UNIVERZITA PARDUBICE
ÚSTAV ELEKTROTECHNIKY A INFORMATIKY

Principy výřezu oblasti v bitmapových grafických editorech.

BAKALÁŘSKÁ PRÁCE

AUTOR PRÁCE: Lubor Mejsnar
VEDOUCÍ PRÁCE: RNDr. Rak Josef
2007

UNIVERSITY OF PARDUBICE
INSTITUTE OF ELECTRICAL ENGINEERING
AND INFORMATICS

Principles of cut out area in bitmap's graphic editors.

BACHELOR WORK

AUTHOR: Lubor Mejsnar
SUPERVISOR: RNDr. Rak Josef
2007

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně Univerzity Pardubice.

V Pardubicích dne 17.5. 2007

Lubor Mejsnar

Poděkování

RNDr. Josefu Rakovi za cenné rady během zpracovávání této práce a za snahu a trpělivost při častých konzultacích na vybrané téma. Ing. Petru Veselému za ochotu pomoci při řešení problémů v praktické části. Také bych rád poděkoval rodičům a všem svým přátelům za pomoc a podporu, které se mě dostalo během studia na této škole.

ABSTRAKT

Cílem této práce je seznámit širší veřejnost s využitím křivek v počítačové grafice, ukázat na výhody použití vektorové grafiky oproti rastrové grafice. Dále ve své práci popisuji rozdílné techniky využití řídících bodů pro křivku, jako je interpolace a aproximace. Část práce také představuje programy, které nám nabízí trh, ať už to jsou programy volně šiřitelné, nebo komerční programy. Uživateli, který hledá vhodný software pro práci s vektorovou grafikou, doporučím z mého pohledu kvalitní grafický editor.

V praktické části představím svůj program pro vykreslování křivek, výběrem jakékoli křivky ve výkresu a úpravu řídících bodů vybrané křivky. Velký důraz je kladen na implementaci vhodné datové struktury pro uchovávání křivky resp. celého výkresu. Uživatel si může vybrat z následujících křivek: lagrangeuv polynom, bézierova kubika, fergusonova křivka, a kubický Coonsův B-splajn. Program se zabývá problémem výřezu oblasti ohraničenou křivkami. Tento problém je představen a následně je ukázána možnost vyřešení tohoto problému krok po kroku skrze algoritmus.

Obsah

1. Úvod do počítačové grafiky

Počítačová grafika je obor *informatiky*, který používá počítače na vyjádření umělých snímků (tzv. *rendering*) a také na úpravu zobrazitelných informací, snímaných z reálného světa (př. digitální fotografie a jejich úprava). Z počátku počítačová grafika byla vyvíjena jen pro akademické zájmy podporované vládou a armádou. Později začala počítačová grafika pronikat i do filmu a televize, protože vývoj a technika začala být konkurence schopná s tradičními speciálními efekty a s animačními technikami používanými do té doby, tudíž i firmy, které v tom viděly příležitost ke zviditelnění a k prosazení v silné konkurenci, začaly přispívat na vývoj tohoto odvětví. Pro informaci jaké to jsou efekty, tak první film, kde se objevila počítačová grafika je považován: *2001 Vesmírná odysea*, který *byl* natočený v roce 1968.

Počítačová grafika se dělí v základu na dvě části 2D grafika a 3D grafika. U 2D grafiky je uchovávání dat prováděno v dvourozměrném systému souřadnic. Tzn., že každý bod má právě dvě souřadnice. 3D počítačová grafika, která je navíc obohacena o třetí rozměr. Pozice je tedy uchovávána třemi souřadnicemi.

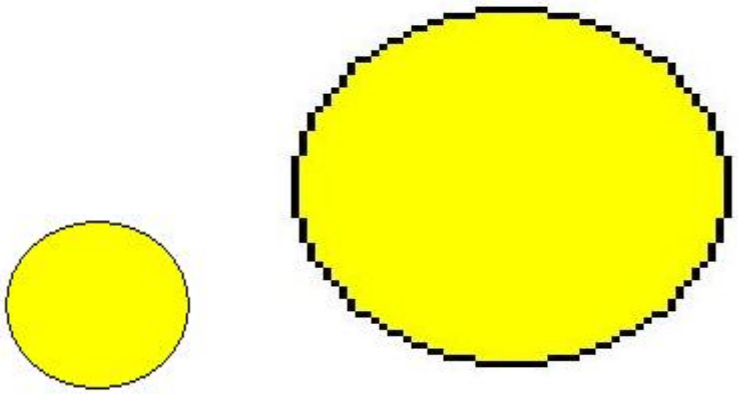
Existují dva různé přístupy k ukládání obrázku v počítačové grafice a to rastrová nebo vektorová grafika.

1.1 Rastrová grafika

Rastrová grafika, nebo-li bitmapa, je pravidelná síť pixelů organizovaná jako dvourozměrná matice bodů. Každý pixel nese specifické informace, o kvalitě obrázku rozhoduje především rozlišení a barevná hloubka.

Barevná hloubka popisuje počet bitu použitých k popisu určité barvy. Větší počet bitu umožňuje použít větší škálu barev, ale také s tím stoupá paměťová náročnost na obrázek či video.

Obrázek v rastrové grafice má omezenou velikost, to je udáváno pomocí rozlišení např. 800x600. Mezi největší nedostatky u rastrového obrázku patří jeho pevná velikost. A s tím spojená nemožnost změny velikosti obrázku, aniž by se nezhoršila obrazová kvalita obrázku. Například při velkém zvětšení je patrná mřížka. Naproti tomu výhoda rastrové grafiky je jeho snadné pořízení třeba prostřednictvím fotoaparátu, snadné přenesení do počítače a velice jednoduché úpravy tohoto obrázku.



Obrázek č. 1.1 a 1.2 kruh (vlevo) pomocí bitmapy, kruh (vpravo) 8x zvětšen

1.2 Vektorová grafika

Vektorová grafika označuje způsob uchovávání a ukládání obrazu. Obraz je reprezentován pomocí základních geometrických objektů, jako jsou bod, přímka, křivka, polygon, atd. Hlavní výhoda uchovávání obrazu ve vektorovém formátu spočívá v možnosti jakkoli měnit velikost obrazu a nikdy se nezmění kvalita obrazu.

Obrázek č. 1.3 a) originální obrázek, b) obrázek zvětšen jako vektorový obrázek, c) obrázek zvětšen jako rastrový obrázek, Zdroj:http://cs.wikipedia.org/

K výhodám určitě patří velikost souboru vektorového obrazu, která je o poznání menší než obrázek uložený v rastrové kvalitě. Samozřejmě zde musíme brát v potaz kompresi při ukládání v rastrovém formátu. Vektorový formát má i své nevýhody, a to je *pořízení vektorového obrázku*, tento obrázek musíme vyrobit v některém z editorů jako např. Adobe Illustrator, Corel Draw, Inkscape. Protože dnešní technologie, zatím neumožňuje snadné převedení obrázku na vektory. Pro zajímavost lidské oko pracuje jako bitmapová grafika, neboť sítnice funguje jako bitmapový rastr, ale mozek zpracovává obraz jako vektorová grafika.

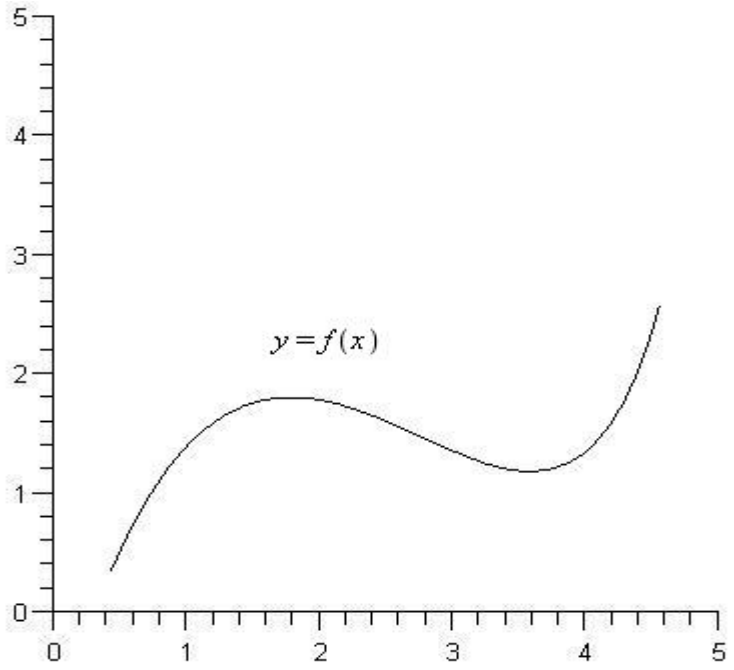
Využití najde vektorová grafika například v typografickém písmu, počítačových animací, ale nyní je velká snaha dosavadní webovou rastrovou grafiku převést na vektorovou grafiku. Velké zastání má vektorová grafika také v CAD systémech, protože je velmi často vyžadováno přesné zaoblení hran. Tento požadavek se může stát velmi složitý v případě, že architekt či designér nezná správné nástroje. K zaoblení, vyhlazení nebo obyčejné spojení více bodů hladkou křivkou bez ostrých hran se využívají křivky. A

tyto možnosti se pokusím objasnit v další části a také ukázat na vlastním demonstračním programu.

2. Vyjádření křivek

Křivky v počítačové grafice se využívají v různých odvětvích, ale vždy je křivka vyjádřena nějakou rovnicí. Existují v podstatě 3 způsoby vyjádření křivky:
a)explicitně
b)implicitně
c)parametricky

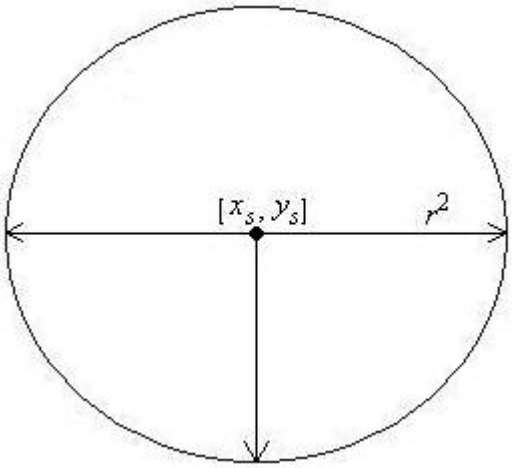
Explicitní zadání křivky, např. *y=f(x)* je pro počítačovou grafiku velmi nevhodné, protože toto zadání můžeme použít pouze pro křivky, které jsou zároveň funkcemi – tzn. pro jednu hodnotu *x* existuje právě jedna funkční hodnota *f(x)*.



Obrázek č. 1.4: Explicitní zadání křivky

Implicitní zadání křivky má tvar *f(x,y)=0*, toto zadání také není vhodné pro práci s křivkami z toho důvodu, že nám neumožňuje v obecnějších případech postupný výpočet křivky. Tento způsob výpočtu je vhodný například pro aplikace, které se zabývají zjištění průsečíku s jinou křivkou nebo pro práci s oblastí vymezenou touto křivkou. Např. rovnice kružnice:

$$f(x,y) = (x - x_s)^2 - (y - y_s)^2 - r^2 = 0$$



Obrázek č. 1.5: Kružnice

Parametrické vyjádření křivky je nejpoužívanější způsob v počítačové grafice. Křivku budeme chápat fyzikálně, jako dráhu pohybujícího se bodu, jehož souřadnice jsou funkcemi parametru *t*. *x=f(t , y= f(t) , z= f(t).* Parametr t je z intervalu *t < tmin , tmax >,* nejčastěji je t zvoleno z rozsahu <0,1>.

2.1 Bodová rovnice

Bodová rovnice křivky je určena výše zmiňovanými funkcemi a její tvar je následující:

$$Q(t) = [x(t), y(t), z(t)]$$

tzn. pro každou souřadnici bodu je jedna parametrická rovnice.

2.2 Vektorová rovnice

Vektorová rovnice má tvar:

$$q(t) = (x(t), y(t), z(t))$$

vektor *q(t)= Q(t) – [0,0,0]* se nazývá polohový vektor, jeho velikost je rovna vzdálenosti *Q(t)* od počátku.

Parametr *t* je chápaný jako čas, tedy pro daný čas můžeme zjistit souřadnice bodu pohybujícího se po křivce. Proto je parametrický zápis křivky velmi vhodný při využití v počítačové grafice, jelikož můžeme zobrazit jakoukoli křivku, včetně případu, když křivka prochází ve více časových okamžicích jedním bodem, např. v prostoru se může křížit, uzavřít apod.

Tečný vektor v bodě $Q(t_0)$ je určen derivacemi parametricky vyjádřené křivky po složkách ve tvaru:

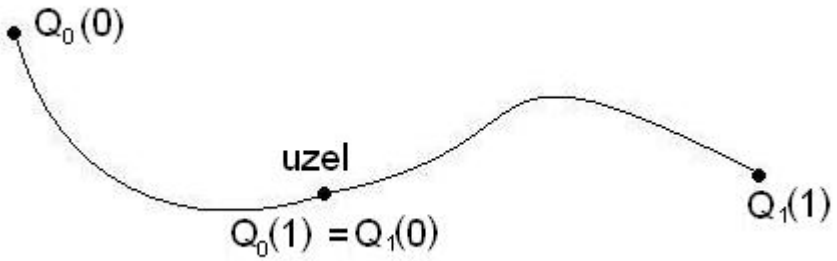
$$\overrightarrow{q'}(t_0) = (x'(t_0), y'(t_0), z'(t_0)) = \frac{\mathrm{d}x(t_0)}{\mathrm{d}t}, \frac{\mathrm{d}y(t_0)}{\mathrm{d}t}, \frac{\mathrm{d}z(t_0)}{\mathrm{d}t}$$

rovnice tečny, tj. přímky, která se křivky v tomto bodě dotýká, se vypočítá z tečného vektoru a bodu na křivce jako

$$P(m) = Q(x_s) + m \cdot \overrightarrow{q'}(t_o)$$

2.3 Spojitost křivek

K výhodám parametrického vyjádření křivky určitě patří snadný výpočet jejich tečných vektorů. Ty jsou důležité při navazování jednotlivých křivek (segmentů) za účelem vytvoření jediné složené křivky. Při navazování jednotlivých segmentů sledujeme *spojitost* (continuitu) v bodu, nebo-li navázání v uzlu. Tj. uzel, ve kterém končí jeden segment složené křivky a začíná další.



