ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1992
Seria: INFORMATYKA z. 16 Nr kol.1140

Ibrahim CHAMI

ALGORYTMY KREŚLENIA WEKTORÓW METODĄ RASTROWĄ W SYSTEMACH INTERAKCYJNYCH

Streszczenie. Praca przedstawia algorytmy opierające się na wcześniej znanym algorytmie Euklidesa i na algorytmach Bresenhama. Celem niniejszego opracowania jest porównanie algorytmów tworzenia wektorów ze względu na dokładność i szybkość, czyli wybór najdokładniejszego i najszybszego algorytmu ze względu na zakres zastosowania.

Praca zawiera również propozycję modyfikacji najnowszego algorytmu Pittewaya-Castle'a (opartego na algorytmie Euklidesa).

Summary. The purpose of this paper is to present, analyse and make a comparison between algorithms of generation vectors based on Euclid's and Bresenham's algorithms from the time and accuracy point of view. In this regard, we select the fastest and the most exact algorithm with taking the application scope into consideration. Also, this paper suggests a modification of Pitteway-Castle's algorithm based on Euclid's algorithm.

Резюме. Работа знакомит с алгоритмами, которые основаны на раньше известном алгоритме Евклида и на алгоритмах Брезенгама. Целью предлагаемой работы является сравнение алгоритмов создания векторов учитывая точность и скорость, что обозначает подбор самого точного и самого скорого алгоритма в зависилости от объема применения.

Работа предлагает также модификацию новейтего алгоритма Питтлея-Кестля (на основании алгоритма Евклида).

1. WSTĘP

Proste w technice rastrowej przedstawiane są na urządzeniach graficznych, takich jak: drukarki mozaikowe, laserowe, monitory, plotery przyrostowe.

Obrazy graficzne zapamiętywane w postaci mapy bitów nazywane są obrazami rastrowymi, natomiast obrazy zapamiętywane jako lista podstawowych elementów

i procedur noszą nazwę obrazów wektorowych. Parametry wszystkich elementów obrazu rastrowego są obliczone i wyświetlane w postaci odpowiadających im punktów ekranu (lub węzłów siatki płaszczyzny obrazowania) zgodnie z zawartością komórek pamięci (pamięć obrazu).

Płaszczyzna obrazu jest płaszczyzną dyskretną ze zdefiniowanymi węzłami całkowitoliczbowej siatki współrzędnych [7]. Wykreślanie wektorów na takich płaszczyznach odbywa się przez zadanie ciągu punktów, które zapalone albo połączone kolejno odcinkami linii prostej aproksymują zadany wektor. Punkty te powinny pokrywać się z węzłami siatki współrzędnych.

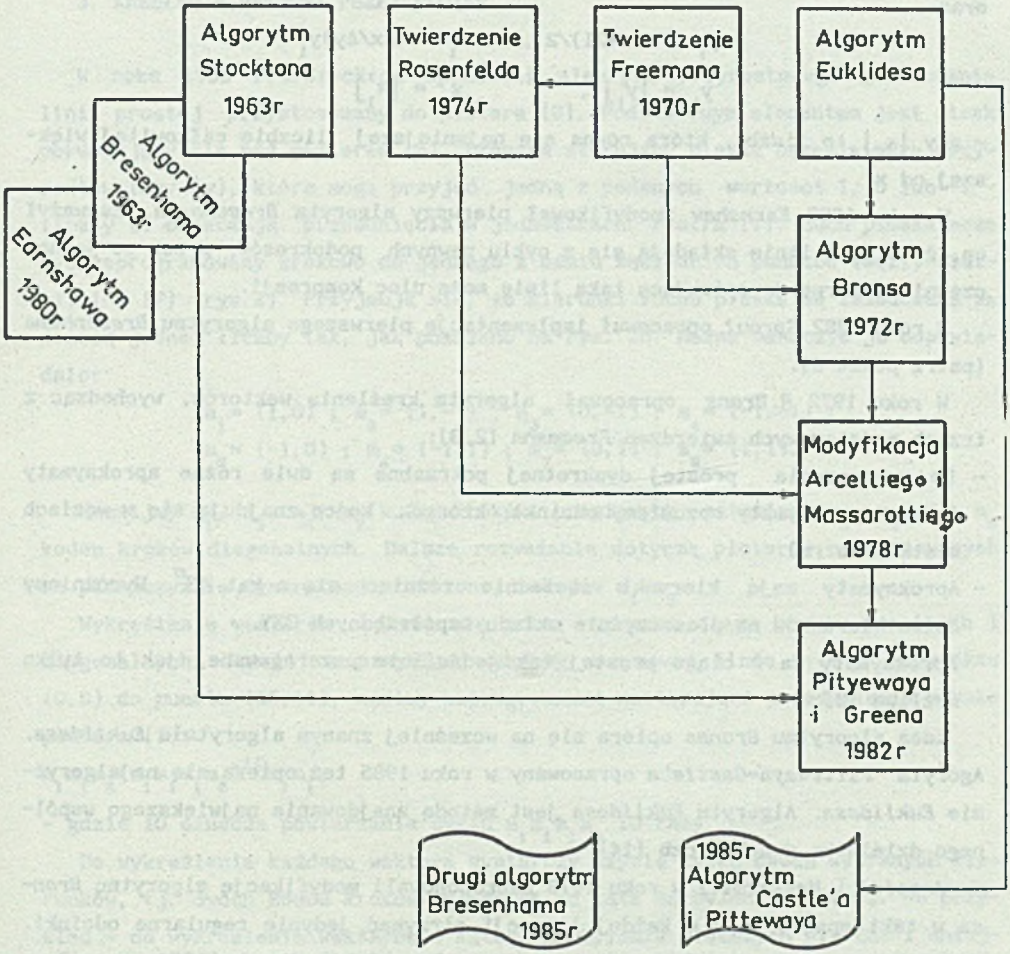
Wybrany algorytm powinien spełniać kilka wymagań, często trudnych do pogodzenia [9]:

- szybkość,
- stała, grubość kreślonej linii,
- aproksymacja powinna dawać linię najbliższą prostej,
- odcinek musi łączyć (lub zapalić) punkty o ustalonych współrzędnych.

2. ALGORYTMY KREŚLENIA WEKTORÓW

F.G.Stocton w roku 1963 jako pierwszy [8] opracował algorytm przedstawienia wektorów na siatce rastrowej. Węzły aproksymujące są tu wybierane na zasadzie minimum odległości mierzonej wzdłuż osi X bądź Y od wektora zadanego. Wyznaczanie punktów ciągu aproksymującego odbywa się na podstawie wyników testowania znaków następujących wielkości: Δx, Δy - rzutów żądanego odcinka linii na osie układu współrzędnych, $(\Delta x + \Delta y)$ i $(\Delta y - \Delta x)$ - sumy i różnicy rzutów; $|\Delta x + \Delta y|$ i $|\Delta y - \Delta x|$ - sumy i różnicy modułów rzutów oraz $[\min(|\Delta x|, |\Delta y|) - \max(|\Delta x|, |\Delta y|)]$ - różnicy między podwojonym mniejszym z modułów a modułem większym [7].

Bresenham opracował dwie metody kreślenia wektorów. Idea pierwszej metody opracowanej w roku 1965 [2] polega na zwiększaniu wartości x o wielkość jednostki rastra i znalezieniu w każdym kroku punktu leżącego najbliżej linii rzeczywistej, czyli metoda ta oparta jest na podobnej zasadzie, co algorytm *Stocktona* (rys.1). Zaletą algorytmu *Bresenhama* jest to, że szybkość realizacji jest o 50% większa od algorytmu *Stocktona* [8].



Rys. 1. Algorytmy kreślenia wektorów w przestrzeni rastrowej

Fig. 1. Methods of generation vectors which based on Euclide's and Bresenham's algorithms

Idea drugiej metody opracowanej w roku 1985 polega na zwiększaniu wartości I o wielkość jednostki rastra [1].

$$I = 0, 1, 2 \ldots\ldots\ldots\ldots \Delta y-1$$

oraz

$$y_1 = (1 + 2I)/2 , \qquad x_1 = (\Delta x/\Delta y)y_1$$

$$y_1 = \lfloor y_1 \rfloor , \qquad x_1 = \lfloor x_1 \rfloor$$

- gdy $\lfloor x_1 \rfloor$ to liczba, która równa się najmniejszej liczbie całkowitej większej od x_1.

W roku 1980 *Earnshaw* zmodyfikował pierwszy algorytm *Bresenhama*. Zauważył on, że długie linie składają się z cyklu pewnych podokresów, przez co jednocześnie informacja opisująca taką linię może ulec kompresji.

W roku 1982 *Sproul* opracował implementację pierwszego algorytmu *Bresenhama* (patrz punkt 6).

W roku 1972 *R.Brons* opracował algorytm kreślenia wektorów, wychodząc z trzech podstawowych twierdzeń *Freemana* [2,8]:

- Do wyznaczenia prostej dyskretnej potrzebne są dwie różne aproksymaty (przez aproksymaty rozumiemy odcinki, których końce znajdują się w węzłach siatki rastra).
- Aproksymaty mają kierunki sąsiednie różniące się o kąt 45°. Wyróżniamy osiem kierunków na płaszczyźnie układu współrzędnych OXY.
- Aproksymaty sa w ciągu prostej tak jednolicie uszeregowane, jak to tylko możliwe [8].

Idea algorytmu *Bronsa* opiera się na wcześniej znanym algorytmie *Euklidesa*. Agorytm *Pittewaya-Castle'a* opracowany w roku 1985 też opiera się na algorytmie *Euklidesa*. Algorytm *Euklidesa* jest metodą znajdowania największego wspólnego dzielnika dwóch liczb [14].

Arcelli i *Massarotti* w roku 1978 zaproponowali modyfikację algorytmu *Bronsa* w taki sposób, aby w każdej iteracji otrzymać jedynie regularne odcinki. Modyfikacja ta opiera się na twierdzeniu *Rosenfelda*. W roku 1974 opisał on strukturę łuków cyfrowych (ang. *digital* arc), która będzie omówiona w rozdziale 4.

W roku 1982 *Pitteway* i *Green* opracowali algorytm opierający się jednocześnie na algorytmie *Euklidesa* i *Bresenhama*. W roku 1985 *Pitteway* przerobił metodę w części opierającej się na algorytmie *Euklidesa* [11] (patrz punkt 7). Na rysunku 1 przedstawiono historię opracowania omówionych algorytmów. Algorytmy te opisano w dalszej części.

