Algoritmy a rasterizace 2D grafických objektů

Algorithms and rasterization of 2D graphical objects

Jan Sečkař

Bakalářská práce
2007

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav aplikované informatiky
akademický rok: 2006/2007

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jan SEČKAŘ**
Studijní program:	**B 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Algoritmy a rasterizace 2D grafických objektů**

Zásady pro vypracování:

1. Vypracujte literární rešerši na zadané téma.
2. Seznamte se s rasterizačními algoritmy úsečky, kružnice a elipsy. V práci je blíže specifikujte a shrňte jejich vlastnosti.
3. Seznamte se s algoritmy pro vykreslování běžných interpolačních a aproximačních křivek. V práci je blíže charakterizujte.
4. Navrhněte program programy, které budou implementovány výše zmíněné algoritmy a budou provádět vykreslení všech uvedených objektů.
5. Navrhněte a realizujte vhodný výstup programu tak, aby se dosažené výsledky daly vhodně prezentovat na přednáškách předmětu Počítačová grafika.

Rozsah práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1.Martišek, D.: Matematické principy grafických systémů.Littera, Brno 2002.
2.Žára, J., a kol.: Moderní počítačová grafika. Computer Press, 2005.
3.Spell, B.: Java Programujeme profesionálně. Bogdan Kiszka. Praha : Com-puter Press, 2002.
4.Štalmach, J.: Aplikace algoritmů počítačové grafiky. Bakalářská diplomová práce. FT UTB Zlín 2004.

Vedoucí bakalářské práce: **Ing. Pavel Pokorný, Ph.D.**
Ústav aplikované informatiky

Datum zadání bakalářské práce: **13. února 2007**

Termín odevzdání bakalářské práce: **24. května 2007**

Ve Zlíně dne 13. února 2007

prof. Ing. Vladimír Vašek, CSc.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.
①

doc. Ing. Ivan Zelinka, Ph.D.
ředitel ústavu

ABSTRAKT

Tato Bakalářská práce se zabývá algoritmy pro rasterizaci základních geometrických primitiv a parametrických polynomiálních křivek. Teoretická část obsahuje několik vybraných rasterizačních algoritmů s jejich stručným popisem. Praktickou částí je program vytvořený v prostředí Java, který využívá teorie těchto algoritmů a názorně předvádí jejich využití včetně průběžných výpočtů. Tento program bude sloužit jako pomůcka předmětu Počítačová grafika.

Klíčová slova: rasterizace, úsečka, kružnice, elipsa, křivka

ABSTRACT

This Bachelor thesis is engaged in rasterization algorithms of basic geometric primitives and parametric polynomials curves. Theoretic part is composed of some sampled rasterizating algorithms with a brief description. Practical part is represented by a program created in Java environment witch uses theory of these algorithms and illustrates their usage including their parallel calculation.

Keywords: rasterization, vector, circle, ellipse, curve

Rád bych tímto poděkoval vedoucímu práce panu Ing. Pavlu Pokornému, Ph.D. za odborný dohled, rady a čas věnovaný této práci. Dále bych chtěl poděkovat celé rodině za podporu při studiích a všem, kteří ve mně vzbudili touhu objevovat.

Prohlašuji, že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků, je-li to uvolněno na základě licenční smlouvy, budu uveden jako spoluautor.

Ve Zlíně …….

Podpis diplomanta

OBSAH

ÚVOD

Jedním z velkých problémů počítačové vědy je vstup dat. V podstatě existují dva způsoby, jak je možné data zadávat. Prvním způsobem je generování dat přímo pomocí počítače. Druhým způsobem je vkládání dat uživatelem za pomocí určitého vstupního zařízení. Druhý způsob je dost náročný, proto je vyvíjeno velké úsilí pro zjednodušení a zefektivnění této činnosti.

S tímto problémem se nejvíce setkáváme v počítačové grafice. Pro většinu prováděných operací s objekty v počítači je nutné jejich přesné vymezení. Objekty v počítačové grafice jsou většinou množiny bodů, které jsou omezeny plochami, v rovině křivkami. Je tedy přirozeným požadavkem zjednodušit vstup právě křivek a ploch. Křivky a plochy jsou nejlépe reprezentovány funkcemi a ty je problematické zadávat. Proto jsou vyvíjeny metody, které umožní uživateli co nejjednodušeji zadat požadovanou křivku, nebo plochu, a to pokud možno tak, aby bylo možné odhadnout její tvar. Uživatel má obyčejně za úkol zadat jen několik řídicích bodů a matematický aparát se o vytvoření křivky postará sám.

Cílem této práce je popsat a podrobně vysvětlit funkci matematických aparátů pro sestrojování křivek v rovině a následného využití poznatků ve vytvořené aplikaci, která názorně předvede vybrané techniky rasterizace křivek a jednoduchých objektů.

I. TEORETICKÁ ČÁST

1 JEDNODUCHÉ OBJEKTY

Mezi jednoduché grafické prvky patří úsečka, kružnice a elipsa. Do této kategorie se taktéž řadí lomené čáry. I když jsou lomené čáry hojně využívány, nejsou zde podrobněji rozebírány, protože se v podstatě jedná o spojení více úseček. Taktéž sem patří kruhový a eliptický oblouk.

Tyto jednoduché objekty je možné reprezentovat v podstatě dvěma způsoby. Prvním je *vektorové vyjádření*, kde je každý tvar určen výčtem všech bodů a spojením mezi nimi. Tato forma zobrazení může být zobrazována například na vektorových kreslících zařízeních, nebo převedena pomocí algoritmů do rastrové podoby. Touto procedurou získáme druhý způsob zobrazení - *zobrazení rastrové*. Tím je posloupnost pixelů, které představují požadovaný grafický prvek. Toto zobrazení využívají například monitory a tiskárny. V podstatě se jedná o proces určování barev a pozic jednotlivých bodů rastru v matici, na základě vektorového popisu objektu. Například máme souřadnice začátku a konce čáry. Rasterizací ji převedeme na mnoho barevných samostatných bodů v barevné mřížce obrázku.

1.1 Úsečka

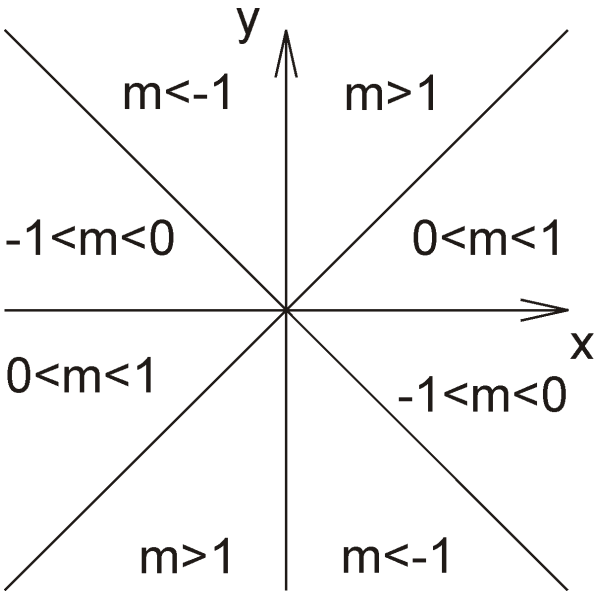
Úsečka je nejjednodušší grafický prvek. V analytické geometrii obsahuje úsečka nekonečně mnoho bodů. Na rastrovém monitoru se však skládá z konečného počtu bodů, které získáme rasterizací. Navzdory tomu, že je poměrně jednoduchá, její rasterizaci je věnována velká pozornost. Pomocí úseček se totiž vykreslují složitější grafické objekty, a proto se snažíme vykreslovat úsečku v co možná nejkratším čase a přitom s největší možnou přesností.

1.1.1 Rasterizace úsečky

Rasterizace úsečky je založený na vzorkovaní s konstantním krokem podle osy *x* a nebo podle osy *y*. To závisí na sklonu úsečky reprezentovaného směrnicí *m*, která se vypočte podle vzorce

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x}. \qquad (1)$$

Pokud je výsledná směrnice $|m| < 1$, potom má úsečka sklon k ose *x* menší jak 45 stupňů, a proto vzorkujeme podle osy *x* s krokem 1 pixel. Jestliže je $|m| > 1$, vzorkujeme podle osy *y*. Úsečku, jejíž výsledná směrnice $m = 1$ nazýváme diagonálou. Tu je možné vzorkovat podle libovolné osy. Osu, podle které vzorkujeme, nazýváme řídící (hlavní) osou. Druhou osu nazveme osou vedlejší. [10]



Obr. 1: Velikost směrnice m pro různé sklony úsečky

1.1.2 DDA algoritmus

Jinak také známý jako jednoduchý přírůstkový algoritmus. Je jedním z prvních v grafice použitých algoritmů. Je založen na opakovaném výpočtu souřadnic, při němž nové souřadnice závisí na souřadnicích předcházejícího bodu. Při každém výpočtu je přičítán konstantní přírůstek k řídící ose, kvůli vzorkování se většinou jedná o 1, k ose vedlejší je přičítán přírůstek neceločíselný. Ten je roven hodnotě *m* v případě že hlavní osou je osa *x* a hodnotě $\frac{1}{m}$ pokud je hlavní osou *y*. Protože je možné vykreslovat pouze celé hodnoty, je potřeba tuto hodnotu před každým vykreslením zaokrouhlit. Výpočet probíhá až do dosažení koncového bodu úsečky.