Obrázek č. 1.6 spojení 2 křivek

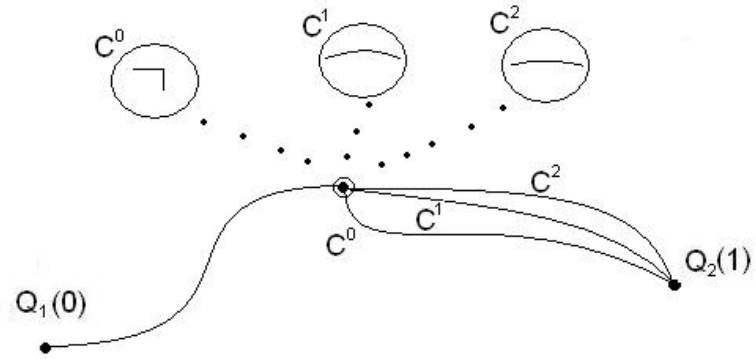
2.3.1 Navazování křivek

Při spojení dvou křivek se klade největší důraz na hladkost navázání. Křivka musí volně přecházet v druhou, aniž by se změnil její tvar. Tento požadavek může být velmi složitý při navazování dvou odlišných křivek. Například jedna interpolační a druhá aproximační. Jistě mnohem lehčí je navázat segmenty jedné křivky. Lehkost navázání vždy záleží na použité křivce a jejích vlastnostech.

2.3.2 Parametrická spojitost stupně n

Říkáme, že výsledná křivka je spojitá, pokud je spojitá ve všech svých bodech a tedy hlavně v navazovacích bodech. Křivka je hladká, pokud jsou, ve všech jejích bodech, spojité i její první derivace. Pro vyšší derivace

říkáme, že křivka má spojitost druhého, třetího a obecně n-tého řádu. Říkáme, že křivka *Q(t)* je třídy C^n, má-li ve všech bodech spojité derivace do řádu *n*.



Obrázek č. 1.7 Spojitost C^0, C^1, C^2

a)spojení třídy C^0:

- pokud je koncový bod prvního segmentu počátečním bodem segmentu druhého, v uzlu se může měnit skokem směr, rychlost, zrychlení

b)spojení třídy C^1:

- v případě, že tečný vektor v koncovém bodě segmentu Q_1 je roven tečnému vektoru v jeho počátečním bodě druhého segmentu. V tomto případě se nemůže měnit směr pohybu a velikost rychlosti, ale zrychlení ano. Jednoduše řečeno v navazujícím bodě musí mít tečny stejný směr, resp. spojovací uzel musí mít shodnou tečnu.

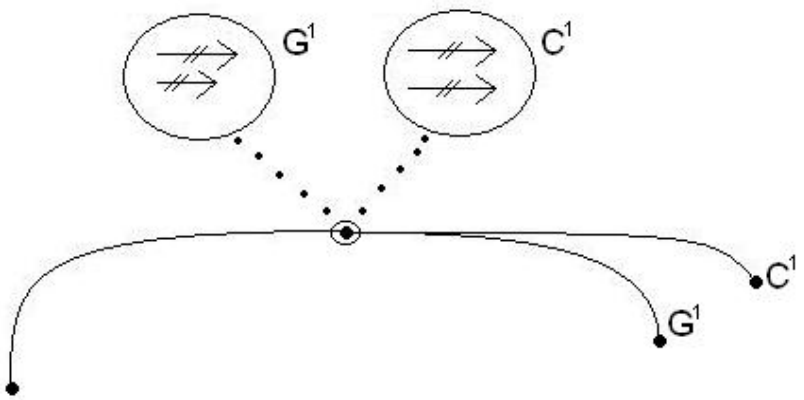
c)spojení třídy C^2:

- je požadována rovnost vektoru první a druhé derivace, výsledkem druhé derivace je „poloměr kružnice“, která v daném bodě kopíruje křivku, v tomto bodě se nemůže měnit směr pohybu, velikost rychlosti zrychlení.

Čím vyšší spojitost je požadována, tím delší „dobu“ (parametr t) se oba segmenty křivky přimykají ke stejnému směru

2.3.3 Geometrická spojitost

Označujeme G^n. Nejčastější používané geometrické spojitosti jsou G^0 a G^1. Dva segmenty křivky Q(t) jsou G^0 spojité, pokud je koncový bod Q_1 totožný s počátečním bodem Q_2. Dva segmenty jsou G^1 spojité, pokud jsou G^0 spojité a současně tečné vektory $q'_1(1)$ segmentu Q_1 a $q'_2(0)$ segmentu Q_2 jsou souhlasně kolineární (rovnoběžné).



Obrázek č. 1.8 Geometrická a parametrická spojitost. Přiblížení ukazuje tečné vektory.

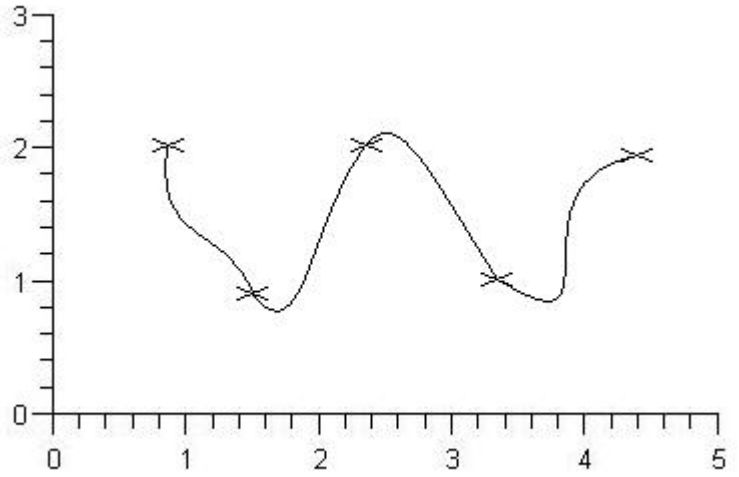
Geometrická spojitost zaručuje totožnost tečen (nikoli tečných vektorů). Pohybující se bod v uzlu nemůže změnit skokem směr, ale může změnit skokem rychlost, křivka je vizuálně hladká. Ze subjektivního hlediska zaručuje spojení $\mathbf{G^1}$ „skoro stejnou“ hladkost jako C^1, z hlediska použití bývá daleko snazší zaručit spojitost $\mathbf{G^1}$ nežli C^1. Spojitost C^1 implikuje $\mathbf{G^1}$ s výjimkou jediného případu, kdy vektor rychlosti v místě spojení dvou segmentů je (0,0,0). Obrácená věta neplatí, neboť geometrická spojitost nepostihuje rychlosti a zrychlení pohybu.

2.4 Modelování křivek

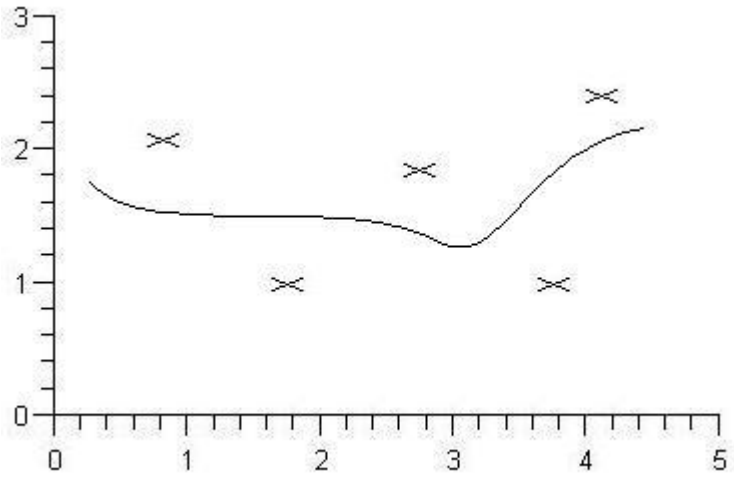
Základním prvkem v teorii křivek v počítačové grafice jsou křivky polynomiální. ($\boldsymbol{Q}_n(\boldsymbol{t}) = \boldsymbol{a}_0 + \boldsymbol{a}_1\boldsymbol{t} + ... + \boldsymbol{a}_n\,\boldsymbol{t}^{\,n}$). Mezi jejich největší výhody patří jejich snadné vyčíslení a to že jsou jednoduše diferencovatelné. Z polynomiálních křivek můžeme skládat křivky *po částech polynomiální,* to jsou křivky, jejichž segmenty jsou polynomiálními křivkami. Nejpoužívanější jsou křivky třetího stupně – kubiky. Jejich výpočet je nenáročný, zajišťuje širokou škálu tvarů a je u nich možné zaručit spojitost C^2, což je často požadováno při modelování v CAD systémech.

Při modelování křivek je nejprve nutné definovat řídící polygon, nebo-li řídící body, dále matematický aparát určí z jejich polohy průběh křivky. Některé metody umožňují zadávání křivek též pomocí tečných vektorů, je možné zaručit spojitost a hladkost navázání aj.

V zásadě existují dva druhy interpretace křivky pomocí řídících bodů. A to interpolace a aproximace. Při interpolaci křivka prochází všemi námi zadanými body a naproti tomu při aproximaci je řídícími body určen její průběh, ale všemi řídícími body nemusí křivka vůbec procházet.



Obrázek č. 1.9 Interpolační křivka

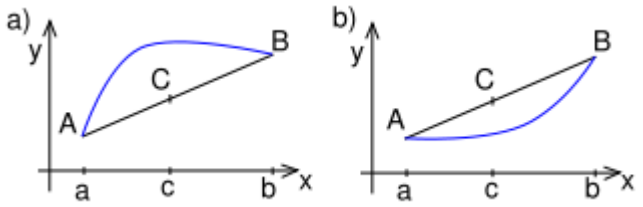


Obrázek č. 2 Aproximační křivka

2.4.1 Význam první a druhé derivace u křivek

Pokud známe funkci popisující nějaký děj, v našem případě průběh křivky, potom první derivace této funkce popisuje okamžitou změnu tohoto děje, tedy okamžitou rychlost děje. Po derivování funkce v určitém čase t známe přírůstek na ose y oproti t-1, čas u nás představuje osu x.

Díky druhé derivaci, to znamená derivaci již derivované funkce, můžeme zjistit průběh funkce, aniž bychom předtím věděli, jak bude křivka vypadat. Pomocí druhé derivace můžeme zjistit konvexnost či konkávnost křivky v daném bodě. Konvexnost a konkávnost je označení pro změnu růstu funkce, jinak řečeno zakřivení jejího grafu.



Obrázek č. 2.1 a) konkávní funkce, b) konvexní funkce
Zdroj:http://cs.wikipedia.org/

2.5 Interpolace

Na intervalu I je dána uspořádaná n-tice bodů, nazýváme opěrné body:

$$a_1(x_1, y_1),\ a_2(x_2, y_2),\ldots,\ a_n(x_n, y_n)$$

Interpolační funkce je taková funkce, která splňuje následující požadavek:

$$f(x_i),\ i = 1,\ 2,\ldots,\ n$$

Aby křivka byla interpolační musí tedy procházet všemi body a_i.

2.5.1 Interpolace jedním polynomem

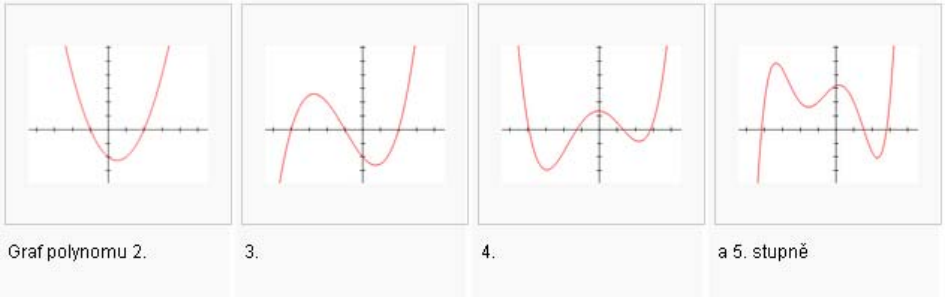
Řešení spočívá v nalezení polynomiální F(x) funkce daného stupně n, která v zadaných bodech $\mathbf{x_0 < x_1 < \ldots . < x_{n-1} < x_n}$ nabývá hodnot $\mathbf{F(x_0) = y_0}$, $\mathbf{F(x_1) = y_1}$, …, $\mathbf{F(x_n) = y_n}$.

Při interpolaci polynomem n-1 řádu hledáme řešení rovnic:

$$y_i = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \vdots & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{bmatrix} \begin{bmatrix} 1 \\ x_i \\ x_i^2 \\ \vdots \\ x_i^{n-1} \end{bmatrix}$$

Vstupem této rovnice jsou body, jež má křivka procházet. A jejím řešením jsou parametry matice $a_{m,n}$ .Do výrazu se postupně dosazují body, jimiž má křivka procházet a tím se obdrží soustava rovnic.

