration++;
                x2=x*x;
                y2=y*y;
        }

        return iteration==MAXITER?0:iteration;
}
```

Fraktale - Zbiór Mandelbrota

Funkcja `calculate_mandelbrot()` wylicza kolejne wyrazy ciągu dla punktu, którego współrzędne zostały jej przekazane przez parametry. Najpierw te współrzędne są skalowane z użyciem funkcji `scale_x0()` i `scale_y0()`, następnie w pętli `while` wyznaczane są kolejne wyrazy ciągu, tak długo, aż licznik iteracji zrówna się wartością ze stałą `MAXITER` lub gdy okaże się, że moduł bieżącego wyrazu ciągu jest większy lub równy dwa. Ten ostatni test sprawdzany jest za pomocą warunku `x2+y2<=4.0`. Można go wyprowadzić ze wzoru $|z_n \leqslant 2|$ następująco: $|z_n \leqslant 2| \Rightarrow |x_n + i \cdot y_n| \leqslant 2 \Rightarrow \sqrt{x_n^2 + y_n^2} \leqslant 2 \Rightarrow x_n^2 + y_n^2 \leqslant 4$. Wewnątrz pętli wyliczane są kolejne wyrazy ciągu i liczona jest liczba iteracji przez tę pętlę wykonanych. Wyrażenia związane z obliczeniami wyrazów ciągu można wyprowadzić przyjmując, że $z = x + i \cdot y$, a $c = x_0 + i \cdot y_0$. Wtedy część rzeczywista kolejnego wyrazu ciągu wynosi $x^2 - y^2 + x_0$, a część urojona $2 \cdot x \cdot y + y_0$. Funkcja zwraca liczbę iteracji wykonanych przez pętlę dla danego punktu lub liczbę zero, jeśli ich liczba osiągnęła wartość stałej `MAXITER`.

Fraktale - Zbiór Mandelbrota

```
int main(void) {
        if(initialize(GFX_AUTODETECT_FULLSCREEN,WIDTH,HEIGHT)<0)
                return -1;
        unsigned int x,y;
        unsigned char color;
        for(y=0;y<SCREEN_H;y++)
                for(x=0;x<SCREEN_W;x++) {
                        color=calculate_mandelbrot(x,y);
                        putpixel(screen,x,y,palette_color[color]);
                }
        while(!(key[KEY_Q]||key[KEY_ESC]))
                ;
        allegro_exit();
        return 0;
}
END_OF_MAIN()
```

Fraktale - Zbiór Mandelbrota

W funkcji `main()` programu, oprócz inicjacji i finalizacji działania biblioteki Allegro wykonywane jest generowanie obrazu zbioru Mandelbrota. W pętlach `for` dla każdego piksela obrazu obliczane są przez `calculate_mandelbro` kolejne wyrazy ciągu i zwracana jest liczba iteracji, po których pętla `while` została zakończona lub zero. Na podstawie tej zwróconej wartości wyznaczany jest kod koloru piksela. W programie skorzystano z gotowej palet kolorów zapisanej w tablicy `palette_color`. Ma ona 256 elementów, zatem liczbę iteracji należy przekształcić na liczbę z zakresu od `0` do 255 włącznie. To przekształcenie dokonywane jest poprzez zapis wyniku funkcji `calculate_mandelbrot()` do zmiennej typu `unsigned char`. Po wygenerowaniu obrazu fraktala program oczekuje w pętli `while` aż użytkownik naciśnie klawisz `Esc` lub `q`. Proszę zwrócić uwagę, że w tej pętli poza badaniem stanu wymienionych klawiszy nie są wykonywane inne czynności.

Fraktale - Zbiór Mandelbrota 2

Obliczenia związane z wyznaczaniem zbioru Mandelbrota powadzone są w dziedzinie liczby zespolonych. Standard ISO C99 definiuje elementy języka, które umożliwiają bezpośrednią obsługę takich liczb. Są one zgromadzone w pliku `complex.h` i użyte w następnym opisywanym programie, który również generuje zbiór Mandelbrota. Te elementy to makro `complex`, które pozwala stworzyć typ `double complex`, stała `I`, której wartość wynosi $\sqrt{-1}$ oraz funkcja `cabs()` obliczająca moduł z liczby zespolonej. Na zmiennych typu `double complex` można wykonywać wszystkie podstawowe działania arytmetyczne korzystając z tych samych operatorów, co w przypadku innych liczbowych typów danych. Ponieważ kod programu jest bardzo podobny do tego opisywanego na poprzednich slajdach, to dalej zostaną opisane tylko różnice między nimi.

Fraktale - Zbiór Mandelbrota 2

```
#include<allegro.h>
#include<allegro/keyboard.h>
#include<complex.h>

#define WIDTH 1366

#define HEIGHT 768

#define MAXITER 8000
```

Fraktale - Zbiór Mandelbrota 2

Do programu włączany jest dodatkowo plik nagłówkowy `complex.h`, który zawiera definicje elementów związanych z obsługą liczb zespolonych.

Fraktale - Zbiór Mandelbrota 2

```
int initialize(int card, int width, int height)
{
    if(allegro_init()) {
        allegro_message("allegro_init: %s\n",allegro_error);
        return -1;
    }
    if(install_keyboard()) {
        allegro_message("install_keyboard: %s\n",allegro_error);
        allegro_exit();
        return -1;
    }
    set_color_depth(32);
    if(set_gfx_mode(card,width,height,0,0)) {
        allegro_message("%s\n",allegro_error);
        allegro_exit();
        return -1;
    }
    return 0;
}
```

Fraktale - Zbiór Mandelbrota 2

```
double scale_x0(int x0)
{
    return 3.5*(((double)x0)/(SCREEN_W-1))-2.5;
}
```

Fraktale - Zbiór Mandelbrota 2

```
double scale_y0(int y0)
{
    return 2.0*(((double)y0)/(SCREEN_H-1))-1.0;
}
```

Fraktale - Zbiór Mandelbrota 2

```
unsigned int calculate_mandelbrot(int xp, int yp)
{
    double complex z;
    unsigned int iteration = 0;
    double complex c = scale_x0(xp) + I*scale_y0(yp);

    z=0.0+0.0*I;

    while(cabs(z)<=2.0 && iteration<MAXITER) {
        z = z*z+c;
        iteration++;
    }

    return iteration==MAXITER?0:iteration;
}
```

Fraktale - Zbiór Mandelbrota 2

Funkcja `calculate_mandelbrot()` używa zmiennych typu `double complex` do wyliczenia kolejnych wyrazów ciągu. Najpierw ze współrzędnych piksela przekazanych przez parametry jest tworzona wartość odpowiadająca stałej `c`. Następnie w zmiennej `z` umieszczana jest wartość pierwszego wyrazu ciągu, a w pętli `while` liczone są kolejne elementy tego ciągu i zliczana jest liczba jej powtórzeń. Zapis kodu funkcji jest bardziej czytelny niż w poprzednim programie, ale czas jego działania jest dłuższy.

Fraktale - Zbiór Mandelbrota 2

