Základy počítačové grafiky

Rasterizace objektů ve 2D

Michal Španěl
Tomáš Milet

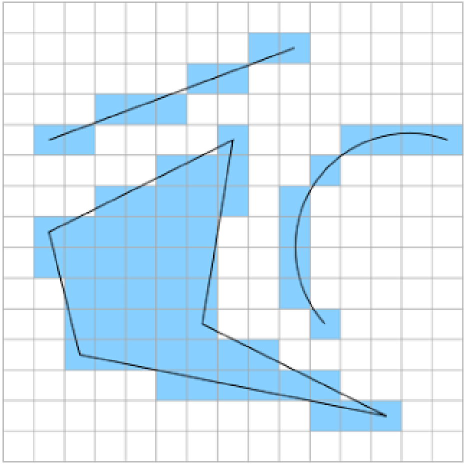
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
Ústav počítačové grafiky a multimédií

Brno 2022

Cíl přednášky

Seznámit se s hlavními algoritmy pro převod základních vektorových entit na rastrové zobrazení.

Prozatím se budeme zabývat úsečkou, kružnicí a elipsou.

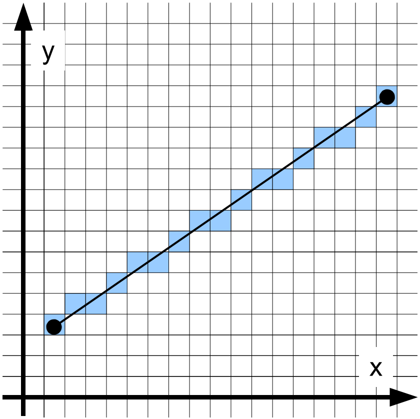


Obsah

Rasterizace

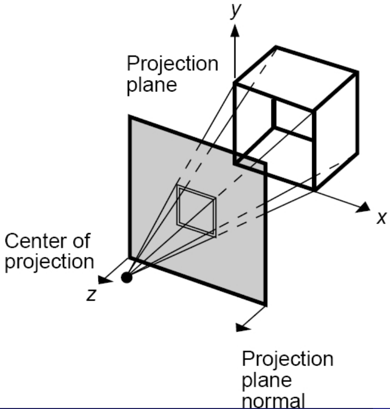
Definice

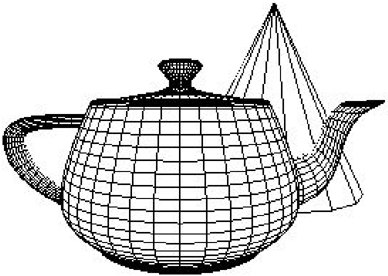
Proces převodu vektorové reprezentace dat na jejich rastrovou formu s cílem dosáhnout maximální možnou kvalitu a zároveň rychlost výsledného zobrazení.



Rasterizace, pokr.

- Při zobrazování je rasterizace velmi často opakovaná operace!
- Realizována v HW grafické karty.





Příklad - Voxelová grafika

- Jak generovat šikmé a ”kulaté”objekty?

Obsah

Úsečka

Definice

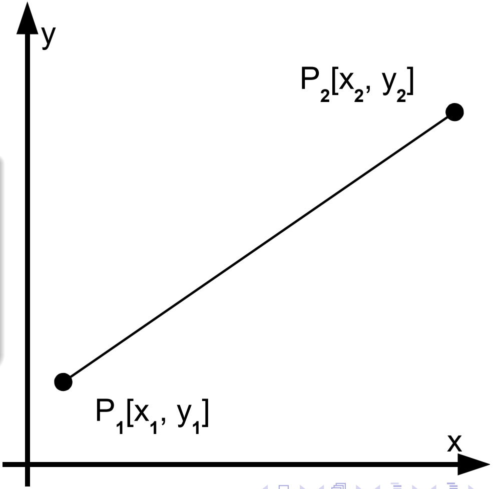
Úsečka je základní geometrická vektorová entita definovaná:

Úsečka

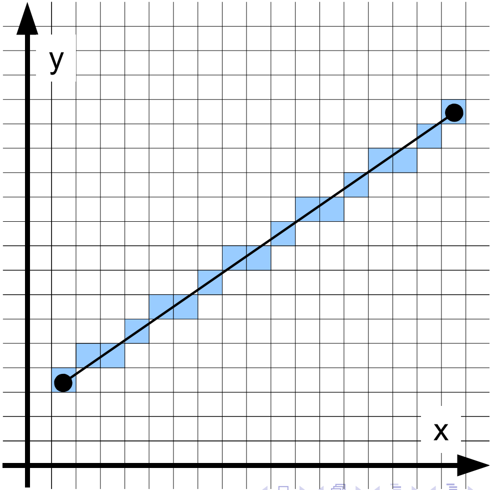
Definice

Úsečka je základní geometrická vektorová entita definovaná:

- souřadnicemi dvou koncových bodů
- rovnicí přímky popisující geometrii



Jak na rasterizaci úsečky?

