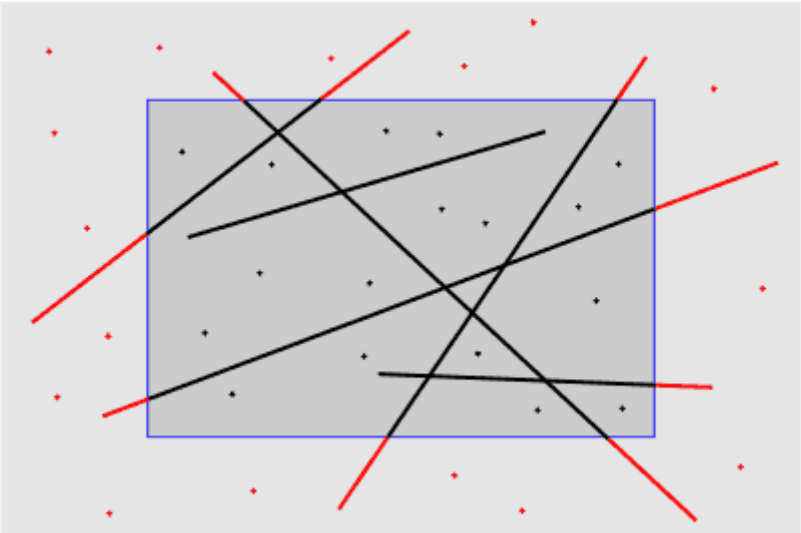
Obcinanie grafiki do prostokąta

Tworząc różnego rodzaju grafikę komputerową bardzo szybko natrafisz na sytuację, gdy rysowane obiekty "wychodzą" poza obszar ekranu. W takim przypadku kontynuowanie rysowania linii bądź punktów prowadzi do błędów graficznych lub błędów dostępu do pamięci - program próbuje wpisywać piksele poza buforem obrazowym, czego oczywiście mu nie wolno robić. Sytuacja ta występuje również wtedy, gdy tworzone rysunki chcesz ograniczyć do rozmiaru prostokątnego okna na swojej powierzchni graficznej.

Rozwiązaniem problemu jest zastosowanie **obcinania** (ang. clipping) linii lub punktów. W przypadku punktów obcinanie polega po prostu na pomijaniu rysowania punktu, jeśli znajduje się on poza dozwolonym obszarem. Natomiast odcinek jest przycinany do fragmentu leżącego wewnątrz dozwolonego obszaru. Obszar nazywamy prostokątem obcinającym (ang. clipping rectangle). Na poniższym rysunku prostokąt obcinający zaznaczono niebieską linią. Punkty i fragmenty odcinków o kolorze czarnym są rysowane na powierzchni graficznej. Punkty i fragmenty odcinków oznaczone kolorem czerwonym nie będą rysowane, ponieważ znajdują się poza prostokątem obcinającym.



Prostokąt obcinający

Do definiowania prostokąta obcinania będziemy stosowali strukturę biblioteki SDL:

```
typedef struct
{
  Sint16 x, y;
  Uint16 w, h;
} SDL_Rect;
```

Pola x i y określają współrzędne lewego górnego narożnika prostokąta. Pola w i h definiują jego szerokość (ang. width) oraz wysokość (ang. height). Na przykład poniższy fragment programu tworzy prostokąt znajdujący się na

środku ekranu w odległości 10 pikseli od każdego z brzegów obszaru graficznego:

```
...

  SDL_Rect * clip = new SDL_Rect; // tworzymy
wskaźnik prostokąta i inicjujemy go

  clip->x = clip->y = 10;          // określamy
położenie lewego górnego narożnika prostokąta
  clip->w = screen->w - 20;        // szerokość
prostokąta
  clip->h = screen->h - 20;        // wysokość
prostokąta
...
```

Prostokąt obcinający umieszczamy w strukturze SDL_Surface naszego ekranu za pomocą wywołania funkcji bibliotecznej:

```
void SDL_SetClipRect(SDL_Surface * surface,
SDL_Rect * clip);
```

Pierwszy parametr wskazuje strukturę SDL_Surface naszego ekranu. Drugi parametr wskazuje prostokąt obcinający. Jeśli drugi parametr ma wartość NULL, to funkcja SDL_SetClipRect() zastosuje cały ekran jako prostokąt obcinający. Pole clip_rect jest inicjowane na rozmiary ekranu przy tworzeniu struktury SDL_Surface. Jeśli zatem chcesz obcinać grafikę tylko do rozmiarów ekranu, nie musisz wywoływać w swoim programie funkcji SDL_SetClipRect(), ale lepiej to zrobić - dla spokojnego sumienia i dla tych, którzy o tym fakcie nie wiedzą.

Obcinanie punktów

W naszej bibliotece SDL_gfx utworzymy funkcję gfxClipPlot(), która będzie rysowała punkty obcięte do przekazanego jej w parametrach wywołania prostokąta obcinającego.

UWAGA: Poniższy kod funkcji gfxClipPlot() dopisz do końca pliku SDL_gfx.cpp.

```
// gfxClipPlot() rysuje piksel 32 bitowy wewnątrz
zdefiniowanego
// prostokąta obcinającego clip_rect
// screen - wskaźnik struktury SDL_Surface
// x,y - współrzędne piksela
// color - 32 bitowy kod koloru piksela
//-----------------------------------------------
-----------------

void gfxClipPlot(SDL_Surface * screen, Sint32 x,
Sint32 y, Uint32 color)
```

```
{
  if((x >= screen->clip_rect.x) && (x < screen-
>clip_rect.x + screen->clip_rect.w) &&
     (y >= screen->clip_rect.y) && (y < screen-
>clip_rect.y + screen->clip_rect.h))
    gfxPlot(screen, x, y, color);
}
```

Funkcja gfxClipPlot() sprawdza, czy rysowany piksel znajduje się wewnątrz prostokąta obcinającego, który ustawiamy przez wcześniejsze wywołanie funkcji SDL_SetClipRect(). Będzie tak, jeśli:

x jest w przedziale <screen->clip_rect.x, screen->clip_rect.x + screen->clip_rect.w)
y jest w przedziale <screen->clip_rect.y, screen->clip_rect.y + screen->clip_rect.h)

Jeśli współrzędne spełniają warunki, to punkt jest rysowany poprzez wywołanie funkcji gfxPlot(), którą utworzyliśmy w rozdziale OL035. Parametry wywołania funkcji gfxClipPlot() są identyczne jak w funkcji gfxPlot().

UWAGA: Na końcu pliku nagłówkowego SDL_gfx.h dopisz:

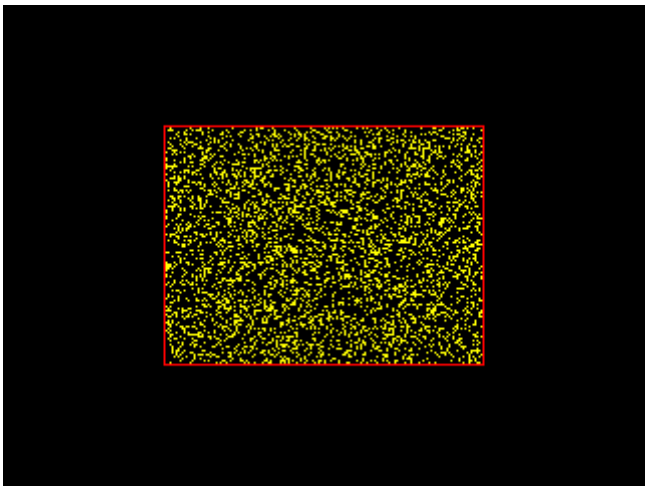
```
void gfxClipPlot(SDL_Surface * screen, Sint32 x,
Sint32 y, Uint32 color);
```

Poniższy program definiuje prostokąt na środku ekranu, rysuje na nim ramkę, pomniejsza go do wnętrza ramki, a następnie generuje 1000000 punktów o pseudolosowych współrzędnych x,y w zakresie od 0 do 1999 i przy pomocy funkcji gfxClipPlot() rysuje je we wnętrzu ramki.

```
//
// P011 - obcinanie punktów
//-------------------------

#include <windows.h>
#include <SDL/SDL_gfx.h>
#include <time.h>      // do inicjalizacji
generatora pseudolosowego

const int SCRX = 320; // stałe określające
szerokość i wysokość
const int SCRY = 240; // ekranu w pikselach

int main(int argc, char *argv[])
{
```

```
  if(SDL_Init(SDL_INIT_VIDEO))
  {
    MessageBox(0, SDL_GetError(), "Błąd
inicjalizacji SDL", MB_OK);
    exit(-1);
  }

  atexit(SDL_Quit);

  SDL_Surface * screen;

  if(!(screen = SDL_SetVideoMode(SCRX, SCRY, 32,
SDL_HWSURFACE)))
  {
    MessageBox(0, SDL_GetError(), "Błąd tworzenia
bufora obrazowego", MB_OK);
    exit(-1);
  }

  if(SDL_MUSTLOCK(screen))
    if(SDL_LockSurface(screen) < 0)
    {
      MessageBox(0, SDL_GetError(), "Błąd blokady
bufora obrazowego", MB_OK);
      exit(-1);
    }

// inicjujemy generator liczb pseudolosowych

  srand((unsigned)time(NULL));

// definiujemy prostokąt obcinający

  SDL_Rect * clip = new SDL_Rect;

  clip->x = screen->w >> 2; // 1/4 szerokości
ekranu
  clip->y = screen->h >> 2; // 1/4 wysokości ekranu
  clip->w = screen->w >> 1; // 1/2 szerokości
ekranu
  clip->h = screen->h >> 1; // 1/2 wysokości ekranu

// rysujemy ramkę
```