3. KREŚLENIE WEKTORA PRZEZ PLOTER

W roku 1963 *F.G.Stockton* opracował algorytm przyrostowego generowania linii prostej przystosowany do plotera [8]. Podstawowym elementem jest pisak poruszający się nad papierem [6]. Program sterujący pisaka określa para przyrostów (Δx, Δy), które mogą przyjąć jedną z podanych wartości 1, 0 lub -1 - liczby te oznaczają przesunięcia w jednostkach rastra [1]. Ruch pisaka może być zaprogramowany krokowo do jednego z ośmiu sąsiednich punktów (węzły siatki) [6] [7] (rys.2). Przyjmuje się, że kierunki ruchu pisaka są zakodowane za pomocą jednej liczby tak, jak pokazano na rys. 2b. Można oznaczyć je odpowiednio:

$$m_1 = (1,0)\ ;\ m_2 = (1,-1)\ ;\ m_3 = (0,-1)\ ;\ m_4 = (-1,-1)\ ;$$
$$m_5 = (-1,0)\ ;\ m_6 = (-1,1)\ ;\ m_7 = (0,1)\ ;\ m_8 = (1,1).$$

Nazwijmy m_1, m_3, m_5, m_7 kodem kroków poosiowych, natomiast m_2, m_4, m_6, m_8 kodem kroków diagonalnych. Dalsze rozważania dotyczą ploterów programowanych na płaszczyźnie dyskretnej utworzonej z kodów m_1-m_8.

Wykreślenie wektorów odbywa się przez zadania ciągu kroków poosiowch i diagonalnych. Jeżeli na przykład chcemy narysować linię prostą od punktu (0,0) do punktu (45,11), musimy zaprogramować następujące przesunięcie pisaka plotera [1]:

$m_1 m_1 m_8 (m_1 m_1 m_1 m_8)^{10} m_1 m_1$,

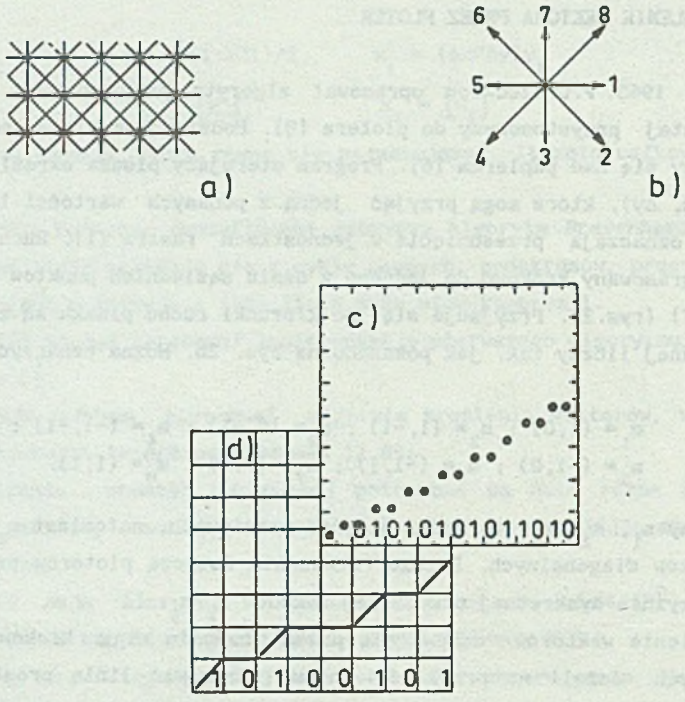
- gdzie 10 oznacza powtarzanie cyklu $m_1 m_1 m_1 m_8$ 10 razy.

Do wykreślenia każdego wektora wystarczy użycie tylko dwóch wybranych kierunków, tj. dwóch kodów kroków zależnych od kąta nachylenia wektora. Na przykład - do wykreślenia wektorów o kątach nachylenia większych niż 225° i mniejszych niż 270° używamy ciągu kodów m_3 i m_4.

Załóżmy że:

$$\Delta x \geq \Delta x \geq 0....$$

Wynika z tego, że nachylenie wektora do osi OX nie może być większe niż 45°. To założenie przyjęto dla wszystkich metod generowania wektorów, które będą omówione w dalszej części. Rozszerzanie zakresu stosowania tych metod dla różnych kątów nachylenia wektora uzyskuje się poprzez wykorzystanie syme-



Rys. 2. Przykłady zaprogramowania kreślenia wektora
a) jednorodne siatki kwadratowe 8-spójne
b) kierunki zaprogramowania ruchu pisaka plotera
c) odwzorowanie prostej na siatce monitora
d) odwzorowanie prostej na siatce plotera

Fig. 2. Examples show the behaviour of drawing vectors
a) homogeneoussquare8-connectedgrid
b) the Freeman chain code for determination the direction of movement (modulo eight)
c) representation of the drawing line on an equivalent graphic display device
d) representation of the drawing line on an

trii w układzie współrzędnych, co oznacza w pierwszm algorytmie *Bresenhama* zmianę symetrii Δx, Δy odpowiednio do kąta nachylenia wektora. Na przykład na linii o kątach nachylenia od 135° do 180° należy zamienić Δx i Δy zgodnie z równaniami:

$$\Delta x \leftarrow -\Delta x$$
$$\Delta y \leftarrow \Delta y$$

Natomiast od 225° do 270° dokonać należy zamiany:

$$\Delta x \leftarrow -\Delta y$$
$$\Delta y \leftarrow -\Delta x$$

Dla wszystkich metod omówionych w dalszej części, czyli dla metod, których wynik wykonania algorytmu jest ciągiem kodów kroków poosiowych i diagonalnych, rozwiązanie problemu rozszerzania dla całego układu kartezjańskiego jest następujące:

Niech macierz:

$$m = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}$$

gdzie:

m_{11} - oznacza wartość współrzędnej x dla kodu kroku poosiowego,

m_{12} - oznacza wartość współrzędnej y dla kodu kroku poosiowego,

m_{21} - oznacza wartość współrzędnej x dla kodu kroku diagonalnego,

m_{22} - oznacza wartość współrzędnej y dla kodu kroku diagonalnego,

czyli (m_{11}, m_{12}) oznacza kod kroków poosiowych, a (m_{21}, m_{22}) kod kroków diagonalnych. Na przykład - dla wektorów o kątach nachylenia większych niż 180° i mniejszych niż 225° mamy:

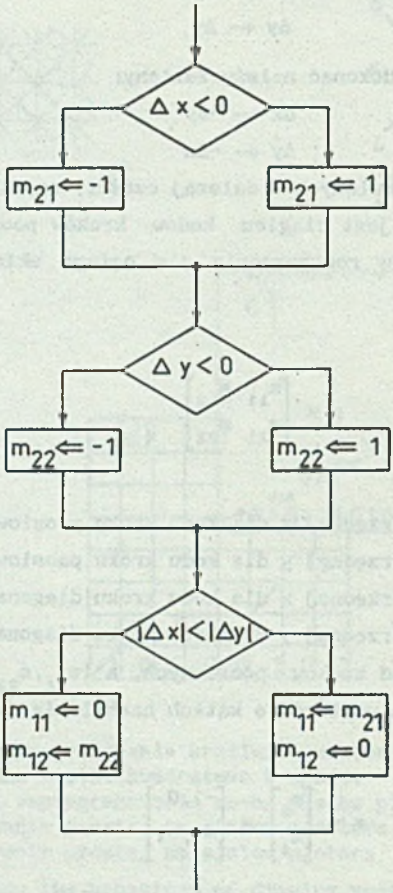
$$m = \begin{bmatrix} m_5 \\ m_4 \end{bmatrix} = \begin{bmatrix} -1,0 \\ -1,-1 \end{bmatrix}$$

czyli $m_{11} = m_{21} = m_{22} = -1, \quad m_{12} = 0$

Na schemacie blokowym (rys. 3) pokazano część algorytmu, który daje odpowiednie wartości kodu kroków m_{11}, m_{12}, m_{21} i m_{22}, zależnie od kąta nachylenia wektorów.

Opierając się na przyjętych wyżej definicjach można przystąpić do przeanalizowania poszczególnych algortmów kreślenia wektorów. W ich opisach przyjęto oznaczenia:

- symbol 0:krok poosiowy
- symbol 1:krok diagonalny



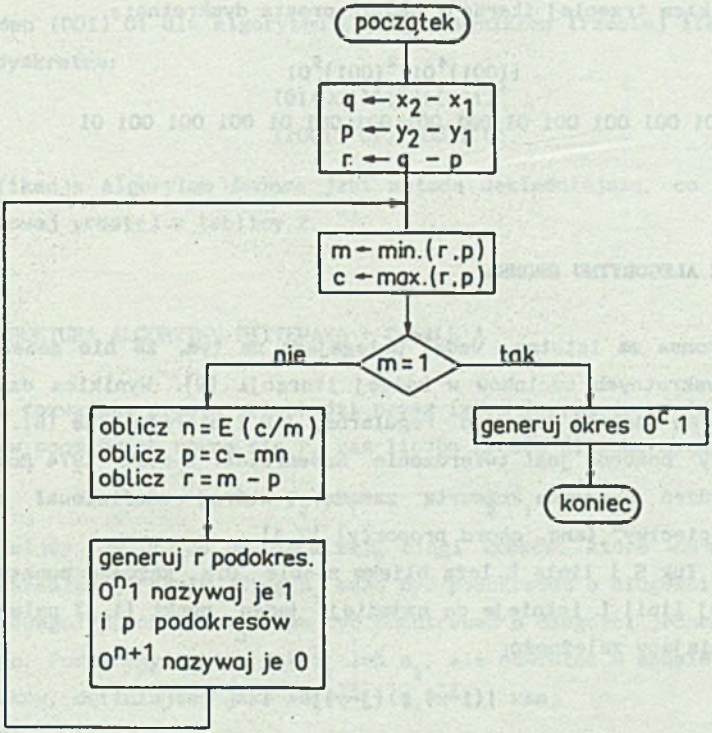
Rys. 3. Schemat blokowy obliczania kodu kroków kreślenia wektora

Fig. 3. The scheme to caculate the constants of direction incremental movement

4. STRUKTURA ALGORYTMU BRONSA

Algorytm *Bronsa* polega na znalezieniu na prostej dyskretnej (opis struktury prostych przybliżonych w dyskretnej siatce rastrowej omówiono w rozdz. 2 na podstawie twierdzenia *Freemana*) powtarzających się podokresów [8]. Wykreś-

lenie prostej powstaje w wyniku złożenia powtarzających się sekwencji podokresów. W kolejnych iteracjach algorytmu zwiększa się podokresy aż do ich ostatecznego określenia. Proces generacji wektora za pomocą algorytmu *Bronsa* jest równoległy [8], co oznacza, iż w kolejnych stadiach generacji dyskretnch odcinków „pozostają one niezależne (tj. nie połączone). Kompletną prostą dyskretną otrzymuje się w końcowym stadium generacji. Algorytm przedstawiony na rys. 4 zakłada, że dla prostej o nachyleniu p/q zachodzi $0 \leq p \leq q$ p i q - całkowite.



