

Test

Adamofus

December 21, 2022

Contents

1	Projekce scény	iii
1.1	Stínění objektů v 3D	iii
1.2	Zoom	iii
2	Rasterizace	iv
2.1	Úsečka	iv
2.2	Zrcadlení	vi
2.3	Kružnice	vi
2.4	Kruh	vii
2.5	Ostatní	vii
2.6	Bezierova křivka	vii
2.7	Vyplňování	vii
3	Analýza	xii
3.1	Bod v polygonu	xii
3.2	Konvexní/konkávní	xiii
3.3	Detekce kolize	xiii
3.4	Oblasti průniku dvou objektů a průřez	xiii
3.5	Obsah polygonu	xv
3.6	Je A v B?	xvi
4	Rotace	xvii
4.1	2D	xvii
4.2	3D	xvii
4.3	Posunutí středu a osy otáčení	xviii
5	Generování tvarů a fraktály	xx
6	Praktická část	xxi

Chapter 1

Projekce scény

Projekcí scény je v této práci myšleno perspektivní vidění. Paprsek jdoucí od promítaného bodu do oka pozorovatele se promítá na určenou rovinu v prostoru, tz. tvoří průnik s touto rovinou. Výsledný bod je přenesený na 2D soustavu souřadnou v této rovině.

1.1 Stínění objektů v 3D

1.2 Zoom

Chapter 2

Rasterizace

Rasterizace je proces převodu vektorově definované grafiky do tzv. rastru, tedy mřížky skládající se z bodů (pixelů). Takový rastr je základem obrazového výstupu na digitálních zařízeních. V následujících kapitolách jsou představeny algoritmy rasterizace 2 objektů: úsečky a kruhu. Samostatná kapitola je pak věnovaná rasterizaci Beziérových křivek.

2.1 Úsečka

```
1 /.MP line A:x:int,y:int B:x:int,y:int
```

Převod vstupů

Úsečka je část přímky definovaná dvěma body: b_1 a b_2 . Pro následující algoritmy platí, že souřadnice x bodu b_1 je menší než x bodu b_2 , aby se mohly algoritmy posouvat o jeden dílek doprava, tj $x + 1$. Směrnice úsečky je $a = dy/dx$. Počítá se s vstupem $a \in (-1, 1)$, takže úsečka svírá s osou x úhel $0 - 45^\circ$ a b_2 je v 1. kvadrantu. Opět pro omezení na jedinou podmínku, zda je potřeba y zvětšit či nikoli.

Tyto nároky umožňují zvolit efektivní algoritmus, ale současně vyžadují převod vstupů a výsledných souřadnic.

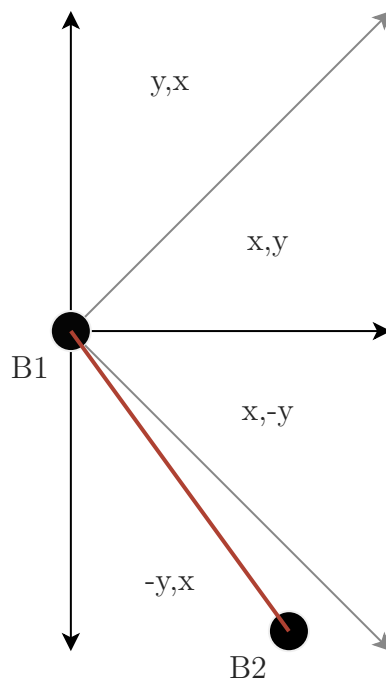


Figure 2.1: Převod souřadnic podle směrnice úsečky.

DDA

DDA využívá zaokrouhlované hodnoty $n * a$ pro výpočet y_{new} , je to prostý přístup. Současně ale zbytečně používá funkci zaokrouhlování či přetypování, kterou lze pro optimalizaci nahradit podmínkou s použitím proměnné, protože jsou pouze 2 možnosti pro y_{next} : $y_{next} = y$, nebo $y_{next} = y + 1$.