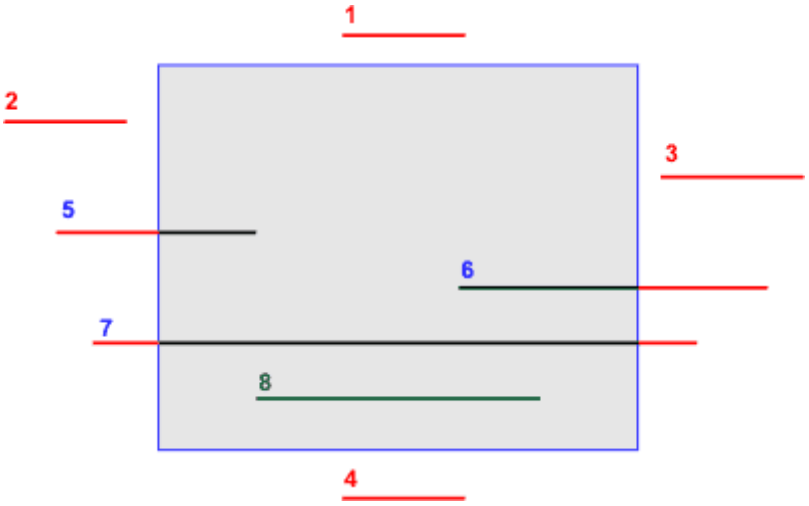
```
  gfxRectangle(screen, clip, 0xff0000);

// pomniejszamy prostokąt obcinający

  clip->x++; clip->y++; clip->w -= 2; clip->h -= 2;

// ustawiamy prostokąt obcinający w strukturze
SDL_Surface

  SDL_SetClipRect(screen, clip);

// rysujemy milion punktów na przypadkowych
współrzędnych obcinając je do prostokąta clip

  for(int i = 0; i < 1000000; i++)
     gfxClipPlot(screen, rand() % 2000, rand() %
2000, 0xffff00);

  if(SDL_MUSTLOCK(screen))
SDL_UnlockSurface(screen);

  SDL_UpdateRect(screen, 0, 0, 0, 0);

  MessageBox(0, "Kliknij przycisk OK", "Koniec",
MB_OK);
  return 0;
}
```



Obcinanie odcinków

Obcinanie odcinków poziomych i pionowych

Na początek zdefiniujemy nowe funkcje rysowania linii poziomych i pionowych oraz ramek. Jest to zadanie łatwe, ponieważ linie poziome i pionowe są rysowane w jeden określony sposób. Dla linii poziomej reguły są następujące (przy linii pionowej postępujemy w identyczny sposób).



Załóżmy, iż odcinek poziomy zdefiniowany jest współrzędnymi lewego punktu początkowego - x,y oraz szerokością len (tak będzie on określony w parametrach wywołania funkcji). Prostokąt obcinający definiuje struktura SDL_Rect o nazwie clip. Istnieje 8 możliwych położeń odcinka poziomego, które przedstawiliśmy na rysunku obok (pamiętaj, iż u nas oś OY jest skierowana w dół):

1. Odcinek ponad górną krawędzią prostokąta obcinającego. Sytuacja ta występuje, gdy y < clip.y.
2. Odcinek przed lewą krawędzią prostokąta: x + len ≤ clip.x.
3. Odcinek za prawą krawędzią prostokąta: x ≥ clip.x + clip.w.
4. Odcinek poniżej dolnej krawędzi prostokąta: y ≥ clip.y + clip.h.

W przypadkach 1...4 odcinek nie może być rysowany, zatem warunki te należy w algorytmie wykluczyć w pierwszej kolejności.

5 i 7. Odcinek przecina lewą krawędź prostokąta: x < clip.x. Zmniejszamy długość len = len - (clip.x - x). Zmieniamy x = clip.x.
6 i 7, Odcinek przecina prawą krawędź prostokąta: x + len > clip.x + clip.w. Zmieniamy długość len = clip.x + clip.w - x.
8. Odcinek w całości zawiera się we wnętrzu prostokąta obcinającego. Jeśli przejdziemy warunki 1..7, to warunek 8 jest wynikowy i nie musimy go testować.

Z podanych rozważań wynika następujący algorytm obcinania odcinka poziomego do prostokąta:

Wejście

x, y - współrzędne ekranowe lewego punktu początkowego kreślonego odcinka
len - szerokość odcinka w pikselach
clip - struktura SDL_Rect definiująca prostokąt obcinający

Wyjście

Rysunek odcinka obciętego do zadanego prostokąta obcinającego clip. Odcinek nie jest rysowany, jeśli znajduje się poza obszarem clip.

Zmienne pomocnicze

x_c - współrzędna x punktu leżącego tuż za prawą krawędzią prostokąta obcinającego
x_p - współrzędna y punktu leżącego tuż za prawym końcem odcinka

Lista kroków

K01: $x_c \leftarrow$ clip.x + clip.w *; obliczamy współrzędną poza prawą krawędzią prostokąta*
K02: $x_p \leftarrow$ x + len *; obliczamy współrzędną poza prawym końcem odcinka*
K03: Jeśli y < clip.y, zakończ *; wykluczamy położenie nr 1*
K04: Jeśli $x_p \leq$ clip.x, zakończ *; wykluczamy położenie nr 2*
K05: Jeśli x $\geq x_c$, zakończ *; wykluczamy położenie nr 3*
K06: Jeśli y $\geq$ clip.y + clip.h *; wykluczamy położenie nr 4*
K07: Jeśli x $\geq$ clip.x, idź do kroku K10 *; położenia 5 i 7*
K08: len $\leftarrow$ len - clip.x + x *; zmniejszamy długość odcinka*
K09: x $\leftarrow$ clip.x *; przesuwamy współrzędną x na lewą krawędź prostokąta*
K10: Jeśli $x_p \leq x_c$, idź do kroku K12 *; położenia 6 i 7*
K11: len $\leftarrow x_c$ - x *; zmniejszamy długość*
K12; Rysuj odcinek poziomy x, y, len
K13: Zakończ

Na podstawie powyższego algorytmu napiszemy funkcje gfxClipHLine(), gfxClipVLine() oraz gfxClipRectangle() i dodamy je do naszej biblioteki SDL_gfx.cpp. Funkcje obcinają grafikę do podanego w parametrach prostokąta clip. Jeśli zamiast wskaźnika przekażemy NULL, to funkcje wykonają obcięcie do bieżącej ramki obrazu.

UWAGA: Poniższy kod funkcji gfxClipHLine(), gfxClipVLine() oraz gfxClipRectangle() dopisz do końca pliku SDL_gfx.cpp.

```
// gfxClipHLine() rysuje linię poziomą z obcięciem
do clip_rect
// screen - wskaźnik struktury SDL_Surface
// x,y    - współrzędne lewego początku lini
// color  - 32 bitowy kod koloru linii
// len    - długość linii w pikselach
//-----------------------------------------------
-----------------

void gfxClipHLine(SDL_Surface * screen, Sint32 x,
Sint32 y, Uint32 color, Uint32 len)
{
  Sint32 x1c, x2c, y1c, y2c, xp;

  x1c = screen->clip_rect.x;
```

```
  y1c = screen->clip_rect.y;
  x2c = x1c + screen->clip_rect.w;
  y2c = y1c + screen->clip_rect.h;

// obcinanie odcinka

  xp = x + len;
  if((y >= y1c) && (xp > x1c) && (x < x2c) && (y <
y2c))
  {
    if(x < x1c)
    {
      len += x - x1c;
      x = x1c;
    }
    if(xp > x2c) len = x2c - x;
    gfxHLine(screen, x, y, color, len);
  }
}

// gfxClipVLine() rysuje linię pionową z obcięciem
do prostokąta clip_rect
// screen - wskaźnik struktury SDL_Surface
// x,y    - współrzędne górnego początku lini
// color  - 32 bitowy kod koloru linii
// len    - długość linii w pikselach
//-----------------------------------------------
-----------------

void gfxClipVLine(SDL_Surface * screen, Sint32 x,
Sint32 y, Uint32 color, Uint32 len)
{
  Sint32 x1c, x2c, y1c, y2c, yp;

  x1c = screen->clip_rect.x;
  y1c = screen->clip_rect.y;
  x2c = x1c + screen->clip_rect.w;
  y2c = y1c + screen->clip_rect.h;

// obcinanie odcinka

   yp = y + len;
```

```
  if((x >= x1c) && (yp > y1c) && (y < y2c) && (x <
x2c))
  {
    if(y < y1c)
    {
      len += y - y1c;
      y = y1c;
    }
    if(yp > y2c) len = y2c - y;
    gfxVLine(screen, x, y, color, len);
  }
}

// gfxClipRectangle() rysuje ramkę na obrysie
prostokąta z obcinaniem
// screen - wskaźnik struktury SDL_Surface
// r      - definiuje prostokąt ramki
// color  - 32 bitowy kod koloru ramki
//-------------------------------------------------
-----------------

void gfxClipRectangle(SDL_Surface * screen,
SDL_Rect * r, Uint32 color)
{
    gfxClipHLine(screen, r->x, r->y, color, r->w);
    gfxClipHLine(screen, r->x, r->y + r->h - 1,
color, r->w);
    gfxClipVLine(screen, r->x, r->y, color, r->h);
    gfxClipVLine(screen, r->x + r->w - 1, r->y,
color, r->h);
}
```