Nevýhoda polynomiální interpolace je ta, že pokud máme polynomy vyššího stupně, tak křivka dostává nepřirozené vlnění. Proto se v praxi používá maximálně polynom stupně 5. Protože čím je větší stupeň polynomu, tím víc je kořenů a každý kořen znamená další průchod s osou x – proto to nazýváme kmitání.



Obrázek č. 2.2 Ukázka použití polynomu vyššího stupně. Zdroj:http://cs.wikipedia.org/

2.5.2 Interpolace po částech

Nejčastěji používaná technika interpolace je interpolace po částech.

Uvažujme n-tici bodů. Interpolací polynomem třetího řádu rozumíme interpolací každých 4 bodů polynomem 3. stupně. Problém nastává při navazování křivek v bodech, kdy jedna přechází v druhou. Tento problém řeší několik metod, např. Akimiho interpolace, interpolace Hermitovskými polynomy a splajn křivkami k-tého řádu který zaručují spojitost k-1.

Tyto metody se spíše používají v statistické matematice a numerické matematice, než v počítačové grafice.

2.6 Aproximace

Na rozdíl od interpolace aproximace bodů znamená, že daná křivka nemusí procházet všemi body, křivka může těmito body být pouze ovlivněná. Aproximaci můžeme chápat několika v zásadě dvěma způsoby.

První způsob je takový, že máme určitý počet bodů a náš cíl je tyto body vhodně interpretovat. Mezi tento způsob patří například metoda nejmenších čtverců, jejímž smyslem je najít hladkou křivku a zároveň čtverec vzdáleností řídících bodů je od ní minimální. Tato metoda má pouze informativní charakter

Druhý způsob, který se využívá v počítačové grafice, nespočívá ve vhodné interpretaci bodů, ale v generování křivky. Křivka může být řízena body, potom mluvíme o tzv. řídícím polygonem, nebo bodu či vektoru. Existuje spoustu metod jak vytvořit aproximační křivku. Každá tato metoda zaručuje určité specifické vlastnosti křivky jako jsou hladkost, spojitost, a jiné.

V počítačové grafice se nejčastěji používá aproximace po částech (podobně jako interpolace) a to obyčejně kubikami (tedy křivkami, které jsou generovány polynomy třetího řádu). Důvody pro to jsou dva. Kubiky jsou dostatečně "pružnými" křivkami, aby se jimi dalo vyjádřit téměř vše, co je v praxi potřeba. Dalším důležitým faktorem je, že stupeň polynomu tři umožňuje velmi rychlý výpočet výsledné křivky, což je výhodné pro její interaktivní tvorbu.

Parametricky lze danou kubiku $Q(t)$ vyjádřit ve tvaru:

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$
$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$$
$$z(t) = a_z t^3 + b_z t^2 + c_z t + d_z$$

můžeme zapsat zkráceně v maticovém tvaru:

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

Derivaci q'(t) získáme derivací vektoru T :

$$q'(t) = \frac{\mathrm{d}}{\mathrm{d}t} q(t) = \frac{\mathrm{d}}{\mathrm{d}t} TC = [t^2, t, 1, 0]C$$

Konstantní matici *C* můžeme rozepsat do součinu:

C=MG

kde matice *M* je typu 4x4 a nazývá se bázová matice. Čtyř-prvkový vektor 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:

$$q(t) = [x(t), y(t), z(t)] = [t^3, t^2, t, 1] \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & m_{1,4} \\ m_{2,1} & m_{2,2} & m_{2,3} & m_{2,4} \\ m_{3,1} & m_{3,2} & m_{3,3} & m_{3,4} \\ m_{4,1} & m_{4,2} & m_{4,3} & m_{4,4} \end{bmatrix} \begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix}$$

Mezi často požadované vlastnosti křivek patří:

- Invariance k afinním transformacím a projekcím, která zaručuje, že například rotace řídicího polygonu a následné generování křivky má stejný výsledek, jako rotace každého bodu z vygenerované křivky.
- Vlastnost konvexní obálky (angl. convex hull property)

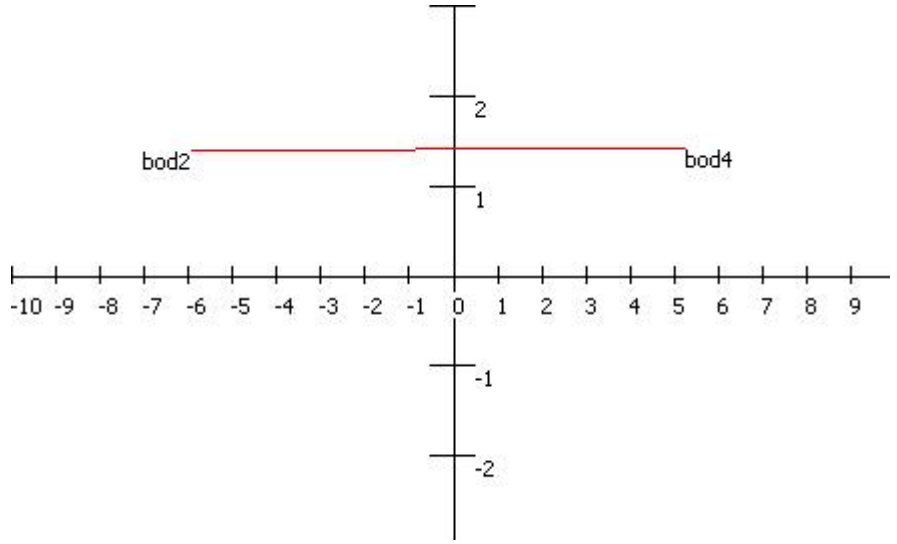
  (a) silná podmínka - celá křivka leží v konvexní obálce všech svých řídicích bodů,

(b) slabá podmínka - část křivky leží v konvexní obálce některých řídicích bodů (typicky segment, v obálce svého generujícího polygonu).

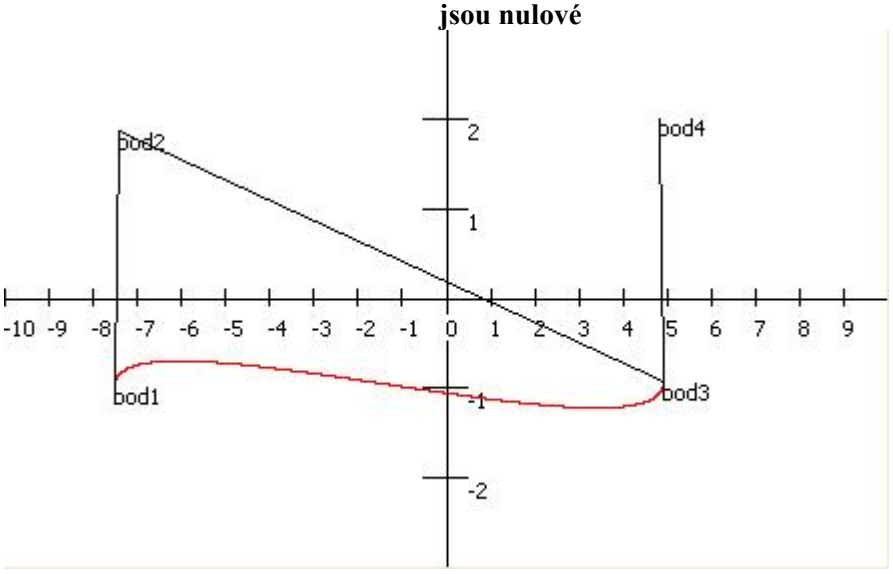
- Lokalita změn - změnou polohy (u racionálních křivek i váhy) řídicího bodu se mění jen část křivky, nikoli křivka celá.
- Křivka může procházet krajními body svého řídicího polygonu

2.6.1 Fergusonova (HERMITOVSKÁ) křivka

Fergusonova křivka je křivka interpolační, fergusonovu křivku tvoří vždy nejméně dva body a dva vektory. Body určují počáteční a koncový bod křivky a vektory zaručují míru vyklenutí křivky, tzn. čím větší jsou vektory, tím déle se křivka k nim přimyká. Pokud tedy jsou vektory nulové, pak se křivka mění v přímku z počátečního do koncového bodu.



Obrázek č. 2.3 Ukázka z programu, Fergusonova křivka, vektory jsou nulové



Obrázek č. 2.4 Ukázka z programu, Fergusonova křivka 4 body

Předpis pro výpočet Hermitovské kubiky :

$$Q(t) = [t^3, t^2, t, 1] \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ \overrightarrow{q'}_0 \\ \overrightarrow{q'}_1 \end{bmatrix}$$

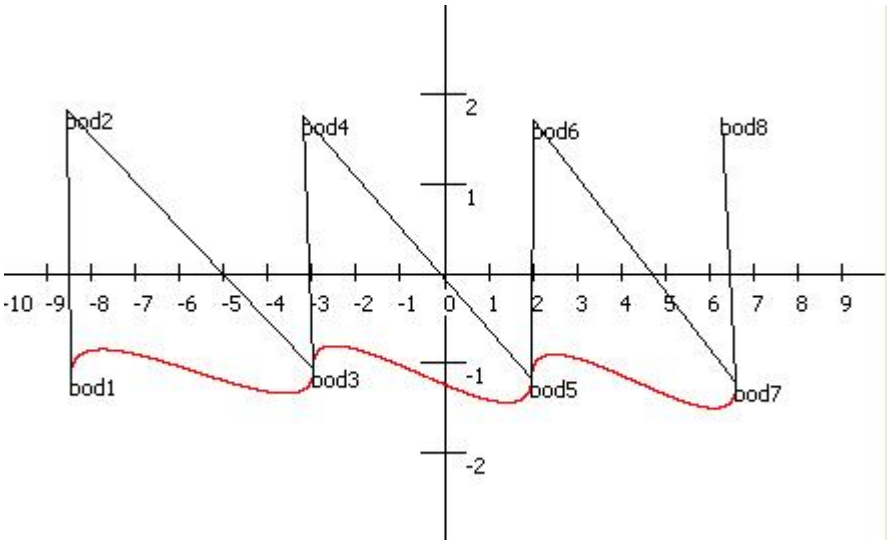
Rozepsané v jediné rovnici by to vypadalo takto:

$$Q(t) = P_0F_1(t) + P_1F_2(t) + \overrightarrow{q'}_0F_3(t) + \overrightarrow{q'}_1F_4(t),$$

Kde F1, F2, F3, F4 jsou tzv. kubické Hermitovské polynomy:

$$F_1(t) = 2t^3 - 3t^2 + 1,$$
$$F_2(t) = -2t^3 + 3t^2,$$
$$F_3(t) = t^3 - 2t^2 + t,$$
$$F_4(t) = t^3 - t^2.$$

Dosazením t=0, resp. t=1 ověříme, že křivka začíná a končí v bodě P_0, resp. P_1 . Největší přednost Hermitovských kubik se projeví při jejich navazování. Vzhledem k tomu, že součástí křivek jsou tečné vektory v jejich koncových bodech, proto můžeme jejich hladké navazování snadno realizovat.



Obrázek č. 2.5 Ukázka z programu, spojení 3 fergusonových křivek, v bod3, bod5, bod7

Spojitost dvou Hermitovských kubik docílíme totožností posledního bodu segmentu $Q_1(1)$ a prvního bodu segmentu $Q_2(0)$. Spojitost C^1 je zaručena identitou tečných vektoru v uzlu. Spojitost G^1 docílíme jejich lineární závislostí, vyjádřeno pomocí geometrických vektorů G dvou po sobě následujících segmentů.

$$G_1 = \begin{bmatrix} P_1 \\ P_2 \\ \overrightarrow{q'}_1 \\ \overrightarrow{q'}_2 \end{bmatrix}, G_2 = \begin{bmatrix} P_2 \\ P_3 \\ k \cdot \overrightarrow{q'}_2 \\ \overrightarrow{q'}_3 \end{bmatrix}, k > 0$$

Nevýhodou těchto křivek je nesnadná editace tečného vektoru ve třech rozměrech.

2.6.2 Bézierovy křivky

Bézierovy křivky jsou křivky aproximační a patří mezi nejpopulárnější křivky o modelování ve dvou rozměrech, ale i pro definici trojrozměrných objektů. Její využití můžeme najít například pro *fonty*(písmo).

Bézierova křivka stupně n je dána n+1 body. Ale v našem případě je Bézierova křivka nevhodná, přesněji řečeno samotná bézierova křivka zadaná n body, se při jakékoli změně i jediného bodu změní celá, což je pro nás nežádoucí, protože můj program, který bude představen níže, se zabývá výběrem určité oblasti. A kdyby se nám při přidání řídícího bodu křivky změnila námi vybraná oblast, kterou jsme předtím pracně vybírali, jistě by to bylo nevhodné. Proto se budu zabývat představením Bézierovy kubiky.