Z parametrického popisu úsečky

$$x = x_1 + t\Delta x, \tag{2}$$

$$y = y_1 + t\Delta y, \tag{3}$$

je možné odvodit pro osu *x* iterační zápis

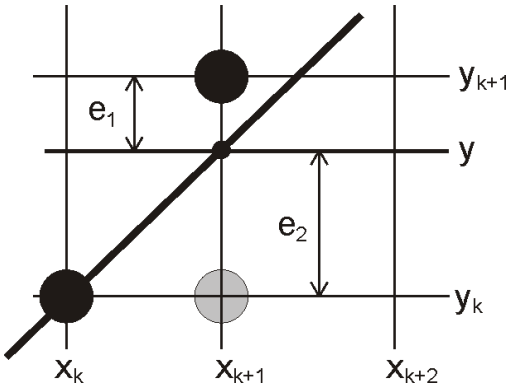
$$x_{k+1} = x_k + 1, \tag{4}$$

$$y_{k+1} = y_k + m, \tag{5}$$

který se provede pro Δx kroků. Pro osu *y* jsou vzorce obdobné.

1.1.3 Bresenhamův algoritmus

V DDA algoritmu byli použita i reálná čísla, která se před vykreslením zaokrouhlovala. Pomocí Bresenhamova algoritmu lze výpočet provést jen pomocí celočíselných operací a díky tomu je algoritmus výpočetně efektivnější. Celý algoritmus spočívá v hledání nejblíže ležících bodů ke skutečné úsečce pouze pomocí celočíselné aritmetiky. Ve směru hlavní osy je pozice následujícího bodu inkrementována opět s krokem 1, ale ve směru vedlejší osy se vypočítává chybový člen a podle něj se určuje pixel, který je blíže k úsečce a který bude vykreslen. Z toho plyne, že následující pixel může být na pozici (x_{k+1}, y_k), nebo (x_{k+1}, y_{k+1}).



Obr. 2: Výběr pixelu na základě velikosti odchylky pro řídící osu x

Pro výpočet odchylky hodnoty y_k od reálné hodnoty y je dán vztah

$$e_1 = y - y_k = \frac{\Delta y}{\Delta x} \cdot (x_k + 1) + y_0 - y_k . \tag{6}$$

Pro výpočet odchylky hodnoty y_{k+1} od reálné hodnoty y

$$e_2 = y_{k+1} - y = y_{k+1} - \frac{\Delta y}{\Delta x} \cdot (x_k + 1) + y_0 . \tag{7}$$

Na základě těchto výpočtů je možné rozhodnout o poloze nového bodu.

Pokud $e_1 > e_2$ bude nová souřadnice $y_k + 1$, jinak y_k.

Pro rychlejší výpočet je odvozena rovnice

$$e_1 - e_2 = \left(2 \cdot \frac{\Delta y}{\Delta x} \cdot (x_k + 1) - 2y_k + 2y_0 - 1 \right). \tag{8}$$

Vzorec (8) používá pro výpočet reálná čísla. Po úpravě do tvaru (9) lze výpočet provést pomocí celých čísel. Tím je výpočet zjednodušen.

$$P_k = \Delta x \cdot (e_1 - e_2) = 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + (2\Delta x \cdot y_0 - 2\Delta x) \tag{9}$$

Pro výpočet následujícího bodu je vzorec upraven do tvaru

$$P_{k+1} - P_k = 2\Delta y \cdot (x_{k+1} - x_k) - 2\Delta x \cdot (y_{k+1} - y_k). \tag{10}$$

Opakovaným výpočtem jsou z této rovnice získány hodnoty pro každý další bod z předcházejícího výpočtu. Podle znaménka rozhodovacího členu P_k se určí poloha dalšího bodu podle pravidel

$$P_k \leq 0 \Rightarrow P_{k+1} = P_k + 2\Delta y\,,\ y_{k+1} = y_k\,, \tag{11}$$

$$P_k > 0 \Rightarrow P_{k+1} = P_k + 2\Delta y - 2\Delta x,\ y_{k+1} = y_k + 1. \tag{12}$$

Uvedený postup rasterizuje pouze úsečky jejichž směrnice je kladná a menší než jedna. Proto před provedením algoritmu je nejprve třeba rozhodnout, která osa je řídící a podle toho vybrat algoritmus určený pro danou směrnici.

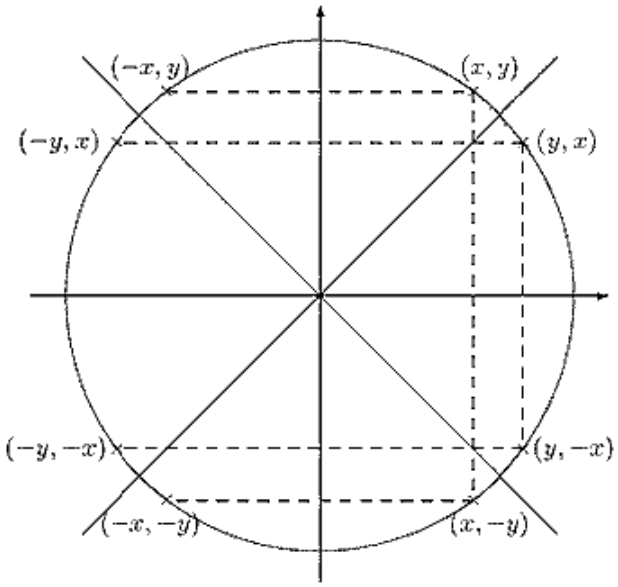
1.2 Kružnice

V euklidovské geometrii je kružnice množina všech bodů v rovině, které leží ve stejné vzdálenosti označované jako poloměr, od pevně daného bodu zvaného střed. Kružnice jsou jednoduché uzavřené křivky. Kružnice bývá nejčastěji zadána svým středem *[x,y]* a poloměrem *r*, ale také například pomocí polárních souřadnic, nebo parametricky.

1.2.1 Rasterizace kružnice

Rasterizace kružnice se většinou provádí pro kružnici se středem v počátku souřadného systému a po skončení výpočtů je posunuta na souřadnice zadaného středu. Rasterizace se provádí pouze pro jednu osminu kružnice (jeden oktan), zbytek není třeba počítat díky symetričnosti kružnice. Při rasterizaci kružnice můžeme využít metod pro kresbu úsečky a

nahradit kružnici lomenou čarou. Tento postup je však nepřesný, na druhou stranu velmi rychlý.



Obr. 3: Symetrické rozdělení kružnice

Další možností rasterizace kružnice je rasterizace pomocí polárních souřadnic.

$$x = x_c + r \cdot \cos\alpha \tag{13}$$

$$y = y_c + r \cdot \sin\alpha \tag{14}$$

Úhel α nabývá hodnot od 0 do $\frac{\pi}{4}$. Při konstantním kroku změny budou body rozložené na kružnici rovnoměrně. Tento krok by měl být roven hodnotě $\frac{1}{r}$, kdy se body liší o jeden pixel. Zbylých 7 oktanů se získá změnou znaménka nebo přehozením souřadnic získaných bodů, jak je patrné z Obr. 3.

Tyto metody jsou však značně neefektivní a výpočetně náročné, protože používají násobení a trigonometrické výpočty.

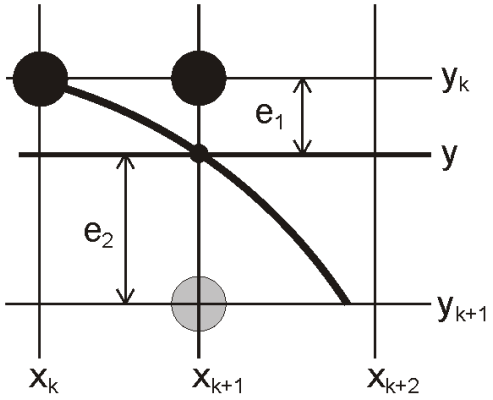
1.2.2 Bresenhamův algoritmus

Bresenhamův algoritmus pro rasterizaci úsečky je s určitými změnami možné aplikovat i na rasterizaci kružnici. V literatuře je nazýván také jako midpoint algorithm.