UWAGA: Na końcu pliku nagłówkowego SDL_gfx.h dopisz:

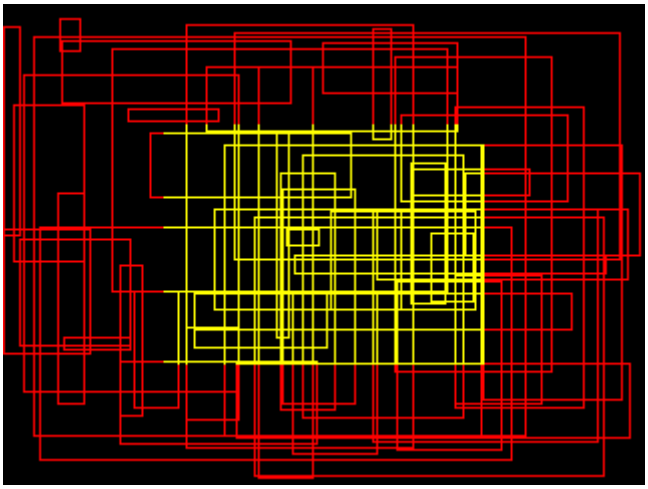
```
void gfxClipHLine(SDL_Surface * screen, Sint32 x,
Sint32 y, Uint32 color, Uint32 len);

void gfxClipVLine(SDL_Surface * screen, Sint32 x,
Sint32 y, Uint32 color, Uint32 len);

void gfxClipRectangle(SDL_Surface * screen,
SDL_Rect * r, Uint32 color);
```

Poniższy program testuje nowe funkcje graficzne. Program definiuje na środku ekranu prostokąt obcinający. Następnie losuje 50 ramek. Wylosowaną ramkę rysuje najpierw bez obcinania w kolorze czerwonym, a następnie z obcinaniem w kolorze żółtym. W efekcie części ramek zawarte w prostokącie obcinającym będą żółte, a części zewnętrzne będą czerwone.

```
//
// P012 - obcinanie prostokątów
//-----------------------------

#include <windows.h>
#include <SDL/SDL_gfx.h>
#include <time.h>      // do inicjalizacji
generatora pseudolosowego

const int SCRX = 320; // stałe określające
szerokość i wysokość
const int SCRY = 240; // ekranu w pikselach

int main(int argc, char *argv[])
{

  if(SDL_Init(SDL_INIT_VIDEO))
  {
    MessageBox(0, SDL_GetError(), "Błąd
inicjalizacji SDL", MB_OK);
    exit(-1);
  }

  atexit(SDL_Quit);

  SDL_Surface * screen;

  if(!(screen = SDL_SetVideoMode(SCRX, SCRY, 32,
SDL_HWSURFACE)))
  {
    MessageBox(0, SDL_GetError(), "Błąd tworzenia
bufora obrazowego", MB_OK);
    exit(-1);
  }

  if(SDL_MUSTLOCK(screen))
    if(SDL_LockSurface(screen) < 0)
    {
```

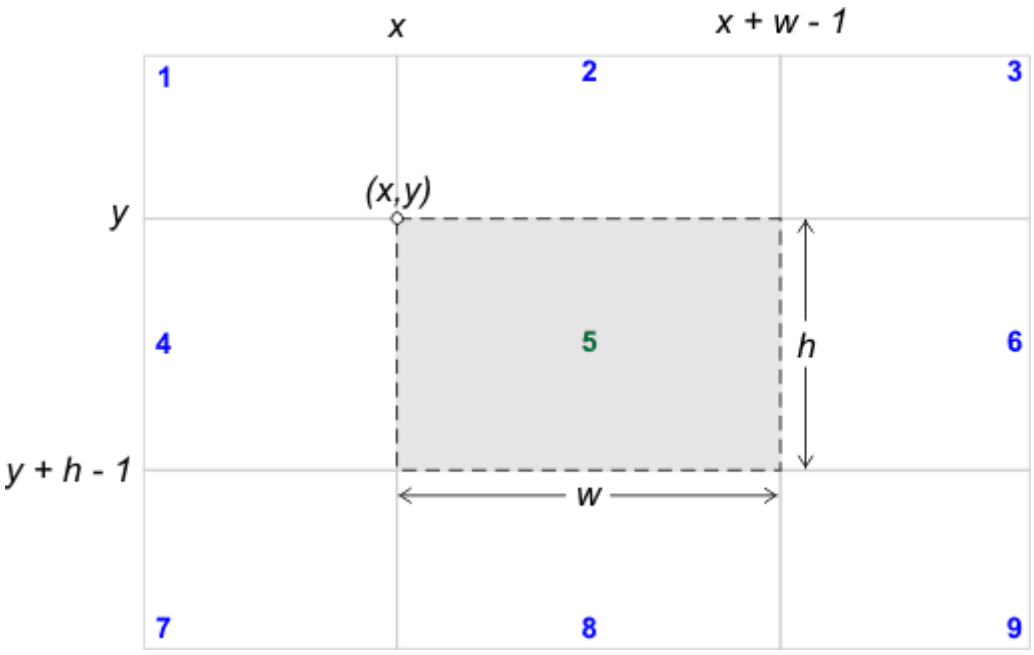
```
      MessageBox(0, SDL_GetError(), "Błąd blokady
bufora obrazowego", MB_OK);
      exit(-1);
    }

// inicjujemy generator liczb pseudolosowych

  srand((unsigned)time(NULL));

// definiujemy prostokąt

  SDL_Rect * clip = new SDL_Rect; // prostokąt
obcinania
  SDL_Rect * r    = new SDL_Rect; // prostokąt
ramki

  clip->x = screen->w >> 2;       // 1/4 szerokości
ekranu
  clip->y = screen->h >> 2;       // 1/4 wysokości
ekranu
  clip->w = screen->w >> 1;       // 1/2 szerokości
ekranu
  clip->h = screen->h >> 1;       // 1/2 wysokości
ekranu

// ustawiamy prostokąt obcinający

  SDL_SetClipRect(screen, clip);

// generujemy 50 przypadkowych ramek

  for(int i = 0; i < 50; i++)
  {
    r->x = rand() % ((screen->w >> 1) + (screen->w
>> 2));
    r->y = rand() % ((screen->h >> 1) + (screen->h
>> 2));
    r->w = 5 + rand() % (screen->w - r->x - 5);
    r->h = 5 + rand() % (screen->h - r->y - 5);
    gfxRectangle(screen, r, 0xff0000);
    gfxClipRectangle(screen, r, 0xffff00);
  }
```

```
  if(SDL_MUSTLOCK(screen))
SDL_UnlockSurface(screen);

  SDL_UpdateRect(screen, 0, 0, 0, 0);

  MessageBox(0, "Kliknij przycisk OK", "Koniec",
MB_OK);
  return 0;
}
```



Obcinanie dowolnych odcinków

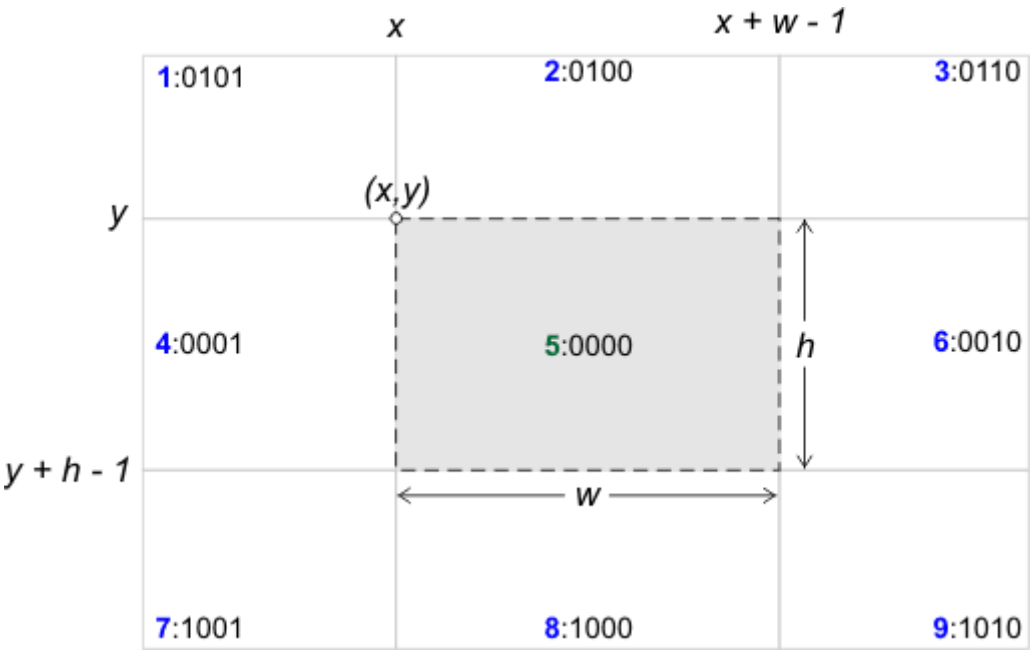
Istnieje wiele algorytmów obcinających odcinek do zadanego prostokąta. My zastosujemy **algorytm Cohena-Sutherlanda**, który jest w miarę prosty do zrozumienia. Algorytm ten dzieli obszar graficzny na 9 części. W środku znajduje się zawsze prostokąt obcinający (nr 5) zdefiniowany przez współrzędne x,y lewego górnego narożnika oraz szerokość w i wysokość h (oś OY u nas jest skierowana w dół):