2.6.3 Bézierovy kubiky

Bézierovy kubiky jsou také aproximační křivky, princip je totožný jako u Bézierových křivek. Bézierova kubika je zadána čtyřmi body, ale to neznamená, že výsledná křivka musí mít pouze 4 body. Vychází z prvního řídícího bodu P_0 a končí v posledním P_3, kubika je určena vztahem:

$$Q(t) = \sum_{i=0}^{3} P_i B_i(t),$$

Bézierovy polynomy mají následující tvar:

$$B_0(t) = (1-t)^3,$$
$$B_1(t) = 3t(1-t)^2,$$
$$B_2(t) = 3t^2(1-t),$$
$$B_3(t) = t^3.$$

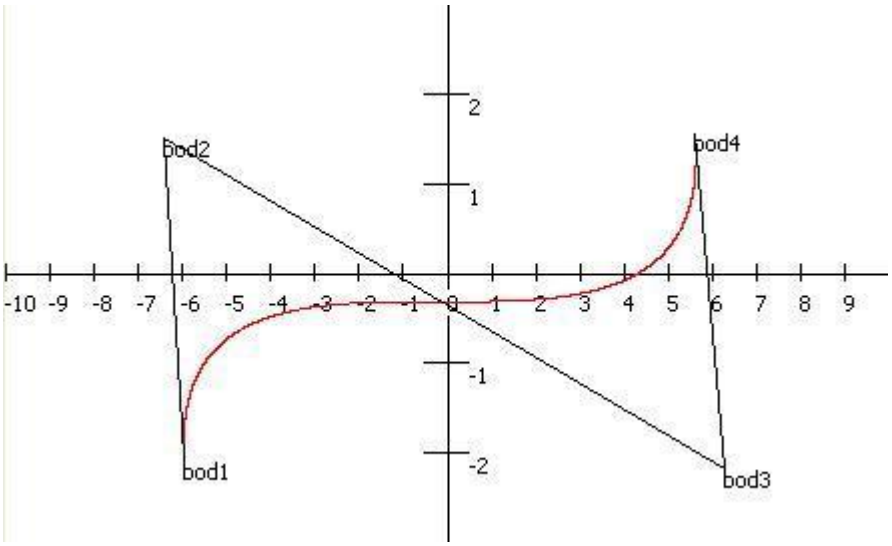
Maticový zápis Bézierovy kubiky je:

$$Q(t) = [t^3, t^2, t, 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

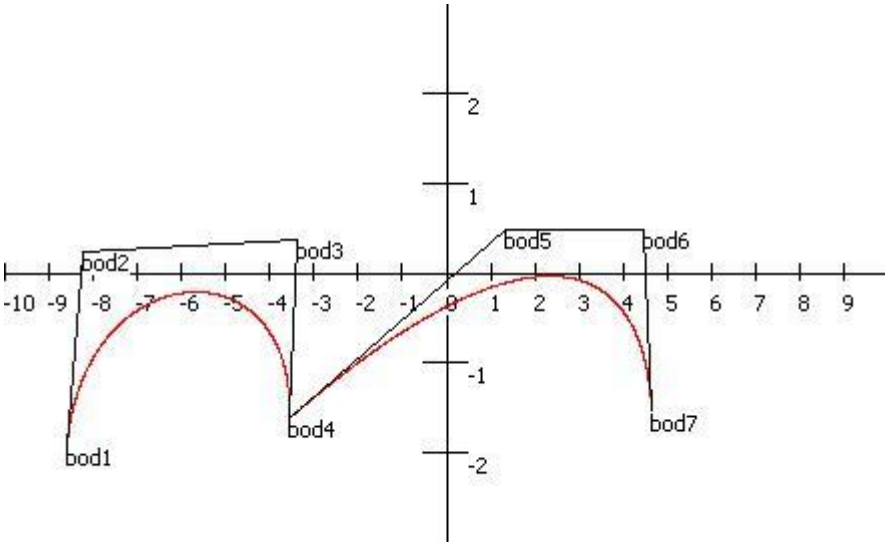
Pro navazování jsou důležité tečné vektory v prvním a posledním bodě:

$$\overrightarrow{p'}(0) = 3(P_1 - P_0) \qquad a \qquad \overrightarrow{p'}(1) = 3(P_3 - P_2)$$

Bézierovy kubiky mají automaticky spojitost třídy C^0, což pro zopakování znamená, že poslední bod G^3 prvního segmentu křivky a první bod P^0 následujícího segmentu jsou totožné. Pokud bychom chtěli zajistit spojitost vyššího řádu, tak by P^1 musel ležet na jedné přímce spolu s P^0=G^3 a G^2, což by zajistilo minimálně spojitost C^1.



Obrázek č. 2.6 Ukázka z programu, Bézierova kubika složená ze 4 bodů



Obrázek č. 2.7 Ukázka z programu, Bézierova křivka složená ze 2 segmentu (kubik), zajištěná spojitost C^0.

3. Výukové programy

Při výběru programu pro práci s vektorovou grafikou máme hodně možností, na softwarovém trhu je celá řada různých výukových programů. Ale každý nemusí odpovídat našim požadavkům a hlavně našim možnostem, protože některé programy jsou sice velice kvalitní a uživatelsky příjemné, ale za tyto vlastnosti se většinou platí. Na trhu jsou v zásadě dva druhy těchto produktů a to *volně šiřitelný programy* nebo *placený software*.

3.1 Volně šiřitelný software (Freeware)

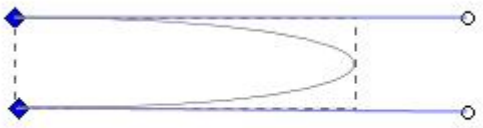
Někdy se říká, že co je volně šiřitelný, tak nemusí být tak kvalitní. Ale to podle mého názoru není pravda. To, že program je volně šiřitelný

sice může naznačovat, že programátorský tým nemusí mít za zády nějakou silnou společnost. Ale historie dává jasně najevo, a to nejen v oboru počítačové grafiky, že i volně šiřitelné programy mohou být stejně kvalitní možná i lepší. Ale co je hlavní vlastnost freeware programů je ta, že je zcela zdarma a můžeme si ho stáhnout, nainstalovat a používat bez jakýchkoli poplatků. Tato vlastnost jistě smaže některé nedostatky programu.

3.1.1 Inkscape

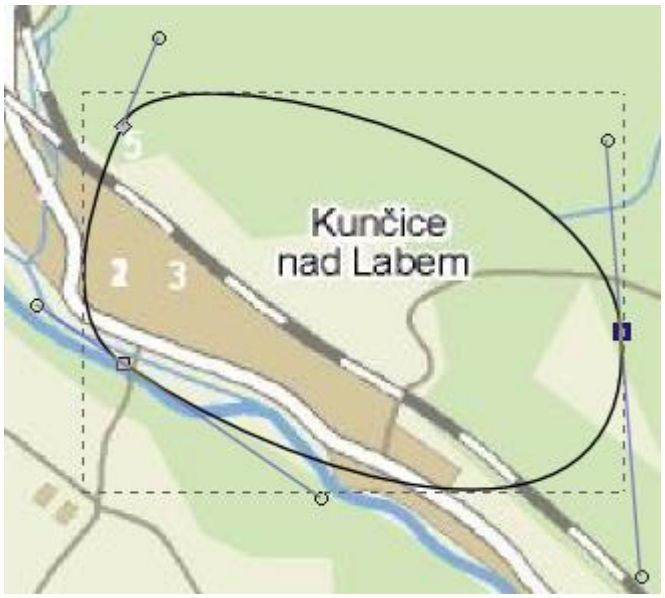
Program, který bych rád přiblížil je *OpenSource vektorový grafický editor*. Inkscape může být používán pod operačním systémem *Windows*, *či Mac OS X*, ale prvořadá operační platforma je systém *Linux*. Já jsem využil možnost si tento produkt stáhnout pro operační systém Windows. Instalační spustitelný soubor má cca. 22MB. Což při rychlosti dnešního internetu není žádný velký problém. Po jednoduché instalaci můžeme hned začít práci v programu. Já jsem se zaměřil konkrétně na realizaci Bézierovy křivky.

V programu klávesovou zkratkou *Shift+F6* se aktivuje volba „Kresba bezierových křivek a přímých čar“. Nyní je vše připraveno, abychom mohly začít kreslit. Při zadávání bodů na naše plátno se nám vykreslují pouze přímé čáry z bodu A do bodu B. Pokud chceme, aby se ze zadaných bodu vytvořila Bézierova křivka, musíme přidržet pravé tlačítko myši a křivka se nám začne vykreslovat.



Obrázek č. 2.8 Ukázka z programu Inkscape, Bézierova křivka složená ze 4 bodů.

Pro jednoduchost jsem si vybral obrázek mé rodné vesnice a chtěl jsem vybrat oblast ohraničenou křivkou. Bohužel musím říci, že se mě to nepodařilo. Tento program je sice výhradně pro práci s vektorovou grafikou, ale taková věc jako výběr ohraničené oblasti by tam rozhodně měla být. Neříkám, že to v tomto programu není obsaženo, ale bohužel jsem to nějak jednoduše nenašel. A nemyslím si, že začínající uživatel s tímto programem nebo dokonce s počítačovou grafikou by to mohl najít rychleji.



Obrázek č. 2.9 Ukázka z programu Inkscape, výběr oblasti pomocí Bézierovy křivky, viditelná konvexní obálka a řídící body.

Program Inkscape dokáže vytvořený obraz uložit do vektorového grafického formátu (*.svd, *.pdf), ale také i export do rastrového formátu (*.jpg, *.bmp).

Tento program je pro uživatele, kteří dávají přednost operačnímu systému Linux. Vzhled, ale i detaily programu připomínají rozhraní Linuxu. Program má velmi pokročilé funkce, navíc tento program je freeware. Což je opravdu velké plus. Tento program není nijak převratný, ale mohu ho vřele doporučit všem, kteří hledají dobrý grafický vektorový editor za málo peněz.

3.1.2 OpenOffice.org Draw

Další program, který se řadí mezi OpenSource vektorový editor je OpenOffice.org Draw. Tento produkt zapadá do balíčku OpenOffice.org. Ten obsahuje

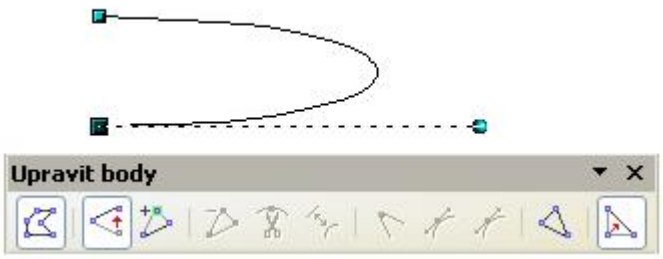
- Tabulkový procesor
- Textový procesor
- Prezentační nástroj
- Grafický editor
- Databázový frontend
- Nástroj na vytváření matematických vzorců

OpenOffice.org je zdařilá freeware aplikace, která se velmi podobá svému největšímu konkurentu Microsoft Office. A jelikož v dnešní době spousta uživatelů přechází z operačního systému Windows na jeho freeware konkurenci Linux, tak i OpenOffice.org získává stále větší oblibu. Ovšem OpenOffice.org není pouze pro operační systém Linux, tento balíček je multiplatformní. Může být používán na téměř všech operačních systémech. Ale také jako u Inkscape jeho prvořadá platforma je operační systém Linux. Aktuální verze je momentálně 2.2.

Po stažení balíku OpenOffice, který má cca. 100MB. Můžeme začít instalaci, během instalace si můžeme vedle standardní instalace vybrat vlastní, a tam nastavit pouze ty balíčky který chceme používat. Ted už by jsme měli mít nainstalovaný balíček OpenOffice.org Draw, po spuštění již můžeme začít kreslit. Zobrazí se nám klasická aplikace, tak jak jsme zvyklý z jiných programů. Na spodní části obrazovky se nachází panel s nástroji pro kreslení a vybereme ikonu křivka, je to sedmá ikona zleva. Při kreslení křivky je kreslena Bézierova křivka.

Postup při kreslení:

- Klepněte na místo, kde má čára začínat, a táhněte myší směrem, kterým má čára vést. Směr ukazuje řídící čára. Pokud chcete omezit směr čáry na 45 stupňů, podržte během tažení myší klávesu Shift.
- Uvolněte myš na místě, kdy by měl být první řídící bod.
- Přesuňte ukazatel na místo, kde chcete, aby segment čáry končil. Křivka se mění podle ukazatele.
- Proveďte jeden z následujících kroků:
- Chcete-li ukončit kresbu křivky, tak stačí kliknout na pozici koncového bodu.Chcete-li vytvořit uzavřenou křivku, poklepejte na počáteční bod.
- Klepněte a uvolněte tlačítko myši pro přidání kotevního bodu. Pohybem myší nakreslíte další segment.
- Klepnutím a tažením v kterémkoliv směru přidáte plynulý přechod.



Obrázek č. 3 Ukázka z programu OpenOffice.org Draw, Bézierova křivka, 4 body, úprava 1 řídícího bodu a jeho vektoru

Programátoři programu OpenOffice.org Draw dali na trh velmi kvalitní vektorový editor. Jistě se v mnoha věcech vyrovná svým konkurentům z komerčních programů. Ale můj názor na provedení bézierovy křivky je drobek zklamání. Hezké grafické provedení ovšem není všechno, u bézierovy křivky není moc dobře odhadnutelný koncový bod.

3.1.3 Srovnání představených freeware programů

To co jsem Vám právě představil, není ani zdaleka celková nabídka na trhu. Ale právě tyto programy podle mého názoru mají největší šanci v silné konkurenci, kterou představují komerční programy, se prosadit. Kdybych si měl vybrat mezi těmito dvěma produkty, tak bych si vybral

program Inkscape. Moje hodnocení se zaměřuje výhradně na to, jak tyto dva programy nabízejí použití aproximačních křivek, a to konkrétně Bézierovy křivky. U programu OpenOffice.org Draw je základní věc, že se nedá moc dobře odhadnout, kde bude naše křivka končit. Naproti tomu program Inkscape nám nabízí po celou dobu naší práce to jak bude křivka vypadat.

3.2 Komerční výukové programy

Komerční program je program, na který si musíme koupit licenci. Po zaplacení poplatku nám zašlou licenční číslo, které nám pomocí internetu vybraný produkt aktivuje. Toto je ale pouze jedno řešení jak lze produkt aktivovat, je také možné zakoupit v obchodě již verzi se sériovým klíčem a tento klíč pouze zadat při instalaci a máme aktivovaný produkt.