```
int main(void)
{
    if(initialize(GFX_AUTODETECT_FULLSCREEN,WIDTH,HEIGHT)<0)
        return -1;
    unsigned int x,y;
    unsigned char color;
    for(y=0; y<SCREEN_H; y++)
        for(x=0; x<SCREEN_W; x++) {
            color=calculate_mandelbrot(x,y);
            putpixel(screen,x,y,palette_color[color]);
        }
    while(!(key[KEY_Q]||key[KEY_ESC]))
        ;
    allegro_exit();
    return 0;
}
END_OF_MAIN()
```

Fraktale - Zbiór Mandelbrota 2

O tworzeniu zbioru Mandelbrota i innych metodach generowania fraktali można przeczytać na stronie http://lodev.org/cgtutor/. Przestawiony na wykładzie opis fraktala Mandelbrota jest częściowo oparty na zawartości polskich i angielskich stron Wikipedii.

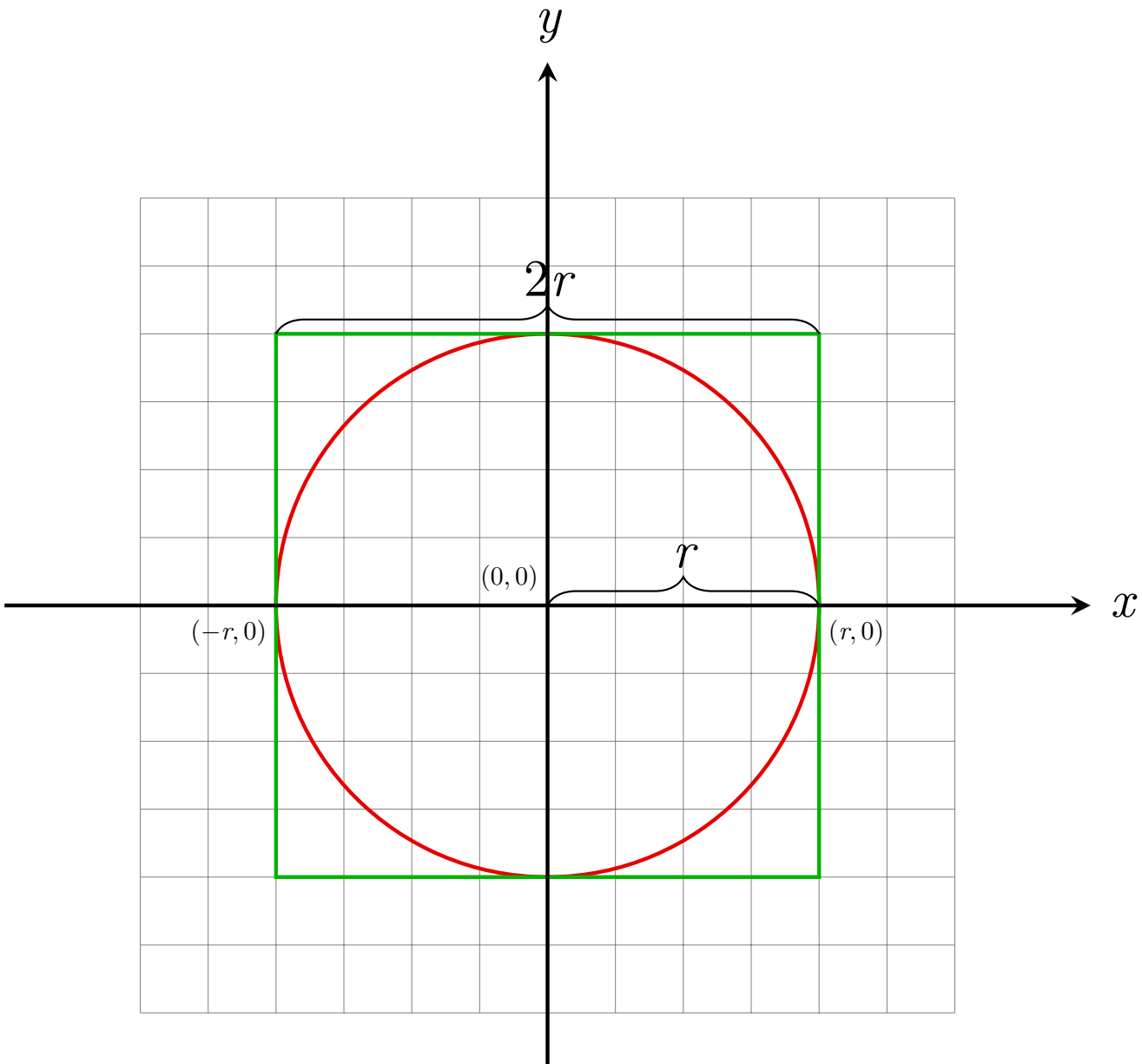
Wyznaczanie liczby π

Liczba π jest jedną z najczęściej pojawiających się w matematyce stałych. Choć jej definicja jest prosta - jest to stosunek długości obwodu koła do długości jego promienia - to wyznaczenie jej przybliżonej wartości nie należy do prostych. Istnieje wiele metod obliczania tej stałej. Jako następny zostanie zaprezentowany program, który dokonuje takiego obliczenia używając jednej z metod statystycznych nazywanych metodami Monte Carlo. Pierwsze z nich odkrył i opracował na potrzeby prac prowadzonych w ramach projektu Manhattan polski matematyk Stanisław Ulam.

Wyznaczanie liczby π

Zastosowana w programie metoda nie należy do szybkich, gdyż jej kroki wymagają wielokrotnego powtórzenia, aby otrzymać choć zgrubne oszacowanie wartości π. Niemniej jednak, jest to metoda, która może być ciekawie zobrazowana. W tej metodzie zakłada się, że dane jest koło o promieniu o długości r wpisane w kwadrat. Środek tego koła znajduje się w początku układu współrzędnych. Bok tego kwadratu ma długość $2r$ (proszę popatrzeć na ilustrację na następnym slajdzie). Stosunek pola kwadratu (P_{kw}) do pola koła (P_{ko}) wynosi zatem $\frac{P_{kw}}{P_{ko}} = \frac{(2 \cdot r)^2}{\pi \cdot r^2}$. Gdybyśmy znali wartość pola kwadratu i pola koła to moglibyśmy wyliczyć wartość liczby π ze wzoru $\pi = \frac{4 \cdot P_{ko}}{P_{kw}}$. Możemy przybliżyć te wartości losując punkty wewnątrz kwadratu i sprawdzając, czy należą one także do koła. Miarą pola kwadratu będzie liczba wszystkich wylosowanych punktów, a miarą pola koła liczba punktów, które należą także do niego. Więcej informacji o tej metodzie można znaleźć na stronie Eve Astrid Andersson (http://www.eveandersson.com/pi/).

Wyznaczanie liczby π



Wyznaczanie liczby π

