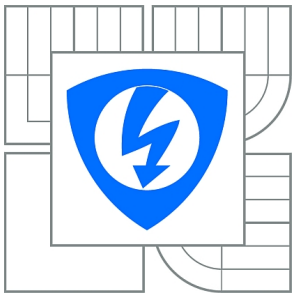
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

VYTVOŘENÍ INTERAKTIVNÍCH STUDIJNÍCH POMŮCEK PRO VÝUKU POČÍTAČOVÉ GRAFIKY

INTERACTIVE STUDY UTILITIES FOCUSED ON COMPUTER GRAPHICS

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

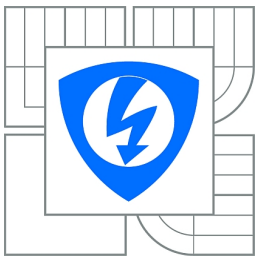
AUTOR PRÁCE — JAKUB MALINA

AUTHOR

VEDOUCÍ PRÁCE — Mgr. PAVEL RAJMIC, Ph.D.

SUPERVISOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student: Jakub Malina ***ID:*** 106613
Ročník: 3 ***Akademický rok:*** 2009/2010

NÁZEV TÉMATU:

Vytvoření interaktivních studijních pomůcek pro výuku počítačové grafiky

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s teorií 2D a 3D grafiky a podrobně s algoritmizací některých úloh dle výběru vedoucího práce (modelování Bézierových křivek, algoritmus de Casteljau pro jejich vykreslování, napojování křivek, modelování ve 3D, vykreslování a transformace objektů).

Vytvořte sbírku interaktivních webových JAVA appletů, které budou zaměřeny na jmenované principy a budou sloužit ke zlepšení úrovně výuky.

DOPORUČENÁ LITERATURA:

[1] Žára, J. a kol.: Moderní počítačová grafika. Druhé vydání. Computer Press, Brno, 2004. ISBN 80-251-0454-0
[2] Farin, G.: Curves and Surfaces for CADG. Fifth edition. Academic Press, 2002. ISBN 1-55860-737-4

Termín zadání: 29.1.2010 ***Termín odevzdání:*** 2.6.2010

Vedoucí práce: Mgr. Pavel Rajmic, Ph.D.

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Abstrakt

V této bakalářské práci se budeme zabývat popisem základních vlastností počítačových křivek a jejich praktickou použitelností. Vysvětlíme si, jak lze křivky chápat obecně, co to jsou polynomiální křivky a způsoby napojování. Poté se zaměříme na popis Bézierových křivek, hlavně pak na Bézierovy kubiky. Podrobněji probereme některé stěžejní algoritmy, které se používají pro vykreslení těchto křivek na počítačích, a ukážeme si jejich praktickou interpretaci. Cílem bakalářské práce je vytvoření souboru interaktivních Java appletů, simulujících některé metody a algoritmy probírané v teoretické části. Tyto applety napomůžou snazšímu pochopení teoretických poznatků a zefektivní výuku. Přednášející i studenti VUT v Brně si budou moci applety vyzkoušet na internetových stránkách.

Klíčová slova

Počítačové křivky, Bézierovy kubiky, de Casteljau, 2D grafika, napojování křivek, polynomiální křivky, stupeň křivky, interpolace, aproximace, Java, GUI, komponenty, applet

Abstract

In this thesis we focus on the basic properties of computer curves and their practical applicability. We explain how the curve can be understood in general, what are polynomial curves and their composing possibilities. Then we focus on the description of Bézier curves, especially the Bézier cubic. We discuss in more detail some of the fundamental algorithms that are used for modelling these curves on computers and then we will show their practical interpretation. The aim of the thesis is the creation of the set of interactive Java applets, simulating some of the methods and algorithms discussed in the theoretical part. These applets will help facilitate understanding of theoretical knowledge and will make the teaching more effective. Lecturers and students of the Brno University of Technology will be able to test these applets on web pages.