Rys. 4. Schemat blokowy algorytmu Bronsa
Fig. 4. Brons' algorithm

Rozważmy teraz następujący przykład generacji prostej przez algorytm *Bronsa*. Niech prosta przechodzi przez punkty (0,0) oraz (39,14). Wartości początkowe dla q, p, r:

$$q = 39,\ p = 14,\ r = 25$$

czyli $q = \Delta x,\ p = \Delta y$ i $r = q-p$

Mamy zatem 25 kroków poosiowych, w tym przypadku horyzontalnych i 14 kroków diagonalnych (0^{25}, 1^{14}). Po pierwszej iteracji otrzymamy jedenaście odcinków 001 i trzy odcinki 01, czyli $(001)^{11}$ $(01)^{9}$. Po drugiej iteracji będziemy mieli dwa odcinki $(001)^{4}01$ i jeden odcinek $(001)^{9}01$, czyli $((001)^{4}01)^{2}$ $(001)^{3}01$. Wynikiem trzeciej iteracji będzie prosta dyskretna:

$$((001)^{4}01)^{2}(001)^{9}01$$

tj.

001 001 001 001 01 001 001 001 001 01 001 001 001 01

MODYFIKACJE ALEGORYTMU BRONSA

Algorytm *Bronsa* ma istotną wadę, polegającą na tym, że nie generuje on regularnych dyskretnych odcinków w każdej iteracji [8]. Wynikiem działania tego algorytmu są odcinki zarówno regularne, jak i nieregularne [8]. W usunięciu tej wady pomocne jest twierdzenie *Rosenfelda*. W roku 1974 *Rosenfeld* dodał do twierdzeń *Freemana* czwartą zasadę, w której zdefiniował pojęcia "przyleżności cięciwy" (ang. *chord proporty*) [2,4].

Załóżmy, że łuk S i linia L leżą blisko siebie. Dla każdego punktu (x,y) na rzeczywistej linii L istnieje co najmniej jeden punkt (i,j) należący do łuku S i spełniający zależność:

$$\max |(i-x)\ ,\ (j-y)| < 1$$

możemy wtedy mówić o zawieraniu się cięciwy w linii $_{L}$ [4].

Wychodząc z powższego *Arcelli* i *Massarotti* zaproponowali modyfikację algorytmu *Bronsa*. Wykazali, że algorytm prostej zaproponowany przez *Bronsa* rzeczywiście generuje odcinek prostej dyskretnej, tj. dyskretyzację segmentu prostej, a jej dyskretna postać spełnia wyżej wymienioną zależność cięciwy

[8]. Modyfikacja wymaga wprowadzenia funkcji pomocniczej:

$$S_i = \begin{cases} 1 \text{ dla } p_i > r_i \\ 0 \text{ dla } p_i < r_i \end{cases}$$

$$t_i = t_{i+1} \cdot s_i \quad ; \quad t_o = 1$$

Schemat blokowy odpowiedniego algorytmu przedstawiono na rys. 5.

Porównując dla prostej o równaniu y = (14 /39)x algorytm *Bronsa* z jego modyfikacją otrzymamy: Po drugiej iteracji dwa odcinki $01(001)^4$ oraz jeden odcinek $01(001)^9$ dla modyfikacji algortmu *Bronsa*, zamiast dwóch odcinków $(001)^4$ 01 i jeden $(001)^9$01 dla algorytmu *Bronsa* i wynikiem trzeciej iteracji będzie prosta dyskretna:

$$(01(001)^4)^2 01(001)^9$$

zamiast:

$$((001)^4 01)^2 (001)^9 01$$

Modyfikacja algorytmu *Bronsa* jest metodą dokładniejszą, co wykazano dla przykładowej prostej w tablicy 2.

5. STRUKTURA ALGORYTMU PITTEWAYA - CASTLE'A

Niech rozważana prosta przechodzi przez (x_1, y_1) oraz (x_2, y_2). Jeżeli liczba kroków poosiowych równa się r, zaś liczba kroków diagonalnych równa się b, to:

$$b = y_2 - y_1 \qquad r = x_2 - x_1 - b$$

Przyjmijmy, że m_1 lub m_2 oznaczają ciągi kroków, które wzrastają według ściśle określonej reguły. Ciąg m_1 może być podokresem o długości jednego kroku poosiowego. Natomiast m_2 może być podokresem o długości jednego kroku diagonalnego. Podokresy takie, jak m_1 lub m_2, ale odwrotne w sensie ciągu kolejnych kroków, definiujemy jako $(m_1)^{-1}$; $(m_2)^{-1}$.

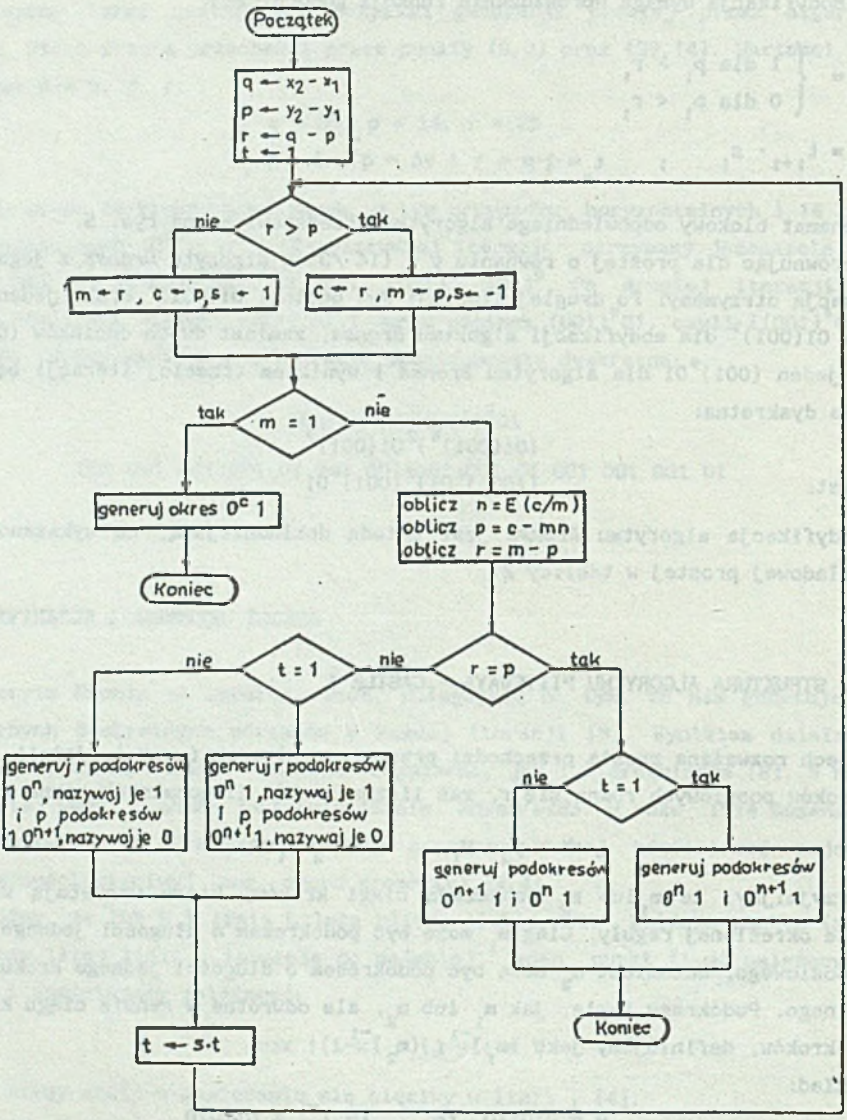
Przykład:

$m_1 = 010110$ to $(m_1)^{-1} = 011010$

$m_2 = 110$ to $(m_2)^{-1} = 011$

$m_1 = 0$ to $(m_1)^{-1} = 0$

$m_2 = 1$ to $(m_2)^{-1} = 1$



Rys. 5. Schemat blokowy modyfikacji algorytmu Bronsa według Arcellego i Massarottiego

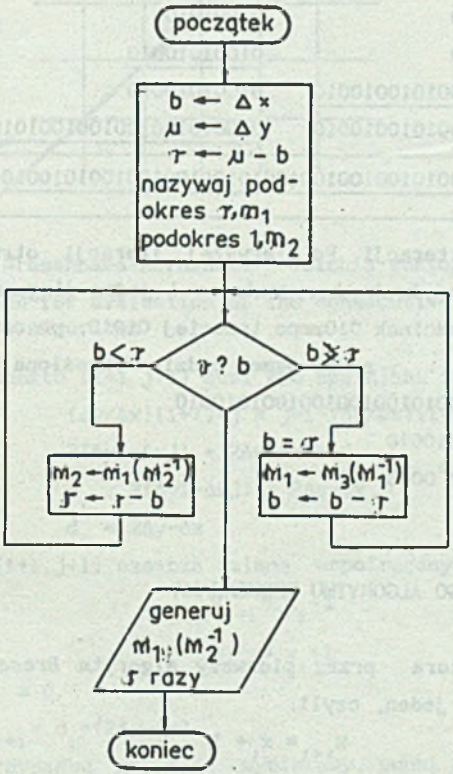
Fig. 5. Arcelli-Massarotti's allgorithm (modifed Brons' algorithm)

W każdym stadium generacji wektora za pomocą algorytmu *Pittewaya-Castle'a* otrzymuje się odpowiednie podokresy połączone z podokresem generowanym we wszystkich poprzednich stadiach. Kompletną prostą dyskretną otrzymuje się w końcowym stadium generacji, przy czym [3]:

$$m_1 = m_2 (m_1)^{-1}$$

lub: $$m_2 = m_1 (m_2)^{-1}$$

Schemat tego algorytmu przedstawiono na rys. 6.



Rys. 6. Schemat blokowy algorytmu Pittewaya-Castleya
Fig. 6. Pitteway-Castle's algorithm

Przykład: dla tej samej prostej, której użyto w przykładzie algorytmu *Bronsa*, mamy:

$$b = y_2 - y_1 = 14$$
$$r = x_2 - x_1 - b = 25$$

r	b	m_1	m_2
25	14	0	01
11	14	010	01
11	3	010	01010
8	3	010	01001010
5	3	010	01001010010
2	3	01001010010010	01001010010
2	1	01001010010010	0100101001001001001010010
1	1	0100101001001001001010010010010010010010	

W przykładzie mamy 8 iteracji. Po pierwszej iteracji otrzymamy odcinek o długości 01, czyli jeden krok horyzontalny i jeden diagonalny. Po drugiej iteracji będziemy mieli odcinek 010, po trzeciej 01010, po czwartej 01001010, po piątej 01001010010 ....., aż po ósmej będzie określona cała prosta dyskretna: 0100101001001001001010010010010010010010

czyli: $01001(01(001)^4)^2010010$

lub: $0100101((001)^401)^20010$

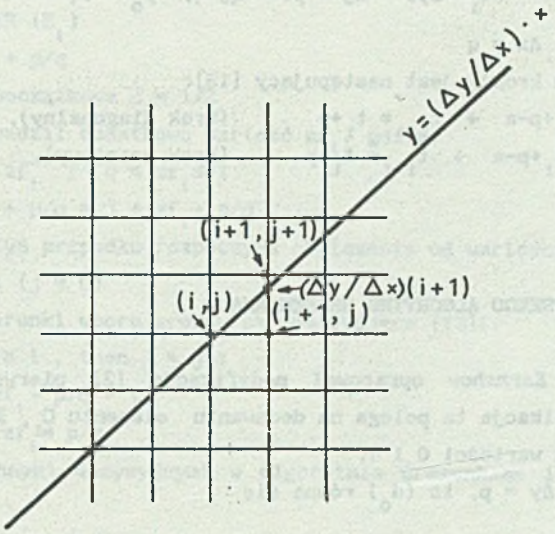
6. STRUKTURA PIERWSZEGO ALGORYTMU BRESENHAMA

Sposób tworzenia wektora przez pierwszy algorytm *Bresenhama* polega na zwiększeniu wartości x o jeden, czyli:

$$x_{i+1} = x_i + 1$$

i znalezieniu w każdym kroku wartości y_{i+1} [2][5][12].