Opět se zde vybírá bod s nejmenší odchylkou od reálné polohy kružnice pomocí rozhodovacího členu, který vypočteme z rovnice

$$P_k = (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2. \tag{15}$$

Pokud je znaménko u P_k záporné bude následující bod vykreslen na stejné souřadnici y_k, v opačném případě bude souřadnice nového bodu $y_k - 1$. Při tomto výpočtu uvažujeme oktan v úseku od $x = 0$ do $x = y$.



Obr. 4: Výběr pixelu kružnice na základě velikosti odchylky

Podobně jako při rasterizaci úsečky jde o iterační algoritmus, a proto se poloha bodu počítá na základě bodu předcházejícího. Vzorec je tedy upraven do tvaru

$$P_{k+1} - P_k = +2x_k + 3 + \left(y_k - \frac{1}{2}\right)^2 + \left(y_{k+1} - \frac{1}{2}\right)^2. \tag{16}$$

Po výpočtu zjistíme podle znaménka polohu následujícího pixelu pomocí kritérií

$$P_k \le 0 \Rightarrow P_{k+1} = P_k + 2x_k + 3,\ y_{k+1} = y_k, \tag{17}$$

$$P_k > 0 \Rightarrow P_{k+1} = P_k + 2x_k + 5 - 2y_k,\ y_{k+1} = y_k + 1. \tag{18}$$

Tento algoritmus je možné aplikovat i na kruhovou výseč. U tohoto výpočtu se pouze mění rozmezí výpočtu podle zadané výseče.

1.3 Elipsa

Elipsa je uzavřená křivka v rovině. Všechny body elipsy mají stejný součet vzdáleností od dvou pevně zvolených bodů, které se nazývají ohniska. Úsečku spojující libovolný bod na elipse s ohniskem nazýváme průvodič. Spojíme-li ohniska úsečkou, v jejich středu je střed elipsy. Nejdelší spojnice středu elipsy a bodu na elipse se nazývá hlavní poloosa. Nejkratší taková spojnice je vedlejší poloosa.

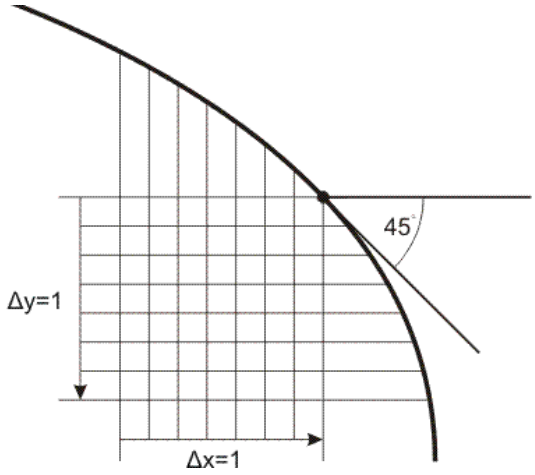
1.3.1 Bresenhamův algoritmus

Rasterizaci elipsy, stejně jako u kružnice, je nejjednodušší provádět se středem v počátku souřadného systému a po provedení výpočtů posunout na zadaný střed elipsy $[x_c, y_c]$.

Stejně jako u kružnice se k rasterizaci elipsy nejčastěji používá Bresenhamův algoritmus s rozhodovacím kritériem. Výpočet bodů musí být proveden pro celý jeden kvadrant. Zbylé body dopočítáme na základě symetrie elipsy.

Implicitní rovnice elipsy se středem v počátku je

$$F(x, y) = b^2x^2 + a^2y^2 - a^2b^2 = 0. \qquad (19)$$



Obr. 5 – Změna řídící osy při rasterizaci elipsy

Při rasterizaci jednoho kvadrantu elipsy dochází ke změně řídící osy. V bodě ve kterém dojde ke změně má tečna elipsy směrnici -1. [10] Tento bod se po vyjádření z rovnice elipsy nachází na souřadnicích, které se vypočítají podle vztahu

$$\left[\frac{a^2}{\sqrt{a^2+b^2}}, \frac{b^2}{\sqrt{a^2+b^2}}\right]. \tag{20}$$

O rozhodovacím členu se rozhodne na základě následujících pravidel. Ty platí pro řídící osu *x*.

$$P_k \leq 0 \Rightarrow P_{k+1} = P_k + b^2\left(2x_k + 1\right) \tag{21}$$

$$P_k > 0 \Rightarrow P_{k+1} = P_k + b^2\left(2x_k + 1\right) - 2a^2 y_k \tag{22}$$

Pro řídící osu y a druhou část algoritmu jsou vztahy pro výpočet podobné.

2 KŘIVKY

Křivky jsou obyčejně v počítači reprezentovány jako soustava parametrů rovnice, která je posléze generativně zobrazována. Toto vyjádření může být v podstatě trojího druhu: explicitní, implicitní, nebo parametrické.

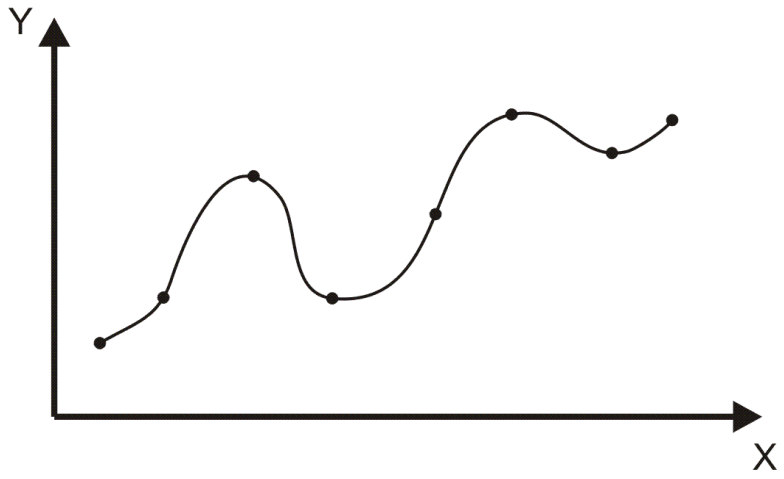
Křivky, které poskytují dostatečně širokou škálu tvarů jsou křivky třetího stupně (kubiky). Tyto křivky jsou také nejčastěji využívány. Jsou jednoduše manipulovatelné a je možné u nich zaručit takzvanou spojitost C^2, která vytváří plynulou návaznost křivek. Výpočet těchto křivek bývá nenáročný.

Modelování probíhá u většiny křivek tak, že pomocí matematického aparátu se určí průběh křivky z polohy předem definovaných řídicích bodů, nebo tečných vektorů.

Existují dva základní druhy interpretace řídicích bodů a to interpolace a aproximace.

2.1 Křivky interpolační

Interpolační křivka k dané množině bodů je taková křivka, která jimi prochází.



Obr. 6 – Interpolační křivka

2.1.1 Interpolace jediným polynomem

Nejčastější interpolační technikou je interpolace polynomem, a to buď jediným polynomem n -1 řádu pro n bodů, nebo po částech. [9]

Interpolace polynomem n -1 řádu znamená najít řešení rovnic

$$y_i = \begin{vmatrix} a_{11} & \cdot & \cdot & a_{1n} \\ \cdot & \cdot & \cdot & \cdot \\ & & & \\ \cdot & \cdot & \cdot & \cdot \\ a_{n1} & \cdot & \cdot & a_{nn} \end{vmatrix} \begin{vmatrix} 1 \\ x_i \\ x_i^{\,2} \\ \cdot \\ x_i^{\,n-1} \end{vmatrix}. \qquad (23)$$

Jako vstupní parametry této rovnice jsou body, jimiž má křivka procházet a jejím řešením jsou parametry matice a_{ii}. Postupně se do výrazu dosazují body, jimiž má křivka procházet a z nich se získá soustava rovnic.

Nevýhodou polynomiální interpolace je, že při polynomech vyššího řádu mohou u křivky vznikat nepřirozené vlnění, a proto bývá polynomiální interpolace používána pro křivky maximálně pátého stupně.

2.1.2 Interpolace po částech

Častější interpolační technikou je interpolace křivky po částech.

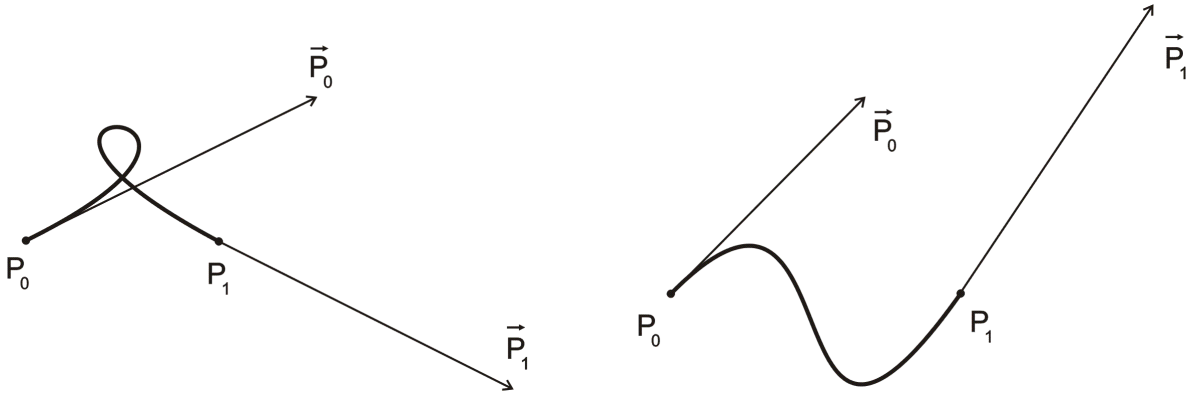
Aproximací polynomem třetího řádu potom rozumíme aproximaci jejích každých čtyř bodů polynomem třetího stupně. Tímto však vzniká problém návaznosti křivek v místě jejich propojení. Existují však metody, pomocí kterých je řešena návaznost v těchto bodech automaticky.