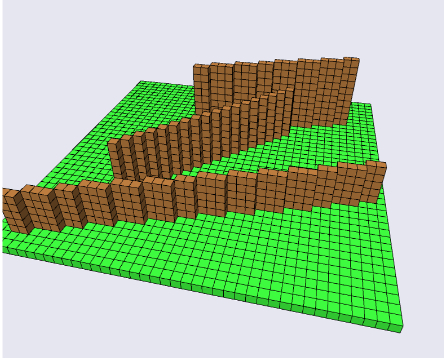


Příklad - Voxelová šikmá zeď v Processing

Nástroj Processing (https://processing.org/)

- Jednoduchý programovací jazyk a prostředí (open source, Java)
- 2D a 3D grafika, interaktivní aplikace, animace,. . .
- Rychlé prototypování řešení

- Jak procedurálně generovat šikmou zeď?



Jak lze definovat úsečku?

Jak lze definovat úsečku?

Obecná rovnice úsečky

$$Ax + By + C = 0, \quad A = (y_1 - y_2), \quad B = (x_2 - x_1)$$

Jak lze definovat úsečku?

Obecná rovnice úsečky

$$Ax + By + C = 0, \quad A = (y_1 - y_2), \quad B = (x_2 - x_1)$$

Parametrické vyjádření

$$x = x_1 + t\,(x2 - x1)\,, \quad y = y_1 + t\,(y2 - y1)\,, \quad t \in < 0, 1 >$$

Jak lze definovat úsečku?

Obecná rovnice úsečky

$$Ax + By + C = 0, \quad A = (y_1 - y_2), \quad B = (x_2 - x_1)$$

Parametrické vyjádření

$$x = x_1 + t\,(x2 - x1)\,, \quad y = y_1 + t\,(y2 - y1)\,, \quad t \in < 0, 1 >$$

Směrnicový tvar

$$y = kx + q, \quad k = \frac{\Delta y}{\Delta x} = \frac{(y_2 - y_1)}{(x_2 - x_1)}$$

DDA algoritmus

Popis

- DDA - Digital Differential Analyser
- Jeden z prvních algoritmů rasterizace úsečky.
- Používá floating-point aritmetiku!

DDA algoritmus

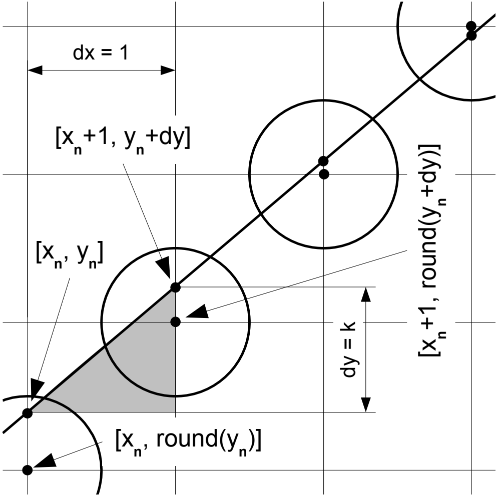
Popis

- DDA - Digital Differential Analyser
- Jeden z prvních algoritmů rasterizace úsečky.
- Používá floating-point aritmetiku!

Princip

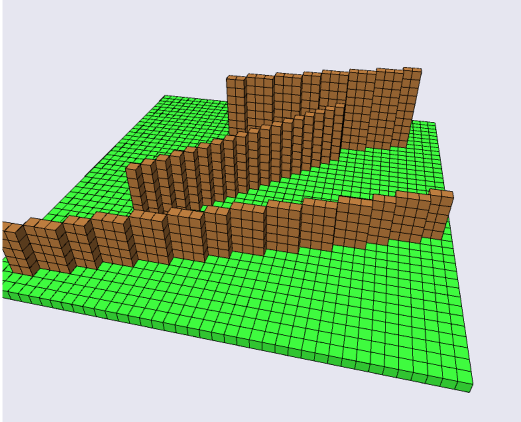
- Vykreslujeme po pixelu od bodu P_1 k bodu P_2.
- V ose X postupujeme s přírůstkem $dx = 1$.
- V ose Y je přírůstek dán velikostí směrnice úsečky.
- Souřadnice Y se zaokrouhluje na nejbližší celé číslo.

DDA algoritmus




```
LineDDA(int x1, int y1, int x2, int y2)
{
        double k = (y2-y1) / (x2-x1);
        double y = y1;

        for (int x = x1; x <= x2; x++)
        {
                draw_pixel( x, round(y));
                y += k;
        }
}
```

Ukázka - Voxelová zeď pomocí DDA algoritmu v Processing



Bresenhamův (Midpoint) algoritmus

Popis

- Nejčastěji používaný algoritmus rasterizace úsečky.
- Používá celočíselnou aritmetiku, sčítání, porovnání.
- Efektivnější a snadnější implementace do HW.