```
#include<allegro.h>
#include<allegro/keyboard.h>
#include<stdlib.h>
#include<stdbool.h>
#include<time.h>
#include<math.h>

#define WIDTH 1366
#define HEIGHT 768
```

Wyznaczanie liczby π

Początek programu jest podobny do tych zaprezentowanych poprzednio. Oprócz plików nagłówkowych związanych z biblioteką Allegro włączane są również pliki, w których zdefiniowane są funkcje konieczne do obsługi generatora liczb pseudolosowych, zawierające definicje użytych funkcji matematycznych oraz typu `bool`.

Wyznaczanie liczby π

```
int initialize(int card, int width, int height)
{
    srand(time(NULL));
    if(allegro_init()) {
        allegro_message("allegro_init(): %s\n",allegro_error);
        return -1;
    }
    if(install_keyboard()) {
        allegro_message("install_keyboard(): %s\n",allegro_error);
        allegro_exit();
        return -1;
    }
    set_color_depth(32);
    if(set_gfx_mode(card,width,height,0,0)) {
        allegro_message("set_gfx_mode(): %s\n",allegro_error);
        allegro_exit();
        return -1;
    }
    return 0;
}
```

Wyznaczanie liczby π

Funkcja `initialize()` jest tak samo zdefiniowana, jak w programie, który jako pierwszy został zaprezentowany na tym wykładzie.

Wyznaczanie liczby π

```
bool is_in_disc(double x, double y, const double radius)
{
    return sqrt(pow(x,2)+pow(y,2))<=radius;
}
```

Wyznaczanie liczby π

Funkcja `is_in_disc()` sprawdza, czy punkt, którego współrzędne zostały jej przekazane przez dwa pierwsze parametry należy do koła o promieniu, którego długość została jej przekazana przez trzeci parametr. Zgodnie z definicją koła, wszystkie punkty należące do niego położne są w stosunku do jego środka w odległości równiej lub mniejszej od długości promienia tego koła. Ponieważ środek koła, które wykorzystywane jest w opisywanej metodzie wyznaczania liczby π, pokrywa się ze środkiem układu współrzędnych, to do wyznaczania odległości punktu od środka koła możemy użyć następującej postaci wzoru Euklidesa $\sqrt{x^2 + y^2}$. Jeśli uzyskana wartość jest mniejsza lub równa długości promienia koła, to punkt należy do koła, w przeciwnym przypadku nie należy. Użyta w kodzie funkcja `pow()` wykonuje działanie potęgowania. Przyjmuje ona dwa argumenty wywołania typu `double`. Przez pierwszy przekazywana jest podstawa potęgi, a przez drugi jej wykładnik. Zwracana przez nią wartość też jest typu `double`.

Wyznaczanie liczby π

```
void draw_pi(void)
{
    const int GREEN_COLOUR = makecol(0,255,0),
              RED_COLOUR = makecol(255,0,0),
              WHITE_COLOUR = makecol(255,255,255),
              SCREEN_MIDDLE_X = SCREEN_W>>1,
              SCREEN_MIDDLE_Y = SCREEN_H>>1;
    const double RADIUS = 300.0;
    unsigned long int in_square=0, in_disc=0;
    while(!keypressed()) {
        double x=-RADIUS+rand()%600+rand()/(1.0+RAND_MAX);
        double y=-RADIUS+rand()%600+rand()/(1.0+RAND_MAX);
        in_square++;
        if(is_in_disc(x,y,RADIUS)) {
            putpixel(screen,x+SCREEN_MIDDLE_X,y+SCREEN_MIDDLE_Y,RED_COLOUR);
            in_disc++;
        } else
            putpixel(screen,x+SCREEN_MIDDLE_X,y+SCREEN_MIDDLE_Y,GREEN_COLOUR);
        if(!(in_square%100000))
            textprintf_ex(screen,font,550,50,WHITE_COLOUR,0,"The PI number is: %1.20lf",
                (4.0*in_disc)/in_square);
    }
}
```

Wyznaczanie liczby π

Funkcja `draw_pi()` obrazuje obliczanie liczby π i wypisuje na ekranie bieżące przybliżenie jej wartości. Na jej początku definiowane są wartości trzech stałych określających kody kolorów odpowiednio: zielonego, czerwonego i białego. Kolejne dwie stałe są składowymi współrzędnych środka ekranu. Będą one użyte do translacji współrzędnych punktów na współrzędne pikseli tak, aby obraz, który one tworzą był rysowany w centrum ekranu. Ostatnią stałą jest stała określająca długość promienia. Przyjmujemy, że będzie ona wynosiła 300 punktów. Deklarowane są również dwie zmienne, które będą zawierały liczbę punków należących do kwadratu (`in_square`) i do koła (`in_disc`). W pętli `while`, która wykonywana jest tak długo, jak długo użytkownik nie naciśnie dowolnego klawisza, losowane są najpierw współrzędne punktów należących do kwadratu i zwiększana jest wartość zmiennej będącej ich licznikiem. Następnie w instrukcji warunkowej sprawdzane jest, czy wylosowany punkt należy także do koła. Jeśli tak, to piksel, który odpowiada temu punktowi jest zaznaczany na czerwono i zwiększana jest wartość zmiennej będącej licznikiem punków należących do koła. W przeciwnym przypadku piksel zaznaczany jest na zielono.

Wyznaczanie liczby π

Proszę zwrócić uwagę, że do współrzędnych punktu dodawane są stałe określające współrzędne środka ekranu. Dodatkowo składowe współrzędnych tych punktów, które są typu `double` niejawnie rzutowane są na typ `int`, co oznacza, że wiele z wylosowanych punktów może być odwzorowanych na ten sam piksel. Komunikat o wyznaczonej wartości liczby π jest wypisywany na ekranie co `100.000` iteracji pętli za pomocą funkcji `textprintf_ex()`. Sama wartość wyznaczana jest ze wzoru przytoczonego na początku opisu programu. Wartości pola kwadratu (P_{kw}) odpowiada wartość zmiennej `in_square`, a wartości pola koła (P_{ko}) wartość zmiennej `in_disc`.

Wyznaczanie liczby π

```
int main(void)
{
    if(initialize(GFX_AUTODETECT_FULLSCREEN,WIDTH,HEIGHT))
        return -1;
    draw_pi();
    allegro_exit();
    return 0;
}
END_OF_MAIN()
```

Wyznaczanie liczby π

W funkcji `main()` wywoływane są wcześniej zdefiniowane w programie funkcje.

Sinusoida

Kolejny program rysuje na ekranie wykres funkcji sinus. Na jej przykładzie prezentowany jest ogólny sposób rysowania wykresów funkcji. Nie wystarczy jedynie obliczyć współrzędne punktów należących do wykresu i zmienić kolor pikselom odpowiadającym tym punktom. W tej sposób otrzymalibyśmy jedynie wykres dyskretny - składający się z wielu niepołączonych ze sobą punktów. Poprawnym rozwiązaniem jest stworzenie wykresu złożonego z bardzo małych odcinków połączonych ze sobą końcami.

Sinusoida