Zadaniem algorytmu Cohena Sutherlanda dla dowolnego odcinka (x_1,y_1) - (x_2,y_2) jest znalezienie jego części zawartej wewnątrz prostokąta obcinającego lub odrzucenie odcinka, jeśli nie posiada on części wspólnej z prostokątem. W tym celu algorytm wylicza dla każdego końca odcinka specjalny, 4-bitowy kod, który identyfikuje położenie tego końca w jednym z 9 powyżej przedstawionych obszarów. Bity w kodzie $b_3b_2b_1b_0$ posiadają następujące znaczenie:

b_0 = 1, jeśli koniec odcinka leży na lewo od prostokąta obcinającego (obszary 1,4,7)
b_1 = 1, jeśli koniec odcinka leży na prawo od prostokąta obcinającego (obszary 3,6,9)
b_2 = 1, jeśli koniec odcinka leży powyżej prostokąta obcinającego (obszary 1,2,3)
b_3 = 1, jeśli koniec odcinka leży poniżej prostokąta obcinającego (obszary 7,8,9)

Jeśli koniec odcinka znajduje się wewnątrz prostokąta obcinającego, to nie zachodzi żaden z powyższych przypadków i kod jest równy 0000.

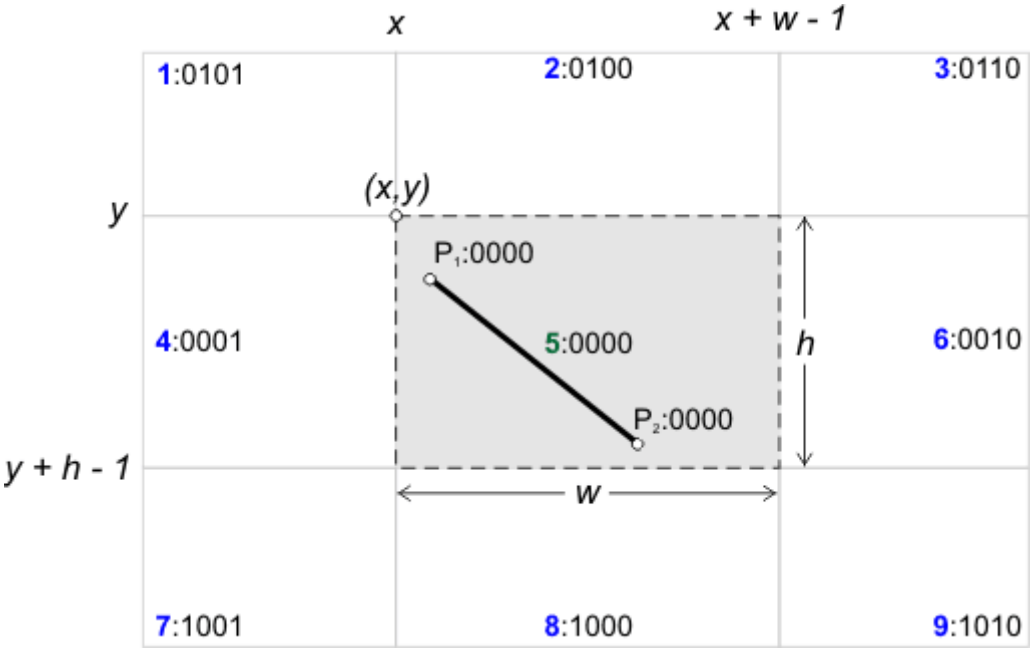


Kod bitowy można wyznaczyć metodą porównań i ustawiania bitów. Przyjmijmy, iż x_p i y_p to współrzędne końca odcinka. Zatem:

$b_0 \leftarrow 1$, jeśli $x_p < x$
$b_1 \leftarrow 1$, jeśli $x_p \geq x + w$
$b_2 \leftarrow 1$, jeśli $y_p < y$
$b_3 \leftarrow 1$, jeśli $y_p \geq y + h$

Jeśli dla obu końców P_1 i P_2 odcinka kody są równe 0000, to odcinek w całości leży w prostokącie obcinającym - w takim przypadku nic nie trzeba odcinać, odcinek akceptujemy w całości.

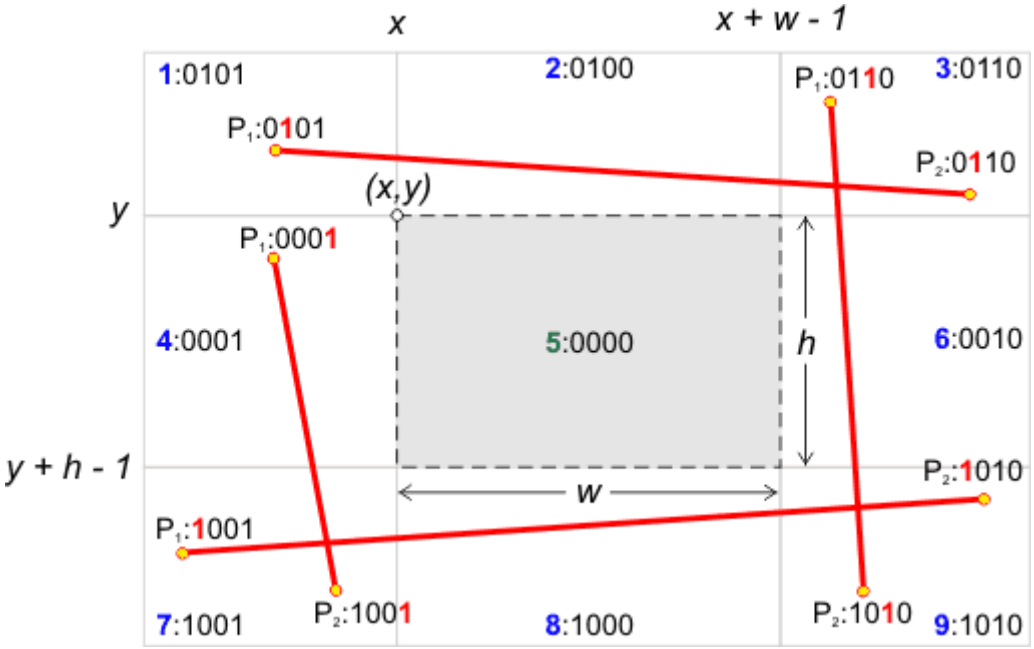
$$kod(P_1) \mid kod(P_2) = 0000$$



Jeśli którekolwiek bity w obu kodach końców odcinka są równe jednocześnie 1, to odcinek w całości leży poza prostokątem obcinającym, zatem go odrzucamy. Wytłumaczenie jest proste - jeśli na przykład w obu kodach bit b_0

jest równy 1, to oba końce odcinka leżą na lewo od prostokąta obcinającego, zatem odcinek nie może przecinać prostokąta. Podobnie jest dla pozostałych bitów:

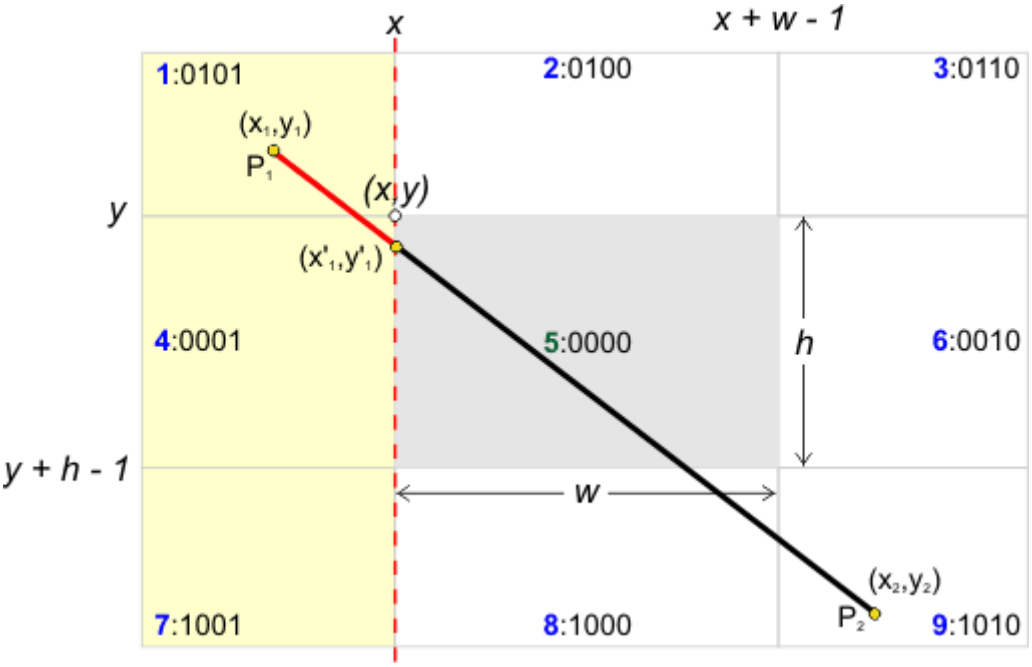
$$kod(P_1) \,\&\, kod(P_2) \neq 0000$$



Jeśli dwa powyższe warunki nie zachodzą, to rysowany odcinek może (ale nie musi) znajdować się częściowo wewnątrz obszaru prostokąta obcinającego. W takim przypadku wybieramy punkt, dla którego kod jest różny od 0000. Załóżmy, iż jest to punkt P_1 (jeśli tak nie jest, to punkty i ich kody zamieniamy ze sobą). Przeglądamy kolejne bity kodu:

$b_0 = 1$

P_1 leży na lewo od prostokąta obcinającego.