Bresenhamův (Midpoint) algoritmus

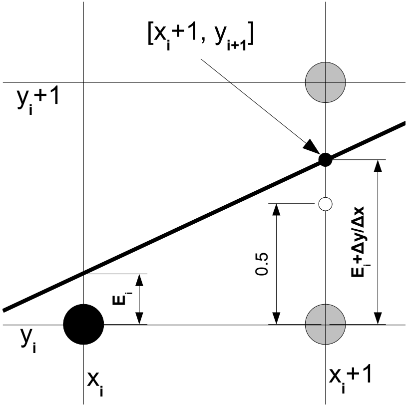
Popis

- Nejčastěji používaný algoritmus rasterizace úsečky.
- Používá celočíselnou aritmetiku, sčítání, porovnání.
- Efektivnější a snadnější implementace do HW.

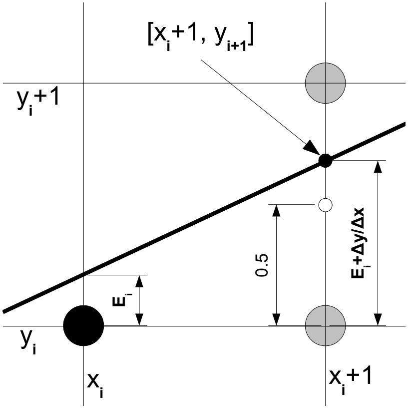
Princip

- Vykreslujeme po pixelu od bodu P_1 k bodu P_2.
- V ose X postupujeme s přírůstkem $dx = 1$.
- O pusunu v ose Y rozhodujeme podle znaménka tzv. prediktoru.

Bresenhamův algoritmus



Bresenhamův algoritmus



Rozhodování a výpočet chyby vykreslování E

$$E_i + \frac{\Delta y}{\Delta x} \begin{cases} < 0.5 & (x_i + 1, y_i) \qquad E_{i+1} = E_i + \frac{\Delta y}{\Delta x} \\ \geq 0.5 & (x_i + 1, y_i + 1) \quad E_{i+1} = E_i + \frac{\Delta y}{\Delta x} - 1 \end{cases}$$

Bresenhamův algoritmus, pokr.

Převod porovnání s 0.5 na test znaménka

- Nerovnice násobíme $2\Delta x$

$$2\Delta x E_i + 2\Delta y - \Delta x \begin{cases} < 0 & E_{i+1} = E_i + 2\Delta y \\ \geq 0 & E_{i+1} = E_i + 2\Delta y - 2\Delta x \end{cases}$$

- Rozhodovací člen nazveme prediktorem P_i

$$P_i = 2\Delta x E_i + 2\Delta y - \Delta x \begin{cases} < 0 & P_{i+1} = P_i + 2\Delta y \\ \geq 0 & P_{i+1} = P_i + 2\Delta y - 2\Delta x \end{cases}$$

Bresenhamův algoritmus, pokr.

Převod porovnání s 0.5 na test znaménka

- Nerovnice násobíme $2\Delta x$

$$2\Delta x E_i + 2\Delta y - \Delta x \begin{cases} < 0 & E_{i+1} = E_i + 2\Delta y \\ \geq 0 & E_{i+1} = E_i + 2\Delta y - 2\Delta x \end{cases}$$

- Rozhodovací člen nazveme prediktorem P_i

$$P_i = 2\Delta x E_i + 2\Delta y - \Delta x \begin{cases} < 0 & P_{i+1} = P_i + 2\Delta y \\ \geq 0 & P_{i+1} = P_i + 2\Delta y - 2\Delta x \end{cases}$$

Počáteční hodnota predikce ($E_0 = 0$)

$$P_0 = 2\Delta y - \Delta x$$

Zjednodušená implementace

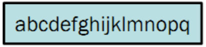
```
LineBres(int x1, int y1, int x2, int y2)
{
        int   dx = x2-x1, dy = y2-y1;
        int   P = 2*dy – dx;
        int   P1 = 2*dy, P2 = P1 - 2*dx;
        int   y = y1;

        for (int x = x1; x <= x2; x++)
        {
                draw_pixel( x, y);
                if (P >= 0)
                        { P += P2; y++; }
                else
                         P += P1;
        }
}
```

DDA s fixed-point aritmetikou

Floating-point aritmetika

- S . . . znaménko
- M . . . mantisa
- E . . . exponent

$X = S \cdot M \cdot 2^{E}$



=0,abcdefghijklmnopq

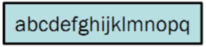
=a*1/2 + b*1/4 + c*1/8 + d*1/16 + ...