Interpolační metody nejsou pro výše uvedené nevýhody příliš často v počítačové grafice využívány. Tyto metody nacházejí uplatnění v numerické matematice a v matematické statistice, kde se na základě interpolací usuzuje na potenciální chování nějakých jevů v budoucnu (potom se hovoří o extrapolaci). [9] Tyto metody byly z těchto oborů také původně odvozeny.

2.1.3 Fergusonovy kubiky

Roku 1964 používal J. C. Ferguson metodu pro generování křivek, která je řízena dvěma body a směrovými vektory.

Krajní body jsou významné pro pozici křivky, protože jimi křivka prochází. Vektory pak určují míru vyklenutí křivky. Čím je velikost vektoru větší, tím více se k němu křivka přimyká.



Obr. 7 – Příklad Fergusonových kubik

Křivka je zadána body P_0, P_1 a vektory P_0', P_1'. Rovnice výsledné křivky má tvar

$$P(t) = P_0 F_1(t) + P_1 F_2(t) + P_0' F_3(t) + P_1' F_4(t), \tag{24}$$

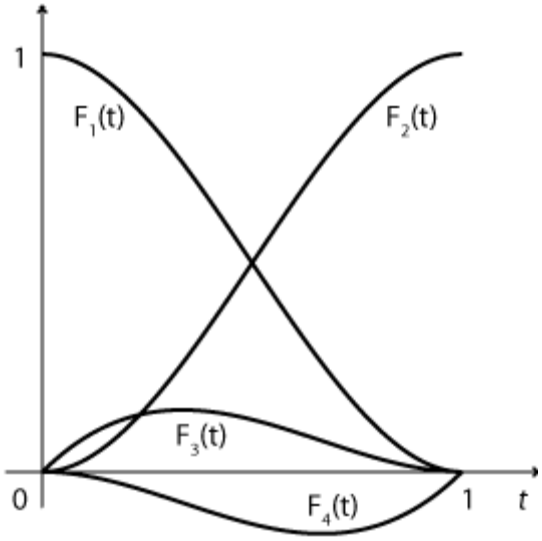
kde F_1, F_2, F_3, F_4 jsou kubické Hermitovské polynomy tvaru

$$\begin{aligned} F_1(t) &= 2t^3 - 3t^2 + 1, \\ F_2(t) &= -2t^3 + 3t^2, \\ F_3(t) &= t^3 - 2t^2 + t, \\ F_4(t) &= t^3 - t^2. \end{aligned} \tag{25}$$

a hodnota t nabývá hodnot z intervalu $\langle 0,1 \rangle$. Pokud položíme $t = 0$ je $P(0) = P_0$. Analogicky pro $t = 1$ je $P(1) = P_1$. To je důkaz že křivka prochází krajními body. Zderivujeme-li $P(t)$ podle t a dosadíme $t = 0$ a $t = 1$, vyplyne, že: $P'(0) = P_0'$ a $P'(1) = P_1'$. Tečné vektory v krajních bodech výsledné křivky jsou identické s těmi, které zadal uživatel.

Maticový zápis:

$$Q(t) = T \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_0' \\ P_1' \end{bmatrix} \tag{26}$$



Obr. 8 - Hermitovské polynomy

P(*t*) je kubikou, neboť z rovnic (24) a (25) plyne, že se jedná o součet polynomů stupně maximálně tři.

Položíme-li $P_0' = P_1' = P_1 - P_0$, aproximujeme touto metodou úsečku.

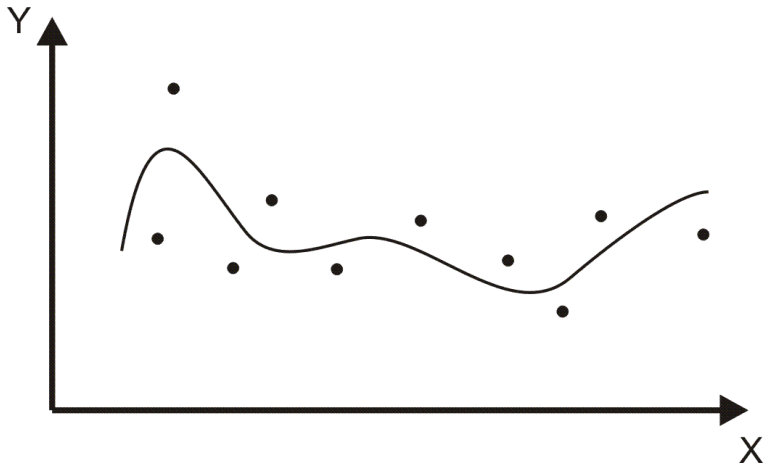
Spojitost při navazování oblouků je zaručena, v případě že je roven poslední bod předchozího oblouku prvnímu bodu oblouku následujícího. Spojitost C^2 je zaručena, pokud jsou velikosti vektorů P_1' předchozího a P_0' následujícího oblouku shodné.

2.2 Křivky aproximační

Aproximací bodů rozumíme vytvoření takové křivky, která je těmito body vhodně řízena. Není kladen požadavek na procházení opěrnými body a dokonce ani prvním a posledním bodem. V podstat existují dva pohledy na aproximaci. [9]

Z matematiky je znám první. Jeho cílem je smysluplná interpretace vstupních dat.

Druhý je používán v počítačové grafice. Jeho účelem je generování křivky požadovaného tvaru. Křivka může být řízena body, nebo vektory. Pokud se jedná o body potom můžeme hovořit o řídicím polygonu. Podle algoritmu, kterým je křivka vytvořena jsou zaručeny její vlastnosti. Základní požadované vlastnosti jsou: hladkost, spojitost a počet inflexí.



Obr. 9 – Aproximační křivka

V počítačové grafice se nejčastěji používá aproximace křivky po částech, které jsou generovány pomocí polynomů třetího řádu (kubiky). Kubiky jsou dostatečně flexibilní, a proto se jimi dá vyjádřit téměř vše, co je v praxi potřeba. Velmi důležitou vlastností kubik je, že stupeň polynomu tři je výpočetně nenáročný a díky tomu je možné tyto křivky vytvářet interaktivně.

Parametricky lze danou kubiku Q(t) vyjádřit ve tvaru

$$\begin{aligned} &x(t) = a_x t^3 + b_x t^2 + c_x t + d_x y(t) = a_y t^3 + b_y t^2 + c_y t + d_y z(t) = \\ &= a_x t^3 + b_x t^2 + c_x t + d_x. \end{aligned} \tag{27}$$

Zkráceně můžeme zapsat v maticovém tvaru

$$Q(t) = TC = \left[t^3, t^2, t, 1\right] \begin{vmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{vmatrix}. \tag{28}$$

Derivaci $q'(t)$ získáme derivací vektoru *T*

$$q'(t) = \frac{d}{dt} q(t) = \frac{d}{dt} TC = \left[3t^2, 2t, 1, 0\right] C . \tag{29}$$