Wyznaczamy współrzędne punktu przecięcia odcinka z prostą zawierającą lewy bok prostokąta obcinającego:

$$\frac{x - x_1}{x_2 - x_1} = \frac{y'_1 - y_1}{y_2 - y_1}$$

$$y'_1 - y_1 = (y_2 - y_1)\frac{x - x_1}{x_2 - x_1}$$

$$y'_1 = y_1 + (y_2 - y_1)\frac{x - x_1}{x_2 - x_1}$$

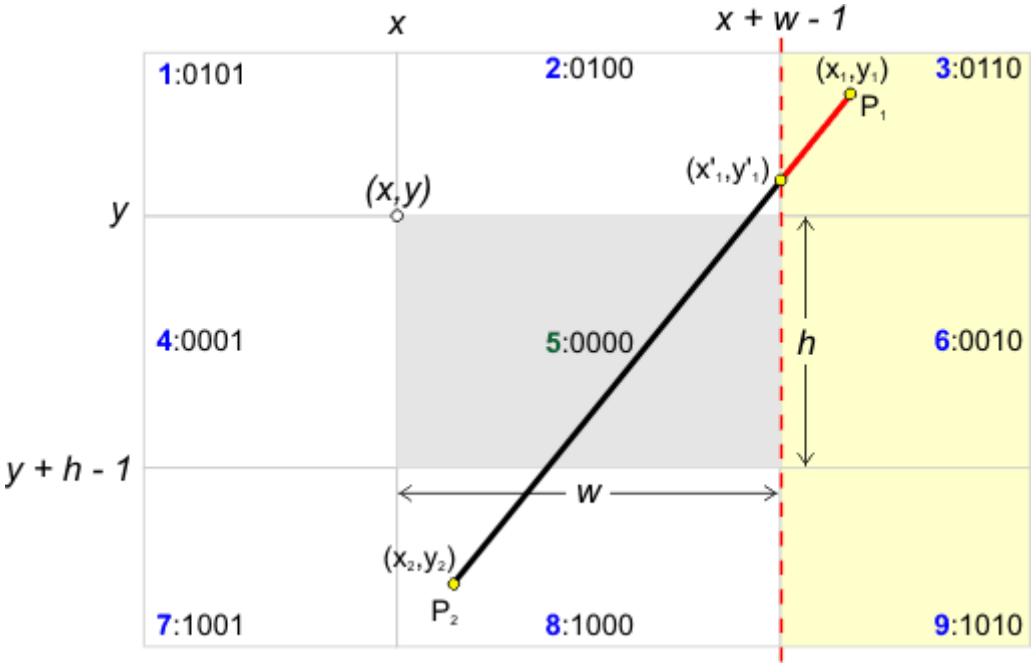
$$x'_1 = x$$

Obliczone współrzędne x'_1, y'_1 przypisujemy do współrzędnych punktu P_1 i ponownie obliczamy kod binarny powracając do początku algorytmu.

Podobnie postępujemy dla pozostałych bitów kodu:

$b_1 = 1$

P_1 leży na prawo od prostokąta obcinającego.



Wyznaczamy współrzędne punktu przecięcia odcinka z prostą zawierającą prawy bok prostokąta obcinającego:

$$\frac{x_1 - x - w + 1}{x_1 - x_2} = \frac{y'_1 - y_1}{y_2 - y_1}$$

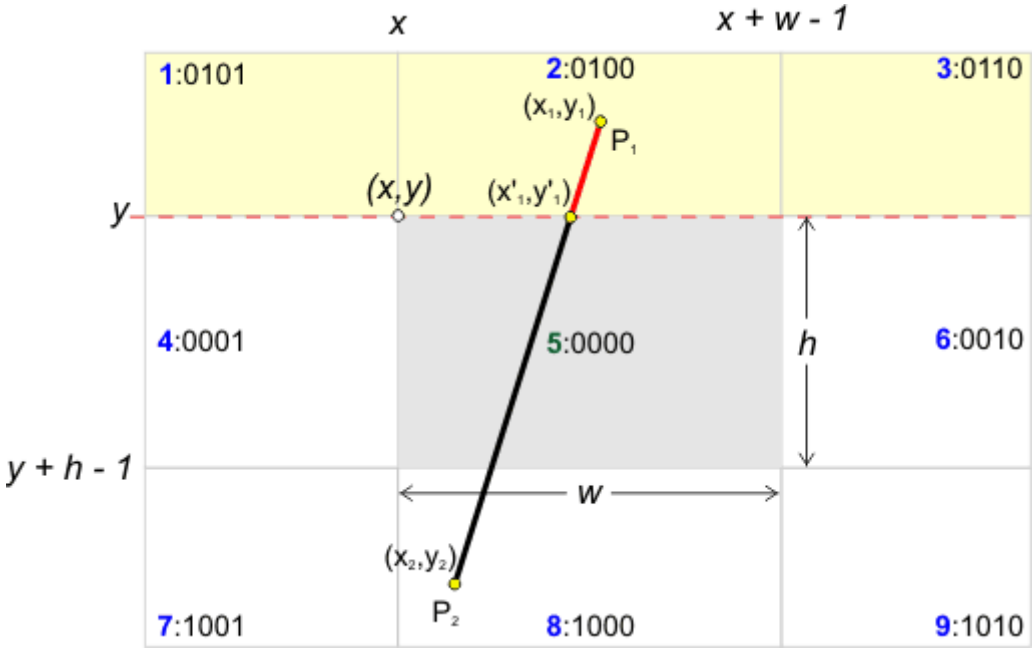
$$y'_1 = y_1 + (y_2 - y_1)\frac{x_1 - x - w + 1}{x_1 - x_2}$$

$$x'_1 = x + w - 1$$

Obliczone współrzędne x'_1, y'_1 przypisujemy do współrzędnych punktu P_1 i ponownie obliczamy kod binarny powracając do początku algorytmu.

$b_2 = 1$

P_1 leży ponad prostokątem obcinającym (oś OY jest u nas skierowana w dół).



Wyznaczamy współrzędne punktu przecięcia odcinka z prostą zawierającą górny bok prostokąta obcinającego:

$$\frac{x_1 - x'_1}{x_1 - x_2} = \frac{y_1 - y}{y_1 - y_2}$$

$$x_1 - x'_1 = (x_1 - x_2)\frac{y_1 - y}{y_1 - y_2}$$

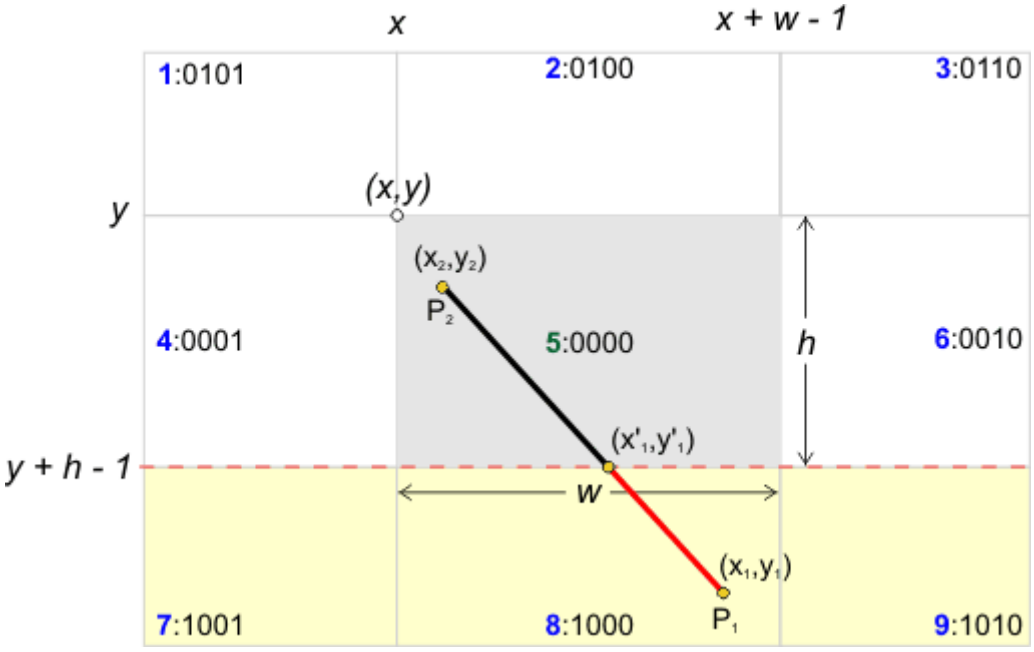
$$x'_1 = x_1 + (x_1 - x_2)\frac{y_1 - y}{y_2 - y_1}$$

$$y'_1 = y$$

Obliczone współrzędne x'_1, y'_1 przypisujemy do współrzędnych punktu P_1 i ponownie obliczamy kod binarny powracając do początku algorytmu.

b_3 = 1

P_1 leży poniżej prostokąta obcinającego (oś OY jest u nas skierowana w dół).