DDA s fixed-point aritmetikou

Floating-point aritmetika

- S ... znaménko
- M ... mantisa
- E ... exponent

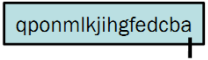
$$X = S \cdot M \cdot 2^E$$



=0,abcdefghijklmnopq

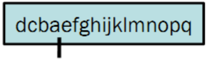
=a*1/2 + b*1/4 + c*1/8 + d*1/16 + ...

Fixed-point aritmetika



=qponmlkjihgfedcba,0

=a*1 + b*2 + c*4 + d*8 + ...



=dcba,efghijklmnopq

=a*1 + b*2 + c*3 + d*4 +
e*1/2 + f*1/4 + g*1/8 + h*1/16 + ...

DDA s fixed-point aritmetikou

```
#define FRAC_BITS 8

LineDDAFixed(int x1, int y1, int x2, int y2)
{
	int y = y1 << FRAC_BITS;
	int k = (y2-y1) << FRAC_BITS / (x2-x1);

	for (int x = x1; x <= x2; x++)
	{
		draw_pixel( x, y >> FRAC_BITS);
		y += k;
	}
}
```

Platnost algoritmů

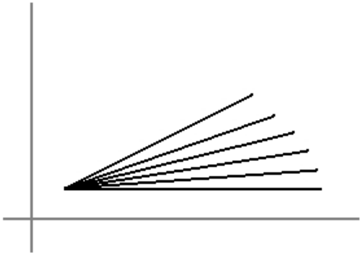
Algoritmy pro vykreslení úsečky jsou odvozeny pro případ, kdy

- úsečka leží v prvním kvadrantu,
- je rostoucí od počátečního bodu P_1 ke koncovému P_2
- a nejrychleji roste ve směru osy X.

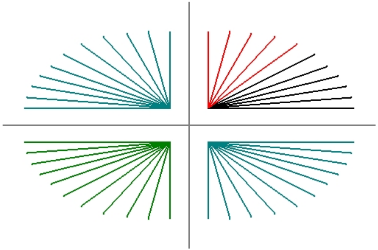
Platnost algoritmů

Algoritmy pro vykreslení úsečky jsou odvozeny pro případ, kdy

- úsečka leží v prvním kvadrantu,
- je rostoucí od počátečního bodu P_1 ke koncovému P_2
- a nejrychleji roste ve směru osy X.



Ostatní polohy úsečky je potřeba převést na tento případ (prohození souřadnic, os, apod.)



Obsah

Kružnice

Definice

Kružnice je základní geometrická vektorová entita definovaná:

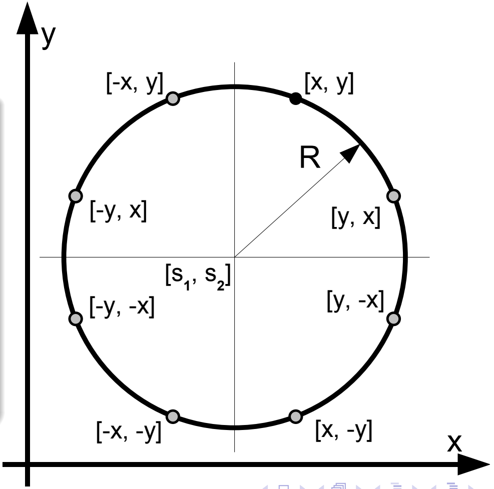
Kružnice

Definice

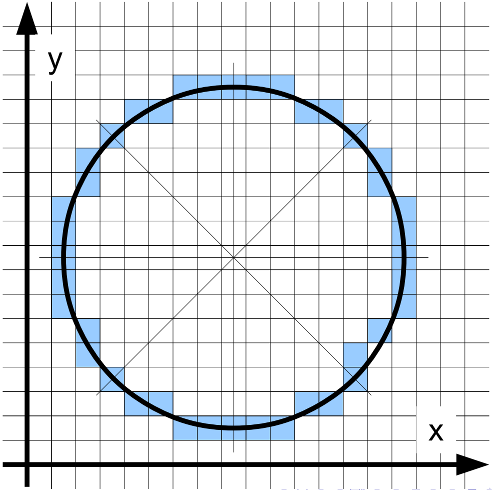
Kružnice je základní geometrická vektorová entita definovaná:

- souřadnicemi středu
- hodnotou poloměru
- rovnicí kružnice popisující geometrii

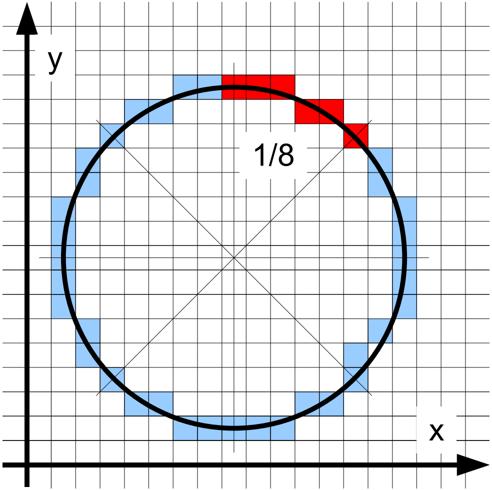
$$(x - s_1)^2 + (y - s_2)^2 - R^2 = 0$$



Jak na rasterizaci kružnice?