Przyjmijmy, że $x_1 = i$; $y_1 = j$, wybór następnego punktu (i+1, j) lub (i+1, j+1) zależy od różnicy odległości pomiędzy punktem określonym analitycznym równaniem prostej a punktami znajdującymi się w siatce rastra (rys.7).



Rys. 7. Algorytm Bresenhama-Earnshowa kreślenia wektora

Fig. 7. Ilustration for evaluation of the consecutive pixels of Bresenham-Earnshow's algorithm

Dla wybrania punktu (i+1, j+1) musi być spełniona zależność:

$$(\Delta y/\Delta x)(i+1)-j \geq j+1-(\Delta y/\Delta x)(i+1)$$

lub: $2(\Delta y i-\Delta x j) + 2\Delta y-\Delta x \geq 0$

niech: $d_i = 2(\Delta y i-\Delta x j) + 2\Delta y-\Delta x$

czyli: $d_o = 2\Delta y-\Delta x$

Wybór punktu (i+1, j+1) oznacza zmianę współrzędnych zgodnie z równaniami:

$$x_{i+1} = x_i + 1$$

$$y_{i+1} = y_i + 1$$

W tym przpadku $d_i \geq 0$,

wtedy: $d_{i+1} = d_i + (2\Delta y-\Delta x)$

Natomiast w przypadku, gdy $d_i < 0$ wybieramy punkt o współrzędnych (i+1, j), zaś wartość (d) dla następnego wyboru będzie: $d_{i+1} = d_i + 2\Delta y$.

Ciekawe będzie to spostrzeżenie, że już w roku 1964 *Thompson* [2, 13] opracował podobną metodę kreślenia wektorów. Jako zmiennej decyzyjnej w swoim algorytmie używał symbolu t_i zgodnie z równaniami [13]:

$$t_i = \Delta yi + \Delta j = pi + qj \quad ; \quad p_o = 0$$

gdzie: $\Delta y = p$, $\Delta x = q$

Warunek wyboru kroków jest następujący [13]:

Jeżeli $t_i + p \geq t_i + p - a \rightarrow t_{i+1} = t_i + p$ (krok diagonalny),

a jeżeli $t_i + p < t_i + p - a \rightarrow t_{i+1} = t_i + p$ (krok poosiowy).

MODYFIKACJE PIERWSZEGO ALGORYTMU BRESENHAMA

W roku 1980 *Earnshow* opracował modyfikację [2] pierwszego algorytmu *Bresenhama*. Modyfikacja ta polega na dodawaniu elementu C_i, który może przyjąć jedną z dwóch wartości 0 i 1.

Gdy: $\Delta x = q$ i $\Delta y = p$, to (d_o) równa się

$$d_o = 2p - q$$

Jeżeli $d_i \geq 0$, to $d_{i+1} = d_i + 2(p-q)$

$$C_{i+1} = 1 \quad \text{(krok diagonalny)}$$

Natomiast, jeżeli $d_i < 0$, to $d_{i+1} = d_i + 2p$

$$C_{i+1} = 0 \quad \text{(krok poosiowy)}$$

Ponadto *Earnshow* stosował następujące równanie [2]:

$$d_i = 2t_i + 2p - q$$

- gdzie t_i to zmienna decyzyjna algorytmu *Thompsona*.

Warunek wyboru kroku określony jest równaniem [2]:

$$(p/q)i > j + 1/2$$

W roku 1982 *Sproul* [2,12] opracował modyfikację pierwszego algorytmu *Bresenhama* i jego implementację w języku PASCAL. W programie swoim stosował następujące instrukcje:

$y_i = (p/q)i$ {gdy $i = 0,1,2....\Delta x$}

j = ENTER $(y_i + 1/2)$

W pierwszej instrukcji obliczamy wartość y_i, która odpowiada zadanej wartości i. Program wyznacza ciąg punktów (i,j) dla prostej o nachyleniu (p/q). W drugiej instrukcji wyznacza się kolejną wartość j. Wartość y_{i+1} określa następujące równanie: $y_{i+1} = y_i + p/q$.

Przez przyjęcie $y_i + 1/2 = Z_i$ otrzymamy następującą instrukcję

$j = \text{ENTER}(Z_i)$

$Z_{i+1} = Z_i + p/q$

gdzie wartość początkowa $Z_o = 1/2$

Sproul wprowadził dodatkowo wartość zf_i, gdzie:

$Z_i = j + zf_i \quad ; \quad 0 \le zf_i \le 1$

$Z_{i+1} = Z_i + p/q = j + zf_i + p/q$

Algorytm w tym przpadku rozpoczyna obliczenia od wartości

$(zf_i = 1/2)$ i $(j = 0)$

Natomiast warunki wboru kroków są następujące [12]:

if $zf_i + p/q \ge 1$, then $j = j+1$

$zf_{i+1} = zf_i + p/q - 1$

else $zf_{i+1} = zf_i = p/q$

Między zmiennymi decyzyjnymi w algorytmie *Bresenhama* i *Sproula* istnieje związek:

$d_i = 2p + 2(zf_i - 1)q$

lub $zf_i = (d_i - 2p)/2p+1$

gdzie przjmuje się następujące wartości początkowe:

$j = 0$ i $zf_i = 1/2$ dla algorytmu *Sproula*

$j = 0$ i $d_o = 2p-q$ dla algortmu *Bresenhama*

7. STRUKTURA ALGORYTMU PITTEWAYA-GREENA

Pitteway i *Green* wykorzystali w swoich rozważaniach pierwszy algorytm *Bresenhama*, w którym:

$$\text{Jeżeli } \begin{cases} 2(\Delta x - \Delta y) = k_i \\ 2\Delta y = l_i \end{cases} \quad \text{to} \quad \begin{cases} d_{i+1} = d_i - k_i & \text{dla } d_i \ge 0 \\ d_{i+1} = d_i - l_i & \text{dla } d_i < 0 \end{cases}$$

$d_o = 2p-q+2eq$, $\quad -1/2 \le e < 1/2$

Ponadto wykorzystali algorytm *Bronsa* i rozszerzyli go o następujący przypadek [10]:

jeżeli $n = c/m$ (lub $n = k/l$) to $n = n-1$

W opracowanym algorytmie zamiast symbolu n będziemy używali symbolu n_i wyznaczanego zgodnie z równaniami:

$$\left.\begin{aligned} n_i &= \text{ENT. } (k_i/l_i) \\ k_{i+1} &= k_i - n_i l_i \end{aligned}\right\} \text{ dla } k_i > l_i$$

wtedy generuje się podokresy $m_2 m_1^{n_i}$, które dla kolejnej iteracji przyjmują symbol m_2.

Analogicznie:

$$\left.\begin{aligned} n_i &= \text{ENT. } (l_i/k_i) \\ l_{i+1} &= l_i - n_i k_i \end{aligned}\right\} \text{ dla } l_i > k_i$$

Wtedy generuje się podokresy $m_1 m_2^{n_i}$, które dla kolejnej iteracji przyjmują symbol m_1,

gdzie m_1 i m_2 podokresy zdefiniowane w punkcie 5.

W roku 1985 *Pitteway* zastąpił n_i we wyżej opisanym algorytmie, powtarzając następujące równanie:

$$l_{i+1} = l_i - k_i \text{ dla } l_i > k_i$$

dopóki $l_{i+1} \leq k_{i+1}$. Przy każdej iteracji podstawia się tu za m_1 podokres $m_1 m_2$, czyli $m_1 m_2 \rightarrow m_1$. W przeciwnym razie, gdy $k_i > l_i$, powtarza się następujące równanie: $k_{i+1} = k_i - l_i$, dopóki $k_{i+1} \leq l_{i+1}$. Przy każdej iteracji podstawia się tu za m_2 podokres $m_2 m_1$, czyli $m_2 m_1 \Rightarrow m_2$.

Wynikiem obu algorytmów są te same proste dyskretne.

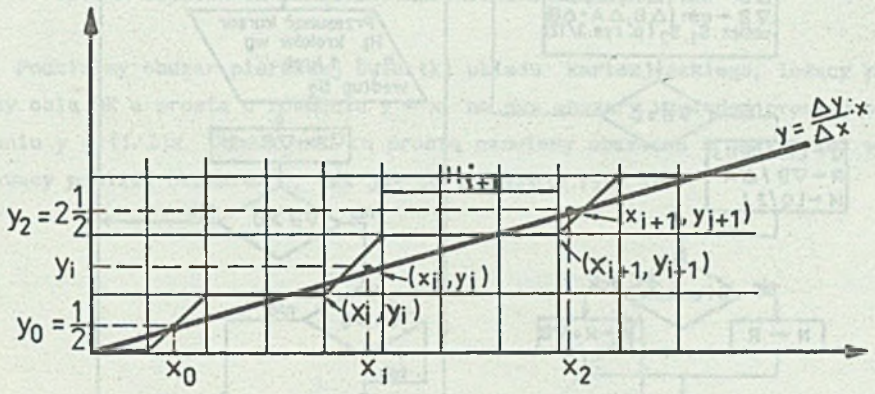
8. STRUKTURA DRUGIEGO ALGORYTMU BRESENHAMA

Załóżmy, że:

$$\Delta x \geq 2\Delta y \geq 0$$

Wynika z tego, że nachylenie wektora do osi OX nie może być większe niż 30°. To założenie przyjęto dla następujących analiz drugiego algorytmu *Bresenhama*. Sposób rozszerzania zakresu stosowania będzie omówiony w punkcie 9.