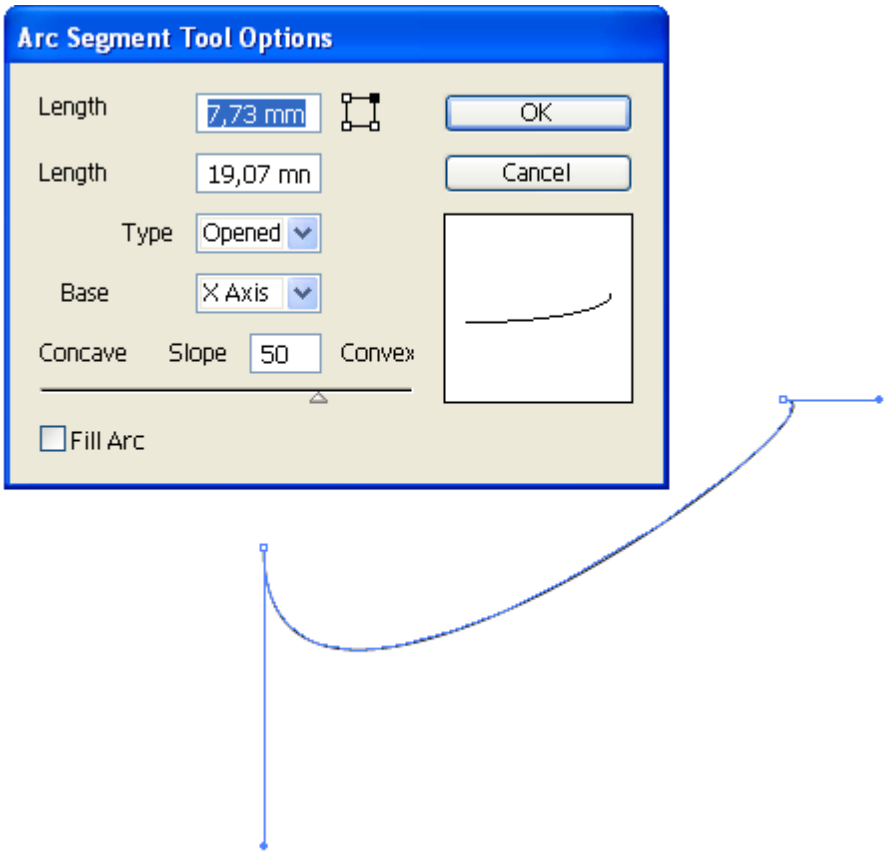
3.2.1 Ochrana autorských práv

Samozřejmě v dnešní době se dá obejít vše, a to i programy, které jsou navrhnutý jako neprolomitelné. Jakmile přijde nový produkt na trh, tak se většinou ve velmi krátkém období podaří ochranu prolomit. Ale je tu spoustu věcí, které hovoří proti používání pirátských verzí. Nejzávažnější je porušení autorských práv, pokud by se přišlo na to, že porušujeme autorské právo. Můžeme za to dostat peněžitý trest nebo propadnutí věci, ale to nejhorší je, že se můžeme se dostat až na dva roky do vězení. Proto všem doporučuji používat legálně zakoupené programy, aby se nedostaly do křížku se zákonem.

3.2.2 Adobe Illustrator

Další produkt, který zde krátce představím je Adobe Illustrator SC2. Představování něčeho, co již bylo mnohokrát představeno. Navíc, asi většina uživatelů, kteří se alespoň drobek zajímají o vektorovou grafiku, tento program dobře znají. Ale přesto se o to ve zkratce pokusím. Adobe Illustrator je nejznámější komerční vektorový editor. Firma Adobe náš trh zásobuje spoustou programů na špičkové úrovni, například Adobe Photoshop.

Pokud se Vám do ruky dostane program jako Adobe Illustrator je těžké něco vytknout, po klasické instalaci a spuštění, si nejdříve vybereme nový dokument a pak již můžeme začít kreslit.



Obrázek č. 3.1 Ukázka z programu Adobe Illustrator CS2, Bézierova křivka, 4 body, úprava křivky

Obrázek 2.3. Nám nejlépe znázorňuje jaký je program Illustrator. Možnosti, které nám nabízí, jsou značné. Křivku můžeme upravovat přímo v grafickém prostředí, uchopením jednoho z řídících bodů nebo dokonce samotné křivky! A podle toho se nám již křivka mění spolu s řídícími body.

U křivky můžeme nastavit zda-li má být uzavřená, z jakého bodu má začínat, řídící osu, procento concave-nosti křivky.

Tento program má ale jednu nevýhodu. Tou je cena. Pro představu o jaké částce zde mluvím, tak licence na Adobe Illustrator CS3 stojí US599$. Upgrade, starší verze US199$. Ale firma Adobe nabízí ke stažení trial verzi, které jsem využil.

3.3 Srovnání představených programů

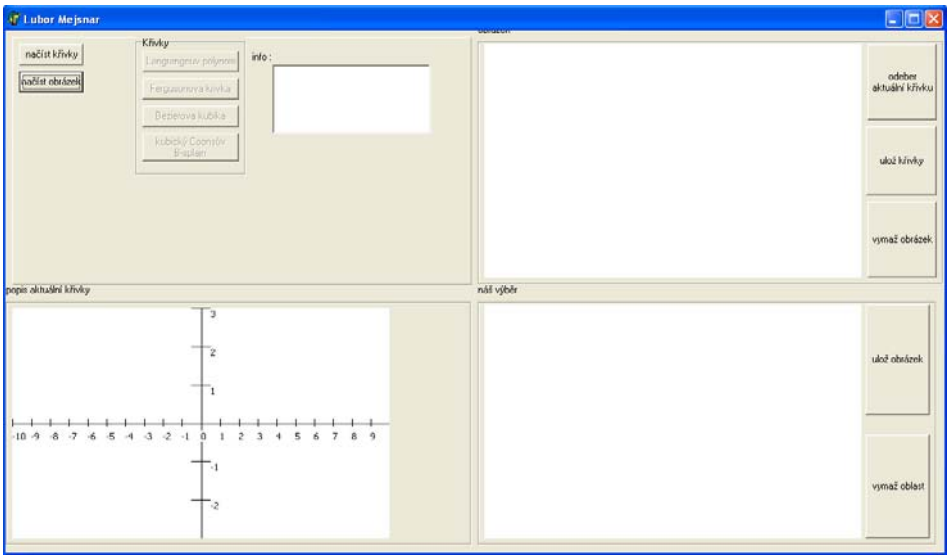
Programy, které jsem tu v rychlosti představil, ani zdaleka nejsou všechny nabízené produkty. Představené programy je jen průřez do vektorových editorů. Kdyby jsem měl zhodnotit a poradit jaký SW si máte vybrat, tak by moje odpověď zněla podle očekávání Adobe Illustrator. Tento program je podle mého názoru velice přehledný, jednoduchý na ovládání a hlavně jeho funkce jsou opravdu velmi dokonalé. Jak jsem již řekl, jeho největší nevýhodou je cena. Tento program je určen opravdu pro profesionály ve svém oboru nebo spíše pro firmy. Málokterý běžný uživatel si může dovolit zaplatit 600$ za licenci. Proto pokud podnikáte v tomto oboru a máte potřebu pracovat s velmi dobrým prostředím, tak neváhejte a

kupte si licenci, jistě nebudete litovat. Jinak doporučuji některý z freeware editorů.

4. Představení vlastního programu

Když jsem vymýšlel program, kterým bych představil křivky pro začínající počítačové grafiky nebo jen studenty, kterým by to pomohlo při studiu, tak hlavní můj požadavek byl jednoduchý kód a přehledné ovládání. Nyní Vám představím program nejprve na uživatelské úrovni a poté se podíváme do zdrojových kódů, které považuji za zajímavé.

4.1 Uživatelské rozhraní

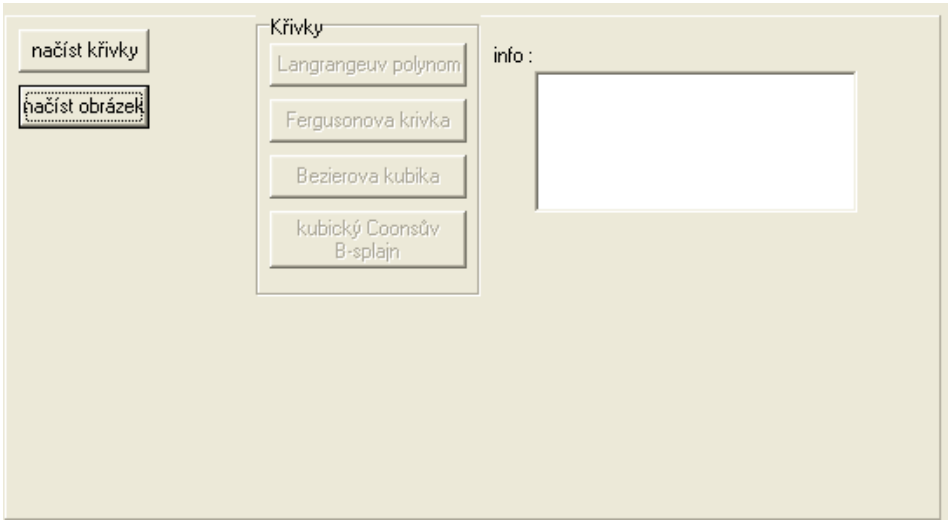


Obrázek č. 3.2. Ukázka z mého programu, Uživatelské rozhraní po spuštění souboru LubkaSoft.exe¨

Po spuštění souboru LubkaSoft.exe se spustí aplikace složená ze 4 částí, které si v několika krátkých kapitolách představíme. A po té představím již zmiňovaný zdrojový kód.

4.2 Ovládací prvky

Když jsem se rozhodoval jak udělat uživatelský přátelské rozhraní pro moji aplikaci, dal jsem přednost tlačítkům, aby uživatel měl všechny možnosti aplikace na očích a nemusel nikde hledat své aktuální další možné kroky.



Obrázek č. 3.3. Ovládací prvky mého programu

Ovládací prvky typu TButton (tlačítko):

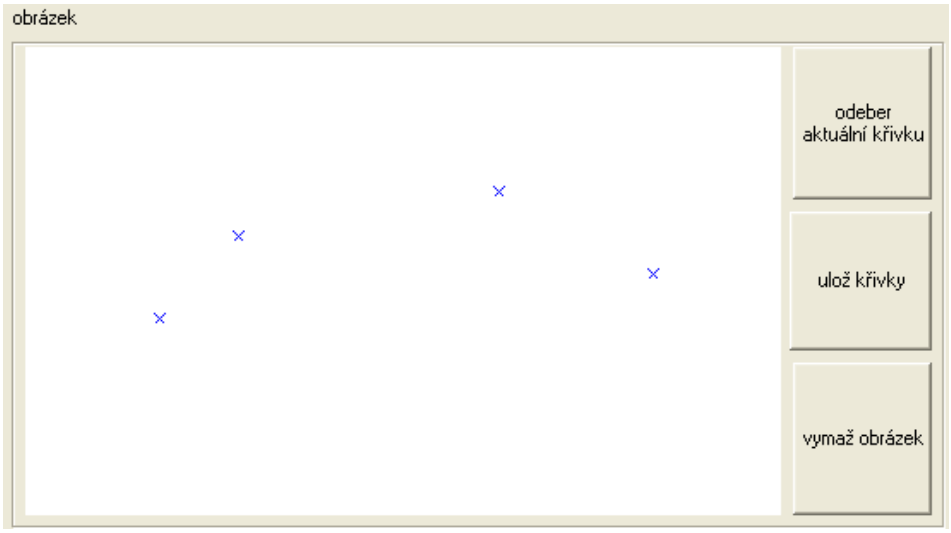
- *Načíst křivky* – po kliknutí na toto tlačítko se nám otevře dialog na otevření textového souboru, pro úspěšné načtení křivek musíme nejdříve takový soubor vytvořit. Křivky se vykreslí do obr 2.3.
- *Načíst obrázek* – otevře se nám dialog pro otevření obrázku formátu *.bmp. Vybraná bitmapa se vloží do obr 2.3.
- *Lagrangeuv polynom* – vykreslí Lagrangeuv polynom pro libovolný počet bodů
- *Fergusonova křivka* – vykreslí Fergusonovu křivku pro počet bodů větší než 4 a zároveň sudý počet bodů
- *Bezierova kubika – vykreslí Bézierovy kubiky, pro 4 ... 4+n*3 body*
- *Aproximační kubický splajn – vykreslí se Kubický splajn po zadání 4 bodů*

Ovládací prvky typu TListBox:

- Listbox – V listboxu (info) se zobrazují aktuální informace o aktivní křivce

4.3 Okno pro zadání řídících bodů

Toto je hlavní část mého programu. Při kliknutí do bílé oblasti se vkládají řídící body pro budoucí křivku. Aby bylo vidět, kam jsme vložily řídící bod, vykreslí se malý křížek do tohoto místa. Viz následující obrázek



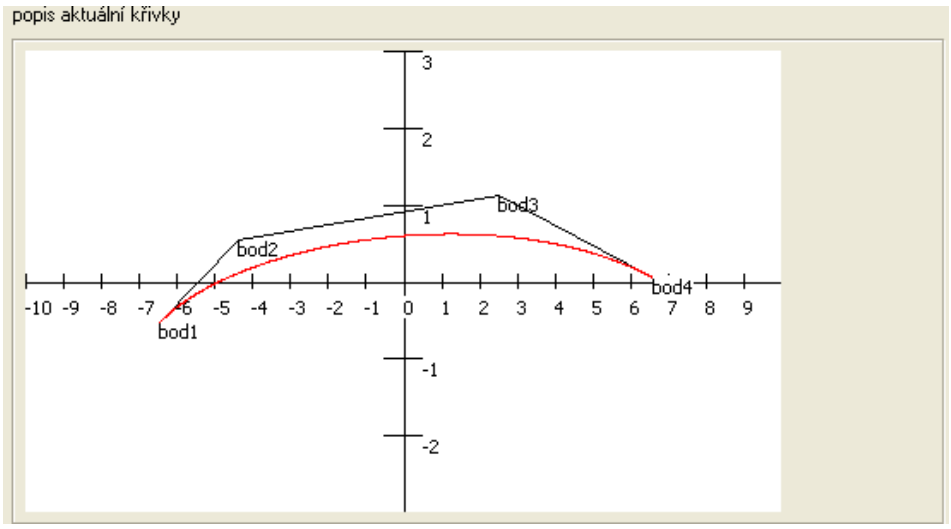
Obrázek č. 3.4. Ukázka výběru bodů pro některou z křivek.