```
#include<allegro.h>
#include<allegro/keyboard.h>
#include<math.h>

#define WIDTH 1366
#define HEIGHT 768
```

Sinusoida

Początek programu jest podobny jak w poprzednich programach. Włączono plik nagłówkowy `math.h` z uwagi na konieczność użycia podprogramu obliczającego wartość funkcji sinus.

Sinusoida

```
int initialize(int card, int width, int height)
{
    if(allegro_init()) {
        allegro_message("allegro_init(): %s\n",allegro_error);
        return -1;
    }
    if(install_keyboard()) {
        allegro_message("install_keyboard(): %s\n",allegro_error);
        allegro_exit();
        return -1;
    }
    set_color_depth(32);
    if(set_gfx_mode(card,width,height,0,0)) {
        allegro_message("set_gfx_mode(): %s\n",allegro_error);
        allegro_exit();
        return -1;
    }
    return 0;
}
```

Sinusoida

Funkcja `initialize()` jest taka sama jak w przypadku wcześniej zaprezentowanych programów, które nie wymagały użycia generatora liczb pseudolosowych.

Sinusoida

```
void draw_sinus(BITMAP* bitmap)
{
    const double DEGREE_TO_RADIAN = M_PI/180.0, HALF_OF_SCREEN = SCREEN_H>>1, X_RATIO = SCREEN_W/360.0;
    double x_start=0.0,y_start=0.0,x_end,y_end;
    const int GREEN_COLOUR = makecol(0,255,0);
    for(x_end=1;x_end<=360;x_end++) {
            y_end = HALF_OF_SCREEN*sin(x_end*DEGREE_TO_RADIAN);
            line(bitmap,x_start*X_RATIO,HALF_OF_SCREEN-y_start,X_RATIO*x_end,HALF_OF_SCREEN-y_end,
                GREEN_COLOUR);
            x_start = x_end;
            y_start = y_end;
    }
}
```

Sinusoida

Funkcja `draw_sinus()` rysuje sinusoidę na ekranie monitora. Na jej początku zdefiniowane są trzy stałe. Stała `DEEGRE_TO_RADIAN` będzie służyła do przeliczania stopni na radiany. Te ostatnie są jednostkami, w jakich mierzona jest miara kąta przyjmowanego jako argumenty wywołania przez funkcję `sin()`. Stała `HALF_OF_SCREEN` (można jej typ zdefiniować jako `int` zamiast `double`) ma wartość połowy wysokości ekranu mierzonej w pikselach, z kolei stała o nazwie `X_RATIO` to wartość stosunku szerokości ekranu mierzonej w pikselach do liczby stopni w pełnym kącie. Innymi słowy, ta stała określa ile pikseli obrazu w poziomie przypada na jeden stopień. Ostatnia stała zawiera kod koloru zielonego. Jest to kolor w jakim zostanie narysowana sinusoida. Zmienne `x_start` i `y_start` służą do przechowywania składowych współrzędnych punktu początkowego bieżąco rysowanego odcinka, a zmienne `x_end` i `y_end` do przechowywania składowych współrzędnych punktu końcowego tego odcinka.

Sinusoida

Wykres jest rysowany w pętli `for`. Punkt początkowy pierwszego odcinka składowego wykresu ma współrzędne $(0, 0)$. Współrzędne punktu końcowego, a dokładniej ich składowa pionowa, wyliczane są podczas pierwszej iteracji pętli. Ponieważ wartości funkcji sinus należą do przedziału $[-1, 1]$, to gdybyśmy próbowali bezpośrednio użyć ich do narysowania wykresu otrzymalibyśmy na ekranie „poszarpany" odcinek. Dlatego składowa pionowa punktu końcowego odcinka jest mnożona przez połowę wysokości ekranu. Dzięki temu wykres obejmuje całą wysokość ekranu. Argumentem wywołania funkcji `sin()` jest miara kąta wyrażona w radianach, dlatego program mnoży miarę kąta wyrażoną w stopniach przez odpowiednią stałą i dopiero wynik takiego wyrażenia przekazuje do tej funkcji. Po wyznaczeniu współrzędnych punktu końcowego odcinka jest on rysowany za pomocą funkcji `line()`. Składowe poziome współrzędnych punktów początkowego i końcowego odcinka otrzymywane są poprzez pomnożenie miary kąta wyrażonej w stopniach przez stałą `X_RATIO`, aby wykres zajmował całą szerokość ekranu. Składowe pionowe odejmowane są od połowy wysokości ekranu.

Sinusoida

Dzięki zabiegowi opisanemu na poprzednim slajdzie otrzymany wykres będzie wyśrodkowany w pionie oraz nie będzie odwrócony. W kolejnej iteracji pętli wyznaczane są współrzędne punktu końcowego drugiego odcinka składowego wykresu. Współrzędne punktu końcowego pierwszego odcinka stają się współrzędnymi punktu początkowego odcinka drugiego. Czynności te powtarzane są tak długo, aż zostanie narysowany na ekranie wykres pełnego okresu funkcji sinus.

Sinusoida

```
void wait_for_any_key(void)
{
    clear_keybuf();
    while(!keypressed())
        ;
}
```

Sinusoida

Funkcja `wait_for_any_key()` wstrzymuje działanie programu do chwili, kiedy użytkownik naciśnie dowolny klawisz na klawiaturze. Aby ta czynność została wykonana poprawnie najpierw czyszczona jest zawartość bufora klawiatury przy użyciu funkcji `clear_keybuf()`. Potem funkcja czeka w pętli `while` aż użytkownik naciśnie jakikolwiek klawisz. Nie wykonuje przy tym żadnych dodatkowych czynności.

Sinusoida

```
int main(void)
{
    if(initialize(GFX_AUTODETECT_FULLSCREEN,WIDTH,HEIGHT))
        return -1;
    draw_sinus(screen);
    wait_for_any_key();
    allegro_exit();
    return 0;
}
END_OF_MAIN()
```

Sinusoida

W funkcji `main()` wywoływane są wszystkie zdefiniowane w programie funkcje.

Animacja - prostokąt

Kolejny program generuje na ekranie komputera animację, w której prostokąt cyklicznie porusza się od lewej krawędzi ekranu do prawej, do momentu naciśnięcia przez użytkownika klawisza q lub `Esc`.

Animacja - prostokąt