Wyznaczamy współrzędne punktu przecięcia odcinka z prostą zawierającą dolny bok prostokąta obcinającego:

$$\frac{x_1 - x'_1}{x_1 - x_2} = \frac{y_1 - y - h + 1}{y_1 - y_2}$$

$$x_1 - x'_1 = (x_1 - x_2)\frac{y_1 - y - h + 1}{y_1 - y_2}$$

$$x'_1 = x_1 + (x_1 - x_2)\frac{y_1 - y - h + 1}{y_2 - y_1}$$

$$y'_1 = y + h - 1$$

Obliczone współrzędne x'_1, y'_1 przypisujemy do współrzędnych punktu P_1 i ponownie obliczamy kod binarny powracając do początku algorytmu.

Z przedstawionych powyżej rozważań wyłania się następujący algorytm Cohena-Sutherlanda:

Wejście

x_1, y_1 - współrzędne ekranowe punktu P_1
x_2, y_2 - współrzędne ekranowe punktu P_2
x, y - współrzędne lewego, górnego narożnika prostokąta obcinającego (oś OY jest skierowana w dół !!!)
w - szerokość prostokąta obcinającego
h - wysokość prostokąta obcinającego

Wyjście

Algorytm rysuje odcinek (x_1,y_1)-(x_2,y_2) obcięty do zadanego prostokąta obcinającego (x,y,w,h). Jeśli odcinek w całości leży poza obszarem prostokąta, to nie będzie rysowany.

Lista kroków

K01: $k_1 \leftarrow kod(x_1,y_1)$ *; wyznaczamy kody binarne końców odcinka*

K02: $k_2 \leftarrow kod(x_2,y_2)$

K03: Jeśli $k_1 \mid k_2 = 0000$, rysuj odcinek $x_1,y_1 - x_2,y_2$ i zakończ *; jeśli odcinek leży wewnątrz prostokąta obcinającego, rysujemy go*

K04: Jeśli $k_1 \& k_2 \neq 0000$, zakończ *; jeśli odcinek leży na zewnątrz prostokąta, odrzucamy go*

K05: Jeśli $k_1 = 0000$, zamień k_1 z k_2 oraz x_1 z x_2 i y_1 z y_2 *; jeśli pierwszy punkt w prostokącie, zamieniamy punkty i kody*

K06: Jeśli $k_1 \& 0001 = 0000$, idź do kroku K10

K07: $y_1 \leftarrow y_1 + (y_2 - y_1) \bullet (x - x_1) / (x_2 - x_1)$ *; punkt na lewo od prostokąta*

K08: $x_1 \leftarrow x$

K09: Idź do kroku K01

K10 Jeśli $k_1 \& 0010 = 0000$, idź do kroku K14

K11: $y_1 \leftarrow y_1 + (y_2 - y_1) \bullet (x_1 - x - w + 1) / (x_1 - x_2)$ *; punkt na prawo od prostokąta*

K12: $x_1 \leftarrow x + w - 1$

K13: Idź do kroku K01

K14: Jeśli $k_1 \& 0100 = 0000$, idź do kroku K18

K15: $x_1 \leftarrow x_1 + (x_1 - x_2) \bullet (y_1 - y) / (y_2 - y_1)$ *; punkt powyżej prostokąta*

K16: $y_1 \leftarrow y$

K17: Idź do kroku K01

K18: $x_1 \leftarrow x_1 + (x_1 - x_2) \bullet (y_1 - y - h + 1) / (y_2 - y_1)$ *; punkt poniżej prostokąta*

K19: $y_1 \leftarrow y + h - 1$

K20 Idź do kroku K01

Na podstawie powyższego algorytmu zaprogramujemy funkcję biblioteczną gfxClipLine() oraz jej pochodną gfxClipLineTo().

UWAGA: Poniższy kod funkcji gfxClipLine() oraz gfxClipLineTo() dopisz do końca pliku SDL_gfx.cpp.

```
// gfxClipLine() rysuje odcinek pomiędzy zadanymi
punktami obcięty do
// obszaru prostokąta obcinającego ustawionego
przez SDL_SetClipRect()
// screen - wskaźnik struktury SDL_Surface
// x1,y1  - współrzędne punktu startowego
// x2,y2  - współrzędne punktu końcowego
// color  - kolor odcinka
//-----------------------------------------------
-----------------

void gfxClipLine(SDL_Surface * screen, Sint32 x1,
Sint32 y1, Sint32 x2, Sint32 y2, Uint32 color)
{
  Sint32 x1c, x2c, y1c, y2c, iax;

  x1c = screen->clip_rect.x; x2c = x1c + screen-
>clip_rect.w - 1;
```

```
  y1c = screen->clip_rect.y; y2c = y1c + screen-
>clip_rect.h - 1;
  for(;;)
  {
    Uint32 k1, k2, uax;
    k1 = (((x1 - x1c) >> 31) & 1) | (((x2c - x1) >>
30) & 2) |
         (((y1 - y1c) >> 29) & 4) | (((y2c - y1) >>
28) & 8);
    k2 = (((x2 - x1c) >> 31) & 1) | (((x2c - x2) >>
30) & 2) |
         (((y2 - y1c) >> 29) & 4) | (((y2c - y2) >>
28) & 8);
    if(!(k1 | k2))
    {
      gfxLine(screen, x1, y1, x2, y2, color);
break;
    }
    if(k1 & k2) break;
    if(!k1)
    {
      uax = k1; k1 = k2; k2 = uax;
      iax = x1; x1 = x2; x2 = iax;
      iax = y1; y1 = y2; y2 = iax;
    }
    if(k1 & 1)
    {
      y1 += ((y2 - y1) * (x1c - x1)) / (x2 - x1);
      x1 = x1c;
    }
    else if(k1 & 2)
    {
      y1 += ((y2 - y1) * (x1 - x2c)) / (x1 - x2);
      x1 = x2c;
    }
    else if(k1 & 4)
    {
      x1 += ((x1 - x2) * (y1 - y1c)) / (y2 - y1);
      y1 = y1c;
    }
    else
    {
      x1 += ((x1 - x2) * (y1 - y2c)) / (y2 - y1);
```

```
      y1 = y2c;
    }
  }
}

// Funkcja gfxClipLineTo() rysuje odcinek od
zapamiętanych współrzędnych
// do nowych współrzędnych, które po operacji są
zapamiętywane. Odcinek
// jest obcinany do podanego prostokąta
// screen - wskaźnik struktury SDL_Surface
// x,y    - współrzędne końca odcinka
// color  - kod koloru odcinka
// clip   - wskaźnik struktury SDL_Rect z
prostokątem obcinającym
//-------------------------------------------------
-----------------

void gfxClipLineTo(SDL_Surface * screen, Sint32 x,
Sint32 y, Uint32 color)
{
   gfxClipLine(screen, gfx_x_coord, gfx_y_coord, x,
y, color);
   gfx_x_coord = x; gfx_y_coord = y;
}
```

UWAGA: Na końcu pliku nagłówkowego SDL_gfx.h dopisz:

```
void gfxClipLine(SDL_Surface * screen, Sint32 x1,
Sint32 y1, Sint32 x2, Sint32 y2, Uint32 color);

void gfxClipLineTo(SDL_Surface * screen, Sint32 x,
Sint32 y, Uint32 color);
```