Ovládací prvky typu TButton:

- *odeber aktuální křivku* – odebere aktuální křivku
- *ulož křivky* – uloží všechny křivky ve výkresu do námi zadaného souboru
- *vymaž obrázek* – vymaže aktuální okno(obr 2.3.) námi definovanými křivkami

4.4 Popis aktivní křivky

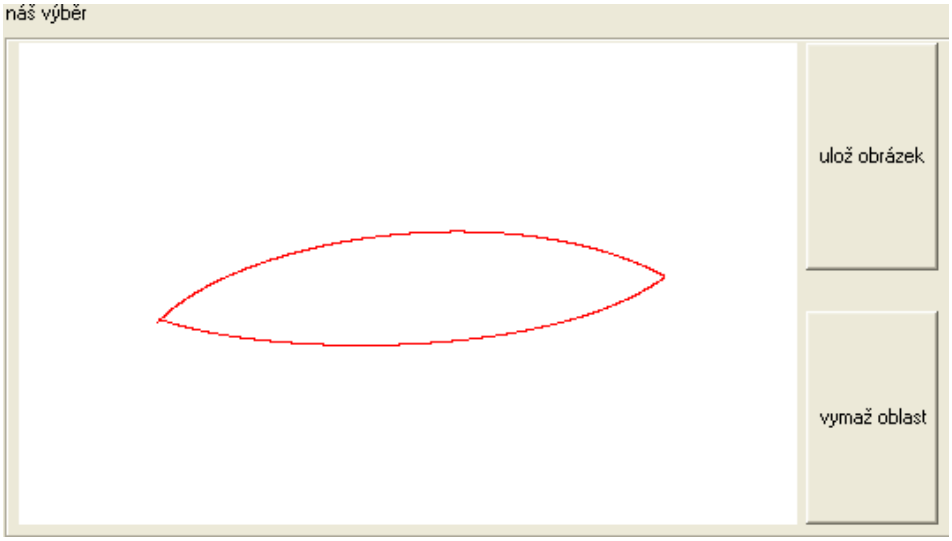
V této části obrazovky se nám vždy vykreslí aktuální křivka v rovinné kartézské soustavě souřadnic. S vykreslenou konvexní obálkou a vypsanými řídícími body.



Obrázek č. 3.5 Popis aktuálně nakreslené nebo modifikované křivky.

4.5 Výsledná vybraná oblast

Následující obrázek nám demonstruje oblast ohraničenou námi definovanými křivkami. Pokud máme načtený obrázek do obr 2.3. Stačí pouze kliknout do ohraničené oblasti a výsledný obrázek bude bitmapa velikosti zadanou křivkami.

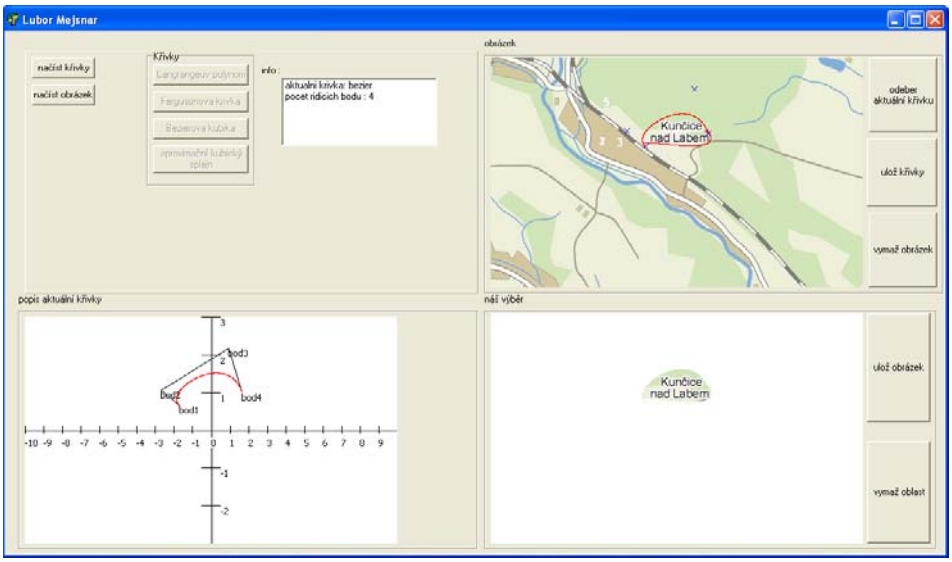


Obrázek č. 3.6 Výsledný obrázek, který je možný uložit, nebo pokud chceme vybrat oblast ohraničenou křivkou, stačí kliknout dovnitř ohraničené oblasti a výsledná oblast se zkopíruje z obr. 2.3.

Ovládací prvky typu TButton:

- Ulož obrázek – po kliknutí na toto tlačítko se nám zobrazí dialog pro uložení obrázku.
- Vymaž vybranou oblast- vymaže vykreslenou oblast.

Abychom viděli přesný postup krok za krokem, následující průřez krok za krokem výběru oblasti nám to přiblíží.



Obrázek č. 3.7 Zde můžeme vidět, jak by mohl vypadat výsledek.

4.6 Programová část

Tato aplikace je naprogramována v prostředí Borland Delphi Professional Second Edition v 7.2. Je to integrované grafické prostředí pro platformu MS Windows v jazyce Object Pascal.

4.6.1 Použité komponenty

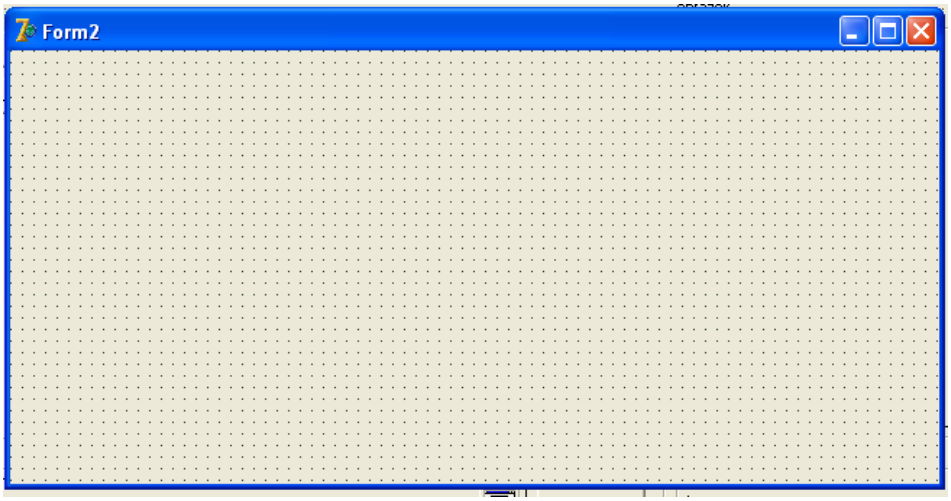
Programování v Delphi je z velké části založeno komponentech. Komponenta je malý program nebo-li, balíček funkcí, který vykonává určitou činnost, jako třeba zobrazuje text, zobrazuje obrázky, přehrává multimedia, komunikuje s databází. Každá komponenta má své události(events) a vlastnosti(properties). Ty se definují v panelu zvaném Object Inspector. Tam je souhrnná nabídka těchto událostí a vlastností. Pokud si vybereme nějakou událost, tak stačí na ni double click a delphi automaticky vytvoří proceduru, do které můžeme začít psát tělo.

Nyní Vám v rychlosti představím některé použité důležité komponenty.

4.6.2 TForm

Při vytvoření formuláře ve Form designeru, tak jsou automaticky implementovány potomci Formu, které mohou být do formu vkládány. Komponenta TForm je hlavní komponentou, která se může také tvářit jako hlavní aplikace (Main Window).

TForm může obsahovat objekty jako TButton, TCheckbox nebo TPaintBox.



Obrázek č. 3.8 Ukázka vytvořeného TForm pojmenovaný Form2

TForm má v základní nabídce velmi mnoho událostí nebo-li evets. Který mohou pracovat s vlastním TFormem. Představím Vám některé události, které jsem využil v mém programu.

Events:

- *OnCreate* – tělo této metody se provede vždy při spuštění projektu nebo při spuštění příslušného Formu.
- *OnResize* – při změně velikosti okna Form se provede tělo této funkce

4.6.3 TImage

TImage je komponenta, která dokáže zobrazit nějaký bitmapový obrázek. Musíme ho vložit do Form-u a po té již s ním můžeme pracovat, můžeme načítat ze souboru, ukládat, kopírovat z nějaké bitmapy. *TImage* používám ve svém programu viz. Obr. 2.5 a 2.3. Je to velmi dobrý nástroj pro práci s libovolnou bitmapou.

Mezi nejdůležitější vlastnosti *TImage* určitě patří *Canvas.* Tato vlastnost nám umožňuje zakreslovat do TImage pomocí vlastností *LineTo*. Nebo dokonce pomocí vlastnosti *Draw* překopírování celé bitmapy do *TImage*.

A mezi důležité události patří:

- OnMouseDown – tělo této procedury se nám provede při kliknutí myši na plátně, této vlastnosti využívám při zadávání bodů

4.6.4 TPaintBox

TPaintbox poskytuje plátno (canvas) pro kreslení libovolného obrázku. Tuto vlastnost jsem využil viz Obr. 2.4. Tato komponenta je téměř stejná jako *TImage*, s rozdílem že *TImage* je jednodušší na ukládání.

TPaintbox také obsahuje vlastnosti Canvas, která nám zpřístupní jako u TImage vlastnosti *Draw*, *LineTo*, *MoveTo.*

4.6.5 TSavePictureDialog

TSavePictureDialog nám zobrazí „Ulož jako“ dialog, který nám umožní uložit obrázek.

Mezi důležité vlastnosti patří:

- Execute – vyvolá dialog
- FileName – do této vlastnosti se nám uloží cesta, kterou jsme si vybrali

4.6.6 TSaveDialog

Téměř stejná komponenta jako TSavePictureDialog s tím rozdílem, TSaveDialog nám umožňuje uložit i jiný soubor než obrázek.

Mezi důležité vlastnosti patří:

- Execute – vyvolá tento dialog
- FileName – do této vlastnosti se nám uloží cesta, kterou jsme si vybrali

4.6.7 TOpenDialog

Opak TSaveDialog, kdy se nám otevře dialog pro otevření souboru. Mezi důležité vlastnosti patří:

- Execute – vyvolá tento dialog
- FileName – do této vlastnosti se nám uloží cesta, kterou jsme si vybrali

4.6.8 TOpenPictureDialog

Otevře se nám dialog, pro otevření obrázkového souboru.

Mezi důležité vlastnosti patří:

- Execute – vyvolá tento dialog
- FileName – do této vlastnosti se nám uloží cesta, kterou jsme si vybrali

4.6.9 TBitmap

TBitmap je grafický objekt, který dokáže vytvořit, změnit nebo uchovávat obrázek v paměti nebo v externím souboru na disku.

Tuto komponentu využívám při kreslení křivek a načítání obrázku, je to rychlejší, než kdybych to rovnou kreslili do TImage, takto to v paměti nakreslím a pomocí vlastnosti Draw u TImage.Canvas přesunu do TImage.

4.7 Představení zdrojových kódů křivek

Před vlastním představením algoritmů křivek je nejprve důležité si uvědomit, že potřebujeme nějakou datovou strukturu na uchovávání křivek a jejich řídících bodů, to celé znamená uchovávat celý výkres. K tomu mě slouží vytvořené třídy (objekty) :

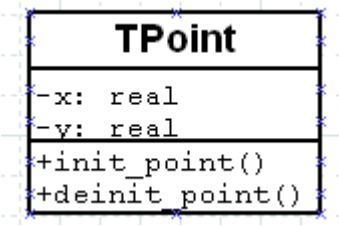
- TPoint

- TKrivka
- TVykres

Nyní si blíže představíme tyto objekty.

4.7.1 TPoint

Object TPoint používám pro uchovávání souřadnic *x* a *y* řídících bodů křivky. Je to důležité z toho hlediska, že kdykoli můžu vědět souřadnice x, y řídícího bodu.