```
#include<allegro.h>
#include<allegro/keyboard.h>

#define WIDTH 1366
#define HEIGHT 768
#define NUMBER_OF_PAGES 4

BITMAP *pages[NUMBER_OF_PAGES];
```

Animacja - prostokąt

W stosunku do programu rysującego sinusoidę w tym programie nie ma użytego pliku nagłówkowego z definicjami funkcji matematycznych, ale zdefiniowano stałą określającą liczbę stron, czyli bitmap używanych przez mechanizm animacji w bibliotece Allegro, oraz zadeklarowano z jej użyciem tablicę, której elementy są wskaźnikami na struktury opisujące bitmapy.

Animacja - prostokąt

```
int initialize(int card, int width, int height)
{
    if(allegro_init()) {
        allegro_message("allegro_init(): %s\n",allegro_error);
        return -1;
    }
    if(install_keyboard()) {
        allegro_message("install_keyboard(): %s\n",allegro_error);
        allegro_exit();
        return -1;
    }
    set_color_depth(32);
    if(set_gfx_mode(card,width,height,0,NUMBER_OF_PAGES*height)) {
        allegro_message("set_gfx_mode(): %s\n",allegro_error);
        allegro_exit();
        return -1;
    }
    return 0;
}
```

Animacja - prostokąt

Funkcja `initialize()` jest podobna do funkcji we wcześniej przedstawionych programach, które nie korzystały z generatora liczb pseudolosowych, ale dodatkowo, w wywołaniu funkcji `set_gfx_mode()` jako dwa ostanie argumenty przekazywane są wymiary ekranu wirtualnego. Jest to działanie niezbędne do uruchomienia mechanizmu animacji i do tworzenia nowych bitmap. Proszę zwrócić uwagę, że wystarczy, aby jeden z tych argumentów był różny od zera.

Animacja - prostokąt

```
int create_pages_array(BITMAP *pages[NUMBER_OF_PAGES])
{
    int i;
    for(i=0; i<NUMBER_OF_PAGES; i++) {
        pages[i] = create_video_bitmap(SCREEN_W,SCREEN_H);
        if(pages[i]==NULL)
            return -1;
    }
    return 0;
}
```

Animacja - prostokąt

Funkcja `create_pages_array()` inicjuje w pętli elementy tablicy wskaźników na bitmapy, wcześniej tworząc te bitmapy. Jeśli którejś z nich nie uda się stworzyć, to działanie funkcji jest przerywane i zwraca ona wartość sygnalizującą wyjątek.

Animacja - prostokąt

```
void destroy_pages_array(BITMAP *pages[NUMBER_OF_PAGES])
{
    int i;
    for(i=0; i<NUMBER_OF_PAGES; i++)
        destroy_bitmap(pages[i]);
}
```

Animacja - prostokąt

Funkcja `destroy_bitmap_array()` działa odwrotnie niż opisana na poprzednim slajdzie `create_bitmap_array()` - usuwa bitmapy.

Animacja - prostokąt

```
void animate_rectangle(BITMAP *pages[NUMBER_OF_PAGES], int speed, int rectangle_width, int rectangle_height)
{
    int page_number = 0;
    int x = 0;
    clear_keybuf();
    const int YELLOW_COLOUR = makecol(255,255,0);
    while(key[KEY_ESC]==0&&key[KEY_Q]==0) {
        BITMAP *active_page = pages[page_number];
        clear_bitmap(active_page);
        rect(active_page,x,SCREEN_H>>1,rectangle_width+x,rectangle_height+(SCREEN_H>>1),YELLOW_COLOUR);
        x=(x+speed)%SCREEN_W;
        if(show_video_bitmap(active_page))
            return;
        page_number = (page_number+1)%NUMBER_OF_PAGES;
    }
}
```

Animacja - prostokąt

Funkcja `animate_rectangle()` tworzy animację prostokąta na ekranie. Przez jej pierwszy parametr przekazywana jest tablica wskaźników na strony, wartość drugiego określa szybkość animacji (prędkość prostokąta), a wartość dwóch ostatnich parametrów, rozmiary prostokąta. Dwie zmienne lokalne funkcji określają odpowiednio: indeks wskaźnika bieżącej strony aktywnej i składową poziomą współrzędnych lewego górnego rogu prostokąta. Zdefiniowana stała ma wartość kodu koloru żółtego. To jest kolor animowanego prostokąta. Przed rozpoczęciem pętli, w której tworzone są kolejne klatki animacji czyszczony jest bufor klawiatury. Wspomniana pętla wykonywana jest do chwili naciśnięcia przez użytkownika klawisza `Esc` lub `q`.

Animacja - prostokąt

We wnętrzu pętli `while` najpierw wybierany jest z tablicy wskaźnik na aktywną bitmapę, która następnie jest czyszczona (kolor jej wszystkich pikseli jest ustawiany na czarny). Potem rysowany jest na niej prostokąt, którego górna krawędź będzie widoczna na połowie wysokości ekranu. Następnie obliczane jest przesunięcie w poziomie lewego górnego rogu prostokąta na kolejnej klatce animacji. Ponieważ stosowana jest arytmetyka modularna, to po „zniknięciu" za prawą krawędzią ekranu prostokąt pojawi się ponownie po jego lewej stronie. Kolejną czynnością jest wyświetlenie na ekranie zawartości aktywnej bitmapy i wyznaczenie indeksu kolejnej, która zajmie jej miejsce w następnej iteracji pętli. Te indeksy też wyliczane są z użyciem arytmetyki modularnej, więc każda bitmapa jest używana co cztery iteracje - bo taką wartość w programie ma stała `NUMBER_OF_PAGES`.

Animacja - prostokąt