Proměnnou vyjadřuje $error(n) = ((n * a) \bmod 1) - 1/2$, kde n je krok iterace, takže $error(0) = -1/2$. Pokud platí zmíněná podmínka $error(n) \geq 0$, platí současně $y_{next} = y + 1$ a $error(n + 1) = error(n) + a - 1$, jinak platí $error(n + 1) = error(n) + a$.

Bresenhamův algoritmus

Další optimalizací je zbavení se desetinné čárky (tzv. float point number). Představme si $error(n) = 2 * dx * error(n)$. Jsou 2 možnosti:

$$error(n + 1) = error(n) + 2 * dy$$

$$error(n + 1) = error(n) + 2 * dy - 2 * dx$$

Nerovnice podmínky se nemění, protože po vynásobení pravé strany $2 * dx$: $error(n + 1) \geq 0 * 2 * dx$ zůstává stejná. Tím jsme se zbavili nutnosti použití desetinné čárky.

$$x \leftarrow x_1$$

$$y \leftarrow y_1$$

while $x \leq x_2$ **do**

```

vykresli bod[x,y]
   $x \leftarrow x + 1$ 
   $error \leftarrow error + 2d_y$ 
  if  $error \geq 0$  then
     $y \leftarrow y + 1$ 
     $error \leftarrow error - 2d_x$ 
  end if
end while

```

2.2 Zrcadlení

```
1 / .MP
```

Zrcadlení je často využívaná operace, například pro rasterizaci kružnice. Využívá bod B a vektor \vec{v} , přes který B zrcadlíme. Výstupem je zrcadlený bod B_z . Platí, že vektor $\overrightarrow{BB_z}$ je kolmý na \vec{v} a jeho délka je dvojnásobná vzdálenosti B od \vec{v} .

$\frac{1}{2}\overrightarrow{BB_z} = (k * x + P_x; k * y + P_y) - (b_x; b_y)$, skalární součin tedy:
 $(k * x + P_x - b_x; k * y + P_y - b_y) * (x; y) = 0$.

Úpravou rovnice vyjde $k = -\frac{x(B_x - P_x) + y(B_y - P_y)}{x^2 + y^2}$, bod získám: $B_z = 2(k * \vec{v} + P) - B = (2(k * x + P_x) - B_x; 2(k * y + P_y) - B_y)$

2.3 Kružnice

```
1 / .MP ring S:x:int,y:int r:int
```

Pro kružnici rovněž existuje Bresenhamův algoritmus. Algoritmus vykreslí 1/8 kružnice. V podmínce je tedy ze znalosti přímky svírající s osou x 45 stupňů: $x \geq y$. K vykreslení této části víme, že iterativně zvětšujeme y . x dekrementujeme, pokud je hodnota d kladná, tj. zajímají nás pouze 2 pixely. Pro vykreslení celku stačí díky symetrii kružnice tuto část 3x zrcadlit.

```

 $d \leftarrow x^2 + y^2 - r^2$ 
 $x \leftarrow r$ 
 $y \leftarrow 0$ 
 $d \leftarrow 0$ 
while  $x \geq y$  do
  vykresli bod[x,y]
   $y \leftarrow y + 1$ 
   $d \leftarrow d + d_y$ 

```

```

if  $d \geq 0$  then
     $x \leftarrow x - 1$ 
     $d \leftarrow d - d_x$ 
end if
end while

```

Drobná modifikace se může zakládat na výběru z 3 sousedících pixelů podle toho, který je nejbliž středu. Takové řešení je ale neefektivní, implementací se zde nezabývám.

2.4 Kruh

```

1  /.MP circle S:x:int,y:int r:int

```

Kruh se vykresluje tak, že se prochází body o souřadnicích od $S - r$ do $S + r$ pro x a y , jakoby byl kruh vepsán do čtverce. Bod se přidá, pokud je jeho vzdálenost od středu $\leq r$.