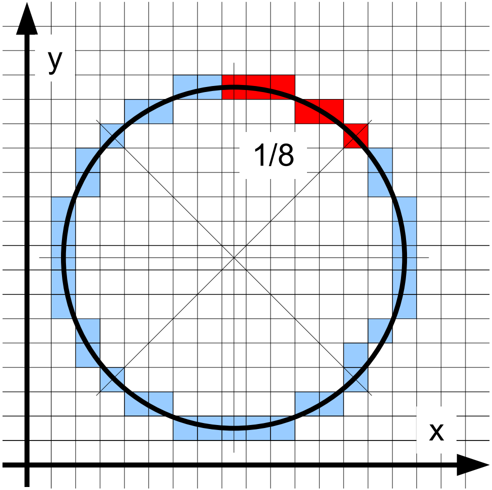
Kružnice



Vlastnosti

- Je 8x symetrická
- Provádíme výpočet pro 1/8 bodů v 1/2 prvního kvadrantu
- Zbylé body získáme záměnou souřadnic

Kružnice



Vlastnosti

- Je 8x symetrická
- Provádíme výpočet pro 1/8 bodů v 1/2 prvního kvadrantu
- Zbylé body získáme záměnou souřadnic
- Algoritmy jsou odvozeny pro kružnici se středem v počátku [0, 0]

Vykreslení kružnice po bodech

Popis

- ”Naivní”algoritmus rasterizace kružnice.
- Používá ”floating-point aritmetiku”.
- Nízká efektivita, náročná implementace v HW.

Vykreslení kružnice po bodech

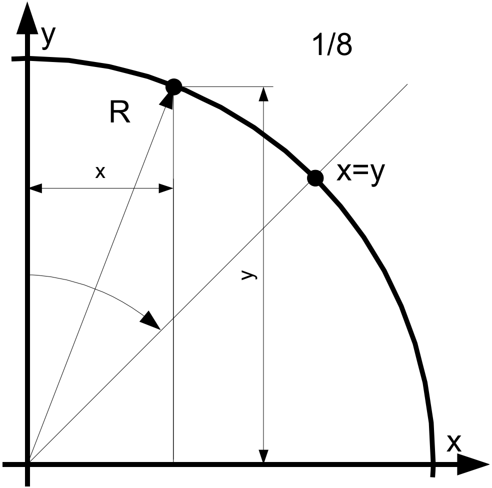
Popis

- "Naivní"algoritmus rasterizace kružnice.
- Používá "floating-point aritmetiku".
- Nízká efektivita, náročná implementace v HW.

Princip

- Vykreslujeme ve směru hodinových ručiček.
- Jdeme po pixelu od bodu $[0, R]$, dokud není $x = y$.
- V ose X postupujeme s přírůstkem $dx = 1$.
- Pozici v ose Y vypočteme podle vztahu $y = \sqrt{R^2 - x^2}$.
- Souřadnice Y se zaokrouhluje na nejbližší celé číslo.

Vykreslení kružnice po bodech




```
CircleByPoints(int cx, int cy, int R)
{
   int x = 0, y = R;

   while (x <= y)
   {
      draw_pixel_circle(x + cx, y + cy);
      x++;
      y = round(sqrt(R*R – x*x));
   }
}
```

Vykreslení kružnice jako N-úhelník

Popis

- Varianta algoritmu DDA pro kružnici.
- Aplikace rotační transformace bodu.
- Používá "floating-point aritmetiku".
- Nízká efektivita, náročná implementace do HW.

Vykreslení kružnice jako N-úhelník

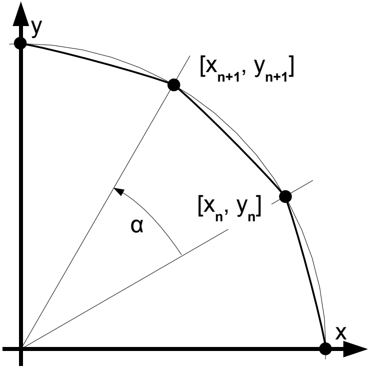
Popis

- Varianta algoritmu DDA pro kružnici.
- Aplikace rotační transformace bodu.
- Používá "floating-point aritmetiku".
- Nízká efektivita, náročná implementace do HW.