Konstantní matici *C* můžeme rozepsat do součinu

$$C = MG , \tag{30}$$

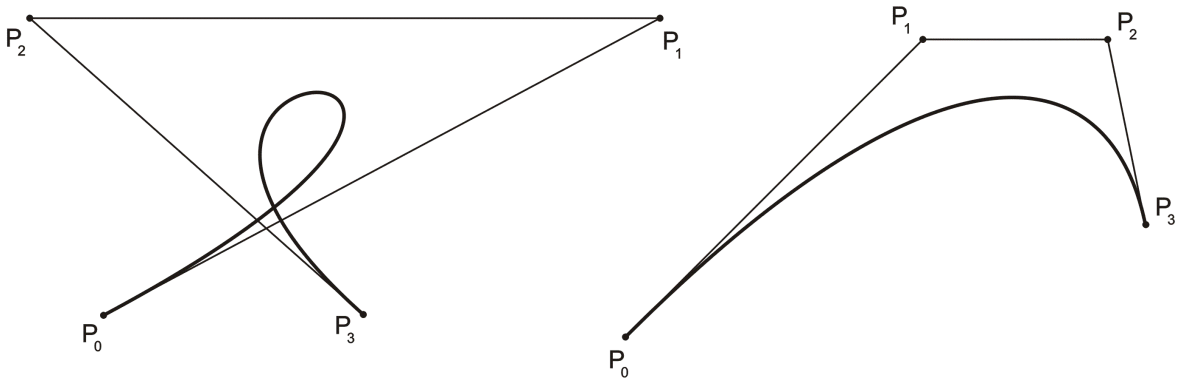
kde matice *M* je typu 4x4 a nazývá se bázová matice. Čtyřprvkový vektor *G* se nazývá geometrický vektor. Geometrický vektor reprezentuje vliv vnějších parametrů. Obsahuje řídicí body, nebo řídicí body a tečné vektory atp. Bázová matice, která je dána použitou metodou,pak určuje výpočet křivky podle vztahu

$$Q(t) = \left[x(t), y(t), z(t)\right] = \left[t^3, t^2, t, 1\right] \begin{vmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{vmatrix} \begin{vmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \end{vmatrix} . \tag{31}$$

2.2.1 Beziérovy kubiky

Beziérovy křivky patří v počítačové grafice mezi jedny z nejpoužívanějších typů parametrických křivek. Jelikož manipulace s vektory u Fergusonových kubik je poměrně nenázorná, je Beziérova metoda podstatně populárnější. Beziérovy křivky stupně dvě a tři, tj. kvadriky a kubiky, jsou použity v mnoha aplikacích a technologiích, například v *PostScriptu* a postscriptových fontech, *TrueType* fontech, formátech aplikace *Corel Draw*, *Adobe Illustrator* apod.

Beziérovy kubiky jsou zadány pomocí čtyř řídících bodů. Křivka přitom prochází prvním a posledním řídícím bodem, druhý a třetí bod určují vyklenutí křivky. Křivka tedy druhým a třetím řídícím bodem obecně neprochází.



Obr. 10 – Příklad Bezpérových kubik

Mezi hlavní výhody Beziérových kubik patří jejich intuitivní zadávání a snadné hladké navazování křivek na sebe. Také výpočet bodů, které leží na křivce, je velmi jednoduchý a rychlý. Pomocí Beziérových kubik však nelze přesně modelovat kuželosečky, zejména kruh a elipsu, což omezuje použití těchto křivek. Také nelze k obecné Beziérově kubice vytvořit offsetovou křivku (tj. křivku, která se od zadané křivky nachází v určité vzdálenosti).

Beziérova kubika je určena vztahem

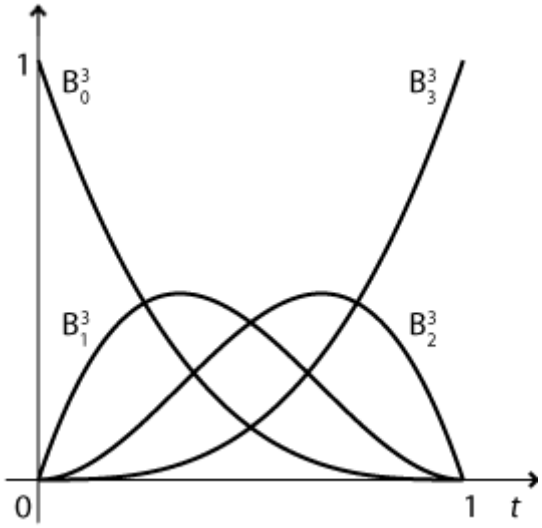
$$P(t) = P_0 B_0(t) + P_1 B_1(t) + P_2 B_2(t) + P_3 B_3(t) = \sum_{i=0}^{3} P_i B_i(t), \tag{32}$$

kde t nabývá hodnot z intervalu $\langle 0,1 \rangle$ a B_0, B_1, B_2, B_3 jsou kubické polynomy tvaru:

$$\begin{aligned} B_0(t) &= (1-t)^3, \\ B_1(t) &= 3t(1-t)^2, \\ B_2(t) &= 3t^2(1-t), \\ B_3(t) &= t^3. \end{aligned} \tag{33}$$

Maticový zápis:

$$Q(t) = T \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix} \tag{34}$$



Obr. 11 - Kubické Bernsteinovy polynomy

Položíme-li $t = 0$ a dosadíme do vztahu (32), je $P(0) = P_0$, analogicky pro $t = 1$ je $P(1) = P_3$. Křivka tedy opět prochází krajními body. Zderivujme $P(t)$ a dostaneme

$$P'(t) = \sum_{i=0}^{3} P_i B'_i(t). \tag{35}$$

Dosazením do tohoto vztahu za $t = 0$ a $t = 1$ vyplyne, že

$$\begin{aligned} P'(0) &= 3(P_0 - P_1), \\ P'(1) &= 3(P_2 - P_3) \end{aligned} \tag{36}$$

Tečné vektory mají vždy směr spojnice dvojice krajních bodů a velikost mají rovnu trojnásobku vzdálenosti bodů.

3 JAVA

Java je objektově orientovaný programovací jazyk, který vyvinula firma Sun Microsystems. Byl představen 23. května 1995.

Java je jedním z nejpoužívanějších programovacích jazyků na světě. Díky své přenositelnosti je používán pro programy, které mají pracovat na různých systémech počínaje čipovými kartami, přes mobilní telefony a různá zabudovaná zařízení, aplikace pro desktop počítače až po rozsáhlé distribuované systémy pracující na řadě spolupracujících počítačů rozprostřené po celém světě. Tyto technologie se jako celek nazývají platforma Java. [7]

3.1 Základní vlastnosti

- **jednoduchý** – jeho syntaxe je zjednodušenou verzí syntaxe jazyka C a C++. Odpadla většina konstrukcí, které způsobovaly programátorům problémy a na druhou stranu přibyla řada užitečných rozšíření.

- **objektově orientovaný** – s výjimkou osmi primitivních datových typů jsou všechny ostatní datové typy objektové.

- **distribuovaný** – je navržen pro podporu aplikací v síti (podporuje různé úrovně síťového spojení, práce se vzdálenými soubory, umožňuje vytvářet distribuované klientské aplikace a servery).

- **interpretovaný** – místo skutečného strojového kódu se vytváří pouze tzv. mezikód. Tento formát je nezávislý na architektuře počítače nebo zařízení. Program pak může pracovat na libovolném počítači nebo zařízení, který má k dispozici interpreter Javy, tzv. virtuální stroj Javy (Java Vitrual Machina).

V pozdějších verzích Javy nebyl mezikód přímo interpretován, ale před prvním svým provedením dynamicky zkompilovány do daného strojového kódu. Tato vlastnost zásadním způsobem zrychlila provádění programů v Javě ale výrazně zpomalila start programů.

V současnosti se převážně používají technologie zvané HotSpot compiler, které mezikód zpočátku interpretují a na základě statistik získaných z této interpretace později provedou překlad často používaných částí do strojového kódu včetně dalších dynamických optimalizací.

- **robustní** – je určen pro psaní vysoce spolehlivého softwaru – z tohoto důvodu neumožňuje některé programátorské konstrukce, které bývají častou příčinou chyb. Veškeré používané proměnné musí mít definovaný svůj datový typ.

Správa paměti je realizována pomocí Garbage collectoru který automaticky vyhledává již nepoužívané části paměti a uvolňuje je pro další použití. To bylo v prvních verzích opět příčinou pomalejšího běhu programů. V posledních verzích běhových prostředí je díky novým algoritmům pro garbage collection a tzv. generační správě tento problém ze značné části eliminován.

- **bezpečný** – má vlastnosti, které chrání počítač v síťovém prostředí, na kterém je program zpracováván, před nebezpečnými operacemi nebo napadením vlastního operačního systému nepřátelským kódem.

- **nezávislý na architektuře** – vytvořená aplikace běží na libovolném operačním systému nebo libovolné architektuře. Ke spuštění programu je potřeba pouze to, aby byl na dané platformě instalován správný virtuální stroj. Podle konkrétní platformy se může přizpůsobit vzhled a chování aplikace.

- **přenositelný** – vedle zmíněné nezávislosti na architektuře je jazyk nezávislý i co se týká vlastností základních datových typů. Přenositelností se však myslí pouze přenášení v rámci jedné platformy Javy. Při přenášení mezi platformami Javy je třeba dát pozor na to, že platforma určená pro jednodušší zařízení nemusí podporovat všechny funkce dostupné na platformě pro složitější zařízení a kromě toho může definovat některé vlastní třídy doplňující nějakou speciální funkčnost nebo nahrazující třídy vyšší platformy, které jsou pro nižší platformu příliš komplikované.

- **výkonný** – přestože se jedná o jazyk interpretovaný, není ztráta výkonu významná, neboť překladače pracují v režimu pomocí kterého se překládá jen ten kód, který je opravdu zapotřebí.