```
int main(void)
{
    if(initialize(GFX_AUTODETECT_FULLSCREEN,WIDTH,HEIGHT)<0)
        return -1;
    if(create_pages_array(pages))
        return -1;
    animate_rectangle(pages,1,100,50);
    destroy_pages_array(pages);
    allegro_exit();
    return 0;
}
END_OF_MAIN()
```

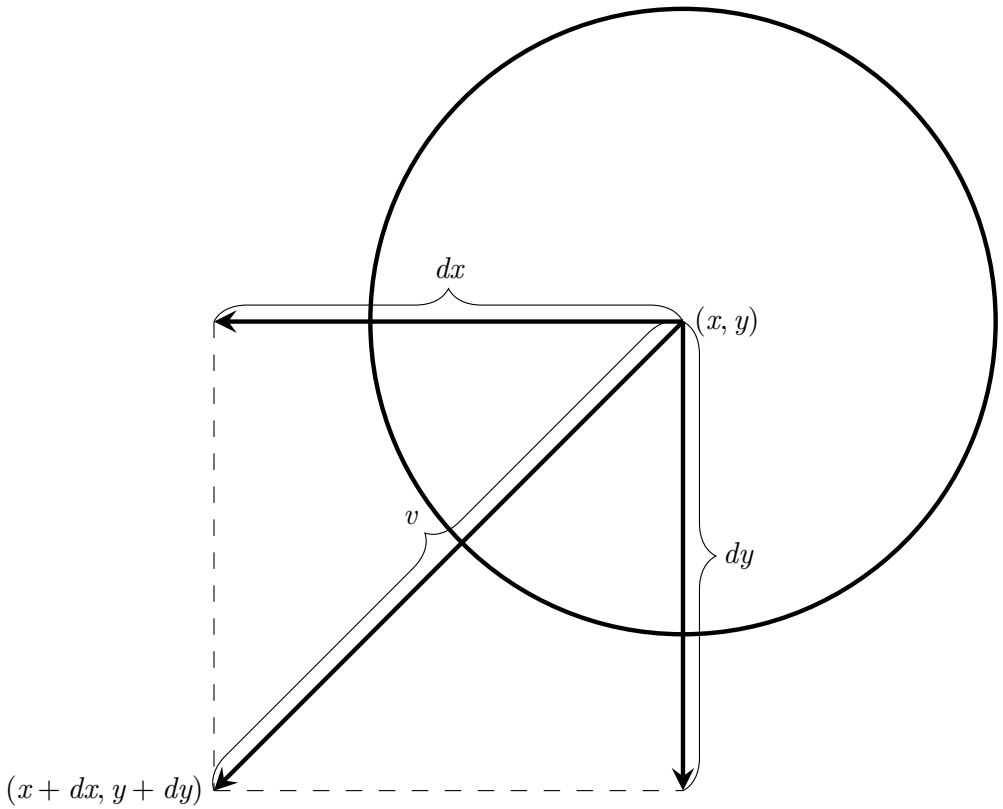
Animacja - prostokąt

W funkcji `main()` wywoływane są wcześniej zdefiniowane w programie funkcje. Proszę zwrócić uwagę na kolejność i umiejscowienie wywołań funkcji `create_pages_array()` oraz `destroy_pages_array()`.

Animacja piłki

Kolejny program również tworzy animację, lecz tym razem piłki, która odbija się od krawędzi ekranu zgodnie z zasadą, że kąt odbicia musi się równać kątowi padania. Aby móc opisać ruch piłki, która w programie będzie reprezentowana przez okrąg, musimy znać współrzędne położenia jej środka (x, y), składowe wektora prędkości (dx, dy), które będą wyznaczały położenie środka piłki w następnej klatce animacji i jednocześnie kąt oraz prędkość z jaką się ona porusza, a także długość jej promienia (r). Zależności między tymi danymi ilustruje rysunek znajdujący się na następnym slajdzie.

Animacja piłki

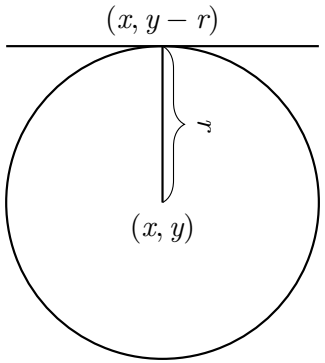


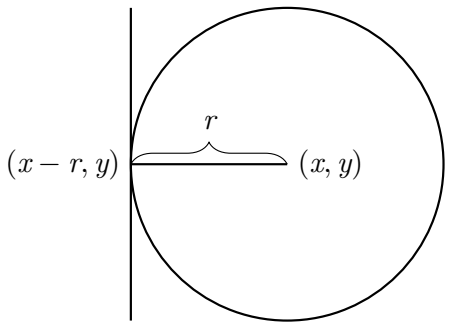
Animacja piłki

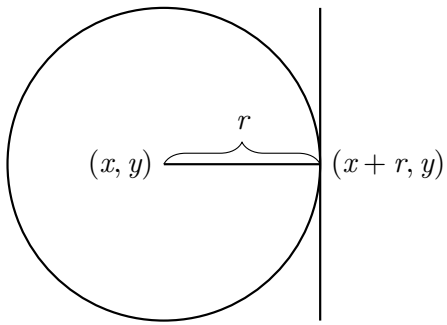
Zderzenie piłki z dowolną z krawędzi ekranu i tym samym konieczność zmiany kierunku jej ruchu wykrywane są poprzez porównanie składowych bieżących współrzędnych jej środka ze składowymi współrzędnych krawędzi pionowych bądź poziomych. Jeśli chcemy sprawdzić, czy piłka nie „dotyka" lewej krawędzi ekranu, to od składowej poziomej współrzędnych jej środka odejmujemy długość jej promienia i porównujemy otrzymaną wartość z zerem. Jeśli będzie ona równa lub mniejsza od zera, to piłka ma kontakt z krawędzią. Podobnie, dla prawej krawędzi dodajemy do składowej poziomej długość promienia piłki i sprawdzamy, czy wartość wynikowa jest większa lub równa stałej `SCREEN_W`[2]. Również w podobny sposób sprawdzamy, czy piłka ma kontakt z krawędziami poziomymi - odpowiednie wyrażenia porównujemy z zerem i wartością stałej `SCREEN_H`. Te zależności ilustruje następny slajd.

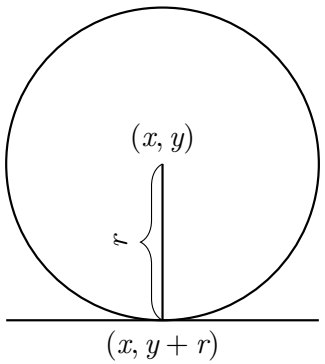
[2] Jeśli chcielibyśmy dokładniejszego porównania, to wartość tej stałej powinniśmy pomniejszyć o jeden. Podobna sytuacja dotyczy także krawędzi dolnej. Taka dokładność nie jest jednak konieczna.

Animacja piłki









Animacja piłki

```
#include<allegro.h>
#include<allegro/keyboard.h>
#include<stdlib.h>
#include<time.h>

#define WIDTH 1366
#define HEIGHT 768
#define NUMBER_OF_PAGES 4
```

Animacja piłki

Początek programu jest podobny do poprzednio zaprezentowanego, jednak tutaj są włączane pliki nagłówkowe konieczne do użycia generatora liczb pseudolosowych.

Animacja piłki

```
struct ball_data {
    int x,y,dx,dy;
    unsigned char radius;

} ball;

BITMAP *pages[NUMBER_OF_PAGES];
```

Animacja piłki

Oprócz tablicy wskaźników na struktury bitmap w programie zadeklarowana jest zmienna `ball`, która jest strukturą typu `ball_data`. Składowe tej struktury przechowują informacje o bieżącym położeniu środka piłki, składowych wektora prędkości oraz o długości promienia piłki.

Animacja piłki