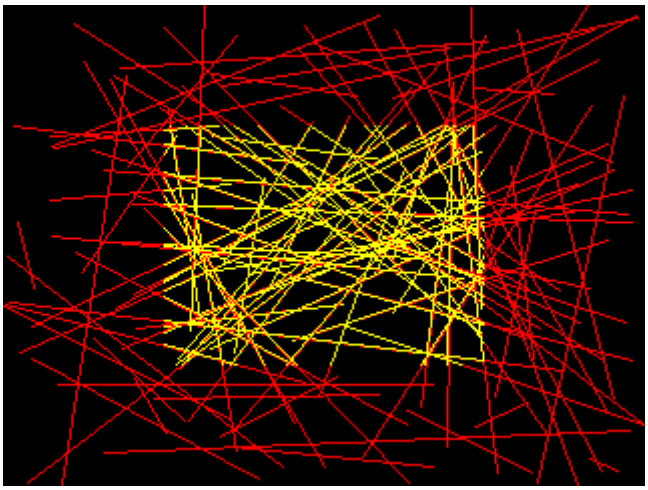
Poniższy program testowy definiuje na środku ekranu prostokąt obcinający, następnie generuje 100 linii o losowych współrzędnych końców i rysuje je raz w kolorze czerwonym bez obcinania i raz w kolorze żółtym z obcinaniem.

```
//
// P013 - obcinanie linii
//-----------------------------

#include <windows.h>
#include <SDL/SDL_gfx.h>
```

```
#include <time.h>     // do inicjalizacji
generatora pseudolosowego

const int SCRX = 320; // stałe określające
szerokość i wysokość
const int SCRY = 240; // ekranu w pikselach

int main(int argc, char *argv[])
{

  if(SDL_Init(SDL_INIT_VIDEO))
  {
    MessageBox(0, SDL_GetError(), "Błąd
inicjalizacji SDL", MB_OK);
    exit(-1);
  }

  atexit(SDL_Quit);

  SDL_Surface * screen;

  if(!(screen = SDL_SetVideoMode(SCRX, SCRY, 32,
SDL_HWSURFACE)))
  {
    MessageBox(0, SDL_GetError(), "Błąd tworzenia
bufora obrazowego", MB_OK);
    exit(-1);
  }

  if(SDL_MUSTLOCK(screen))
    if(SDL_LockSurface(screen) < 0)
    {
      MessageBox(0, SDL_GetError(), "Błąd blokady
bufora obrazowego", MB_OK);
      exit(-1);
    }

// inicjujemy generator liczb pseudolosowych

  srand((unsigned)time(NULL));

// definiujemy prostokąt
```

```
  SDL_Rect * clip = new SDL_Rect; // prostokąt
obcinania

  clip->x = screen->w >> 2;       // 1/4 szerokości
ekranu
  clip->y = screen->h >> 2;       // 1/4 wysokości
ekranu
  clip->w = screen->w >> 1;       // 1/2 szerokości
ekranu
  clip->h = screen->h >> 1;       // 1/2 wysokości
ekranu

// ustawiamy prostokąt obcinający

  SDL_SetClipRect(screen, clip);

// generujemy 100 losowych linii

  for(int i = 0; i < 100; i++)
  {
    Sint32 x1 = rand() % screen->w;
    Sint32 x2 = rand() % screen->w;
    Sint32 y1 = rand() % screen->h;
    Sint32 y2 = rand() % screen->h;

// Rysujemy czerwoną linię bez obcinania

    gfxLine(screen,x1,y1,x2,y2,0xff0000);

// Rysujemy żółtą linię z obcinaniem

    gfxClipLine(screen, x1, y1, x2, y2, 0xffff00);

  }

  if(SDL_MUSTLOCK(screen))
SDL_UnlockSurface(screen);

  SDL_UpdateRect(screen, 0, 0, 0, 0);

  MessageBox(0, "Kliknij przycisk OK", "Koniec",
MB_OK);
  return 0;
```

```
}
```



Zwróć uwagę, iż pod liniami żółtymi widać linie czerwone. Również końce niektórych linii żółtych nie pokrywają się dokładnie z liniami czerwonymi. Powodów jest kilka. Najważniejszy z nich to użycie przez nas arytmetyki liczb całkowitych do wyznaczania współrzędnych punktów przy przycinaniu odcinka. Tymczasem właściwy algorytm Cohena-Sutherlanda operuje na liczbach rzeczywistych, co znacznie zwiększa dokładność wyznaczania obcięć. Drugi powód jest taki, iż algorytm rysowania linii nie przechodzi zwykle przez wszystkie te same punkty, gdy rozpoczniemy rysowanie od punktu leżącego w innym miejscu linii - również arytmetyka liczb całkowitych ma tutaj swój wpływ. Nasza wersja algorytmu Cohena-Sutherlada nie nadaje się zatem do zastosowań profesjonalnych, ale w grafice szkolnej czy amatorskiej można z powodzeniem go wykorzystywać.

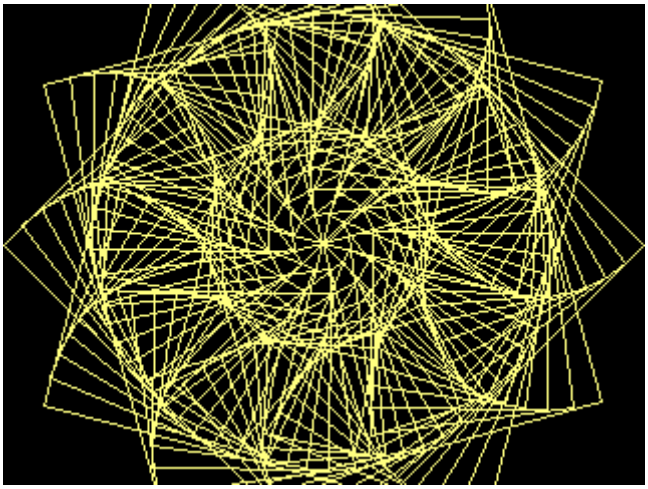
Poniższy program wykorzystuje obcinanie do swobodnego narysowania figury geometrycznej, która nie mieści się w całości na obszarze graficznym.

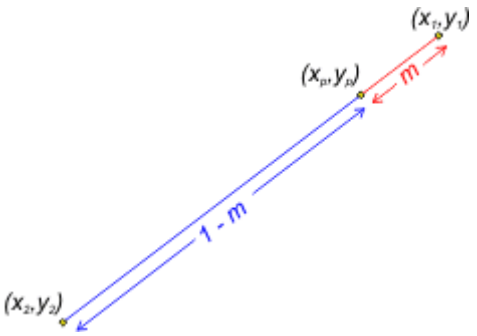
```
//
// P014 - przykładowa figura
//--------------------------

#include <windows.h>
#include <SDL/SDL_gfx.h>
#include <time.h>      // do inicjalizacji
generatora pseudolosowego
#include <math.h>      // udostępnia funkcje
trygonometryczne

const int SCRX = 320; // stałe określające
szerokość i wysokość
const int SCRY = 240; // ekranu w pikselach

int main(int argc, char *argv[])
{
```

```
  if(SDL_Init(SDL_INIT_VIDEO))
  {
    MessageBox(0, SDL_GetError(), "Błąd
inicjalizacji SDL", MB_OK);
    exit(-1);
  }

  atexit(SDL_Quit);

  SDL_Surface * screen;

  if(!(screen = SDL_SetVideoMode(SCRX, SCRY, 32,
SDL_HWSURFACE)))
  {
    MessageBox(0, SDL_GetError(), "Błąd tworzenia
bufora obrazowego", MB_OK);
    exit(-1);
  }

  if(SDL_MUSTLOCK(screen))
    if(SDL_LockSurface(screen) < 0)
    {
      MessageBox(0, SDL_GetError(), "Błąd blokady
bufora obrazowego", MB_OK);
      exit(-1);
    }

// inicjujemy generator liczb pseudolosowych

  srand((unsigned)time(NULL));

// Współrzędne środka ekranu

  Sint32 xs = screen->w >> 1;
  Sint32 ys = screen->h >> 1;

// Promień figury

  Sint32 rf = (screen->w >> 2);

  int const MAXF = 8;  // liczba figur w serii
```

```
// składowe koloru

  Uint32 r = rand() % 256;
  Uint32 g = rand() % 256;
  Uint32 b = rand() % 256;
  Uint32 color = (r << 16) | (g << 8) | b ;

// ustawiamy prostokąt obcinający na cały ekran (to
wywołanie nie jest konieczne, ale...)

  SDL_SetClipRect(screen, NULL);

// Obrót figury co 30 stopni

  for(int i = 0; i < 12; i++)
  {
    double fi = 6.283185 * i / 12;

// współrzędne środka figury

    Sint32 xfs = xs + rf * cos(fi);
    Sint32 yfs = ys + rf * sin(fi);

// Współrzędne wierzchołków

    Sint32 x[5], y[5], nx[4], ny[4];

    for(int j = 0; j < 4; j++)
    {
      x[j] = xfs + rf * cos(6.283185 * j / 4 + fi);
      y[j] = yfs + rf * sin(6.283185 * j / 4 + fi);
    }

// Rysujemy figurę

    for(int k = 0; k < MAXF; k++)
    {
      x[4] = x[0]; y[4] = y[0];

// Rysujemy kwadrat

      gfxMoveTo(x[0], y[0]);
```

```
      for(int j = 1; j < 5; j++)
gfxClipLineTo(screen, x[j], y[j], color);

// wyznaczamy nowe współrzędne na bokach obecnego
kwadratu

      for(int j = 0; j < 4; j++)
      {
        nx[j] = (x[j + 1] * (MAXF - 1) + x[j]) /
MAXF;
        ny[j] = (y[j + 1] * (MAXF - 1) + y[j]) /
MAXF;
      }

// Nowe współrzędne stają się aktualnymi

      for(int j = 0; j < 4; j++)
      {
        x[j] = nx[j]; y[j] = ny[j];
      }
    }
  }

  if(SDL_MUSTLOCK(screen))
SDL_UnlockSurface(screen);

  SDL_UpdateRect(screen, 0, 0, 0, 0);

  MessageBox(0, "Kliknij przycisk OK", "Koniec",
MB_OK);
  return 0;
}
```