- **víceúlohový** – podporuje zpracování vícevláknových aplikací

- **dynamický** – Java byla navržena pro nasazení ve vyvíjejícím se prostředí. Knihovna může být dynamicky za chodu rozšiřována o nové třídy a funkce, a to jak z externích zdrojů, tak vlastním programem.

- **elegantní** – velice pěkně se v něm pracuje, je snadno čitelný, přímo vyžaduje ošetření výjimek a typovou kontrolu.

3.2 Nevýhody jazyka Java

Proti programovacím jazykům, které provádějí tzv. statickou kompilaci (např. C++), je start programů psaných v Javě pomalejší, protože prostředí musí program nejprve přeložit a potom teprve spustit. Je však možnost využít mechanismů JIT a HotSpot, kdy se často prováděné nebo neefektivní části kódu přeloží do strojového kódu a program se zrychlí. Na zrychlení se také podílí nové přístupy ke správě paměti, viz výše popsaná generační správa paměti.

Další nevýhodou projevující se hlavně u jednodušších programů je větší paměťová náročnost při běhu způsobená nutností mít v paměti celé běhové prostředí.

3.3 Eclipse

Eclipse je open source vývojová platforma, která je známa jako vývojové prostředí určené pro programování v jazyce Java. Flexibilní návrh této platformy dovoluje rozšířit seznam podporovaných programovacích jazyků za pomoci pluginů, například o C++ nebo PHP.

Projekt Eclipse spustila firma IBM a spravuje jej Eclipse Foundation. Pro účely tohoto projektu se mimo jiné vytvořil grafický framework SWT, náhrada za AWT/Swing. Díky SWT je Eclipse, co do vzhledu a chování, stejný jako nativní aplikace operačního systému. Nevýhodou SWT, a tudíž Eclipse, představuje platformová závislost grafického rozhraní. Pro každý cílový operační systém je proto nutné stáhnout rozdílnou verzi instalačního archivu. [2]

II. PRAKTICKÁ ČÁST

4 PROGRAMOVÁ ČÁST

Praktická část této práce je tvořena programem, který využívá výše popsané algoritmy v praxi. Pro vytvoření programu byl zvolen programovací jazyk Java a vývojové prostředí Elipse.

4.1 Struktura programu

Byla vytvořena struktura programu, ve které jsou jednotlivé rasterizační algoritmy odděleny od hlavní části programu. Rozhraní *IRazserizer* implementují všechny rasterizační třídy. Toto rozhraní obsahuje všechny funkce, které jsou potřebné pro rasterizaci uvedených jednoduchých objektů. Pro rasterizaci křivek je od rozhraní *IRasterizer* odvozeno rozhraní *ICurveRasterizer*, které zahrnuje další funkce, které rasterizace křivek vyžadují.

Hlavní část programu je tvořena třídou *Rasterize* a pomocnými třídami *Canvas*, *IntInput* a *Writer*.

Dále program obsahuje třídu *Scene* vytvořenou pro obsluhu ukládání a opětovné načítání připravených scén pomoci funkcí *save()* a *load()*.

Aby bylo možné ukládat proměnné typu *Point*, byla upraveny třídy *DataInputStream* a *DataOutputStream*, které byly odvozeny od standardních tříd *ObjectInputStream* a *ObjectOutputStream*.

4.2 Popis jednotlivých tříd

4.2.1 Rasterize

Jak již bylo uvedeno, tato třída je třídou hlavní. Slouží pro vytvoření hlavního okna, plochy pro data, plátna pro rasterizaci a ovládací prvky. Dále obsahuje dva časovače. První slouží pro překreslování scény. Pomocí druhého je docíleno zpomalení průběhu rasterizačních algoritmů. Také jsou v ní ošetřeny akce provedené uživatelem a spouštění jednotlivých algoritmů.

4.2.2 Canvas

Třída *Canvas* slouží pro obsluhu grafického výstupu programu. Obsahuje funkce pro vykreslení mřížky a osového kříže, dále funkce pro přepočet souřadnic zadaných a

vykreslovaných bodů a hlavně funkci zabezpečující vykreslování všech vypočtených bodů. Tyto body jsou uloženy ve vektoru a jsou postupně vykreslovány jako čtverce reprezentující pixely. Jejich velikost je určena velikostí mřížky (*size*).

4.2.3 IntInput

Slouží pro vytváření vstupního okna, pomocí kterého jsou nastavovány různé číselné parametry programu. Parametry jsou omezeny minimální a maximální hodnotou aby nedocházelo k nežádoucím stavům.

Při vytváření je zadán popis okna, původní hodnota a maximální a minimální hodnota.

Pokud je stlačeno tlačítko *OK* návratová hodnota je rovna číslu zadanému v textovém poli. Při jakémkoliv jiném způsobu zavření okna hodí funkce *getIntInput()* výjimku, která informuje program o zrušení operace. Je-li zadán znak, nebo číslo mimo daný rozsah, vypíše se chybová zpráva s maximální a minimální hodnotou zadávané proměnné.

4.2.4 Writer

Pomocí třídy *Writer* je obsluhován datový výstup programu. Díky podpoře *html* a *css* v objektech *JTextPane* je použito pro formátování výstupu kaskádových stylů.

V proměnné *beginText* jsou uložený styly, které nastavují vzhled zobrazovaných dat. Velikost písma je implicitně nastavena v proměnné *fontSize* na velikost 10. Tuto velikost lze v průběhu měnit pomocí okna *IntInput*, které je voláno ze třídy *Rasterize*.

4.2.5 Scene

Třída *Scene* je vytvořena pro ukládání a načítání scén vytvořených v tomto programu. Pro tento účel obsahuje funkce *save()* a *load()*.

Při ukládání je pomocí komponenty *JFileChooser* vybrán, popřípadě vytvořen soubor do kterého jsou následně uloženy parametry scény (zoom, rychlost animace atp.) a instance vytvořené rasterizační třídy pomocí serializace. Uloženy jsou pouze potřebné data pro vytvoření objektu. Data jsou ukládána v pevně daném pořadí.

Načítání také využívá *JFileChooser* pro výběr souboru. Poté jsou ve stejném pořadí v jakém byla data uložena ze souboru načítána a přiřazována hodnotám scény. Nakonec

jsou zavolány funkce pro smazání textu (*clearText()*), všech bodu (*points.clear()*) a pro výpočet základních parametrů načteného objektu.

4.2.6 IRasterizer

Tato třída slouží jako rozhraní (interface) pro rasterizační algoritmy, aby bylo dosaženo jednotného stylu realizace rasterizačních algoritmů a také z důvodu jednodušší implementace těchto algoritmů do programu.

Třída obsahuje funkce:

- add(Point p)
  - přidává pozice body potřebné pro sestrojení daného objektu
  - je volána třídou *Rasterize* při kliknutí na plátno (*Canvas*), pokud je počet zadaných bodů menší než je potřebný počet bodu pro sestrojení daného objektu
- draw(Graphics g)
  - vykresluje daný objekt
  - je volána funkcí *paint(Graphics g)* při překreslování scény
- rasterizeMain()
  - vypočte a vypíše inicializační hodnoty daného algoritmu
  - proběhne pouze pokud je zadán potřebný počet bodů, pokud je rasterizace restartována, nebo pokud je načtena scéna
- rasterizeNext()
  - vypočte další bod objektu, pokud nebyla rasterizace dokončena
  - funkce provede výpočet dalšího bodu pokud je stlačeno tlačítko step
  - při animaci je funkce prováděna automaticky po daných časových intervalech časovače

- rasterizeRestart()
    - smaže všechny vypočtené body objektu a vypočte inicializační hodnoty algoritmu
    - je vykonána po stlačení tlačítka restart, po editaci zadaných bodů vytvořeného objektu a při načtení scény
- isEnterFinished()
    - vrací hodnotu proměnné *enterFinished*, která určuje zda byly zadány všechny body objektu
- getMaxPointCount()
    - vrací počet potřebných bodů pro sestrojení daného objektu
- getPoint(int i)
    - vrací bod z pole uživatelem zadaných bodů
    - je užívána při editaci vytvořených objektů
- setMouseSelectedPoint (int i)
    - nastavuje index vybraného bodu do proměnné *mouseSelectedPoint* pomocí kterého se provádí editace objektu
- getMouseSelectedPoint ()
    - vrací hodnotu bodu který uživatel vybral pro editaci
    - pokud má zápornou hodnotu nebyl vybrán žádný bod
- movePoint (Point p)
    - posune bod který je určen proměnnou *mouseSelectedPoint* na nově určenou pozici podle kritérií, která jsou určena pro každý objekt

4.2.7 IRasterizerDetail

Toto rozhraní je pouze rozšířením rozhraní *IRasterizer* o funkce které jsou využívány k rasterizaci křivek. Jedná se o funkce *setStep(int i)* a *getStep()*, pomocí kterých je

nastavována a získávána úroveň rozdělení křivky (*step*). Jedná se o počet přímek, na které je daná křivka interpolována.