```
int initialize(int card, int width, int height)
{
    srand(time(NULL));
    if(allegro_init()) {
        allegro_message("allegro_init(): %s\n",allegro_error);
        return -1;
    }
    if(install_keyboard()) {
        allegro_message("install_keyboard(): %s\n",allegro_error);
        allegro_exit();
        return -1;
    }
    set_color_depth(32);
    if(set_gfx_mode(card,width,height,0,NUMBER_OF_PAGES*height)) {
        allegro_message("set_gfx_mode(): %s\n",allegro_error);
        allegro_exit();
        return -1;
    }
    return 0;
}
```

Animacja piłki

Funkcja `initialize()` w porównaniu z funkcją o takiej samej nazwie z poprzedniego programu zawiera wywołanie funkcji inicjujących generator liczb pseudolosowych.

Animacja piłki

```
void set_ball(struct ball_data *ball)
{
    ball->radius = 20;
    ball->x = rand()%(SCREEN_W>>1)+ball->radius;
    ball->y = rand()%(SCREEN_H>>1)+ball->radius;
    ball->dx = 10;
    ball->dy = -10;
}
```

Animacja piłki

Funkcja `set_ball()` inicjuje przekazaną jej przez parametr wskaźnikowy strukturę opisującą piłkę. Długość promienia piłki jest określana na 20 pikseli, składowe wektora prędkości są inicjowane liczbami odpowiednio 10 i −10. Ponieważ są one równe co do wartości bezwzględnej, to piłka będzie poruszała się pod kątem $45°$. Ze względu na znaki tych składowych początkowo piłka będzie podążała w kierunku prawego górnego rogu ekranu. Początkowe współrzędne środka piłki losowane są w ten sposób, aby znalazła się ona całą powierzchnią wewnątrz ekranu i dodatkowo w jego pierwszej ćwiartce.

Animacja piłki

```
int create_pages_array(BITMAP *pages[NUMBER_OF_PAGES])
{
    int i;
    for(i=0; i<NUMBER_OF_PAGES; i++) {
        pages[i] = create_video_bitmap(SCREEN_W,SCREEN_H);
        if(pages[i]==NULL)
            return -1;
    }
    return 0;
}
```

Animacja piłki

```
void destroy_pages_array(BITMAP *pages[NUMBER_OF_PAGES])
{
    int i;
    for(i=0; i<NUMBER_OF_PAGES; i++)
        destroy_bitmap(pages[i]);
}
```

Animacja piłki

Funkcje przedstawione na dwóch poprzednich slajdach zostały opisane w poprzednim programie.

Animacja piłki

```
void draw_ball(BITMAP *page, struct ball_data ball)
{
    clear_bitmap(page);
    circle(page,ball.x,ball.y,ball.radius,makecol(255,255,0));
}
```

Animacja piłki

Funkcja `draw_ball()` rysuje na bitmapie, do której wskaźnik jest jej przekazywany przez pierwszy parametr, żółty okrąg, na podstawie danych ze struktury przekazanej jej przez drugi parametr. Wcześniej kolor każdego piksela tej bitmapy jest ustawiany na czarny.

Animacja piłki

```
void update_ball_position(struct ball_data *ball)
{
    ball->x += ball->dx;
    ball->y += ball->dy;
    if(ball->x-ball->radius<=0||ball->x+ball->radius>=SCREEN_W)
        ball->dx = -ball->dx;
    if(ball->y-ball->radius<=0||ball->y+ball->radius>=SCREEN_H)
        ball->dy = -ball->dy;

}
```

Animacja piłki

Funkcja `update_ball_position()` wylicza na podstawie zawartości przekazanej jej przez parametr struktury współrzędne środka piłki w następnej klatce animacji. Sprawdza ona także, czy piłka nie ma kontaktu z którąkolwiek z krawędzi ekranu. Jeśli tak jest, to piłka musi się od niej odbić, a więc zmienić kierunek poruszania. W programie takie zachowanie uzyskiwane jest następująco:

- Jeśli piłka ma kontakt z którąś z krawędzi poziomych, to zmieniany jest znak składowej pionowej jej wektora prędkości na przeciwny.
- Jeśli piłka ma kontakt z którąś z krawędzi pionowych, to zmieniany jest znak składowej poziomej jej wektora prędkości na przeciwny.

Animacja piłki

```
void animate_ball(struct ball_data ball, BITMAP *pages[NUMBER_OF_PAGES])
{
    int page_number = 0;
    clear_keybuf();
    while(key[KEY_ESC]==0&&key[KEY_Q]==0) {
        BITMAP *active_page = pages[page_number];
        draw_ball(active_page,ball);
        if(show_video_bitmap(active_page))
            return;
        update_ball_position(&ball);
        page_number = (page_number+1)%NUMBER_OF_PAGES;
    }
}
```

Animacja piłki

Funkcja `animate_ball()` pełni podobną rolę, co `animate_rectangle()` z poprzedniego programu. Metoda zatrzymywania pętli i użycia mechanizmu animacji pozostała taka sama jak w poprzednim programie. Zmiana polega na tym, że zamiast prostokąta rysowany jest okrąg poprzez wywołanie `draw_ball()`, a jego położenie na następnej klatce animacji jest wyznaczane za pomocą wywołania funkcji `update_ball_position()`.

Animacja piłki

```
int main(void)
{
    if(initialize(GFX_AUTODETECT_FULLSCREEN,WIDTH,HEIGHT)<0)
        return -1;
    if(create_pages_array(pages))
        return -1;
    set_ball(&ball);
    animate_ball(ball,pages);
    destroy_pages_array(pages);
    allegro_exit();
    return 0;
}
END_OF_MAIN()
```

Animacja piłki

W funkcji `main()` wywoływane są funkcje wcześniej zdefiniowane w programie. Przed funkcją animującą ruch piłki wywoływana jest funkcja `set_ball()`, która inicjuje strukturę opisującą dane tej piłki, a dokładniej okręgu, który ją reprezentuje w programie.

Proste efekty graficzne

Aby uzyskać ciekawe efekty graficzne, które mogą posłużyć np. jako tekstura dla obiektów w grach, nie trzeba wcale pisać skomplikowanych fragmentów kodu. Przy pomocy wyrażeń zbudowanych z prostych operatorów oraz składowych współrzędnych pikseli możemy uzyskać na ekranie ciekawie wyglądające wzory, co demonstruje następny program.

Proste efekty graficzne

```
#include<allegro.h>
#include<allegro/keyboard.h>

#define WIDTH 1366
#define HEIGHT 768
```

Proste efekty graficzne

Początek programu zawiera instrukcje włączające pliki nagłówkowe niezbędne do korzystania z biblioteki Allegro oraz definicje stałych określających rozmiar ekranu.

Proste efekty graficzne

```
int initialize(int card, int width, int height)
{
    if(allegro_init()) {
        allegro_message("allegro_init(): %s\n",allegro_error);
        return -1;
    }
    if(install_keyboard()) {
        allegro_message("install_keyboard(): %s\n",allegro_error);
        allegro_exit();
        return -1;
    }
    set_color_depth(32);
    if(set_gfx_mode(card,width,height,0,0)) {
        allegro_message("set_gfx_mode(): %s\n",allegro_error);
        allegro_exit();
        return -1;
    }
    return 0;
}
```

Proste efekty graficzne

Funkcja `initialize()` jest tak samo zdefiniowana, jak w poprzednio opisanych programach, które nie korzystają z generatora liczb pseudolosowych, ani z mechanizmu animacji.

Proste efekty graficzne