Obrázek č. 3.9 UML diagram reprezentující object TPoint

Private atributy:

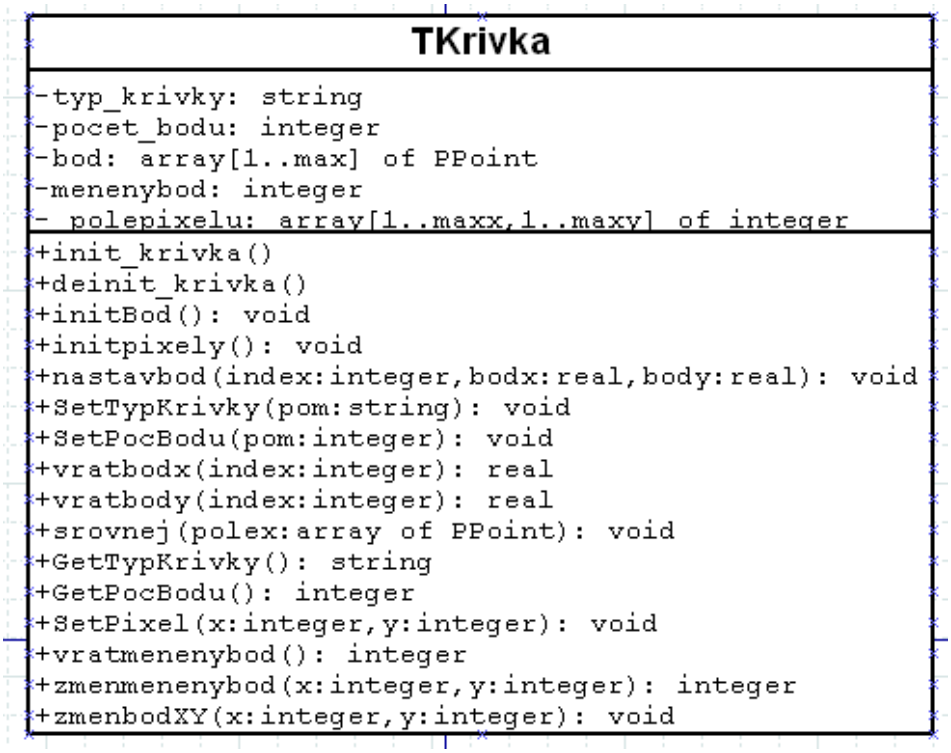
- x a y jsou typu real

Public metody zde jsou pouze :

- Constructor init_point()
- Destructor deinit_point

4.7.2 TKrivka

V objektu TKrivka uchovávám veškeré informace o křivce. Následující UML diagram nám to přiblíží. Tento způsob uchovávání křivky nám velmi usnadní její modifikování, vykreslování a vyhledávání ve výkresu.



Obrázek č. 4 UML diagram reprezentující object TKrivka

Vzhledem k obsáhlosti zdrojových kódů představím pouze některé metody nebo atributy, které jsou z mého pohledu velmi důležité a usnadňují práci s křivkami. U většiny metod nebo atributů je z jejich názvu poznat jejich účel.

Private atributy:

- typ_krivky – údaj říká o jaký typ křivky jde
- bod - je typu PPoint to je ukazatel na TPoint
- polepixelu – dvojrozměrné pole, které využívám pro výběr křivky z výkresu
- menenybod – index bodu, který budeme měnit

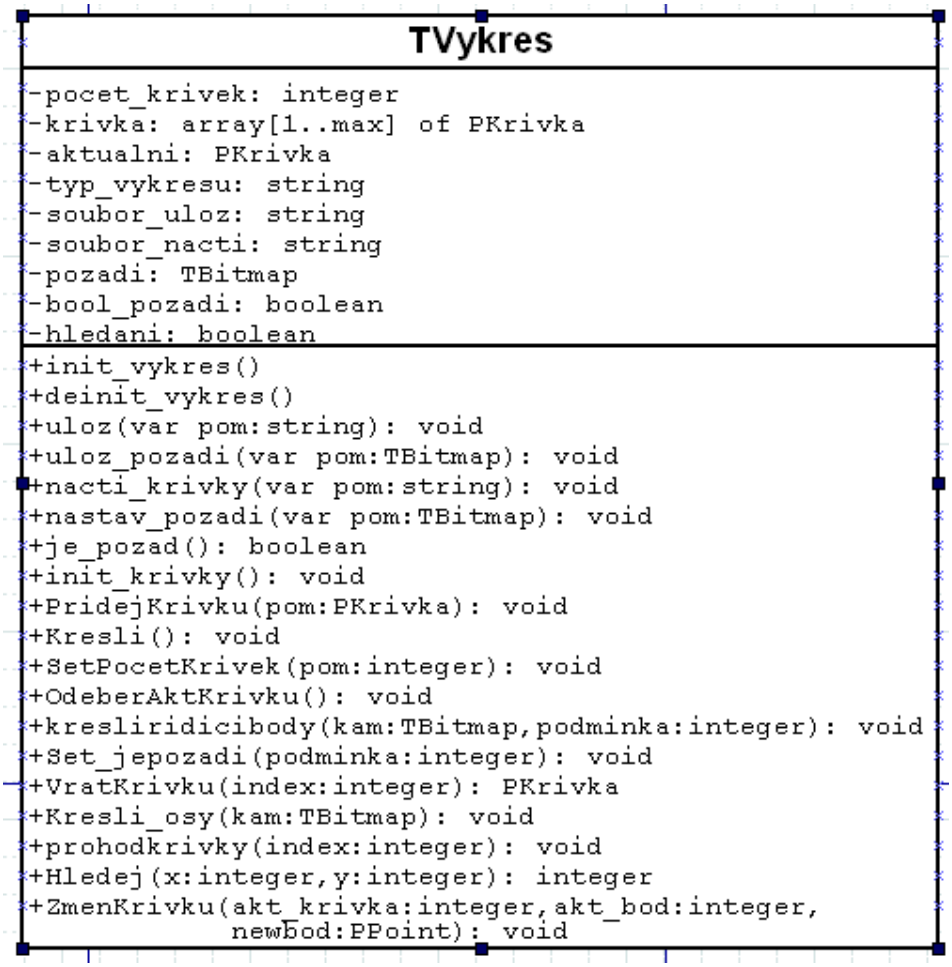
Public metody:

- Constructor init_krivky
- Destructor deinit_krivky
- Srovnej(polex : array of PPoint) – v případě, že se jedná o lagrangeuv polynom srovná souřadnice podle x, z důvodu, abychom mohly zadávat body nezávisle na pořadí podle osy x
- SetPixel(x,y : integer) – nastavuje atribut polepixelu souřadnici x +/- n, y +/- n na 1. Kde n je konstanta, jak moc velká vzdálenost bude vyplněná v okolí křivky, z důvodu nepřesného odhadu souřadnic x, y při výběru křivky pomocí kurzoru
- ZmenMenenyBod(x, y:integer) – vypočítá vzdálenost k nejbližšímu řídícímu bodu křivky, nastaví měněný bod na nejbližší bod

- ZmenBodXY(x,y :integer) – nastaví souřadnici x a y aktuálnímu nastavenému měněnému bodu

4.7.3 TVýkres

V objektu TVykres uchovávám veškeré informace o výkresu. Tato datová struktura nám umožňuje jednoduché ukládání, načítání a vykreslování všech křivek. Ale také přidání nové křivky nebo odebrání křivky.



Obrázek č. 4.1 UML diagram reprezentující object TVykres

Tato třída je řídící strukturou mého programu. Opět představím pouze nějaké hlavní metody a atributy, které jsou ve výkresu základním kamenem mé aplikace.

Private atributy:

- pocet_krivek – udává počet křivek ve výkresu
- krivka – pole typu PKrivka(ukazatel na TKrivka),přidání křivky přidá adresu do tohoto pole
- pozadi – typu TBitmap, v případě, že je nastaveno, tak vykreslování křivek předchází načtení tohoto pole

Public metody:

- Constructor init_vykres
- Destructor deinit_vykres
- Init_krivky – nastaví všechny prvky pole krivka na nil
- Pridej_krivku – přidá křivku na konec pole
- Kresli – vykreslí všechny prvky pole krivka
- KresliRidicibody(kam:TBitmap,podmínka:integer) – vykresli do bitmapy kam řídící body křivky
- ProhodKrivky(index:integer) - vymění prvek pole krivka s indexem index, za poslední prvek v poli křivka
- Hledej(x,y :integer):integer – hledá podle souřadnic x a y, v atributu polepixelu objektu TKrivka, nastavenou 1. V případě, že je nalezena, vrací index křivky kde je tato 1 nastavena. Používám tuto funkci při vyhledávání křivky pomocí kurzoru.

4.7.4 Lagrangeuv polynom

Lagrangeuv polynom je křivka interpolační, musí procházet všemi body.

```
for ax:=round(krivka.vratbodx(1)) to round(krivka.vratbodx(pocetbodu)) do
 begin
 y:=0;
 for k:=1 to pocetbodu do
  begin
   cit:=1;
   jmen:=1;
   for j:=1 to pocetbodu do
    begin
     if k<>j then
      begin
       jmen:=jmen*(krivka.vratbodx(k)-krivka.vratbodx(j));
       cit:=cit*(ax-krivka.vratbodx(j));
      end;
    end;
   if (cit<>0) and (jmen<>0) then y:=y+krivka.vratbody(k)*(cit/jmen);
  end;
   krivka.SetPixel(round(real2pixx(ax)),round(real2pixy(y)));
   pom_bitmap.Canvas.LineTo(round(real2pixX(ax)),round(real2pixY(y)));
  if poslednikrivka=krivka then
  akt_krivka_bitmap.Canvas.LineTo(round(real2pixX(ax)),round(real2pixY(y)));
   pom_obr_bitmap.Canvas.LineTo(round(real2pixX(ax)),round(real2pixY(y)));
  end;
  form1.img_obr.Canvas.Draw(0,0,pom_bitmap);
  form1.img_oblast.Canvas.Draw(0,0,pom_obr_bitmap);
  if poslednikrivka=krivka then form1.pntbx.Canvas.Draw(0,0,akt_krivka_bitmap);
```

Obrázek č. 4.2. Algoritmus Lagrangeova polynomu

To, co není vidět v tomto algoritmu pro správné fungování, je pouze nastavení kurzoru v jednotlivých komponentách TBitmap, kam chceme kreslit, na počáteční bod. A nastavení barvy pera na červenou. Algoritmus není nic jiného, než zapsání vzorce pomocí podmínkových funkcí do programu.

Červená přímka na obrázku 3., označuje část programu, která zaručuje, že část algoritmu ohraničená příkazy begin a end proběhne pro všechny body x mezi počátečním a koncovým bodem, to znamená možnost zjištění y pro každý x.

Kombinace podmínek označené modrou a žlutou přímkou, vyjadřuje zápis vzorce, podle kterého počítáme lagrangeuv polynom.

Světle modrá přímka označuje vlastní tělo programu, které počítá čitatele a jmenovatele, nutný pro výpočet souřadnice y.

Poslední část algoritmu je už jen vykreslení aktuálně vypočítané souřadnice y. Nastavení pole pixelu křivce pro výběr křivky z výresu. A pomocí metody draw vykreslení bitmapy do TImage.

4.7.5 Fergusonova křivka

Fergusonova křivka, je křivka interpolační, určují ji dva body a dva vektory. Viz kapitola 3.

```
detail:=0.0001;
i:=1;
pom_obr_bitmap.Canvas.Pen.Color:=clRed;
pom_bitmap.Canvas.Pen.Color:=clRed;
if posledniKrivka=krivka then akt_krivka_bitmap.Canvas.Pen.Color:=clRed;
for j:=1 to (krivka.GetPocBodu div 2)-1  do
  begin
    t:=0.0;
   while t<=1 do
     begin
          t3:=power(t,3);
          F1:=2*t3 - 3*sqr(t) + 1;
          F2:=-2*t3 + 3*sqr(t);
          F3:= t3 - 2*sqr(t) + t;
          F4:= t3 - sqr(t);
x:=krivka.vratbodx(i)*F1+krivka.vratbodx(i+2)*F2+(krivka.vratbodx(i+1)
-krivka.vratbodx(i))*F3+(krivka.vratbodx(i+3)-krivka.vratbodx(i+2))*F4;
y:=krivka.vratbody(i)*F1+krivka.vratbody(i+2)*F2+(krivka.vratbody(i+1)
-krivka.vratbody(i))*F3+(krivka.vratbody(i+3)-krivka.vratbody(i+2))*F4;
          roundX:=round(real2pixX(x));
          roundY:=round(real2pixY(y));
          krivka.SetPixel(roundx,roundy);
          pom_obr_bitmap.Canvas.LineTo(roundX,roundY);
          pom_bitmap.Canvas.LineTo(roundX,roundY);
          if posledniKrivka=krivka then akt_krivka_bitmap.Canvas.LineTo(roundX,roundY);
          t:=t+detail;
     end;
    inc(i,2);
  end;
```

Obrázek č. 4.3. Algoritmus Fergusonovy křivky

I zde existuje část kódu, který není vidět, u každé křivky je to stejné, je to nastavení před kreslením. To znamená nastavení kurzoru v bitmapě na první bod a nastavení kreslící barvy.

Modrá barva ukazuje výpočet Hermitovských polynomů. Které pak vynásobím určitými řídícími body. Což ukazuje světle modrá přímka. Žlutá

přímka představuje část kódu kde vše je závislé na parametru *t*. Podle vzorce musí být v rozmezí 0 až 1. Protože při dosazení *t=0* tak by nám měl vyjít počáteční bod, a *t=1* nám vyjde koncový bod křivky. Červená přímka reprezentující *for* cyklus, zaručuje že celý kód probíhá pro (počet bodů děleno 2)-1.