4.2.8 RasterizerBasicCircle

Třída slouží k rasterizaci kružnice pomocí polárních souřadnic. Pro sestrojení kružnice a následnou rasterizaci je potřeba zadat dva body. První bod určuje polohu středu, pomocí druhého bodu je vypočítán poloměr.

Pokud jsou všechny body zadány je zavolána funkce *rasterizeMain()*, která vypočte poloměr *r*. Pomocí poloměru je určen krok (*step*). Je nastavena pomocná proměnná *a*, která slouží jako parametr rovnic. Dále jsou vypsány základní vztahy pro tento typ rasterizace a pomocí vzorců (13) a (14) které jsou uvedeny v teoretické části je vypočten první bod. Tento bod je osmkrát přidán do vektoru vykreslovaných bodů, pokaždé však se změněnými znaménky, protože u kružnice je vypočítáván pouze jeden oktan a další jsou odvozeny.

Funkce *rasterizeNext()* pak při volání vypočítává a vypisuje další body kružnice dokud $a < \frac{\pi}{4}$. Každý vypočtený bod je použit osmkrát, pokaždé pro jiný oktan.

4.2.9 RasterizerBezier

Tato třída využívá algoritmu pro Beziérovi kubiky. Pro vytvoření Beziérovi kubiky jsou potřebné čtyři body pomocí kterých je křivka sestrojena. Rasterizace je zde rozdělena do dvou hlavních částí. V první části jsou vypočítávány přímky, které interpolují zadanou křivku. Tyto přímky jsou poté rasterizovány pomocí Bresenhamova algoritmu pro úsečku, který tvoří druhou část.

Provedením funkce *rasterizeMain()* je nastavena pomocná proměnná *actualStep* na velikost kroku, který může být uživatelsky změněn. Pomocí tohoto kroku je vypočtena první přímka. Jako první bod je nastaven bod *points[0]*, druhý bod přímky je vypočten pomocí vzorců (32) a (33). Tyto vzorce jsou také vypsány do panelu *data*. Proměnné *lineFinished* a *lineSetup* jsou nastaveny na hodnotu *false*. Hodnota proměnné *lineFinished* určuje zda má být vypočtena další úsečka. Proměnná *lineSetup* udává zda pro danou přímku byla provedena inicializace pomocí funkce *rasterizeLineMain()*.

Funkce *rasterizeNext()* se poté stará o volání funkce *rasterizeLineMain()*, vykreslování dalších bodu úsečky pomocí funkce *rasterizeLineNext()*, také o výpočty dalších úseček.

4.2.10 RasterizerBresenhamCircle

Pro rasterizaci kružnice pomocí Bresenhamova algoritmu byla napsána třída *RasterizerBresenhamCircle*. Stejně jako třída *RasterizerBasicCircle* potřebuje pro sestrojení a výpočet kružnice 2 body.

Funkcí *rasterizeMain()* jsou nastaveny počáteční hodnoty pomocných proměnných a přidány do vektoru bodu čtyři body kružnice a jsou vypsány vzorce do panelu *data*.

Další body jsou vypočítány pomocí funkce *rasterizeNext()* dokud platí podmínka $actualPoint.x+1<actualPoint.y$. Pokud tato podmínka není splněna znamená to, že rasterizace byla dokončena.

4.2.11 RasterizerBresenhamEllipse

Pomocí této třídy je aplikován algoritmus pro rasterizaci elipsy pomocí Bresenhamova algoritmu.

Pro vytvoření elipsy je potřeba zadat 3 body. Prvním zadaným bodem je střed, další dva body jsou poloosy. U těch je třeba rozhodnout o které poloosy se jedná. Výběr je proveden pomocí diference souřadnic od středu pokud je $\Delta x > \Delta y$, potom daný bod patří poloose *x*, v opačném případě se jedná o poloosu *y*.

Po zadání všech bodů je provedena funkce *rasterizeMain()*. Ta vypočte velikosti poloos a nastaví počáteční hodnoty proměnných. Je zde také přidán první bod. Ten je přidán do vektoru dvakrát, díky symetričnosti elipsy. Nakonec jsou vypsány vzorce a parametry rasterizované elipsy.

Funkce *rasterizeNext()* se dále stará o výpočet dalších bodů a to ve dvou fázích. v první fázi se vypočítávají body s řídící osou *x* dokud platí podmínka $a^2(y_i - 0{,}5) > b^2(x_i - 1)$. Po změně řídící osy se provádí výpočty dokud je $ActualPoint.y > 0$.

4.2.12 RasterizerBresenhamLine

Slouží pro rasterizaci úsečky Bresenhamovým algoritmem. Pro sestrojení jsou potřeba zadat 2 body pomocí funkce *add()*.

Po zadání těchto bodů je zavolána funkce *rasterizeMain()* která vypočte parametry potřebné k rasterizaci. Dále rozhodne zda je potřeba zadané body přehodit. Body je nutné přehodit pokud je hodnota *x* prvního bodu větší než hodnota bodu druhého. Algoritmus je totiž sestrojen pro rasterizaci zleva doprava.

Podle proměnné *k* se rozhodne o hlavní ose, která určí větvení funkce. Poté je přidán do vektoru vykreslovaných bodů první bod (*points[0]*). Následuje vypsání vzorců do panelu *data*.

Následující body jsou vypočítávány podle vzorců které jsou k dané řídící ose sestrojeny. Tyto vzorce jsou obsaženy ve funkci *rasterizeNext()*. Rasterizace probíhá až do chvíle kdy je vypočtený bod roven druhému zadanému bodu.

4.2.13 RasterizerDDALine

Třída byla vytvořena pro rasterizaci úsečky pomocí DDA algoritmu.

Opět je potřeba zadat 2 body pro sestrojení a zahájení rasterizace. Pro rasterizaci jsou využívány vztahy (4) a (5).

Algoritmu je sestrojen pro rasterizaci úsečky zleva doprava. O případné přehození bodů se stará funkce *rasterizeMain()*, která dále vypočte hodnoty potřebné pro vypočet bodů a vypíše vzorce do panelu *data*.

Při rasterizaci může nastat několik situací. Ty jsou ovlivněny směrnicí *m*. Podle toho je program větven jak ve funkci *rasterizeMain()*, tak v *rasterizeNext()*.

Body jsou vypočítávány pomocí funkce *rasterizeNext()*, dokud není dosaženo koncového bodu úsečky (*points[1]*).

4.2.14 RasterizerFerguson

Fergusonovu kubiku rasterizuje třída *RasterizerFerguson*. Třída využívá vzorců uvedených v teoretické části. Konkrétně se jedná o vzorce (24) a (25).

K sestrojení kubiky jsou potřebné dvě dvojce bodů. Každá dvojce určuje vrchol a vektor pomocí kterého je určeno vyklenutí křivky.

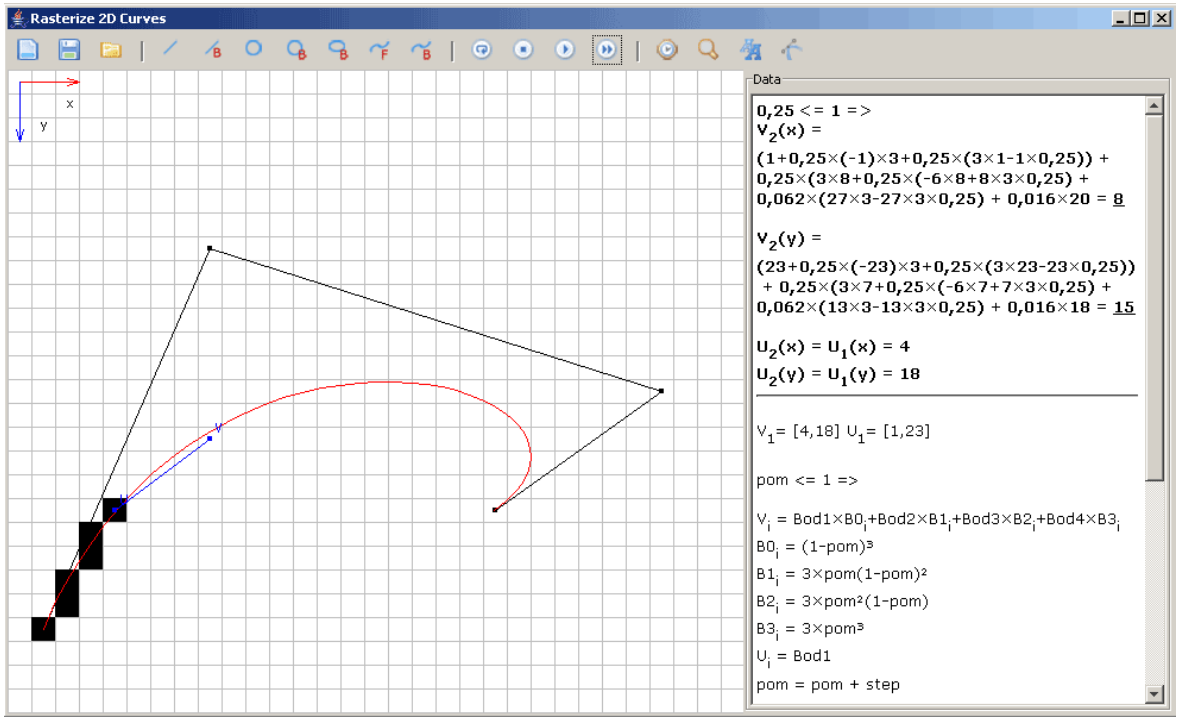
Proces rasterizace je obdobný jako u rasterizace Beziérovy kubiky. Po zadání všech bodů je volána funkce *rasterizeMain()*. Ta vypočte body první přímky a zobrazí užité vzorce do panelu *data*.