2.5 Ostatní

2.6 Bezierova křivka

2.7 Vyplňování

Řádkovací metoda

```

1  /.MP polygon-fill point*

```

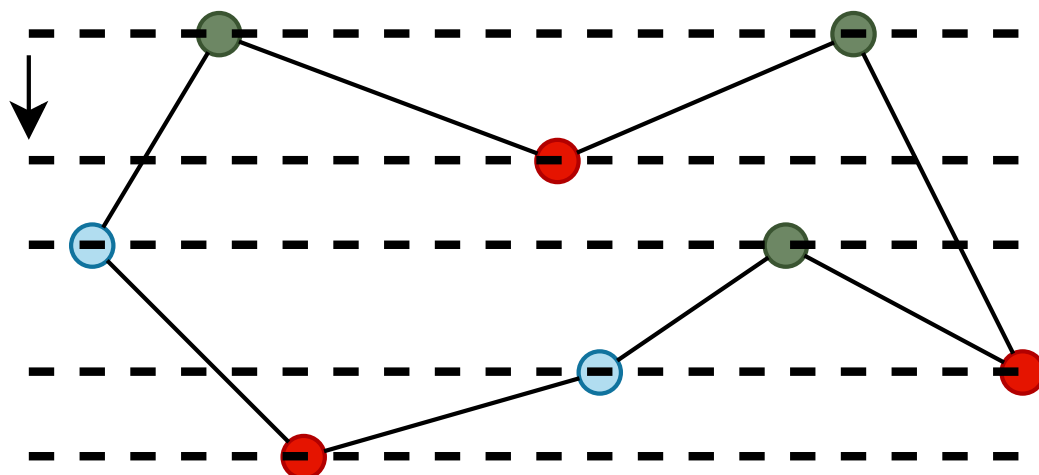


Figure 2.2: Vyplňování řádkováním ze shora dolů. Červené vrcholy nevedou na žádnou další úsečku, zelené vedou na 2 úsečky a modré vedou na druhou přiléhající úsečku.

Vyplňování rozbitím do trojúhelníků

Metoda rozbije libovolný konkávní polygon do trojúhelníků, které už stačí vyplnit úspornou metodou vyplňování trojúhelníka. Algoritmus omezuje vstup konkávního polygonu, protože konvexní polygon vyžaduje odlišný přístup - zejména ověření úhlu, který hrany svírají (konkávní/konvexní?). Podoba algoritmu by v intencích původní myšlenky nutně zahrnovala komplexnější přístup.

Flood fill

Algorithm 1 Řádkovací metoda

```

function SOLVE( $x, y$ )
   $row \leftarrow \text{maxY}(\text{points})$ 
   $\text{points}[], \text{init}[], \text{arr\_a}[], \text{lines\_points}[] \leftarrow []$ 
   $\text{points}[] \leftarrow [\text{points}[\dots], \text{points}[0]]$ 
  for  $i = 0; i = \text{len}(\text{init}); i++$  do
    for  $b\_idx = 1; b\_idx < \text{len}(\text{init}[i]); b\_idx++$  do
       $b\_val \leftarrow \text{init}[i][b\_idx]$ 
      for  $b\_tmp = 0; b\_tmp < \text{len}(\text{tmp}); b\_tmp++$  do
        if  $b\_val = \text{tmp}[b\_tmp]$  then
           $found \leftarrow \text{true}$ 
          if  $y(\text{points}[b\_val - 1]) < y(\text{points}[b\_val])$  then
             $\text{tmp}[b\_tmp] \leftarrow b\_val - 1$ 
             $\text{arr\_a}[b\_tmp] \leftarrow \text{get\_dx}(b\_val, b\_val - 1)$ 
          else if  $y(\text{points}[b\_val + 1]) < y(\text{points}[b\_val])$  then
             $\text{tmp}[b\_tmp] \leftarrow b\_val + 1$ 
             $\text{arr\_a}[b\_tmp] \leftarrow \text{get\_dx}(b\_val, b\_val + 1)$ 
          else if  $y(\text{points}[b\_val + 1]) = y(\text{points}[b\_val])$  then
             $\text{remove}(\text{tmp}[b\_tmp], \text{arr\_a}[b\_tmp], \text{lines\_points}[b\_tmp])$ 
          else
             $\text{remove}(\text{tmp}[b\_tmp], \text{arr\_a}[b\_tmp], \text{lines\_points}[b\_tmp])$ 
             $\text{remove}(\text{tmp}[b\_tmp], \text{arr\_a}[b\_tmp], \text{lines\_points}[b\_tmp])$ 
          end if
          break
        end if
      end for
    if  $!found$  then
       $\triangleright$  jsou zde 2 nové (neuvažované) úsečky. každá s tímto bodem
       $\text{tmp.add}(b\_val + 1, b\_val - 1)$ 
       $\text{arr\_a.add}(\text{get\_dx}(b\_val, b\_val + 1), \text{get\_dx}(b\_val, b\_val - 1))$ 
       $\text{new\_x} \leftarrow x(\text{points}[b\_val])$ 
       $\text{lines\_poins.add}(\text{new\_x}, \text{new\_x})$ 
    end if
  end for
  for  $y = 0; y < \text{init}[i][0]; y++$  do
     $row \leftarrow row - 1$ 
    for  $k = 0; k < \text{len}(\text{lines\_points}); k++ = 2$  do
       $A_x \leftarrow \text{lines\_points}[k] \leftarrow \text{lines\_points}[k] + \text{arr\_a}[k]$ 
       $B_x \leftarrow \text{lines\_points}[k+1] \leftarrow \text{lines\_points}[k+1] + \text{arr\_a}[k+1]$ 
      for  $x = A_x; x < B_x; x++$  do
         $\triangleright$  přidej bod  $[x, row]$ 
      end for
    end for
  end for
end function

function GET_DX( $A\_idx, B\_idx$ )
   $\text{pointA} \leftarrow \text{points}[A\_idx]$ 

```

Algorithm 2 Vyplňování rozbitím do trojúhelníků

```

points[]  $\leftarrow$  []
function SOLVE
  nB  $\leftarrow$  len(points)
  b_i  $\leftarrow$  0
  C  $\leftarrow$  0
  while nB > 3 do
    A  $\leftarrow$  C
    while !arr[(+ + b_i)%len(points)] do
    end while
    B  $\leftarrow$  b_i
    arr[B]  $\leftarrow$  false
    while !arr[(+ + b_i)%len(points)] do
    end while
    C  $\leftarrow$  b_i
    nB  $\leftarrow$  nB - 1
    fillTriangle(A, B, C)
  end while
  B  $\leftarrow$  C
  while !arr[(+ + C%len(points))] do
  end while
  fillTriangle(A, B, C)
end function
function GET_DX(A_idx, B_idx)
  pointA  $\leftarrow$  points[A_idx]
  pointB  $\leftarrow$  points[B_idx]
  return  $\frac{x(\textit{pointB})-x(\textit{pointA})}{y(\textit{pointB})-y(\textit{pointA})}$ 
end function

```

Algorithm 2 Vyplnění trojúhelníka

function FILLTRIANGLE(A,B,C)

 $a \leftarrow \min Y(A, B, C)$
 $b \leftarrow \text{middle} Y(A, B, C)$
 $c \leftarrow \max Y(A, B, C)$
 $a_1 \leftarrow \text{get_dx}(B, C)$
 $a_2 \leftarrow \text{get_dx}(A, B)$
 $a_3 \leftarrow \text{get_dx}(A, C)$
 $fP \leftarrow x(C)$
 $sP \leftarrow x(C)$
 $y \leftarrow y(C)$
for $y_p = y(c); y < y(b); y - -$ **do**
 $fP \leftarrow fP + a_1$
 $sP \leftarrow sP + a_3$
 $x \leftarrow fP$
while $++ x < sP$ **do**
 \triangleright přidej bod $[x, y_p]$
end while
end for
for $y_p = y(c); y < y(b); y - -$ **do**
 $fP \leftarrow fP + a_2$
 $sP \leftarrow sP + a_3$
end for
end function

Chapter 3

Analýza

3.1 Bod v polygonu

Pokud je polygon otevřený, žádný bod by správně neměl ležet v polygonu. V takovém případě stačí buď ověřit, zda je polygon úplný a neprůnikuje se. Prakticky v programu k tomu nedojde, přesto je možné obě věci obejít tv. winding číslem. Bod leží mimo polygon, pokud je 0, jinak leží uvnitř.

3.2 Konvexní/konkávň

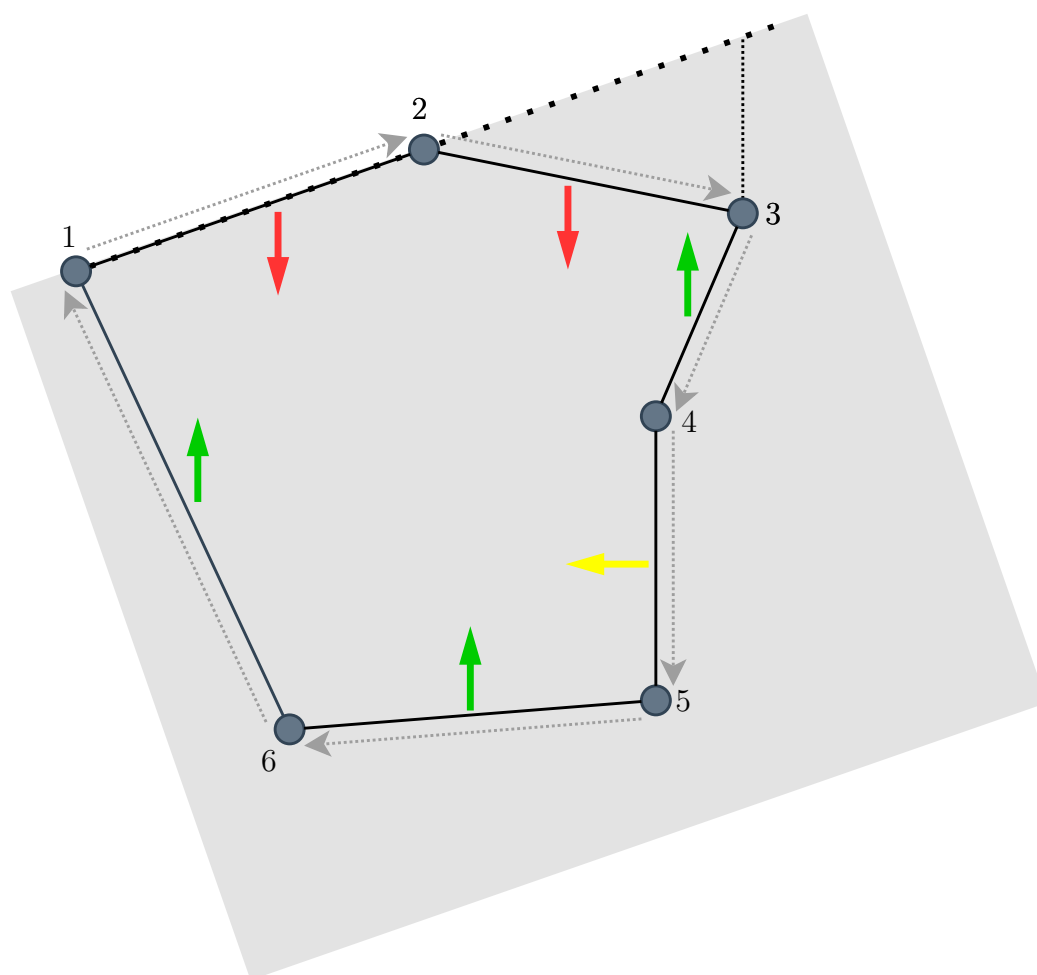


Figure 3.1: Identifikace konvexního/konkávňho polygonu

3.3 Detekce kolize

3.4 Oblasti průniku dvou objektů a průřez

Pro jasnost uvádím, že polygon A je polygon, který průnikuje polygon B. Výsledný průnik (polygon) označuju C.

Sutherland–Hodgman algoritmus

Pokud pomyslně prodloužíme hrany A, můžeme dělit strany této hrany na stranu, kde se nachází zbytek A a tedy i na stranu, v které se nenachází ani

jeden vrchol A. Jakýkoli bod na druhé straně polygonu není určitě uvnitř A, takže ho nepoužijeme pro C. Pokud je tomu naopak, bod necháme. Postup se totiž opakuje pro každou další stranu A, tz. že na konci zbydou pouze body (vrcholy B), které uvnitř A leží. Mezi body se navíc při procházení zjišťuje, jestli průnikují B, v takovém případě se přidává navíc i bod jako průsečík hran. Algoritmus je omezen pouze na konvexní A. V opačném případě nemusí fungovat koncept (nebo by nutně vyžadoval zbytečně náročnou modifikaci), protože bychom mohli vyloučit body, které jinak v polygonu leží. Současně B je nutně konvexní. V opačném případě může dojít k překrytí (tz. overlapping) hran C - v tomto případě by vzniklo výsledků víc jako víc polygonů/samostatných průnikových oblastí. Problém řeší Vattiho algoritmus, který zaznamenává body při vstupu B do A končí při výstupu B z A. Potom přidá body na A mezi I_{vstup} a I_{vystup} ve směru procházení, kterým procházíme vrcholy B.

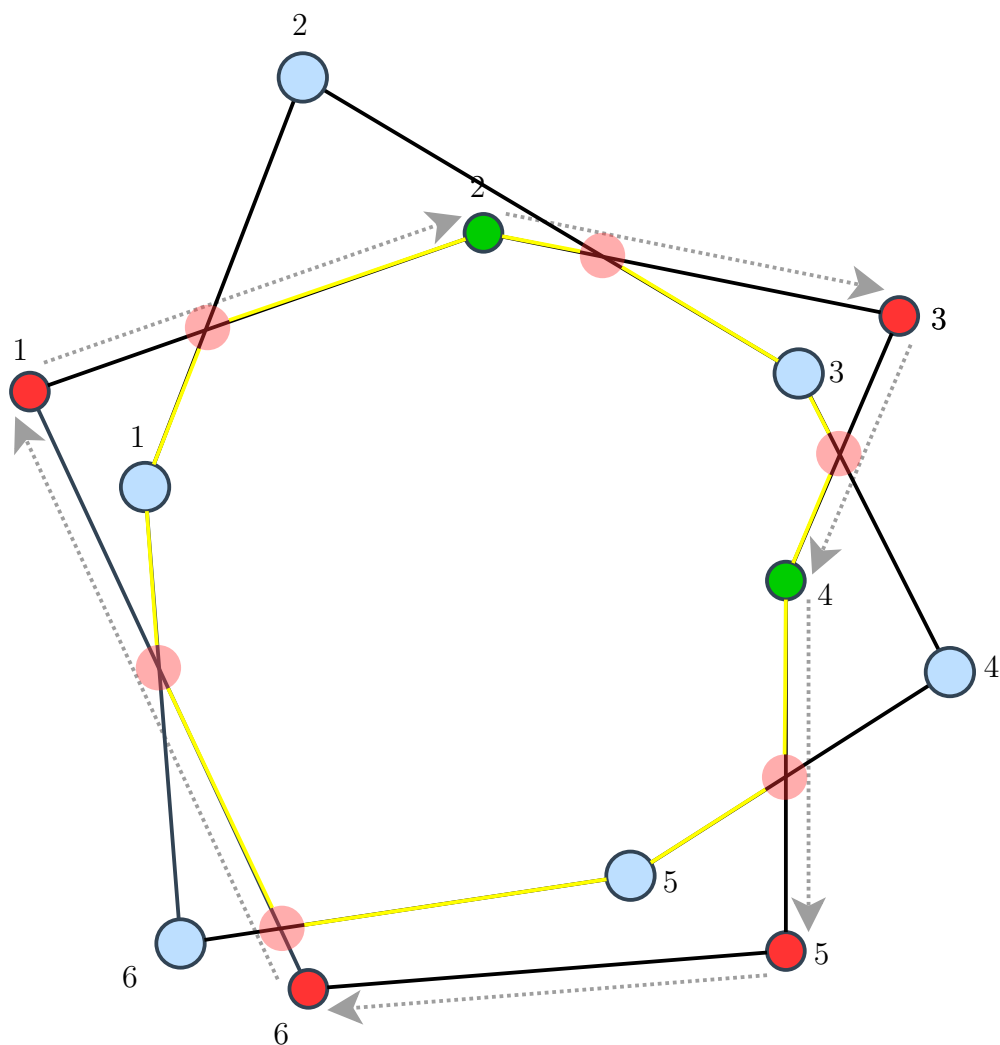


Figure 3.2: Vatti clipping algoritmus

3.5 Obsah polygonu

Obsah polygonu si lze $S_i = (x_1 - x_2) * \frac{y_1 + y_2}{2}$

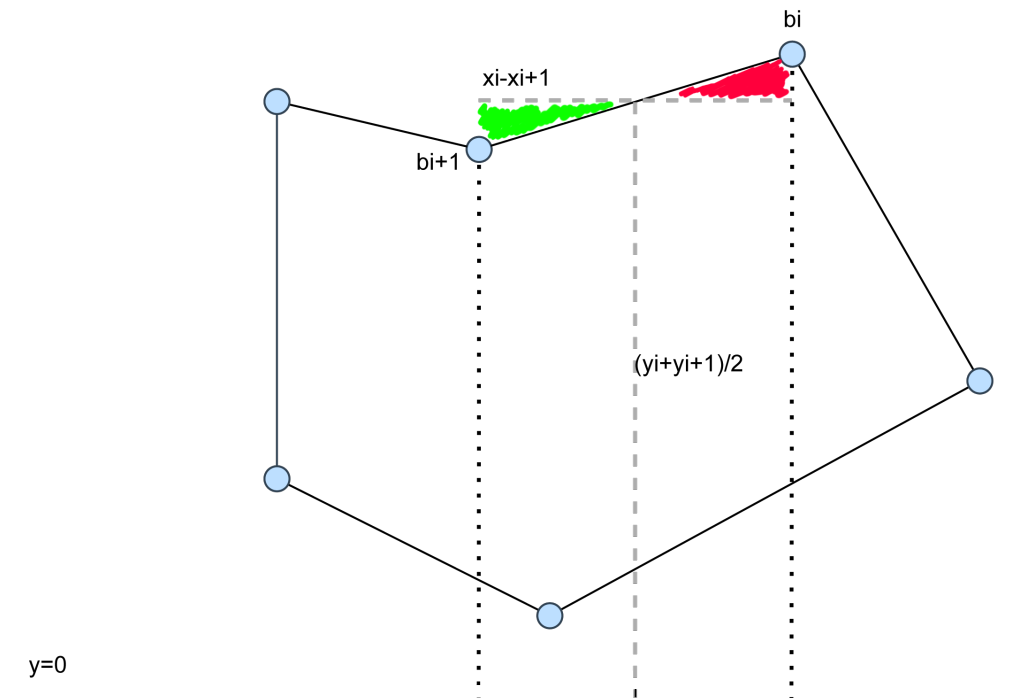


Figure 3.3: Vatti clipping algorithm

3.6 Je A v B?

Chapter 4

Rotace

1 `/.MP cube-example`

Otáčet bod vůči středu soustavy souřadné je jako nanášet ho na otočenou soustavu souřadnou, tedy násobit vektory udávající osy x , y , atd. takové soustavy. Tyto vektory mají délku 1. Lze zapsat takto:

$$p_n = \begin{pmatrix} X_1 & Y_1 \\ X_2 & Y_2 \end{pmatrix} \vec{p}$$

4.1 2D

Otočené souřadnice můžeme vyjádřit takto:

$$X_1 = \cos(-\alpha) = \cos(\alpha)$$

$$X_2 = \sin(2\pi - \alpha) = -\sin(\alpha)$$

$$Y_1 = \cos(\pi/2 - \alpha) = \sin(\alpha)$$

$$Y_2 = \sin(\pi/2 - \alpha) = \cos(-\alpha) = \cos(\alpha)$$

$$\begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

4.2 3D

Rotace bodu vůči ose Z ve směru hodinových ručiček:

$$\begin{pmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Rotace bodu vůči zvolené ose:

Máme vstup osu o a úhel α .

Mějme 2 soustavy souřadné A a B popsané jednotkovými vektory. Přitom A je naše výchozí, na které vykreslujeme:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Matice B má tvar:

$$\begin{pmatrix} a_x & b_x & o_x \\ a_y & b_y & o_y \\ a_z & b_z & o_z \end{pmatrix}$$

3. řádek B tvoří souřadnice osy o , takže 3. souřadnice p_b je právě vzhledem k o .

Čili je z následující rovnice zaručeno, že získáme přesně takový bod p_b , kde jeho 3. souřadnice je vzhledem k o . To potřebujeme, protože jsme zvolili matici rotace podle osy Z (respektive 3. osy...), kterou chceme bod násobit. Takže tato souřadnice bodu v B po rotaci bude stejná.

Z rovnice $p_a = p_b B$ vyjádříme tedy p_b vynásobením B^{-1} : $p_b = p_a B^{-1}$

Zbývající vektory a a b v B musí být jednotkové a vzájemně kolmé. takový a dostaneme třeba ignorováním z a přehozením souřadnic následovně: $\vec{a} = \{-y, x, 0\}$, kde x a y jsou souřadnice o . b je už jen vektorovým součinem a a b .

To je tedy převod relativních souřadnic ze soustavy A do soustavy B .

Rotace probíhá následovně:

Vstup: osa otáčení o , úhel α

1 převedeme souřadnice z A do B

2 zrotujeme (například přes matici 1.1, tj. vůči 3. ose)

3 vykreslujeme v A , takže převedeme z B do A

4.3 Posunutí středu a osy otáčení

Pro posunutý střed v 2D platí, že vektor \vec{XS} , kde S je střed a X bod, má fakticky souřadnice bodu \vec{p} . K výpočtu stačí potom přičíst S .

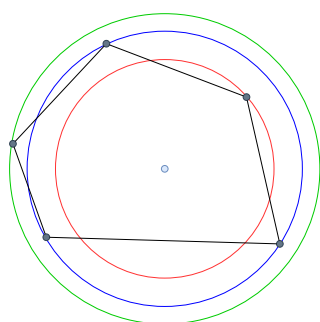
Stejně u posunuté osy v 3D, je takový vektor vlastně bod $p_a - S$. V praxi, grafickém editoru, určíme osu typicky dvěma body. Právě jeden z nich

(třeba pro intuici první určený) je totiž tento střed S . Nakonec stačí opět přičíst S .

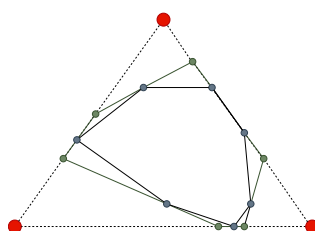
Chapter 5

Generování tvarů a fraktály

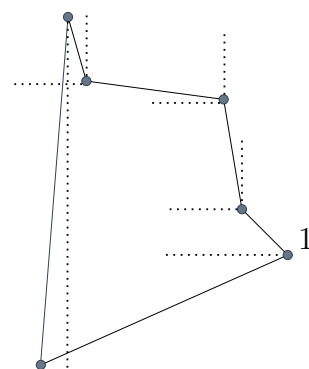
```
1 /.MP polygon polygon-rnd:n:int
```



(a) Caption for first figure



(b) Caption for second figure



(c) Caption for second figure

Figure 5.1: Schémata generování polygonu

Chapter 6

Praktická část

Výstupem programu je okno a textový výstup terminálu, v kterém program běží. Program je psán v jazyce C, tedy je před spuštěním kompilován. Pro kompilace

Syntax, ovládání, funkce