Princip

- Rekurentně se posouváme o konstantní přírůstek úhlu.
- Funkce *sin* a *cos* jsou vypočítány pouze jednou!
- Souřadnice X a Y se zaokrouhlují na nejbližší celé číslo.
- Vypočtené souřadnice spojujeme úsečkami.

Vykreslení kružnice jako N-úhelník



$$
\begin{aligned}
x_{n+1} &= x_n \cos\alpha - y_n \sin\alpha \\
y_{n+1} &= x_n \sin\alpha + y_n \cos\alpha
\end{aligned}
$$

```
CircleDDA(int R, int N)
{
   double cosa = cos(2*PI/N);
   double sina = sin(2*PI/N);
   int  x1 = R, y1 = 0, x2, y2;

   for (int i = 0; i < N; i++)
   {
      x2 = x1*cosa – y1*sina;
      y2 = x1*sina + y1*cosa;
      draw_line(x1, y1, x2, y2);
      x1 = x2;
      y1 = y2;
   }
}
```

Midpoint algoritmus pro kružnici

Popis

- Variace na Bresenhamův algoritmus, stejný přístup.
- Určování polohy "Midpointu"vůči kružnici (in, out).
- Používá celočíselnou aritmetiku, sčítání, porovnání.
- Velmi efektivní, snadná implementace v HW.

Midpoint algoritmus pro kružnici

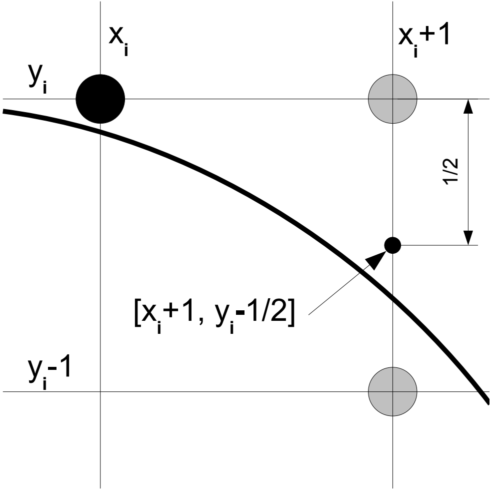
Popis

- Variace na Bresenhamův algoritmus, stejný přístup.
- Určování polohy ”Midpointu”vůči kružnici (in, out).
- Používá celočíselnou aritmetiku, sčítání, porovnání.
- Velmi efektivní, snadná implementace v HW.

Princip

- Vykreslujeme po pixelu od bodu $[0, R]$, dokud není $x = y$.
- V ose X postupujeme s přírůstkem $dx = 1$.
- O pusunu v ose Y rozhodujeme podle znaménka prediktoru.

Midpoint algoritmus pro kružnici




```
CircleMid(int s1, int s2, int R)
{  int x = 0, y = R;
   int P = 1-R, X2 = 3, Y2 = 2*R-2;

   while (x < y)
   {
      draw_pixel_circle(x, y);

      if (P >= 0)
         { P += -Y2; Y2 -= 2; y--; }

      P += X2;
      X2 += 2;
      x++;
   }
}
```

Midpoint algoritmus pro kružnici

Odvození prediktoru

$$\begin{aligned} F(x,y) &= x^2 + y^2 - R^2 = 0 \\ p_i &= F(x_i + 1, y_i - \frac{1}{2}) \\ p_i &= (x_i + 1)^2 + (y_i - \frac{1}{2})^2 - R^2 \end{aligned}$$

$$\begin{aligned} p_i < 0 &\implies y_{i+1} = y_i \\ p_i \geq 0 &\implies y_{i+1} = y_i - 1 \end{aligned}$$

Midpoint algoritmus pro kružnici

Odvození rekurentního prediktoru

$$p_{i+1} = (x_{i+1}+1)^2 + \left(y_{i+1} - \frac{1}{2}\right)^2 - R^2$$

$$\begin{aligned} p_{i+1} &= p_i + 2x_i + 3 \quad \Longleftarrow \quad p_i < 0 \\ p_{i+1} &= p_i + 2x_i - 2y_i + 5 \quad \Longleftarrow \quad p_i \geq 0 \end{aligned}$$