Funkce *rasterizeNext()* se taká zde stará o volání funkcí pro rasterizaci vypočtené úsečky, její inicializaci a o vypočet nové úsečky pomocí zadaných vztahů.

5 UŽIVATELSKÉ ROZHRANÍ

Jelikož je program vytvořen v programovacím jazyku Java je tento program *multiplatformní*. To znamená ze je možné spustit ho v jakémkoliv operačním systému. Po menších úpravách je možné tento program vložit také jako *applet* na webové stránky.

Ve Windows se program spouští pomocí souboru *start.bat*. Po spuštění programu se otevře okno programu. V jeho horní části se nacházejí ikony. Zleva ikony pro novou scénu, uložení a otevření scény. Dále ikony pro výběr rasterizačního algoritmu, ovládání animace rasterizace a nakonec ikony pro nastavení parametrů. Vpravo je umístněn panel pro výpis dat a vzorců vybraných rasterizačních algoritmů. Největší část okna zabírá plocha pro vykreslování a následnou rasterizaci objektů.



Obr. 12 – Uživatelské rozhraní

5.1 Parametry

Uživatelsky je možné nastavovat některé parametry programu v určeném rozsahu:

- **rychlost animace**
  - nastaví dobu vykreslování jednoho bodu při automatické rasterizaci
  - doba se nastavuje v milisekundách v rozsahu od 50ms do 10 000ms

- **velikosti mřížky**
  - určuje velikost mřížky která reprezentuje velikost pixelů
  - je možné jej měnit také pomocí kolečka myši
- **velikost písma**
  - nastavuje velikost písma zobrazovaných dat
- **úroveň rozdělení křivky**
  - je aktivní pouze při rasterizaci křivek
  - určuje počet úseček na které je křivka rozdělena

5.2 Ovládání

Program se ovládá pomocí myši. Po zvolení ikony příslušného rasterizačního algoritmu musí uživatel kliknutím na plátno zadat pozici bodů potřebných pro sestrojení vybraného objektu. Při zadávání bodů se vpravo vypisuje jejich pozice. Po dokončení zadávací sekvence se v poli *data* vypíšou obecné vzorce pro daný rasterizační algoritmus. Po spuštění rasterizace se postupně vypisují aktuální vzorce a hodnoty vypočteného bodu.

Vytvořené objekty je možno upravovat tažením vyznačených bodů. Při změně tvaru objektu se rasterizace automaticky restartuje.

Rasterizaci objektu je možné provádět v podstatě dvěma způsoby. Prvním způsobem je automatická animace. Po stisknutí ikony *play* je každých 500ms vypočten a zobrazen jeden pixel. Tuto dobu je možné nastavit pomocí tlačítka *rychlost animace*. Druhým způsobem je manuální posouvání animace pomocí tlačítka *step*.

ZÁVĚR

Bakalářská práce se zabývá rasterizací 2D grafických objektů. Teoretická část pojednává o křivkách a jednoduchých objektech. Popisuje je a uvádí některé z možných rasterizačních algoritmů.

Po seznámení s rasterizačními algoritmy pro úsečku, kružnici a elipsu byly vybrány nejpoužívanější způsoby rasterizace daných objektů. Byla popsána jejich funkčnost a efektivita. Bresenhamův algoritmus byl popsán u všech jednoduchých objektů z důvodu využití celočíselné aritmetiky, která je výpočetně efektivnější než aritmetika reálných čísel.

V další kapitole byli charakterizovány zástupci algoritmů pro metody interpolace a aproximace křivek. Po domluvě byly zvoleny Beziérovy a Fergusonovy kubiky, které jsou probírány i na přednáškách předmětu Počítačová grafika. Podrobně bylo popsáno jejich sestrojování a vliv jednotlivých bodů na vzhled křivky.

Praktickou částí je aplikace v jazyce Java, která využívá všechny algoritmy popsané v teoretické části. Software je hlavní výstup bakalářské práce a bude sloužit jako výuková pomůcka pro potřeby kabinetu Aplikované informatiky FAI-UTB Zlín. Zdrojový kód aplikace je multipatformní díky použití programovacího jazyku Java. Program je možné jednoduše rozšiřovat o další rasterizační algoritmy dopsáním dané třídy obsahující matematický aparát pro rasterizaci nových objektů.

Prezentace algoritmů na přednáškách předmětu Počítačová grafika je umožněna tímto programem za použití předem připravených scén a nastavení, které jsou pomocí programu uloženy do externích souborů.

CONCLUSION

This Bachelor thesis is engaged in rasterization of 2D graphic objects. Theoretic part discusses curves and primitive objects. It describes and shows some of possible rasterization algorithms.

After a description of rasterization algorithms for line, circle and ellipse, there was chosen most used ways of rasterization for these objects. Their functionality and efficiency was described. Bresenham algorithm was described for all simple objects because of using integer arithmetic which is more computationally efficient than the arithmetic of real numbers.

Reprezentation of algorithms for methods of interpolation and approximation of a curve were characterized in the next chapter. Bezier and Ferguson cubic curves were chosen after consultations, because they are explained at the lessons of the Computer graphic. Their construction and the effect of particular points to the shape of the curve were also described in this chapter.

Practical part is formed by an application in Java language, which uses all the algorithms described in the theoretical part. The software is the main output of this thesis and it will serve as an educational instrument for the Department of Applied Informatics at the FAI-UTB Zlín. The source code of the application is multiplatform because of using the Java language. Program is easy to extend by other rasterization algorithms. They can be added as a class implementing a mathematical theorem for rasterization of new objects.

This program allows the presentation of the mentioned algorithms at the lessons of the Computer graphics, using prepared scenes and setups stored in external files.

SEZNAM POUŽITÉ LITERATURY

[1] DRDLA, J. *Geometrické modelování křivek a ploch*. UP, Olomouc, 2001

[2] *Eclipse* [online]. 2007 [cit. 2007-04-28]. Dostupný z WWW: http://www.eclipse.org/

[3] HUDEC, B. *Základy počítačové grafiky*. ČVUT, Praha,2001, ISBN 80-01-02290-0

[4] MARTÍŠEK, D. *Matematické principy grafických systémů*. Brno : Littera, 2002. 296 s., 1 CD-ROM. ISBN 80-85763-19-2

[5] POKORNÝ, P. *Základy počítačové grafiky*. UTB, Zlín, 2004. ISBN 80-7318-161-4

[6] Sochor, J., et al. Algoritmy počítačové grafiky. Skripta ČVUT, Praha 1996

[7] SPELL, B. *Java Programujeme profesionálně*. Bogdan Kiszka. Praha : Computer Press, 2002. 1022 s. ISBN 80-7226-667-5

[8] *The Source for Java Developers* [online]. 1994-2007 [cit. 2007-04-28]. Dostupný z WWW: http: //java.sun.com/

[9] ŽÁRA, J., et al. *Počítačová grafika - principy a algoritmy*. Praha : Grada a.s., 1992. 472 s. ISBN 80-85623-00-5

[10] ŽÁRA, J., et al. *Moderní počítačová grafika*. 2. přeprac. vyd. Brno : Computer Press, 2004. 609 s. ISBN 80-251-0454-0

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

P1 Zdrojový kód programu. Příloha se nachází na přiloženém CD v adresáři *P1_zdrojovy_kod*.

P2 Spustitelný program. Nachází se na přiloženém CD v adresáři *P2_program*.

P3 Scény programu určené pro prezentaci rasterizačních algoritmů. Uložené na CD v adresáři *P3_sceny*.

P4 Dokumentace ke zdrojovému kódu ve formátu *HTML*. Nachází se na přiloženém CD v adresáři *P4_dokumentace*.

P5 Vývojové prostředí Eclipse. Uložené na CD v adresáři *P5_eclipse*.

P6 Java Runtime Environment. Softwarový balík, nutný pro spuštění aplikace. Na přiloženém CD v adresáři *P6_jre*.