Keywords

Computer curves, Bézier cubic, de Casteljau, 2D graphics, composition of the curves, polynomial curves, degree of the curve, interpolation, approximation, Java, GUI, components, applet

MALINA, J. *Vytvoření interaktivních studijních pomůcek pro výuku počítačové grafiky*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2010. 26 s. Vedoucí bakalářské práce Mgr. Pavel Rajmic, Ph.D.

Děkuji vedoucímu práce Mgr. Pavlu Rajmicovi, Ph.D. za velmi užitečnou metodickou pomoc a cenné rady při zpracování bakalářské práce.

V Brně dne ....

podpis autora

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Tématem předložené bakalářské práce je vytvoření interaktivních studijních pomůcek pro výuku počítačové grafiky. Při výuce počítačové grafiky jsou často vysvětlovány abstraktní pojmy, které lze jen stěží pochopit bez vizuálních pomůcek. Interaktivní applety to mohou usnadnit. Jejich přínos bude spočívat hlavně v grafické názornosti při výuce a ke snadnějšímu pochopení jinak složitých matematických aparátů pro vykreslování křivek na počítačích. Přednášející i studenti na VUT v Brně budou moci měnit v těchto appletech hodnoty a sledovat simulaci různých průběhů počítačových křivek.

Tato práce vychází ze semestrálního projektu, který se zabýval podobným tématem. Jeho výsledkem bylo vytvoření interaktivního appletu pro vykreslení Bézierových křivek třetího stupně. Tento applet zde bude rozšířen o další funkce a přibudou i další applety. Soubor těchto appletů celistvě popisuje problematiku počítačových křivek a jejich základních vlastností.

V rámci této práce se postupně seznámíme s teorií 2D a 3D grafiky a podrobně si prohlédneme algoritmizaci některých úloh, například modelování Bézierových křivek, algoritmus de Casteljau nebo algoritmus pro zvýšení stupně křivky. V další části probereme problematiku 2D modelování v jazyce Java a knihovny, které modelování umožňují.

1.1 Struktura práce

S nástupem počítačů a počítačové grafiky se začaly uplatňovat různé techniky vykreslování vektorové grafiky.

Druhá kapitola 2D grafika a Bézierovy křivky se zabývá 2D grafikou, obecným vysvětlením některých vlastností křivek, jejich napojováním, algoritmem de Casteljau a asi nejznámějšími polynomiálními křivkami ve 2D i 3D grafice, Bézierovými křivkami. Kapitola je členěna do několika podkapitol: Křivky obecně, Křivky a jejich základní vlastnosti, Způsoby napojování křivek, kde se popisuje několik základních způsobů napojení jednotlivých segmentů na sebe, díky čemuž jsme schopni vytvářet i velmi složité křivky. Dále se seznámíme v podkapitole Definice polynomiálních křivek s křivkami, které patří mezi nejpoužívanější druhy v počítačové grafice. Následující dvě podkapitoly Bézierovy křivky a Algoritmus de Casteljau jsou postaveny na znalostech z předchozích kapitol, které jsou pro jejich pochopení nezbytné. Bézierovy křivky jsou tématem zasahujícím takřka do celé bakalářské práce. Podrobně si je ale probereme v podkapitole Bézierovy křivky. Podkapitola Algoritmus de Casteljau se zabývá jedním ze způsobů výpočtu Bézierových křivek. Tato podkapitola zahrnuje i metody pro rozdělení křivky na dvě vzájemně napojené a vykreslování křivek postupným dělením. Poslední dvě podkapitoly popisují zvýšení a snížení stupně křivky, kde jsou vysvětleny algoritmy realizující tuto problematiku.

Třetí kapitola Java je zaměřená na praktické využití jazyka Java při modelování výukových ukázek. Podkapitola Grafické uživatelské rozhraní otevírá téma vykreslování grafického uživatelského prostředí, jako jsou okna a rozvržení. Také jsou zde popsány rozdíly mezi knihovnami AWT a Swing. Na ni navazuje podkapitola Návrhy a rozvržení

okna GUI, která toto téma rozšiřuje a dále popisuje jednotlivá rozvržení okna nezávislá na platformě či operačním systému. V podkapitole Grafické komponenty v Javě jsou popsány jednotlivé komponenty využité pro realizaci Java appletů.