Stosownie do sytuacji zobrazowanej na rys. 8, kwadrat zawierający punkt (x_i, y_i) powinien mieć krok diagonalny, natomiast pomiędzy krokiem (x_i, y_i) a krokiem (x_{i+1}, y_{j+1}) jest kilka kroków horyzontalnych, które nazwiemy H_{i+1}:



Rys. 8. Drugi algorytm Bresenhama kreślenia wektora

Fig. 8. Ilustration for evaluation of the consecutive pixels of second Bresenham's algorithm

$$H_{i+1} = X_{i+1} - X_i + 1$$

gdzie: $X_i = \lfloor x_i \rfloor$, $i = \lfloor y_i \rfloor$

Zmienną decyzyjną oznaczamy przez ∇; oznacza się ją z następującej zależności [1]:

Wartość początkowa $\nabla_1 = (2\Delta y|^{\Delta x} + 2(\Delta y|^{\Delta x}) - 2\Delta y$

gdzie: $\Delta y^{\Delta x} = \Delta x - \Delta y(\lfloor \Delta x/\Delta y \rfloor)$

Jeżeli: $2\Delta y|^{\Delta x} = N, \Delta y|^{\Delta x} = R$, to $\nabla_1 = N+2R-2\Delta y$

$\nabla_{i+1} = \nabla_i + 2R$ {dla $\nabla_i < 0$} $\rightarrow H_i = \Delta x/\Delta y - 1$

$\nabla_{i+1} = \nabla_i + 2R - 2\Delta y$ {dla $\nabla_i \geq 0$} $\rightarrow H_i = \Delta x/\Delta y$

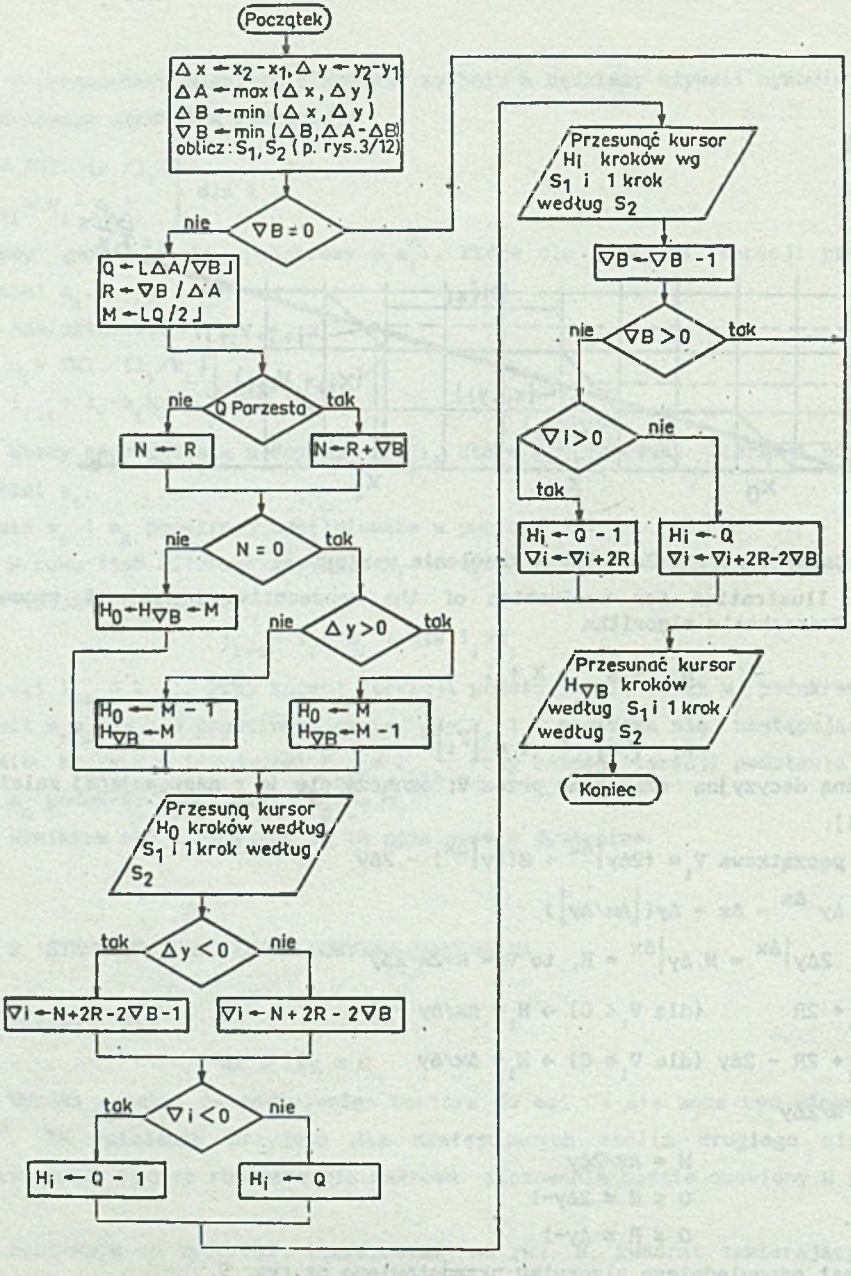
$H_o = M + N/2\Delta y$

gdzie: $M = \Delta x/2\Delta y$

$0 \leq N \leq 2\Delta y-1$

$0 \leq R \leq \Delta y-1$

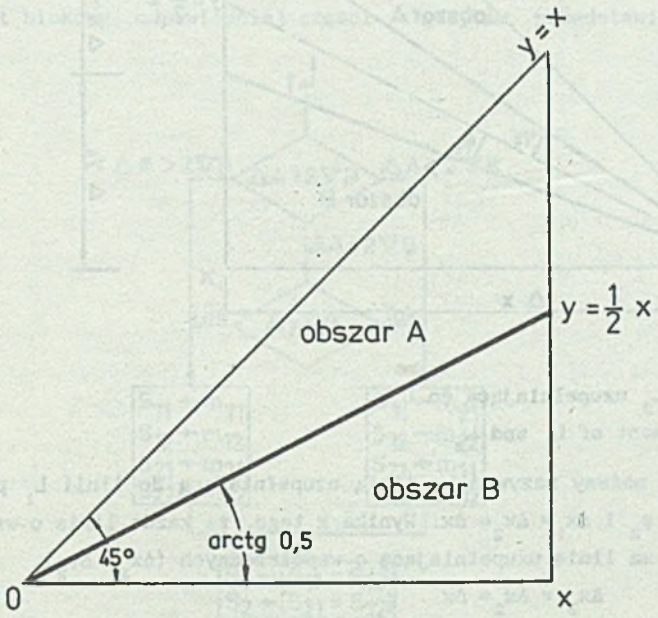
Schemat odpowiedniego algorytmu przedstawiono na rys. 9.



Rys. 9. Schemat blokowy drugiego algorytmu Bresenhama
Fig. 9. The second Bresenham's algorithm

9. OBSZAR STOSOWANIA DDRUGIEGO ALGORYTMU BRESENHAMA

Podzielmy obszar pierwszej ćwiartki układu kartezjańskiego, leżący pomiędzy osią OX a prostą o równaniu y = x, na dwa obszary względem prostej o równaniu y = (1/2)x. Obszar nad tą prostą nazwiemy obszarem A, natomiast obszar leżący poniżej obszarem B, tak jak to ilustruje rys. 10.

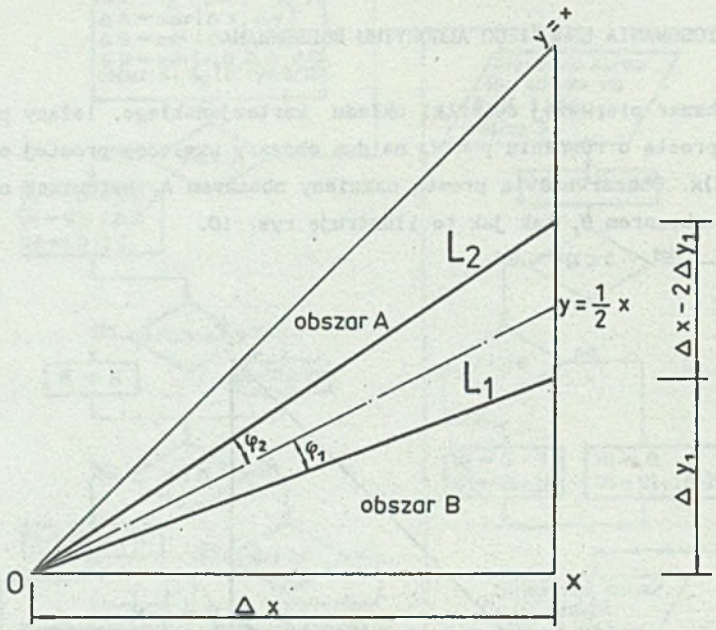


Rys. 10. Podział obszaru kreślenia
Fig. 10. Determination the field of action

Linia prosta L_1 leży w obszarze B, a linia prosta L_2 leży w obszarze A. Jeżeli mamy następujące symbole (rys. 11):

φ_1 - kąt pomiędzy prostą o równaniu y = (1/2)x a prostą L_1

φ_2 - kąt pomiędzy prostą o równaniu y = (1/2)x a prostą L_2



Rys. 11. Linia L_2 uzupełniająca do L_1

Fig. 11. Complement of L_1 and L_2

W tym przypadku możemy nazywać L_2 linią uzupełniającą do linii L_1 pod dwoma warunkami: $\varphi_1 = \varphi_2$ i $\Delta x_1 = \Delta x_2 = \Delta x$. Wynika z tego, że każda linia o współrzędnych $(\Delta x_1, \Delta x_2)$ ma linię uzupełniającą o współrzędnych $(\Delta x_2, \Delta y_2)$.

$$\Delta x_2 = \Delta x_2 = \Delta x$$

$$\Delta y_2 = \Delta x - \Delta y_1 \rightarrow \Delta y_1 = \Delta x - \Delta y_2$$

Drugi algorytm *Bresenhama* może być stosowany tylko dla linii leżących w obszarze B [1]. Stosowanie tego algorytmu dla linii leżących w obszarze A wymaga:

- określenia sekwencji kroków potrzebnych do wyznaczenia linii uzupełniającej $(\Delta x - \Delta y) \rightarrow \Delta y$,
- zmiany kroków diagonalnych na sekwencji kroków potrzebnych do wykreślenia linii $(\Delta x, \Delta y)$.

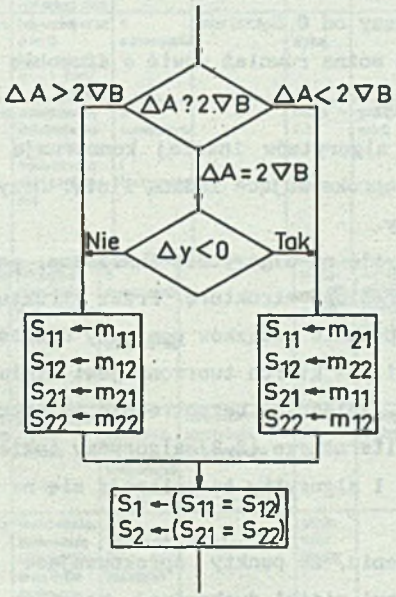
Na przykład: nich rozważana będzie linia prosta o współrzędnych początkowych (0,0) i końcowych (131,16), czyli linia leżąca w obszarze B. Drugi algorytm *Bresenhama* daje wtedy następującą prostą dyskretną:

$$0^4(10^7)^4 10^8 (10^7)^5 10^8 (10^7)^4 10^4$$

Linia uzupełniająca do powyższej będzie prostą o współrzędnych początkowych (0,0) i końcowych (131,115). Uzyskana prosta dyskretna dla linii uzupełniającej wymaga zamiany kroków, czyli:

$$1^4(01^7)^4 01^8 (01^7)^5 01^8 (01^7)^4 01^4$$

Schemat blokowy odpowiedniej części algorytmu przedstawiono na rys. 12.