Startovací hodnota prediktoru je $p_i = 1 - R$

Midpoint algoritmus pro kružnici

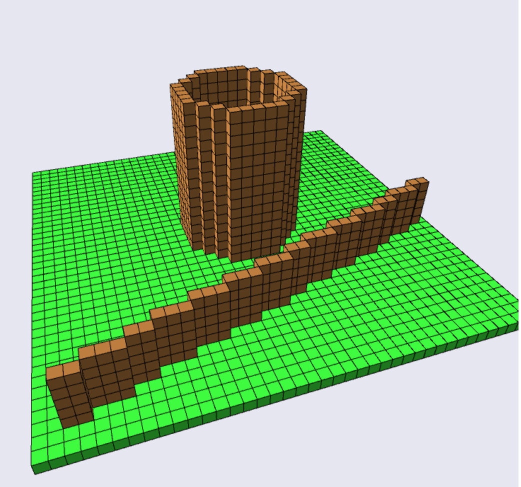
Odvození rekurentního prediktoru

$$p_{i+1} = (x_{i+1} + 1)^2 + (y_{i+1} - \frac{1}{2})^2 - R^2$$

$$\begin{aligned} p_{i+1} &= p_i + 2x_i + 3 \quad \Longleftarrow \quad p_i < 0 \\ p_{i+1} &= p_i + 2x_i - 2y_i + 5 \quad \Longleftarrow \quad p_i \geq 0 \end{aligned}$$

Startovací hodnota prediktoru je $p_i = 1 - R$???

Ukázka - Voxelová věž pomocí Midpoint algoritmu v Processing



Obsah

Elipsa

Definice

Elipsa je základní geometrická vektorová entita definovaná:

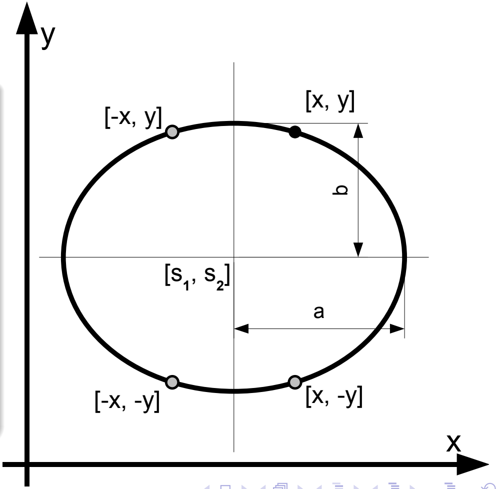
Elipsa

Definice

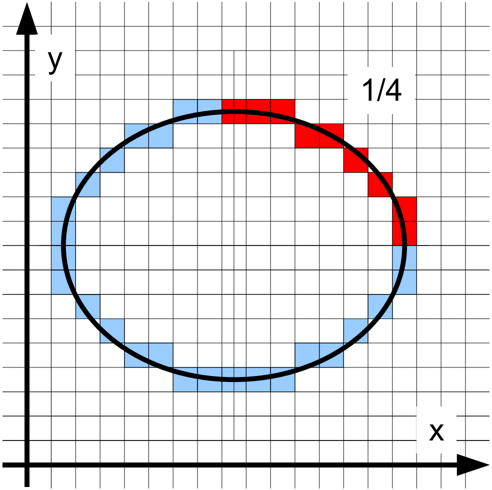
Elipsa je základní geometrická vektorová entita definovaná:

- souřadnicemi středu
- hodnotami hlavní a vedlejší poloosy
- úhlem natočení hlavní poloosy
- rovnicí elipsy popisující geometrii

$$F(x, y) : b^2x^2 + a^2y^2 - a^2b^2 = 0$$



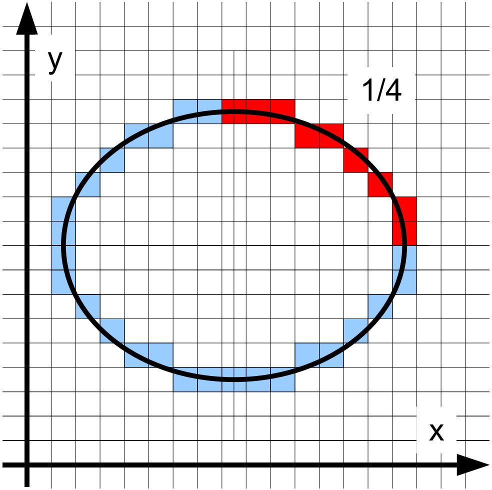
Elipsa



Vlastnosti

- Je 4x symetrická
- Provádíme výpočet pro 1/4 bodů
- Zbylé body získáme záměnou souřadnic

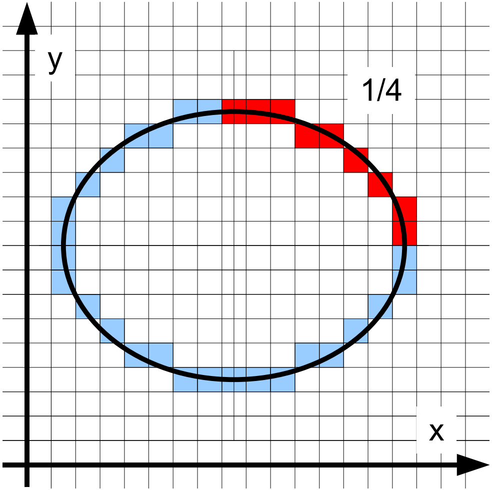
Elipsa



Vlastnosti

- Je 4x symetrická
- Provádíme výpočet pro 1/4 bodů
- Zbylé body získáme záměnou souřadnic
- Dvě oblasti výpočtů - jak je poznáme?

Elipsa



Vlastnosti

- Je 4x symetrická
- Provádíme výpočet pro $1/4$ bodů
- Zbylé body získáme záměnou souřadnic
- Dvě oblasti výpočtů - jak je poznáme?
- Algoritmy jsou odvozeny pro elipsu se středem v $[0, 0]$ a nulovým otočením

Midpoint algoritmus pro elipsu

Popis

- Ekvivalent Midpoint algoritmu pro kružnici.
- Určování polohy ”Midpointu”vůči elipse (in, out).
- Používá celočíselnou aritmetiku, sčítání, porovnání.

Midpoint algoritmus pro elipsu

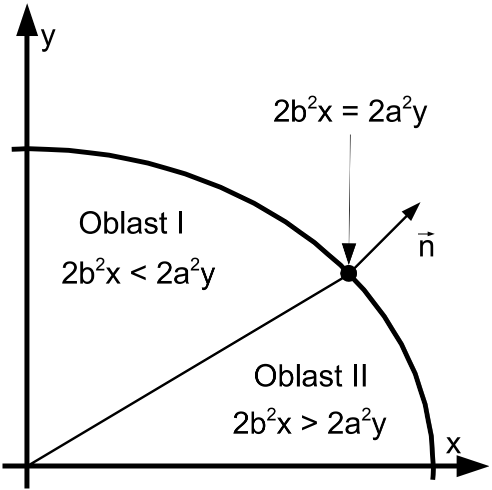
Popis

- Ekvivalent Midpoint algoritmu pro kružnici.
- Určování polohy "Midpointu"vůči elipse (in, out).
- Používá celočíselnou aritmetiku, sčítání, porovnání.

Princip

- Pro oblast I jdeme po pixelu od bodu $[0, b]$, dokud nejsou parciální derivace podle x a y rovny ($2b^2x = 2a^2y$).
- Pak pro oblast II až do bodu $[a, 0]$.
- V ose X/Y postupujeme s přírůstkem $dx/dy = 1$ (oblast I/II)
- Pusun v ose Y/X určuje znaménko prediktoru

Midpoint algoritmus pro elipsu

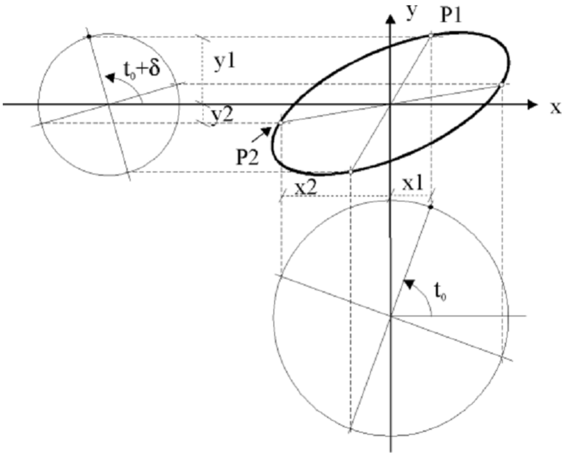



```
ElipseMid(int A, int B)
{   int x = 0, y = B, AA = A*A, BB = B*B;
    int P = BB – AA*B + AA/4;

    while (AA*y > BB*x)
    {   draw_pixel_elipse(x, y);
        if (P < 0)
            { P += BB*(2*x+3); x++; }
        else
            { P += BB*(2*x+3) + AA*(2-2*y); x++; y--; }
    }
    P = BB*(x+0,5)*(x+0,5)+AA*(y-1)*(y-1)-AA*BB;
    while (y >= 0)
    {   draw_pixel_elipse(x, y);
        if (P < 0)
            { P += BB*(2*x+2) + AA*(3-2*y); x++; y--; }
        else
            { P += AA*(3-2*y); y--; }
    }
}
```

Jak vykreslit elipsu v obecné poloze?

Elipsa pomocí dvou kružnic

- Složení dvou rotačních pohybů se stejnou úhlovou rychlostí
- Dvě kružnice s různým poloměrem
- x . . . dána polohou bodu na kružnici s poloměrem A
- y . . . dána polohou bodu na kružnici s poloměrem B



Elipsa pomocí dvou kružnic

Parametrické vyjádření elipsy se středem v počátku

- $f(t) = f(x(t), y(t))$
- $f(t) = (A.sin(t), B.sin(t + \delta))$