4.7.6 Bézierova kubika

Bézierova kubika je tvořená ze 4 řídících bodů, je to křivka aproximační. Začíná v prvním a končí v posledním bodě.

```
detail:=0.001;
i:=1;
for j:=1 to akt_krivka.GetPocBodu do
begin
t:=0.0001;
  pom_obr_bitmap.Canvas.MoveTo(roundpocX,roundpocY);
  pom_bitmap.Canvas.MoveTo(roundpocX,roundpocY);
  if posledni krivka=akt_krivka then
begin
  akt_krivka_bitmap.Canvas.MoveTo(roundpocX,roundpocY);
end;
 while t<=1 do
  begin
  x:=akt_krivka.vratbodx(i)*power((1-t),3)+akt_krivka.vratbodx(i+1)*3*t*sqr(1-t)+
  akt_krivka.vratbodx(i+2)*3*sqr(t)*(1-t)+akt_krivka.vratbodx(i+3)*power(t,3);
  y:=akt_krivka.vratbody(i)*power((1-t),3)+akt_krivka.vratbody(i+1)*3*t*sqr(1-t)+
  akt_krivka.vratbody(i+2)*3*sqr(t)*(1-t)+akt_krivka.vratbody(i+3)*power(t,3);
  roundX:=round(real2pixX(x));
  roundY:=round(real2pixY(y));
  akt_krivka.SetPixel(roundx,roundy);
  pom_obr_bitmap.Canvas.Pen.Color:=clRed;
  pom_obr_bitmap.Canvas.LineTo(roundX,roundY);
  pom_bitmap.Canvas.Pen.Color:=clRed;
  pom_bitmap.Canvas.LineTo(roundX,roundY);
  t:=t+detail;
  end;
end;
```

Obrázek č. 4.4 Algoritmus Bezierovy kubiky

Červená přímka označující for cyklus, ukazuje zdrojový kód, který proběhne pro každý řídící bod. Modrá přímka představuje zdrojový kód, jehož provedení závisí na parametru t, ke kterému je při každém proběhnutí tohoto algoritmu přičten detail, parametr musí být vždy menší nebo roven 1. Zelená přímka představuje, vypočtení souřadnic x a y závislé na parametru x a na řídícím bodu. A zbytek kódu je pouze vykreslení do TBitmap a přičtení detailu.

4.7.7 Kubický Coonsův B-splajn

kubický Coonsův B-splajn je křivka aproximační, ale neprochází ani jedním zadaným řídícím bodem.

```
for i:=4 to krivka.pocet_bodu do
begin
u:=0;
prirustek:=0.0001;
while u<=1 do
  begin
    Be03:=-1/6*power(u,3)+1/2*power(u,2)-1/2*u+1/6;
    Be13:=1/2*power(u,3)-power(u,2)+2/3;
    Be23:=-1/2*power(u,3)+1/2*power(u,2)+1/2*u+1/6;
    Be33:=1/6*power(u,3);
    x:=krivka.vratbodx(i-3)*Be03+krivka.vratbodx(i-2)*Be13+krivka.vratbodx(i-1)*Be23+
    krivka.vratbodx(i)*Be33;
    y:=krivka.vratbody(i-3)*Be03+krivka.vratbody(i-2)*Be13+krivka.vratbody(i-1)*Be23+
    krivka.vratbody(i)*Be33;
    roundX:=round(real2pixX(x));
    roundY:=round(real2pixY(y));
    krivka.SetPixel(roundx,roundy);

    if u=0 then
    begin
    pom_obr_bitmap.Canvas.MoveTo(roundX,roundY);
    pom_bitmap.Canvas.MoveTo(roundX,roundY);
    if posledni krivka=krivka then akt_krivka_bitmap.Canvas.MoveTo(roundX,roundY);
    end;
    u:=u+prirustek;
    pom_obr_bitmap.Canvas.LineTo(roundX,roundY);
    pom_bitmap.Canvas.LineTo(roundX,roundY);
    if posledni krivka=krivka then akt_krivka_bitmap.Canvas.LineTo(roundX,roundY);
  end;
end;
```

Obrázek č. 4.5 Algoritmus Kubický Coonsův B-splajn

Modrá přímka ohraničuje for cyklus, pomocí kterého zaručujeme vytvoření spojité křivky z n bodů. Protože spojitou křivku zajistíme pokud, každý následující segment křivky je tvořen třemi řídícími body předchozího segmentu a jednoho nového bodu.

Červená přímka ohraničuje zdrojový kód, který probíhá, dokud proměnná u je menší nebo se rovná 1, na konci každého proběhnutého cyklu se parametr *t* zvýší o *přírustek*. Zelená přímka ukazuje hlavní část algoritmu, která spočívá ve vypočtení polynomů a vynásobením řídícími body. Tím nám vyjde bod *x* a *y* do kterých následně vykreslíme přímku pomocí příkazu *LineTo (x , y)*.

5. Principy používané u výřezů oblasti ohraničené křivkami

Zásadní problém je ten, jak poznat jestli je určitý pixel uvnitř oblasti, kterou ohraničují křivky. Máme zadané křivky parametricky, na první pohled nejsme schopni určit, jak bude křivka probíhat. Tím pádem přesně určit, který pixel je nebo není uvnitř dané oblasti.

Nejprve si musíme uvědomit, jak uchováváme křivky, jestli je pouze vykreslujeme, nebo je vykreslíme a poté uložíme do nějaké struktury. V mém programu je hlavní řídící objekt TVykres, ten má jako atribut pole křivek.

Tím vzniká první řešení pro každý pixel vypočítat, jestli je ohraničen křivkou z každé strany. Toto řešení je samozřejmě na první pohled nevhodné, bylo velmi časově náročné.

Další možnost je ta, že si pomůžeme dvojrozměrným polem bytů. Kde bude 1bit znázorňovat 1 pixel na výkresu. Zde budeme pomocí bitového posunu, tyto jednotlivé bity nastavovat. Jednotlivé bity můžeme, vzhledem k úspoře časové náročnosti, nastavovat při vykreslení křivky. Toto řešení je již vhodné, ale vzhledem k lepší přehlednosti jsem zvolil ještě drobek jiné řešení.

Moje aplikace využívá pomocný atribut typu TBitmap. Tam já si nejprve vykreslím křivky.

Obrázek č. 4.6 Takhle může vypadat pomocná bitmapa vykreslená do TImage

Následující krok je zavedení „semínka“ do oblasti ohraničené křivkami. Tím se spustí následující algoritmus.

```
vysledek:=kontrola(x,y,pom_bitmap);
  if vysledek= true then
  begin
    Seed_Fill(x,y,pom_obr_bitmap);
    kopiruj(foto_img,orig_bitmap,pom_obr_bitmap);
    pom_obr_bitmap:=orig_bitmap;
  end
  else
  MessageDlg('Neklik jste do oblasti ohraničené křivkami!'
  ,mtWarning,mbOKCancel,1);
```

Obr.3.6. Algoritmus který se provede po kliknutí do oblasti ohraničené křivkami.

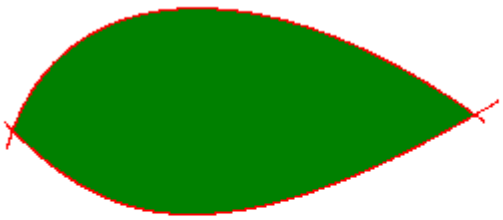
Kontrola(x,y,pom_obr_bitmap) je funkce která vrací true pokud je od bodu x a y směry nahoru, dolu, vlevo a vpravo nalezena hraniční barva, která je v našem případě červená. Pokud ano provede se ta část programu mezi begin a end následující po podmínce if. Pokud ne je zobrazena chybová zpráva.

Seed_Fill(x,y,pom_obr_bitmap) je procedura, která v proměnné pom_obr_bitmap vybarví rekurzivně všechny bílé pixely.

```
hranicni_barva:=clred;
vyplnovaci_barva:=clred;
 if ((x>1)and(y>1))and((x<(kam.Width-1))and(y<(kam.Height-1)))then
begin
  skutecna_barva := kam.Canvas.Pixels[x, y];
 if (skutecna_barva <> hranicni_barva)
 and (skutecna_barva <> vyplnovaci_barva) then
   begin
      if ((x>1)and(y>1))and((x<(kam.Width-1))
      and(y<(kam.Height-1)))then kam.Canvas.Pixels[x, y]:=clred;
     begin
     Seed_Fill (x - 1, y,pom_obr_bitmap);
     Seed_Fill (x + 1, y,pom_obr_bitmap);
     Seed_Fill (x, y - 1,pom_obr_bitmap);
     Seed_Fill (x, y + 1,pom_obr_bitmap);
     end;
    end;
 end
 //continue;
else exit;
```

Obrázek č. 4.7. Algoritmus Seed_Fill

Takto může vypadat pomocná proměnná pom_obr_bitmap po vykreslení do TImage.



Obrázek č. 4.8 pom_obr_bitmap po vykreslení do TImage.

Poslední složitější krok vypadá následovně. Procedura Kopiruj(foto_img,orig_bitmap,pom_obr_bitmap) a jeho algoritmus jsou:

```
sirka:=form1.img_obr.Width;
vyska:=form1.img_obr.Height;
for by:=1 to vyska do
  for bx:=1 to sirka do
    begin
      if pom.Canvas.Pixels[bx,by]=clgreen then
        begin
          barva:=odkud.Canvas.Pixels[bx,by];
          kam.Canvas.Pixels[bx,by]:=barva;
        end
        else if pom.Canvas.Pixels[bx,by]=clred then
        begin
          barva:=clwhite;
          kam.Canvas.Pixels[bx,by]:=barva;

        end;
    end;
Form1.img_oblast.canvas.draw(0,0,cisty_bitmap);
Form1.img_oblast.canvas.draw(0,0,kam);
```

Obrázek 4.9. algoritmus kopiruj

Tato procedura nedělá nic jiného, než když z proměnné pom čte pixel po pixelu a hledá jestli je clgreen, pokud ano, zkopíruje tento pixel z proměné odkud(originální obrázek) do proměné kam(náše vybraná oblast). Pokud daný pixel je clred, vykreslí do cílové bitmapy tento pixel bíle.

Výsledný obrázek může vypadat takto:

Obrázek č. 5 Výsledný obrázek

6. Závěr

V dnešní době kdy je velká snaha použít vektorovou grafiku téměř ve všech odvětvích počítačové grafiky, a to aktuálně platí hlavně v případě grafiky na webu(SVG formát), proto je dobré znát o vektorovém formátu základní věci, které nám umožní lépe pochopit pozdější vykreslování na obrazovku a objasní značné výhody.

Cílem této práce je ukázat, jak se dají naprogramovat základní křivky, které jsou nezbytnou součástí často velmi drahých komerčních výukových grafických editorů. U všech použitých křivek je zároveň k dispozici jejich algoritmus, s popisem nejdůležitějších částí. Při návrhu vlastního kreslícího programu je důležité si uvědomit, jak budeme uchovávat křivky. V této práci nabízím vlastní řešení pomocí tříd, s popisem nejdůležitějších atributů či metod. Můj program také nabízí vybrat rastrovou oblast, která je ohraničena křivkami, tuto možnost také ukazuji postupně pomocí ukázek zdrojového kódu z mého programu. Program je naprogramován v prostředí Borland Delphi professional, toto prostředí jsem ve zkratce ukázal a také představil použité komponenty a jejich metody usnadňující práci s grafikou.

Zdroje:

[1] Alexandr, L. (2006). *Výuka počítačové grafiky cestou WWW*. Získáno 15. 4 2007, z http://lubovo.misto.cz/_MAIL_/curves/obsah.html

[2] Root.cz. (2007). *Vektorový grafický formát DXB a binární varianta DXF*. Získáno 10. 5 2007, z http://www.root.cz/clanky/vektorovy-graficky-format-dxb-a-binarni-varianta-dxf/

[3] Root.cz. (2007). *Vytváříme křivky v PostScriptu*. Získáno 20. 5 2007, z http://www.root.cz/clanky/vytvarime-krivky-v-postscriptu/

[4] Wikipedie, o. e. (1. 5 2007). *Křivka*. Získáno 10. 5 2007, z http://cs.wikipedia.org/wiki/K%C5%99ivka

[5] Wikipedie, o. e. (2006). *Křivost Křivky*. Získáno 8. 5 2007, z http://cs.wikipedia.org/wiki/K%C5%99ivost_k%C5%99ivky

[6] Wikipedie, o. e. (2006). *Rastrová grafika*. Získáno 1. 5 2007, z http://cs.wikipedia.org/wiki/Rastrov%C3%A1_grafika

[7] Wikipedie, o. e. (2006). *Vektorová grafika*. Získáno 1. 5 2007, z http://cs.wikipedia.org/wiki/Vektorov%C3%A1_grafika

[8] Žára, J., Beneš, B., Sochor, J., & Felkel, P. (2004). *Moderní počítačová grafika.* Brno: Computer Press ISBN 80-251-0454-0.

[9] www.km.sjf.stuba.sk. (2006). *APROXIMAČNÉ A INTERPOLAČNÉ ČIARY*. Získáno 10. 5 2007, z http://www.km.sjf.stuba.sk/Geometria/PREDNASKY/prednaska4.htm

Seznam obrázků

ÚDAJE PRO KNIHOVNICKOU DATABÁZI

Název práce	Principy výřezu oblastí v bitmapových grafických editorech
Autor práce	Lubor Mejsnar
Obor	Informační technologie
Rok obhajoby	2007
Vedoucí práce	RNDr. Rak Josef
Anotace	Studium aproximačních křivek, demonstrace komerčních programů a využití křivek při výběru oblastí. Vlastní program na výběr oblastí.
Klíčová slova	Splajn, bézierovy křivky, fergusonova křivka, spjan, popis křivek, algoritmus křivek, Illustrator, Incscape, Seed_fill, výřez oblasti, interpolace, aproximace, barevná hloubka, delphi, algoritmus.