Rys. 12. Schemat blokowy obliczenia: S_1, S_2

Fig. 12. The scheme for calculation S_1 and S_2

Schemat opracowany przez *Bresenhama* [1]. Używał on dwóch symboli: s i m. Można również użyć w zapisie algorytmu jednego symbolu zamiast dwóch.

Opisane wyżej algorytmy wyczerpują stosowane znane do dzisiaj metody generacji wektorów w siatce rastrowej.

10. PORÓWNANIE

Podstawowe cechy i porównanie omówionych algorytmów przedstawiono w tab.1. Algorytmy te różnią się między sobą pod wieloma względami. Najlepszym sposobem oceny algorytmów byłoby ich analityczne porównanie [7]. Jednak w większości przypadków jest to trudne do przeprowadzenia. Z tego też względu najwłaściwsze będzie porównanie algorytmów pod względem *szybkości* i *dokładności*: przez szybkość rozumieć będziemy liczbę operacji głównych w jednym cyklu generacyjnym oraz maksymalną liczbę tych cykli, co przede wszystkim zależy od operacji podstawowych (głównych). Natomiast dokładność określona jest poprzez: średnią wartość błędu, maksmalną wartość błędu oraz liczbę przypadków, w których błąd może być większy od 0,5.

W odniesieniu do prostych można również mówić o sposobie dyskretyzacji.

10.1. Sposoby dyskretyzacji

Każdy z przedstawionych algorytmów inaczej konstruuje linię dyskretną, czyli inaczej wybiera węzły aproksymujące zadaną linię. Oczywiście, każdy algorytm ma swoje wady i zalety.

Algorytmy, które opierają się na algorytmie *Euklidesa*, generują linie dyskretne, które mają ściśle określoną strukturę. Przez strukturę linii dyskretnej będziemy rozumieli występowanie związków pomiędzy długością i nachyleniem poszczególnych segmentów linii, z których tworzona jest linia dyskretna, oraz kolejnością ich występowania, zależnie, bezpośrednio i jednoznacznie od parametrów prostej zadanej. W literaturze [2,8] algorytmy takie nazwano *strukturalnymi*. Zgodnie z rysunkiem 1 algorytmy te opierają się na twierdzeniu *Freemana*.

Algorytmy oparte na założeniu, że punkty aproksymujące prostą wybrane za pomocą pewnej reguły są węzłami siatki dyskretnej, będziemy nazywać algorytmami *warunkowymi* i zgodnie z rysunkiem 7, algorytmy te opierają się na algorytmach *Bresenhama*.

Pozostałe algorytmy nazywać będziemy algorytmami *warunkowo-strukturalnymi*, np. algorytm *Pittewaya* i *Greena*.

Tabela 1

PODSTAWOWE CECHY I PORÓWNANIE METODY GENERACJI WEKTORÓW

METODA GENERACJI	SPOSÓB DYSKRETYZACJI	SZYBKOŚĆ GENERACJI (złożoność obliczeń)			DOKŁADNOŚĆ GENERACJI			WADY METODY	ZALETY METODY
		używane operacje podstawowe	liczba operacji głównych w cyklu generacji	maksymalna liczba iteracji	średnia wartość błędu	liczba błędów większych niż 0.5	maksymalna wartość błędu		
wg Stoctona (1963r)	warunkowa	sumowanie, testowanie znaków (dla liczb) obliczanie modułów liczb	6 sumowań	max (Δx,Δy)	około max błąd/2	0	0.5	duża liczba cyklu generacji	dokładna i prosta
wg Bresenhama (1963r)	warunkowa	sumowanie, testowanie znaków, obliczanie modułów i mnożenie przez dwa	5 sumowań	max (Δx,Δy)	jak wyżej	0	0.5	jak wyżej	jak wyżej oraz szybsza niż wyżej o 50%
wg Earnshawa (1980r)	warunkowa	jak wyżej oraz $c \leftarrow 0$ lub gdy 0.1 $c \leftarrow 1$ kody ruchu	4 sumowania	max (Δx,Δy)	jak wyżej	0	0.5	jak wyżej	jak wyżej oraz daje jako wynik łuk cyfrowy
wg Bronsa (1972r)	strukturalna	sumowanie, dzielenie, testowanie znaków, mnożenie i porównanie	3 sumowania i testowanie	3	około 0.5	[max(Δx,Δy)/Z] ≤ 0.5	około 1	bardzo duża niedokładność (wg parametrów podanych obok) brak przypadku n=c/m	szybka generacja
wg Arcelego Massarottiego (1978r)	strukturalna	jak wyżej oraz mnożenie przez -1	3 sumowania 4 testowania	3	mniejsza niż wyżej	mniejsza niż wyżej, maksymalnie M.L.L.*	$\left[\frac{M.L.L.*}{\max(\Delta x,\Delta y)}\right]^{\cdot} \cdot \frac{\max(\Delta x,\Delta y)}{M.L.L.*} \leq$	duża niedokładność oraz brak przypadku n=c/m	jak wyżej oraz dokładniejsza niż wyżej
wg Pitteway Greena (1982r)	warunkowo strukturalna	jak dla algorytmu Bronsa	6 sumowań 3 testowania	mniejsza niż w algorytmie Bresenhama	mniejsza niż wyżej	mniejsza niż wyżej	$1 - \frac{M.L.L.*}{\max(\Delta x,\Delta y)}$	dokładność gorsza niż w algorytmie Bresenhama	dodany przypadek n=c/m a dokładniejsza niż wyżej
wg Bresenhama (1985r)	warunkowa	sumowanie, testowanie i dzielenie oraz dwie nowe operacje	8 sumowań 8 testowań	M.L.L.*	około max. błąd/2	0	0.5	stosunkowo duża złożoność dla krótkich wektorów o kącie nachyl. $30° \pm 90°$	jest najszybszą metodą warunkową, ponadto dokładna
wg Pitteway Castle'a (1985r)	strukturalna	sumowanie, testowanie, przemieszczanie ciągu binarnego i przemieszczenie wraz z odwróceniem ciągu binarnego	2 sumowania 1 testowanie	proporcjonalna do $min\left(\frac{\Delta y}{\Delta x}, \frac{\Delta x - \Delta y}{\Delta x}\right)$	jak wyżej	0	0.5	stosunkowo duża złożoność dla długich wektorów o kącie nachyl. $0° \pm 90°$ mb $45° \pm 90°$	dokładna, z najmniejszą liczbą operacji głównych w cyklu generacji
wg zmodyfikowanej wyżej (1990r)	strukturalna	jak dla algorytmu Bronsa oraz jak wyżej	jak wyżej	mniejsza niż wyżej o części całkowitej	jak wyżej	0	0.5	-	lepsza niż metoda wyżej wymieniona

M.L.L. oznacza min(Δx,Δy,Δx−Δy)*

10.2. Dokładność generacji

Ocena dokładności generacji powinna uwzględniać następujące wielkości:

- maksymalna wartość błędu N_{max} (maksymalne odchylenie pomiędzy generowaną a zadaną prostą [5],
- wartość średnia błędu $\mathcal{S}$, czyli suma wartości średniej ($\Delta x+1$) zmiennych błędów wyrażona poniższym wzorem:

$$\mathcal{S} = (\sum_{i=o}^{\Delta x} N_i)/(\Delta x + 1)$$

- liczba błędów większych niż 0.5 rastra $\mathcal{X}$.

W tabeli 2 pokazano wartości tych wielkości dla linii o równaniu y = (14/39)x.

Mówiąc o dokładności generacji wektorów, celowe jest wyróżnienie czterech klas uporządkowanych względem dokładności:

Klasa 1: Algorytmy klasy 1 są algorytmami dokładnymi, to znaczy takimi, w których maksymalny błąd N_{max} jest mniejszy od 0.5 rastra, co możemy zapisać: $N_{max} \leq 0.5$, czyli liczba błędów większych niż 0.5 rastra równa się zero. Algorytmy zaliczane do tej klasy mają najmniejsze średnie wartości błędu wyznaczone za pomocą poniższego wzoru.

$$\mathcal{S} = (\sum_{i=o}^{\Delta x} N_i)/(\Delta x + 1) = N_{max}/2$$

gdzie: $\Delta x \geq \Delta y \geq 0$

Do klasy tej należą algorytmy: *Stocktona, Bresenhama* i *Pittewaya-Castle'a.*

Klasa 2: Zawiera algorytmy, dla których:

$$N_{max} = 1 - [\min(\Delta x, \Delta y, \Delta x - \Delta y)/\max(\Delta x, \Delta y)]$$

Maksymalny błąd jest większy niż w algorytmach klasy 1. Liczba błędów większych niż 0.5 może być większa niż zero, a wartość średnia błędu spełnia nierówność:

$$\mathcal{S}_{klasa2} \geq \mathcal{S}_{klasa1}$$

Do tej klasy należą algorytmy *Pittewaya-Greena* i modyfikacja tego algorytmu dokonana przez *Pittewaya* w roku 1985.

Tabela 2

PRZYKŁAD OBLICZANIA WARTOŚCI BŁĘDU APROKSYMACJI PRZYBLIŻENIA WEKTORA O RÓWNANIU

$$y = \frac{14}{39} * x$$

RÓŻNYMI METODAMI GENERACJI WEKTORÓW

METODA GENERACJI	WARTOŚĆ BŁĘDU APROKSYMACJI KOLEJNYCH PUNKTÓW WEKTORA ZADANEGO WYŻEJ						ŚREDNIA WARTOŚĆ BŁĘDU	MAKSYMALNA WARTOŚĆ BŁĘDU	LICZBA BŁĘDÓW > 0.5	MINIMALNA WARTOŚĆ BŁĘDU	UWAGI
	x=0+5	x=6+11	x=12+18	x=19+25	x=26+ 32	33+39					
wg Stocktona, Bresenhama i Pittewaya Castle'a	0.00	0.15	0.31	0.18	0.33	0.15	0.2437	0.4871795	0	0.025641	Najlepsza dokładność. Ciąg błędów od 0 do $\Delta x/2$ powtarzany od Δx do $\Delta x/2$.
	0.36	0.49	0.31	0.18	0.31	0.21					
	0.28	0.13	0.33	0.18	0.05	0.44					
	0.08	0.23	0.38	0.1	0.41	0.08					
	0.44	0.41	0.26	0.26	0.22	0.28					
	0.21	0.05	0.1	0.38	0.13	0.36					
			0.46	0.02564	0.487179	0.00					
wg Pittewaya i Greena	0.00	0.15	0.31	0.18	0.33	0.15	0.2597	0.615385	5	0.025641	Gorsza dokładność niż wyżej. Błędy większe niż 0.5 nie są wspólne z wyższymi metodami.
	0.36	0.48717	0.33	0.18	0.31	0.21					
	0.28	0.13	0.02564	0.46	0.05	0.44					
	0.08	0.23	0.61538	0.1	0.59	0.08					
	0.56	0.41	0.26	0.26	0.22	0.28					
	0.21	0.05	0.1	0.38	0.13	0.36					
			0.54	0.025641	0.51	0.00					
wg Arcellego Massarottiego	0.00	0.15	0.31	0.18	0.33	0.15	0.2847	0.69230076	8	0.025641	Lepsza dokładność niż niżej. Poprawiła kilka błędów większych niż 0.5.
	0.36	0.51	0.67	0.18	0.692307	0.21					
	0.28	0.13	0.02564	0.54	0.05	0.56					
	0.08	0.23	0.38	0.1	0.41	0.08					
	0.44	0.59	0.26	0.26	0.22	0.28					
	0.21	0.05	0.1	0.62	0.13	0.64					
			0.46	0.025641	0.487179	0.00					
wg Bronsa	0.00	0.15	0.31	0.82	0.33	0.15	0.4752	0.9743589	19	0.025641	Najgorsza dokładność. Każdy błąd z (0) do ($\Delta x/2$) ma uzupełniający do 1 z (Δx) do ($\Delta x/2$).
	0.36	0.51	0.67	0.18	0.692307	0.21					
	0.72	0.87	0.02564	0.54	0.05	0.56					
	0.08	0.23	0.38	0.9	0.41	0.92					
	0.44	0.59	0.74	0.26	0.78	0.28					
	0.79	0.95	0.1	0.62	0.13	0.64					
			0.4	0.974389	0.487179	0.00					

Klasa 3: Zawiera algorytmy, dla których błąd:

$$N_{max} \leq \left\lfloor \frac{\max(\Delta x, \Delta y)}{\min(\Delta x, \Delta y, \Delta x - \Delta y)} \right\rfloor \cdot \frac{\min(\Delta x, \Delta y, \Delta x - \Delta y)}{\max(\Delta x, \Delta y)} \leq 1$$

Algorytm *Arcelliego-Massarottiego* należy do tej klasy.

Liczba błędów większych niż 0.5 rastra jest większa niż w algorytmach należących ddo klasy 4.

Klasa 4: Zawiera algorytmy o najmniejszej dokładności, w których błąd:

$$\mathcal{E}_{max} \leq 1$$

$$\mathcal{X} \cong \{\max(\Delta x, \Delta y) - 1\}/2$$

$$\mathcal{E} = \mathcal{E}_{max}/2 \qquad \text{(patrz tab. 1)}$$

Algorytm *Bronsa* należy do tej klasy.

Dokładność generacji wektorów podzieliliśmy zatem na 4 klasy. Dla przykładu w tabeli 3 przedstawiono, jak wektor o równaniu y = (14/39)x może zostać przybliżony przez algorytmy należące do różnych klas.

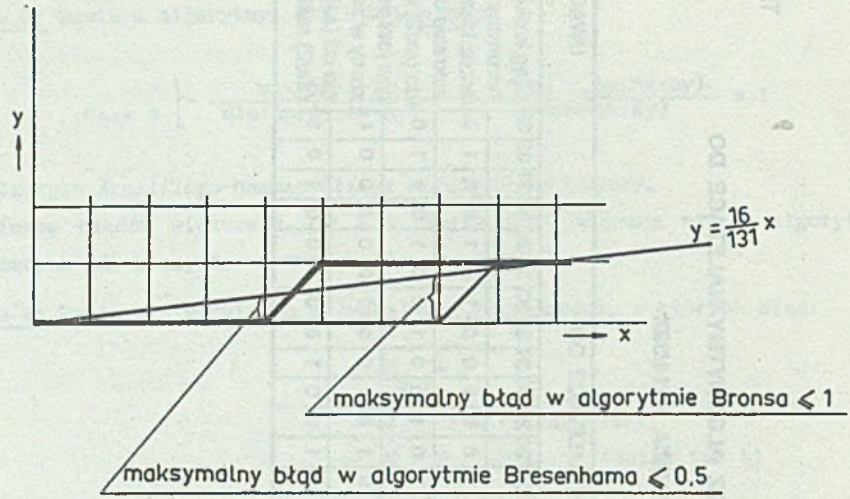
10.3. Szybkość generacji (złożoność obliczeniowa)

Algorytm *Stoctona* wykorzystuje następujące podstawowe operacje: sumowanie, testowanie znaków i liczb i obliczanie modułów liczb. Są to operacje z punktu widzenia realizacji układowej niezwykle proste i łatwo realizowalne technicznie. Jednak liczba niezbędnych do wykonania operacji w trakcie wyznaczania punktów ciągu jest duża (tab. 1), stąd można powiedzieć, że szbkość generacji nie jest najwyższa [7]. Pierwsza metoda *Bresenhama* jest lepsza niż metoda *Stocktona*. Na przykład: liczba operacji głównych w jednym cyklu generacyjnym w algorytmie *Bresenhama* wynosi 5 sumowań, natomiast w algorytmie *Stocktona* wynosi 6 sumowań (tab. 1). Dla obu algorytmów liczba cykli generacji jest duża i w niektórych przypadkach wynosi max (Δx, Δy), natomiast w drugim algorytmie *Bresenhama* liczba ta jest o wiele mniejsza, gdyż równa się min (Δx, Δy, Δx-Δy), zatem drugi algorytm *Bresenhama* jest tym szybszy, im mniejszy jest kąt nachylenia prostej:

Tabela 3

PRZYKŁAD LINII DYSKRETNYCH WYZNACZONYCH PRZEZ ALGORYTMY NALEŻĄCE DO RÓŻNYCH KLAS DOKŁADNOŚCI Z RÓWNANIA Y=14/39X

KLASA	CIĄGI KROKÓW WYZNACZONYCH PRZEZ ALGORYTMY NALEŻĄCE DO RÓŻNYCH KLAS DOKŁADNOŚCI																																										UWAGI
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39				
KLASA 1	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0	1	0				Ciąg kroków generowany przez klasę I w zakresie od (0) do (Δx/2) rastrów jest powtórzony w zakresie od (Δx) do (Δx/2) rastrów.
KLASA 2	0	1	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0	1	0				
KLASA 3	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0	1				
KLASA 4	0	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0	1	0	1				



Rys. 13. Przykład przybliżenia wektora o równaniu $y = \frac{16}{131} x$ algorytmu Bresenhama i Bronsa w zakresie od (0) do (9) rastrów

Fig. 13. The approximation of the vector "$y = \frac{16}{131} x$" by Bresenham's and Brons algorithm for integration range 0-9

$$\mathcal{F} = \text{arctg}\,[\min(\Delta x/\Delta y\ ,\ (\Delta x-\Delta y)/\Delta x)]$$

gdzie: ($\mathcal{F}$) to kąt nachylenia prostej.

Wynika z tego, że prosta dyskretna ma najdłuższe podokresy 0^n1, gdy kąt nachylenia prostej $\mathcal{F}$ ma najmniejszą wartość: $n \sim 1/\mathcal{F} \rightarrow n \sim \max(\Delta x/\Delta y\ ,\ \Delta x/\Delta x-\Delta y)$, gdzie: (n) oznacza powtarzanie kroków poosiowych (0) lub diagonalnych (1) n razy.

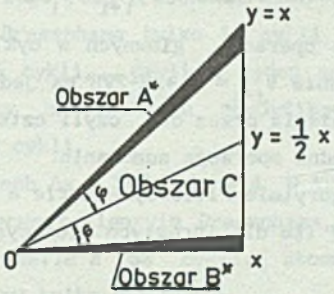
Algorytmy *Bronsa* i *Arcellego-Massarottiego* są najlepsze ze względu na szybkość generacji - w niektórych przypadkach algorytmy te mają 3 iteracje - (tab. 1).

Szybkość generacji w algorytmie *Pittewaya-Castle'a* nie jest lepsza niż w algorytmie *Bronsa*, a w niektórych przypadkach liczba cykli generacji jest proporcjonalna do kątów nachylenia prostych (dla linii leżących w obszarze B na rys. 10). Natomiast w drugim algorytmie *Bresenhama*, jak wykazaliśmy, obo-

wiązuje zależność przeciwna. Podobnie w algorytmie *Pittewaya-Castle'a* mamy:

$$n \sim \mathcal{F} \rightarrow n \sim \min(\Delta x/\Delta y\ ,\ (\Delta x-\Delta y)/\Delta x)$$

Wynika z tego, że istnieje obszar C (rys. 14), w którym algorytm *Pittewaya-Castle'a* jest lepszy niż drugi algorytm *Bresenhama*. Są natomiast dwa obszary A^* i B^*, w których drugi algorytm *Bresenhama* jest lepszy niż algorytm *Pittewaya-Castle'a*. Problem stanowi określenie obszaru C, czyli wyznaczenie granicy obszaru C. Z przeprowadzonych rozważań możemy wyciągnąć dwa wnioski:



Rys. 14. Obszar C, w którym algorytm Pittewaya Castle'a jest najlepszy
Fig. 14. Ilustration of the best field "C" of Pitteway-Castle's algorithm

1. Najszybszą generację prostych otrzymuje się przez stosowanie algorytmu *Bronsa*, zaś najwolniejszą - przez stosowanie algorytmu *Stocktona*. Algorytmy *Pittewaya-Castle'a* i *Bresenhama*, które opracowano w roku 1985, mają dobrą szybkość w zależności od zakresu zastosowania.
2. Algorytmy *Bresenhama*, *Pittewaya-Castle'a* i *Stocktona* mają najlepszą dokładność generacji.

11. PODSUMOWANIE

W celu zwiększenia szybkości generacji prostych przez drugi algorytm *Bresenhama* i przez algorytm *Pittewaya-Castle'a*, można przyjąć następujące założenia (*modyfikacje*):

- W algorytmie *Bresenhama* (rys. 9) mieliśmy zmienną decyzjną dla wyboru liczby kroków w każdej iteracji algorytmu, pod symbolem ∇_i, która ma wartość początkową ∇_i i w każdej iteracji nową wartość ∇_{i+1}, czyli:

$$\nabla_i = N+2R-2VB \quad \text{lub} \quad \nabla_i = N+2R-2VB-1$$
$$\nabla_{i+1} = \nabla_i +2R \quad \text{lub} \quad \nabla_{i+1} = \nabla_i +2R-2VB$$

Możemy obliczyć wartość 2R-2VB na początku, co będzie w wielu przypadkach szybsze. Niech: $L = 2R-2VB$ $C = 2R$.

W tym przypadku będzie: $\nabla_i = N+L$ lub $\nabla_i = N+L-1$

$$\nabla_{i+1} = \nabla_i +C \quad \text{lub} \quad \nabla_{i+1} = \nabla_i +L$$

Wynika z tego, że liczba operacji głównych w cyklu generacyjnym będzie mniejsza. Na przykład: równanie $\nabla_{i+1} = \nabla_i +2R-2VB$ ma jedną operację sumowania, jedną odejmowania i dwie mnożenia przez dwa, czyli cztery. Natomiast w równaniu $\nabla_{i+1} = \nabla_i +L$ jest tylko jedna operacja sumowania.