```
void wait_for_any_key(void)
{
    clear_keybuf();
    while(!keypressed())
        ;
}
```

Proste efekty graficzne

Funkcja `wait_for_key()` została opisana w programie rysującym sinusoidę.

Proste efekty graficzne

```
void draw_text(BITMAP *bitmap, FONT *font, const char *message)
{
    int red = makecol(255,0,0);
    textout_centre_ex(bitmap,font,message,SCREEN_W>>1,SCREEN_H>>1,red,-1);
}
```

Proste efekty graficzne

Funkcja `draw_text()` wypisuje na środku ekranu w kolorze czerwonym tekst przekazany jej przez ostatni z parametrów.

Proste efekty graficzne

```
void or_draw(BITMAP* bitmap)
{
    int x,y;
    for(x=0; x<SCREEN_W; x++)
        for(y=0; y<SCREEN_H; y++) {
            int expression = (x|y)&255;
            int colour = makecol(expression,expression,expression);
            putpixel(bitmap,x,y,colour);
        }
}
```

Proste efekty graficzne

Funkcja `or_draw()` rysuje na ekranie wzór składający się z powtarzających się „kafelków”. Uzyskany obraz jest w odcieniach szarości oraz czerni i bieli. Większość pikseli ma jednak jaśniejsze barwy. Dzieje się tak, ponieważ ich kolor jest wyznaczany na podstawie składowych ich współrzędnych z użyciem operatora sumy bitowej `|`. Wynik takiej sumy współrzędnych jest następnie poddawany działaniu modulo, w którym zamiast operatora reszty z dzielenia użyto operatora iloczynu bitowego. Dzięki temu uzyskiwana jest wartość z zakresu od `0` do `255`. Następnie ta wartość traktowana jest jak każda składowa koloru piksela i przekazywana jest do wywołania funkcji `makecol()`, która zwraca kod tego koloru. To działanie jest wykonywane dla każdego piksela na ekranie monitora.

Proste efekty graficzne

```
void and_draw(BITMAP* bitmap)
{
    int x,y;
    for(x=0; x<SCREEN_W; x++)
        for(y=0; y<SCREEN_H; y++) {
            int expression = (x&y)&255;
            int colour = makecol(expression,expression,expression);
            putpixel(bitmap,x,y,colour);
        }
}
```

Proste efekty graficzne

Funkcja `and_draw()` różni się tym od poprzednio opisywanej `or_draw()`, że do wyznaczenia koloru pikseli używany jest operator iloczynu bitowego. Dzięki temu uzyskany obraz jest podobny do poprzedniego, ale ciemniejszy.

Proste efekty graficzne

```
void xor_draw(BITMAP* bitmap)
{
    int x,y;
    for(x=0; x<SCREEN_W; x++)
        for(y=0; y<SCREEN_H; y++) {
            int expression = (x^y)&255;
            int colour = makecol(expression,expression,expression);
            putpixel(bitmap,x,y,colour);
        }
}
```

Proste efekty graficzne

Funkcja `xor_draw()` tworzy obraz na ekranie wyznaczając kolory pikseli przy pomocy operatora bitowej różnicy symetrycznej. Uzyskiwany obraz pod względem jasności plasuje się między dwoma poprzednimi.

Proste efekty graficzne

```
void multiply_draw(BITMAP* bitmap)
{
    int x,y;
    for(x=0; x<SCREEN_W; x++)
        for(y=0; y<SCREEN_H; y++) {
            int expression = (x*y)&255;
            int colour = makecol(expression,expression,expression);
            putpixel(bitmap,x,y,colour);
        }
}
```

Proste efekty graficzne

Funkcja `multiply_draw()` używa do wyznaczania kolorów pikseli operatora mnożenia arytmetycznego. Uzyskany obraz jest inny niż te, które generowane są przez poprzednio opisane funkcje. Obraz widoczny na ekranie jest nazywany efektem mory.

Proste efekty graficzne

```
void draw_sierpinski_triangle(BITMAP* bitmap)
{
    int x,y;
    int white = makecol(255,255,255);
    for(x=0; x<SCREEN_W; x++)
        for(y=0; y<SCREEN_H; y++) {
            if((x&y)==0)
                putpixel(bitmap,x,y,white);
        }
}
```

Proste efekty graficzne

Ostatnia z funkcji zdefiniowanych w programie tworzy na ekranie obraz fraktala nazywanego trójkątem Sierpińskiego, od nazwiska jego odkrywcy, polskiego matematyka Wacława Sierpińskiego. To nie jedyna metoda uzyskania takiego fraktala, ale ta jest bardzo prosta. Jeśli iloczyn bitowy współrzędnych piksela jest równy zero, to taki piksel jest zaznaczany na ekranie na biało. Czynności te są powtarzane dla wszystkich pikseli na ekranie.

Proste efekty graficzne

```
int main(void)
{
    if(initialize(GFX_AUTODETECT_FULLSCREEN,WIDTH,HEIGHT))
        return -1;
    or_draw(screen);
    draw_text(screen,font,"or");
    wait_for_any_key();
    and_draw(screen);
    draw_text(screen,font,"and");
    wait_for_any_key();
    xor_draw(screen);
    draw_text(screen,font,"xor");
    wait_for_any_key();
    multiply_draw(screen);
    draw_text(screen,font,"*");
    wait_for_any_key();
    clear(screen);
    draw_sierpinski_triangle(screen);
    wait_for_any_key();
    allegro_exit();
    return 0;
}
END_OF_MAIN()
```

Proste efekty graficzne

W funkcji `main()` wywoływane są po kolei wszystkie zdefiniowane wcześniej w programie funkcje. Po wywołaniu każdej z nich, oprócz ostatniej, wypisywana jest przy pomocy funkcji `draw_text()` nazwa/symbol operatora, który posłużył do wygenerowania obrazu. Po tej czynności wykonanie programu jest wstrzymywane do czasu naciśnięcia przez użytkownika dowolnego klawisza. Przed wywołaniem funkcji rysującej trójkąt Sierpińskiego kolor każdego piksela ekranu jest ustawiany na czarny poprzez wywołanie funkcji `clear()`. Po narysowaniu tego fraktala wykonanie programu ponownie jest wstrzymywane do momentu naciśnięcia przez użytkownika jakiegokolwiek klawisza na klawiaturze. Po wystąpieniu tego zdarzenia finalizowana jest praca biblioteki Allegro i kończone jest działanie programu.

Pytania

?

KONIEC

Dziękuję Państwu za uwagę.