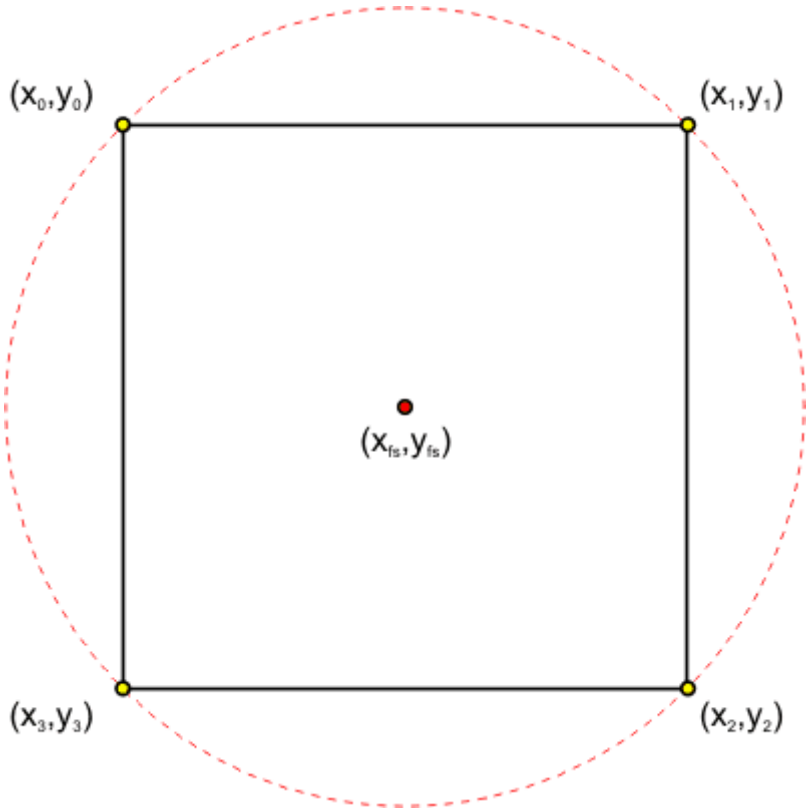
Do narysowania tej figury wykorzystujemy prostą metodę znajdowania punktu podziałowego na odcinku. Jeśli mamy odcinek (x_1,y_1) - (x_2,y_2) i chcemy znaleźć na nim punkt P dzielący ten odcinek w stosunku m : (1 - m), gdzie $m \in R$, $m \in (0,1)$, to stosujemy następujące wzory:

$x_p = (1 - m)x_1 + mx_2$
$y_p = (1 - m)y_1 + my_2$

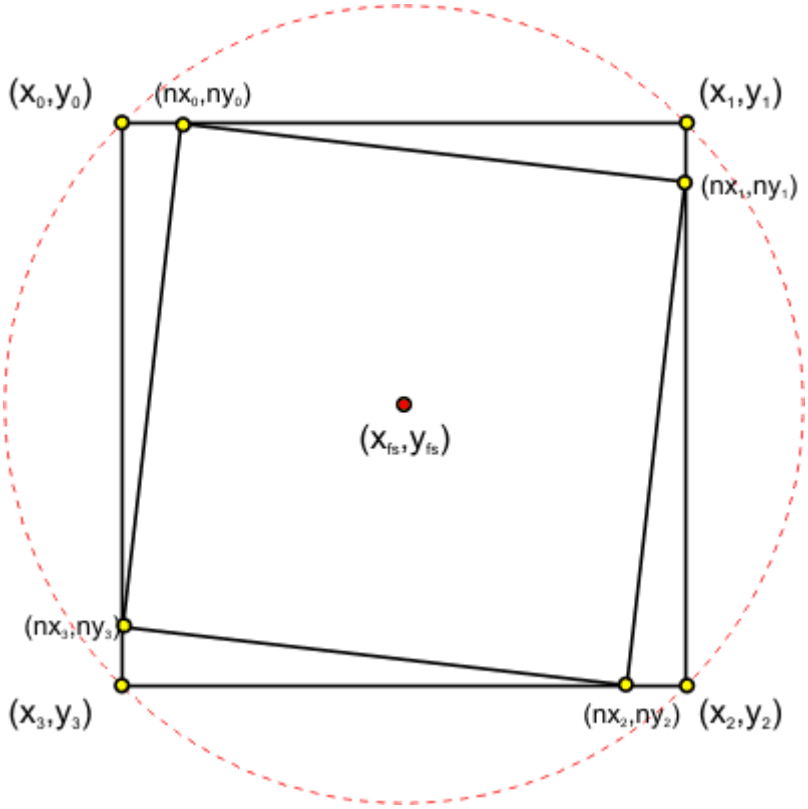
Jeśli m wyrazimy w postaci ułamka, to powyższe działania możemy wykonać w dziedzinie liczb całkowitych (tak robimy w programie), jednakże musimy się liczyć z błędami zaokrągleń.

Program wyznacza najpierw cztery wierzchołki kwadratów. Ponieważ kwadraty będą obracane o kąt fi, wykorzystujemy postać parametryczną równania okręgu do wyliczenia współrzędnych tych punktów:

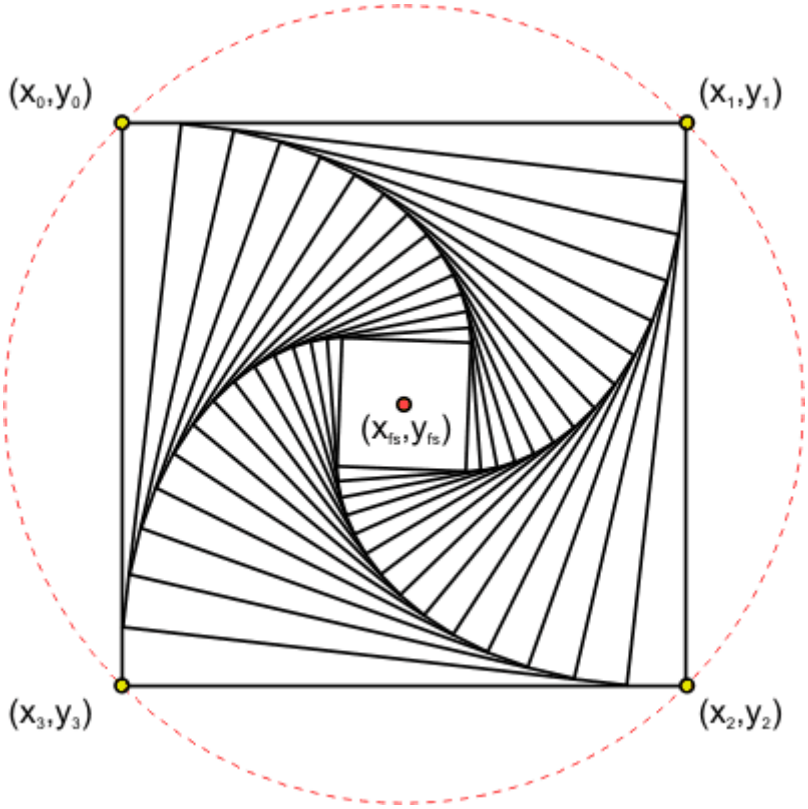
$x_i = x_{fs} + r_f\cos(2\pi i/4 + \varphi)$
$y_i = y_{fs} + r_f\sin(2\pi i/4 + \varphi)$ gdzie x_{fs}, y_{fs} są środkiem figury, r_f jest promieniem okręgu a φ to kąt obrotu



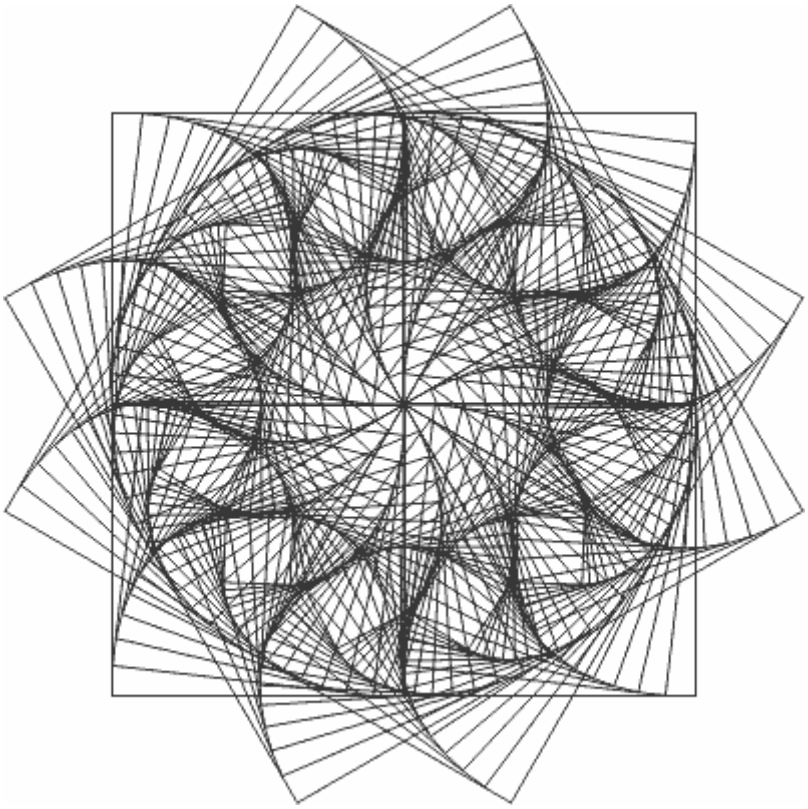
Po narysowaniu pierwszego kwadratu jego boki dzielimy wg opisanej wyżej metody. Znalezione punkty podziałowe są wierzchołkami kolejnego kwadratu.



Procedurę powtarzamy dla nowego kwadratu odpowiednią liczbę razy. Otrzymujemy poniższą figurę:



Teraz wystarczy dla każdej figury wyznaczyć środki x_{fs}, y_{fs} leżące na okręgu o promieniu r_f i środku w środku ekranu, a otrzymamy figurę złożoną:



Wykorzystaj podane informacje do napisania programów rysujących poniższe figury: