Řešení vybraných úloh grafických algoritmů

Autor
Adam Malíř
Vedoucí práce
Mgr. Josef Horálek, Ph.D.

DELTA – Střední škola informatiky a ekonomie Pardubice,
Ke Kamenci 151, 530 03 Pardubice I
Informační technologie 18-20-M/01
2022/2023
4.A

Zadání maturitního projektu z informatických předmětů

Téma práce: Řešení vybraných úloh grafických algoritmů

Způsob zpracování, cíle práce, pokyny k obsahu a rozsahu práce:

Cílem maturitního projektu je vytvořit sadu implementaci grafických algoritmů v objektově orientovaném jazyce. Autor práce podrobně popíše vybrané grafické algoritmy (zejména rasterizaci úsečky, Flood fill, seed seed a mandelbrotovu množinu). Navrhne OOP model pro implementaci vybraných algoritmů a realizuje jejich implementaci.

Anotace

Cílem a výsledkem práce je implementace některých základních algoritmů počítačové grafiky v jazce C. U některých problémů jsem navrhl vlastní postup (řádkování, generování polygonu).

Klíčová slova/keywords

line, polygon, circle, polygon-filling, SDL, polygon-intersection, rotation úsečka, polygon, kružnice, vyplňování, SDL, průnik, rotace

Prohlašuji, že jsem maturitní projekt vypracoval(a) samostatně, výhradně s použitím uvedené literatury.

Pardubicích dne 31.3.2023 ....

Obsah

1 Projekce scény **v**

2 Rasterizace **v**
2.1 Úsečka v
2.2 DDA vi
2.3 Bresenham vi
2.4 Zrcadlení vii
2.5 Kružnice vii
2.6 Kruh vii
2.7 Vyplňování viii
2.8 Řádkovací metoda viii
2.9 Rozbití do trojúhelníků x
2.10 Flood fill x

3 Analýza **xi**
3.1 Bod v polygonu xi
3.2 Konvexní/konkávní polygon xii
3.3 Průnik tvarů xii
3.4 Sutherland–Hodgman a Vattiho algoritmus xii
3.5 Obsah polygonu xiii

4 Rotace **xiv**
4.1 2D xiv
4.2 3D xv
4.3 Posunutí středu a osy otáčení xv

5 Generování tvarů **xvi**

6 Praktická část **xvii**
6.1 Syntax, ovládání, funkce xvii
6.2 Požadavky, omezení xviii
6.3 Struktura programu xviii

7 Závěr **xix**

Kapitola 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. Výsledné 2 souřadnice jsou souřadnicemi bodu v 2D, 3. se ignoruje.

Kapitola 2

Rasterizace

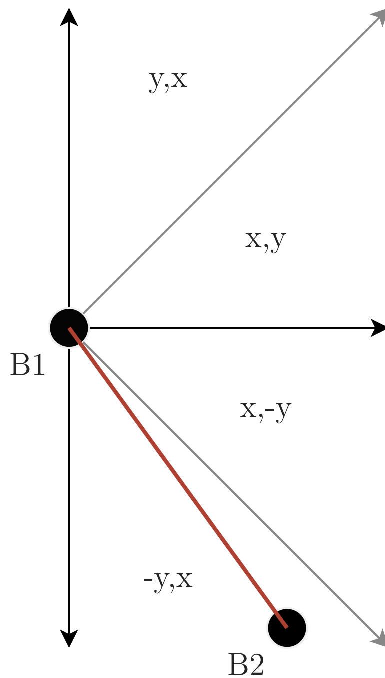
Rasterizace je proces převodu vektorově definované grafiky do tzv. rastru, tedy mřížky skládající se z bodů (pixelů). Takový rastr je základem obrazového výstupu na digitálních zařízení. V následujícíh kapitolách jsou představeny algoritmy rasterizace 2 objektů: úsečky a kruhu.

2.1 Úsečka

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

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 $a \subset < 0; 1 >$, takže úsečka svírá s osou x úhel $0 - 45°$ a b_2 je v 1. kvadrantu. Opět pro omezení na jedinou podmínku, zda je nutné přičíst 1 k y či nikoli při přičítání 1 k x.

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



Obrázek 2.1: Převod souřadnic podle směrnice úsečky.

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

2.3 Bresenham

[1] 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 $2 * dx$: $error(n + 1) >= 0 * 2 * dx$ zůstává stejná. Tím jsme se zbavili nutnosti použití desetinné čárky.

Algorithm 1 Bresenhamův algoritmus

```
x ← x_1
y ← y_1
while x <= x_2 do
vykresli bod[x,y]
    x ← x + 1
    error ← error + 2d_y
    if error ≥ 0 then
        y ← y + 1
        error ← error − 2d_x
    end if
end while
```

2.4 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 jako: $B_z = 2(k * \vec{v} + P) - B) = [\boldsymbol{(2(k * x + P_x) - B_x; 2(k * y + P_y) - B_y)}]$

2.5 Kružnice

```
1 ./MP ring S:<point> r:<int>
```

Algorithm 2 Kružnice

$d \leftarrow x^2 + y^2 - r^2$
$x \leftarrow r$
$y \leftarrow 0$
$d \leftarrow 0$
while $x >= y$ **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

Pro kužnici rovněž existuje Bresenhamův algoritmus. Algoritmus vykreslí 1/8 kružnice. V podmínce je tedy ze znalosti přímky svírající s osou x 45 stupnu: $x >= y$. 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. Drobná modifikace se může zakládat na výběru z 3 sousedících pixelů podle toho, který je nejblíž středu. Takové řešení je ale neefektivní, implementací se zde nezabývám.

2.6 Kruh

```
1 ./MP circle S:<point> 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 $<= r$.

Algorithm 3 Kruh

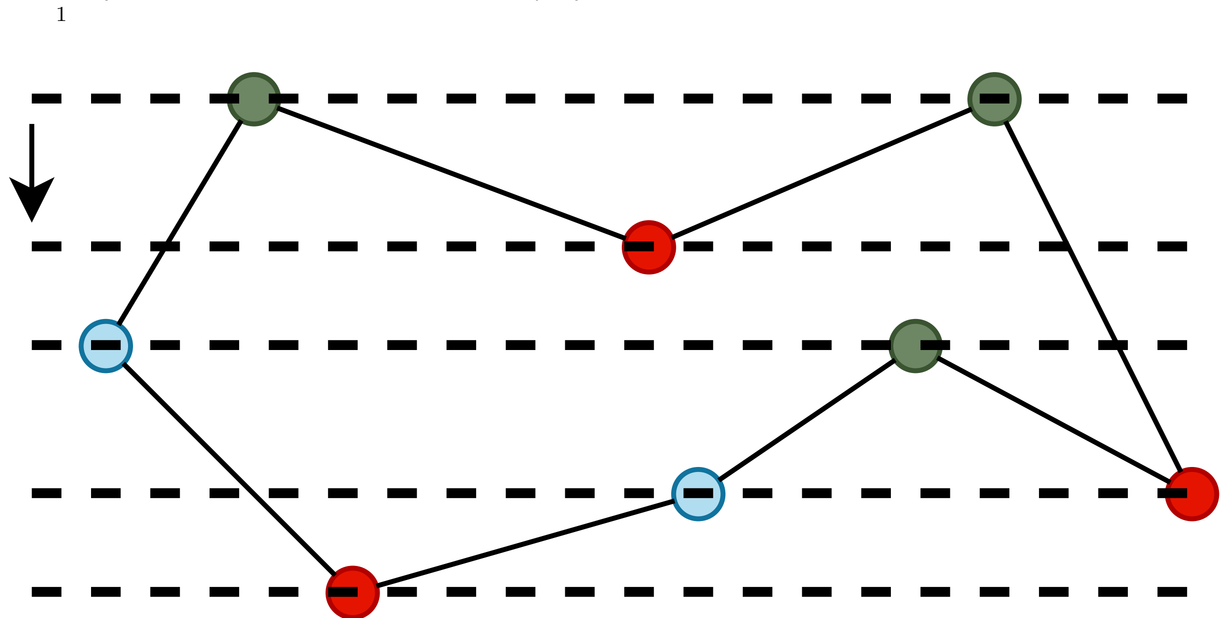
1: $x \leftarrow 0$
2: $y \leftarrow 0$
3: **for** $x = -r$ to r **do**
4: **for** $y = -r$ to r **do**
5: **if** $x^2 + y^2 \leq r^2$ **then**
6: vykresli bod[sx + x,sy + y]
7: **end if**
8: **end for**
9: **end for**

2.7 Vyplňování

2.8 Řádkovací metoda

```
1 ./MP polygon-fill point:<point>*
```

Pokud si představíme vodorovnou přímku, která protíná libovolný polygon. Pak řádkovací metoda vyplňuje $[row, x_i]- > [row, x_{i+1}]...[row, x_{i+2}]- > [row, x_{i+3}]$,..atd., kde x_i je x souřadnice i-tého bodu, když je vzestupně seřadíme podle x souřadnice. [1]



Obrázek 2.2: Vyplňování řádkováním ze shora dolů. Červené vrcholy nejsou součástí žádné další nezpracované hrany, zelené vedou na 2 hrany k zpracování a modré vedou na právě jednu vedlejší/přiléhající nezpracovanou hranu.

[1] Pokud $dx > 1$, může docházet k nespojeným oblastem (vynechaný jeden pixel na řádku).

Algorithm 4 Řádkovací metoda

```
row ← maxY(points)
points[], init[], arr_a[], lines_points[] ← []
points[] ← [points[...], points[0]]
for n_points_in_row = 0; n_points_in_row < init[i]; n_points_in_row + + do
    for b_idx = 1; b_idx < len(init[i]); b_idx + + do
        b_val ← converted_points[b_idx+n_points_in_row].pos
        next ← (b_val + 1)  mod n_points
        prev ← (b_val == 0)?n_points − 1 : b_val − 1
        found ← false
        for b_tmp = 0; b_tmp < len(tmp); b_tmp++ do
            if b_val=tmp[b_tmp] then
                found ← true
                if y(points[b_val − 1]) < y(points[b_val]) then
                    tmp[b_tmp] ← b_val − 1
                    arr_a[b_tmp] ← get_dx(b_val, b_val − 1)
                else if y(points[b_val + 1]) < y(points[b_val]) then
                    tmp[b_tmp] ← b_val + 1
                    arr_a[b_tmp] ← get_dx(b_val, b_val + 1)
                else if y(points[b_val + 1]) == y(points[b_val]) || y(points[b_val + 1]) = y(points[b_val]) then
                    remove(tmp[b_tmp],arr_a[b_tmp],lines_points[b_tmp])
                else
                    remove(tmp[b_tmp],arr_a[b_tmp],lines_points[b_tmp])
                    remove(tmp[b_tmp],arr_a[b_tmp],lines_points[b_tmp])
                end if
                break
            end if
        end for
        if not found then
            insert_pos ← 0
            while points[b_val].x > lines_points[insert_pos] and insert_pos < arr_size do
                insert_pos ← insert_pos + 1
            end while
            a1 ← get_dx(b_val, prev)
            a2 ← get_dx(b_val, next)
            tmp1 ← prev
            if a1 > a2 then
                prev ← next
                next ← tmp1
                tmp2 ← a1
                a1 ← a2
                a2 ← tmp2
            end if
            if points[prev].y ≠ points[b_val].y then
                insert(tmp, insert_pos, prev)
                insert(arr_a, insert_pos, a1)
                insert(lines_points, insert_pos, points[b_val].x)
                arr_size ← arr_size + 1
                insert_pos ← insert_pos + 1
            end if
            if points[next].y ≠ points[b_val].y then
```

2.9 Rozbití do trojúhelníků

Metoda rozbije libovolný konkávní polygon do trojúhelníků, které už stačí vyplnit úspornou metodou vyplňování trojúhelníka. Algoritmus omezuju na 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.

Algorithm 5 Vyplňování rozbitím do trojúhelníků

```
points[] ← []
function SOLVE
    nB ← len(points)
    b_i ← 0
    C ← 0
    while nB > 3 do
        A ← C
        while !arr[(++b_i)%len(points)]
do
        end while
        B ← b_i
        arr[B] ← false
        while !arr[(++b_i)%len(points)]
do
        end while
        C ← b_i
        nB ← nB − 1
        fillTriangle(A, B, C)
    end while
    B ← C
    while !arr[(++C%len(points))] do
    end while
    fillTriangle(A, B, C)
end function
function GET_DX(A_idx, B_idx)
    pointA ← points[A_idx]
    pointB ← points[B_idx]
    return (x(pointB)−x(pointA)) / (y(pointB)−y(pointA))
end function
```

Algorithm 5 Vyplnění trojúhelníka

```
function FILLTRIANGLE(A,B,C)
    P1 ← maxY(A, B, C)
    P2 ← middleY(A, B, C)
    P3 ← minY(A, B, C)
    a1 ← get_dx(P1, minX(P2, P3))
    a2 ← get_dx(P1, maxX(P2, P3))
    f ← P1
    s ← P1
    for row = P1; row > P2; row −− do
        f ← f + a1
        s ← s + a2
        x ← f
        while ++x < s do
                                    ▷ přidej bod [x,row]
        end while
    end for
    if X(P2) < X(P3) then
        a1 ← get_dx(P2, P3)
    else
        a2 ← get_dx(P2, P3)
    end if
    for row > P3; row −− do
        f ← f + a1
        s ← s + a2
        x ← f
        while ++x < s do
                                    ▷ přidej bod [x,row]
        end while
    end for
end function
```

2.10 Flood fill

Narozdíl od výše zmíněných algoritmů, algoritmus flood fill počítá s vstupem jednoho bodu, který je součástí oblasti, kterou vyplňujeme. Sousední body potom přidáme do fronty, pokud nejsou vyplněné nebo nemají barvu ohraničení. Postup se opakuje dokud není fronta prázdná. Vhodnější použití algoritmu je však v úpravě obrázku než vyplnění polygonů.

Algorithm 6 Flood Fill

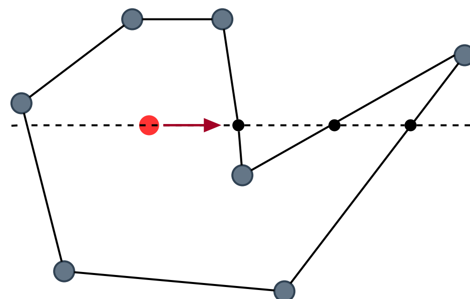
```
procedure FloodFill(x, y)                          ▷ Vstupem je výchozí bod (x, y)
    q ← prázdná fronta                             ▷ Fronta sousedních bodů k vyplnění
    add(q, (x, y))                                 ▷ Přidání výchozího bodu do fronty
    while q není prázdná do                        ▷ Dokud fronta není prázdná, opakuj:
        (a, b) ← první bod z q                     ▷ Odebere první bod z fronty
        změň barvu bodu (a, b) na požadovanou
        if (a − 1, b) není vyplněný a nemá barvu ohraničení then
            add(q, (a − 1, b))                     ▷ Přidá sousední bod do fronty
        end if
        if (a + 1, b) není vyplněný a nemá barvu ohraničení then
            add(q, (a + 1, b))                     ▷ Přidá sousední bod do fronty
        end if
        if (a, b − 1) není vyplněný a nemá barvu ohraničení then
            add(q, (a, b − 1))                     ▷ Přidá sousední bod do fronty
        end if
        if (a, b + 1) není vyplněný a nemá barvu ohraničení then
            add(q, (a, b + 1))                     ▷ Přidá sousední bod do fronty
        end if
    end while
end procedure
```

Kapitola 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. V programu k tomu však prakticky nedojde. Problém je možné řešit pomocí představy .tzv winding čísla - bod leží mimo polygon, pokud je 0, pro 1 leží uvnitř, jinak neurčujeme.



Obrázek 3.1: Rozhodnout, zda se bod nachází v polygonu, můžeme toutou vizualizací. Podle počtu průniků vodorvné přímky $x = B_x$ s hranami polygonu v nějakém směru. Lichý počet znamená, že leží uvnitř.

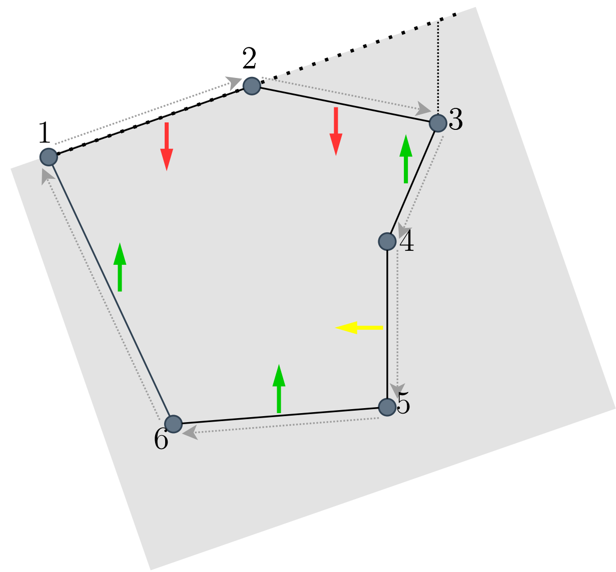
3.2 Konvexní/konkávní polygon

```
1 ./MP polygon-convex? A:<polygon> B:<polygon>
```

Algorithm 7 Určení konvexnosti polygonu

$p \leftarrow minX(...points)$ ▷ p je index bodu v points[]
if $Y(p+1) > Y(p-1)$ **then**
  $p_u \leftarrow 1$
else
  $p_u \leftarrow -1$
end if
$k \leftarrow p + p_u$
while $x(points[k+p_u]) > x(k)$ **do**
  if $is_above(line(k-p_u, k), point(k+p_u))$ **then**
    return 0
  end if
end while
while $x(points[k+p_u]) < x(k)$ **do**
  if $!is_above(line(k-p_u, k), point(k+p_u))$ **then**
    return 0
  end if
end while
if $k = p$ **then**
  return 1
end if
return 0

Konvexnost se nedá určit pouze z matematického výpočtů úhlů mezi každou dvojicí sousedících hran a ověřením, zda jsou všechny $< 180°$, protože výpočet nezohledňuje podobu celého polygonu čili výstupem výpočtu může být vnější úhel.



Obrázek 3.2: Identifikace konvexního/konkávního polygonu

3.3 Průnik tvarů

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

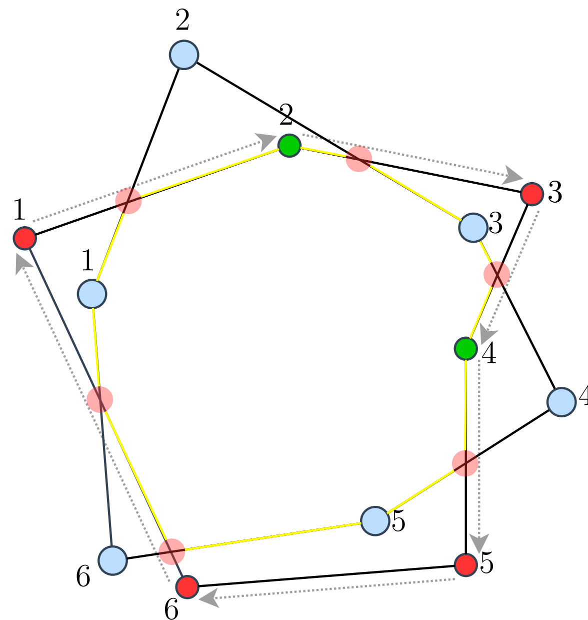
3.4 Sutherland–Hodgman a Vattiho algoritmus

```
1 ./MP polygon-shutherland-hodgman A:<polygon> B:<polygon>
```

```
1 ./MP polygon-vatti A:<polygon> B:<polygon>
```

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. overlaping) 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. Pokud žádný vrchol nepřidáme, ověříme, pokud nějaký bod B leží uvnitř A či nikoli, tedy jestli A leží uvnitř B.

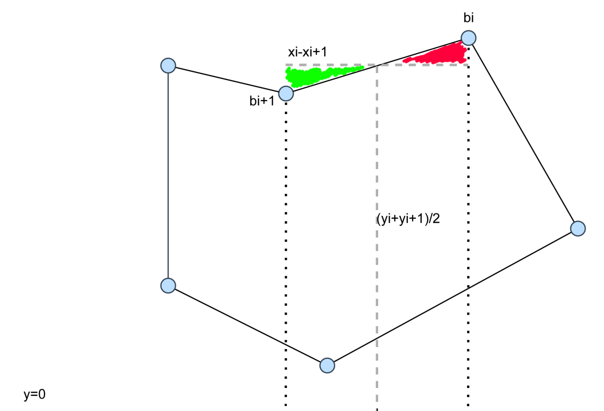


Obrázek 3.3: Shutherland-hodgman algoritmus

3.5 Obsah polygonu

```
1 ./MP polygon-S? A:<polygon> B:<polygon>
```

Obsah polygonu je $S = |\sum_{i=1}^{n}(x_i - x_{i+1})\frac{y_i+y_{i+1}}{2}|$. Pokud se prohodí x_i s x_{i+1}, prochází se defacto opačným směrem, takže výsledek je taky opačný, proto je třeba absolutní hodnoty.

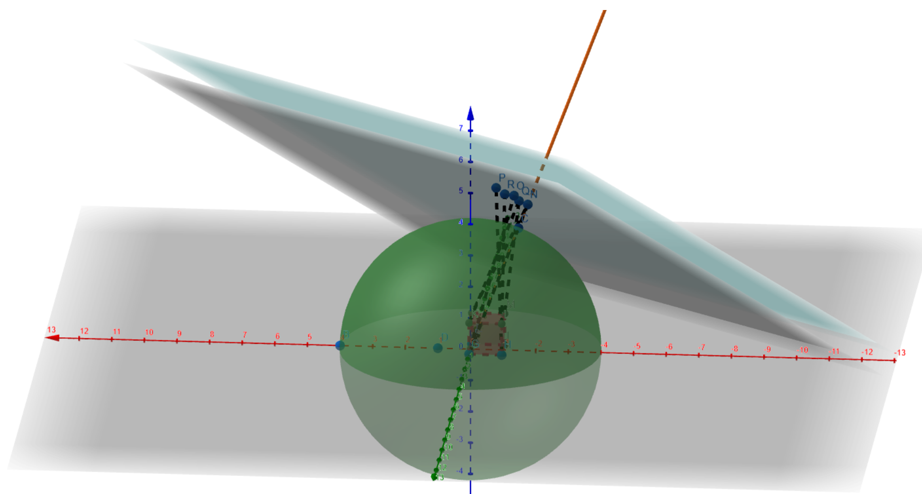


Obrázek 3.4: Vizualizace vzorce obsahu polygonu.

Kapitola 4

Rotace

```
./MP cube-example
```



Obrázek 4.1: 3D rotace a projekce funguje v programu jako na obrázku. Pozorovatel (ohnisko) se pohybuje na sféře (převod souřadnic z jedné báze do druhé a rotace) a současně se dívá do středu souřadnic. 2D souřadnice se promítají na rovinu kolmou k spojnici střed-pozorovatel, která je posunuta dál od středu.

Otáčet bod vůči středu soustavy souřadné je jako nanášet ho na otočenou soustavu souřadnou, tedy násobit vektory udávájící 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:[3]

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 vynasobením 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

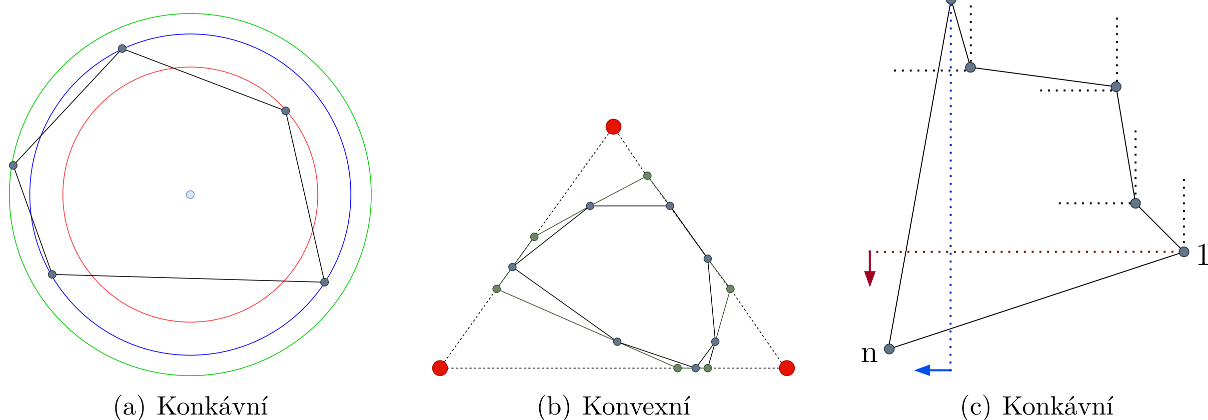
Pro posunutý střed v 2D platí, že vektor $\vec{XS}$, kde S je střed a X bod, má fakticky souřadnice bodu $\vec{p}$. K výpočtu stačí potom přičíst S.

Stejně u posunuté osy v 3D, je takový vektor vlastně bod $p_a - S$. V praxi, grafickém editoru, určujeme osu typicky dvěma body. Právě jeden z nich (třeba pro intuici první určený) je totiž tento střed S. Nakonec stačí opět přičíst S.

Kapitola 5

Generování tvarů

```
1 ./MP polygon|polygon-fill polygon-rnd:n:<int>
```



(a) Konkávní (b) Konvexní (c) Konkávní

Obrázek 5.1: Metody generování polygonu

Generování náhodných polygonů není to samé co pouhé generování náhodných bodů - strany se mohou protínat či svírat úhel 180 stupňů, což bývá v praxi nežádoucí. Metoda (a) je generování bodů na kružnici a následně výběr náhodné pozice na pomyslné úsečce střed-bod. První a poslední bod na kružnici zúžuje úhlový rozsah, v kterém další bod generujeme. Posledním bodem se pak stává generovaný bod a situace se opakuje. Hrany se nemohou protnout, protože se vždy nová hrana nachází celá ve zmíněné oblasti, kde žádný jiný bod neleží. Metoda může generovat konkávní polygony. Metoda (b) generuje na postupně procházených hranách trojúhelníku 1-2 náhodných bodů, v případné další iterace algoritmu se generuje nad výsledným tvarem atd. Následující iterací se dá generovat až dvojnásobný počet vrcholů tvaru z předchozí iterace. Výsledkem je konvexní polygon. Metoda (c) itereativně generuje body v určeném kvadrantu kartézských souřadnic, kde středem se následně stává generovaný bod. Díky tomu má generovaný bod ze všech nejmenší x souřadnici. První bod má nejmenší y. Pro spojení prvního a posledního bodu se určí bod $[x, y]$, kde $x < minX$ a $y < minY$.

Metoda (a):

1 generuj n náhodných úhlů

2 seřaď tyto úhly

3 konečnou pozici bodu urči jako $random_cislo * B \vec{-} S$

Kapitola 6

Praktická část

Výstupem programu je okno a text v terminálu, v kterém program běží. Program je psán v jazyce C, tedy je před spuštěním kompilován následovně.

```
1 git clone https://github.com/malirl/MP.git && cd resources/
    program; sudo make MP
```

6.1 Syntax, ovládání, funkce

Program běží ve dvou módech - první příkazový (PM), druhý vývojářský (VM). Rozdíl mezi nimi je však jen v datech, které zpracovává. Pro uvedení programu do PM je následující syntax:

```
1 ./MP nazev_obejktu parametr:hodnota parametr:hodnota ...
```

Pro uvedení programu do VM stačí čistě spustit:

```
1 ./MP
```

PM zpracovává jeden objekt/problém z příkazu, avšak VM jen volá soubor devtest.c, zdrojový kód, v kterým je možné nastavit příslušné vstupy a volat příslušné funkce. [1]

- Parametrem může být objekt, jehož parametr:hodnota jsou odděleny čárkami.
- Pořadí parametrů může být libovolné.
- Nadbytečné parametry program ignoruje.
- Nevalidní vstup program vyrozumí a skončí.
- Posouvání šipkami, zoom kolečkem myši/touchpadem.

[1] -.

- Speciální objekt example slouží stejně jako VM, ale vstup podobjektů nastavuje přes pomocné funkce. Nakonec libovolný výstup podobjektů zabaluje do vlastního čili výstupu jednoho objektu.

6.2 Požadavky, omezení

Pro úspěšnou kompilaci je nutná instalace překladače gcc a multimediální knohovny SDL2 na Linuxu (X-window system/Wayland). Jiné platformy nebyly testovány (stačí pouze přepsat Makefile).

6.3 Struktura programu

SDL2 řeší v programu pouze vytvoření okna, vykreslení bodů na něm a zachytávání uživatelského vstupu.[2] Zpracování regulárních výrazů řeší knihovna (modul) tiny-regex.

Klíčové vlastnosti

- Projekt se skládá z kontroleru, který validuje typ vstupu, nastavuje scénu, volá jednotlivá řešení v podsložkách jako /polygon ap.
- Dílčí řešení zapisuje do objektu body nebo podobjekty pro vykreslení. V případě, že je objekt součástí řešení a nevykresluje se, zapisuje do výstupu skládajícího se z dalších možných výstupů.
- Správně nastavený vstup objektu pro vykreslení je v syntaxi PM prostého/úplného tvaru vypsán před zpracováním.
- Vstup je předán main.c v podsložce dané úlohy, kde je případně upraven pro účely daného algoritmu.

Kapitola 7

Závěr

Literatura

[1] Tom Carter. The bresenham line algorithm, computer science csu stanislaus. 2014.

[2] Benedict HENSHAW. Intro to software rendering with sdl2. 2023.

[3] Rostislav HORCIK. Výpočet matice rotace, http://uivty.cs.cas.sz. 2009.