- Szybkość generacji $\mathcal{G}$ w algorytmie *Pittewaya-Castle'a* jest proporcjonalna do kątów nachylenia prostych $\mathcal{F}$ (to dla prostych leżących w obszarze B, rozdz. 10), czyli:

$$\mathcal{G} \sim \min(\Delta y/\Delta x\ ,\ (\Delta x-\Delta y)/\Delta x)$$

Jeżeli: $\Delta x = q,\ \Delta y = p,\ \Delta x-\Delta y = r$ to:

$$\mathcal{G} \sim \min(p/q,\ r/q)$$

Dla prostych leżących w obszarze B mamy: $\mathcal{G} \sim p/r$. Natomiast dla prostych leżących w obszarze A mamy: $\mathcal{G} \sim r/p$. W algorytmie można pominąć (m-1) cykli generacji, kiedy $m = E(r/p)$ lub $m = E(p/r)$ zależy od prostej, czy leży w obszarze B, czy w A. W tej modyfikacji można rozpocząć obliczenia dla podokresów prostej dyskretnej od następujących:

$$\left.\begin{aligned} m_1 &= 0^k 1\ 0^{n+1} \\ m_2 &= 0^k 1\ 0^n \end{aligned}\right\} \text{dla } r > p$$

$$\left.\begin{aligned} m_1 &= 1^k 0\ 1^{n+1} \\ m_2 &= 1^k 0\ 1^n \end{aligned}\right\} \text{dla } r < p$$

W przypadku r = p algorytm będzie jak bez modyfikacji, dlatego efektywność modyfikacji jest lepsza dla prostych mających największy kąt z prostą y = x/2.

W przypadku r = 0 po pierwszej iteracji ma to znaczenie takie, że algorytm *Pittewaya-Castle'a* miał już ostatnią iteracją, jaka była poprzednio, czyli:

$m_1 = 0^k 1\ 0^n$ } dla r = 0 (gdy na początku r > p)

$m_2 = 1^k 0\ 1^n$ } dla p = 0 (gdy na początku r < p)

Schemat odpowiedniego zmodyfikowanego algorytmu *Pittewaya-Castle'a* przedstawiono na rys. 15.

Na przykład: jeżeli wektor ma następujące przyrosty

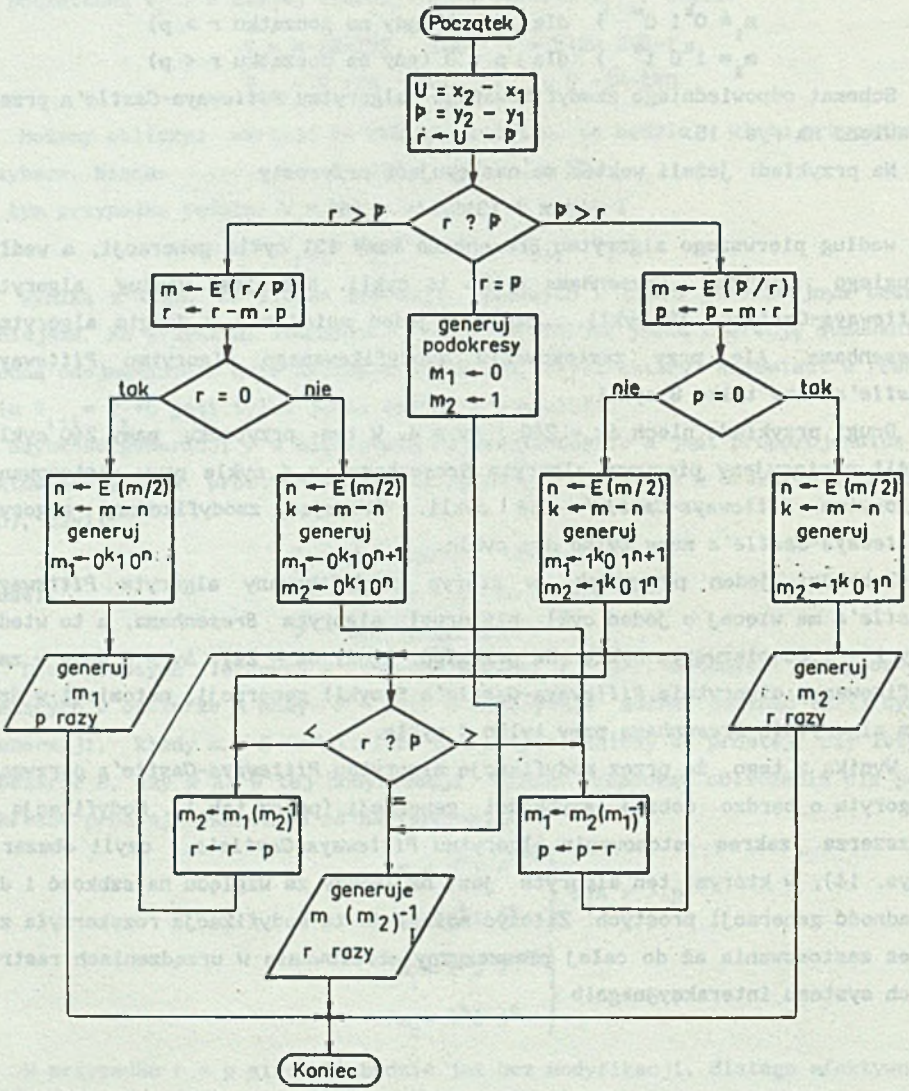
$$\Delta x = 131\ ,\ \Delta y = 16$$

to według pierwszego algorytmu *Bresenhama* mamy 131 cykli generacji, a według drugiego agorytmu *Bresenhama* tylko 16 cykli. Natomiast według algorytmu *Pittewaya-Castle'a* 15 cykli, czyli o jeden mniej niż w drugim algorytmie *Bresenhama*. Ale przy zastosowaniu zmodyfikowanego algorytmu *Pittewaya-Castle'a* mamy tylko 8 cykli.

Drugi przykład: niech $\Delta x = 240$, $\Delta y = 4$. W tym przypadku mamy 240 cykli, jeśli zastosujemy pierwszy algorytm *Bresenhama*, a 4 cykle przy zastosowaniu algorytmu *Pittewaya-Castle'a* 58 cykli. Stosując zmodyfikowany algorytm *Pittewaya-Castle'a* mamy tylko dwa cykle.

Zachodzi jeden przypadek, w którym zmodyfikowany algorytm *Pittewaya-Castle'a* ma więcej o jeden cykl niż drugi algorytm *Bresenhama*, a to wtedy, gdy r = 1 po pierwszym cyklu. Na przykład, kiedy $\Delta x = 241$, $\Delta y = 4$ mamy w zmodyfikowanym algorytmie *Pittewaya-Castle'a* 5 cykli generacji, natomiast w drugim algorytmie *Bresenhama* mamy tylko 4 cykle.

Wynika z tego, że przez modyfikację algorytmu *Pittewaya-Castle'a* otrzymamy algorytm o bardzo dobrej szybkości generacji (patrz tab.1). Modyfikacja ta rozszerza zakres stosowania algorytmu *Pittewaya-Castle'a*, czyli obszar C (rys. 14), w którym ten algorytm jest najlepszy ze względu na szbkość i dokładność generacji prostych. Założyć można, że ta modyfikacja rozszerzyła zakres zastosowania aż do całej płaszczyzny obrazowania w urządzeniach rastrowych systemu interakcyjnego.



Rys. 15. Schemat blokowy zmodyfikowanego algorytmu Pittewaya-Castle'a
Fig. 15. The suggested modification of Pitteway-Castle's algorithm

LITERATURA

1. Bresenham J.E.: Run length slice algorithm for incremental lines Fund. Algor. for Comp.grap. Springer-Verlag, 1985, s.59-104.
2. Brons R.: Theoretical and linguistc methods for describing straight lines. Fund.Algor. for Comp.Grap. Springer-Verlag, 1985, s.21-40.
3. Castle C.M.A., Pitteway M.L.V.: An application of Euclid's algorithm to drawing straight lines. Fund.Algor. for Comp.Grap. Springer-Varlag, 1985, s.135-137.
4. Creutzburg E., Hubler A., Sykora O.: Geometric methods for on-line re-recognition of digital straight line segements. Comp. and Artificial Intelligence, vol.7, no.3, Bratislava 1988, s.270-274.
5. Hearn D., Backer M.P.: Computers graphics. P.H.I.1986, s.58-61.
6. Kleiber M., Szuniewicz R.: komputer osobisty typu IBM PC, Warszawa 1988, s.70-71.
7. Mokrzycki W.: Analiza cyfrowa metod generacji układowej krzywych drugiego stopnia, Warszawa 1978, s.24-32.
8. Mokrzycki W.: Całkowitoliczbowe dyskretyzacje krzywych algebraicznych w komputerowym obrazowaniu informacji, Warszawa 1985, s.13-15.
9. Newman W.M., Sproull R.F.: Principles of interactive computer graphics. Mc Graw Hill, 1979, s.21-23.
10. Pitteway M.L.V., Green J.R.: Bresenham's algorithm with run line coding shortcut. Computer journal, vol.25, no.1, 1982, s.114-115.
11. Pitteway M.L.V.: The relationship between Euclid's algorithms and run-length encoding. Fund.Agor. for Comp. Grap. Springer-Verlag, 1985, s.105-112.
12. Sproull R.F.: Using program transformations to derive line-drawing algorithms, ACM Trans. Graph.1, 1982, s.259-273.
13. Thmpson J.R.: Straight lines and graph plotters. Comp.Journal,4, 1964, s.227.
14. Węgrzyn S.: Podstawy informatyki, Warszawa 1982, s.19-20.

Recenzent: doc.dr inż. Wojciech Świder

Wpłynęło do Redakcji: 15.03.1990 r.

ALGORITHMS OF VECTORS GENERATION FOR INTERACTIVE RASTER GRAPHICS SYSTEMS

Abstract

The purpose of this paper is to present, analyse and make a comparision between algorithms of vectors generation based on Euclid's and Bresenham's algorithms from the time and accuracy point of view. In this regard we select the fastest and the most exact algorithm with taking the application scope into considenation. This paper suggests also a modification of Pitteway-Castle's algorithm which is based on Euclid's algorithm.

The suggested modification relies on omit (m) loops from Pitteway-Castle's algorithm according to the vector's slope and the angle (φ) between vectors represention and basic vector "y = x/2" (look to figurs 10,11).

(m) can be calculated from integer starting point (X_s, Y_s) and integer ending point (X_t, Y_t), i.e.

$$m = \frac{\text{maximum } \{\Delta Y,\ \Delta X-\Delta Y\}}{\text{minimum } \{\Delta Y,\ \Delta X-\Delta Y\}}$$

where,

$$\Delta X = X_t - X_s \quad , \quad \Delta Y = Y_t - Y_s$$

It's noticed from the above equation that the value of (m) and consequentyly the speed of generation is directly proportional with the angle (φ).

The suggested algorithm contains three outputs (fig.15): first one for vectors " y > x/2 ", second for vectors " y < x/2 " and third for vectors " y = x/2 " (Pitteway-Castle's algorithm without modification).