Praktická část je kapitola, kde si představíme Java applety vytvořené v rámci této bakalářské práce. Najdeme zde popis funkčnosti i návod k ovládání appletů.

Celá bakalářská práce je shrnuta v páté kapitole nazvané Závěr. Zde se seznámíme s výsledky, kterých bylo dosaženo.

2 Bézierovy křivky a 2D grafika

2.1 Křivky obecně

Pokud mluvíme o křivce a jejích vlastnostech, budeme ji chápat fyzikálně, jako dráhu pohybujícího se bodu, ať už ve dvourozměrném či trojrozměrném prostoru. V moderní počítačové grafice používáme křivky v mnoha oblastech. Jejich využití je možné nalézt např. v technických oborech, jako jsou strojírenství nebo architektura. Setkáváme se s nimi při modelování ve dvou i ve třech rozměrech, při definici a vykreslování počítačových fontů, při určování dráhy pohybujících se objektů v počítačové animaci a při tvorbě animovaných filmů.

2.2 Křivky a jejich základní vlastnosti

Pro výpočet aktuální pozice pohybujícího se bodu, který modeluje křivku, používáme tři různá matematická vyjádření: explicitní (2.1), implicitní (2.2) a parametrické (2.3). V praxi je nejčastěji využíván matematický zápis pomocí parametrického vyjádření. Jeho výhodou je závislost pouze na parametru *t*, který je fyzikální interpretací času.

$$y = f(x) \tag{2.1}$$

$$F(x, y) = 0 \tag{2.2}$$

$$Q(t)\begin{cases} x = x(t) \\ y = y(t) \end{cases}, \text{kde } t \text{ je čas}, t \in \langle t_{\min}, t_{\max} \rangle \tag{2.3}$$

Souřadnice pohybujícího se bodu jsou funkcemi parametru *t*. Parametr *t* je proměnná zastupující čas a je v intervalu od $t_{\min}$ do $t_{\max}$. Promítnutím rovnic (2.4) na osu *x* a (2.5) na osu *y* dostaneme bodové vyjádření průběhu křivky *Q(t)* v čase *t*, viz rovnice (2.6). Vektorové vyjádření udává polohový vektor v čase *t*, jehož velikost je rovna vzdálenosti bodu *Q(t)* od počátku.

$$x = x(t) \tag{2.4}$$

$$y = y(t) \tag{2.5}$$

$$Q(t) = [x(t), y(t)] \tag{2.6}$$

$$\vec{q}(t) = [x(t), y(t)] \Rightarrow \vec{q}(t) = Q(t) - [0,0] \tag{2.7}$$

První derivací polohového vektoru dostaneme směr tečny ke křivce v bodě $Q(t_0)$. Derivace polohového vektoru se provádí derivacemi jednotlivých složek.

$$\vec{q}'(t_0) = [x'(t_0), y'(t_0)] = \left[\frac{\mathrm{d}x(t_0)}{\mathrm{d}t}, \frac{\mathrm{d}y(t_0)}{\mathrm{d}y}\right] \tag{2.8}$$

Nyní jsme schopni pomocí tečného vektoru a bodu na křivce vypočítat rovnici tečny, tj. přímky, která se v tomto bodě křivky dotýká. Zápis rovnice (2.9) je pro bod $Q(t_0)$ v čase t_0. Proměnná s náleží do množiny reálných čísel.

$$p(s) = Q(t_0) + s.\vec{q}'(t_0) \tag{2.9}$$

Druhá derivace polohového vektoru nám udává zrychlení, což je změna okamžité rychlosti pohybu po křivce. Zrychlení, tj. zakřivení, označujeme $\vec{q}''$.

$$\vec{q}''(t_0) = [x''(t_0), y''(t_0)] = \left[\frac{\mathrm{d}x^2(t_0)}{\mathrm{d}t}, \frac{\mathrm{d}y^2(t_0)}{\mathrm{d}y}\right] \tag{2.10}$$

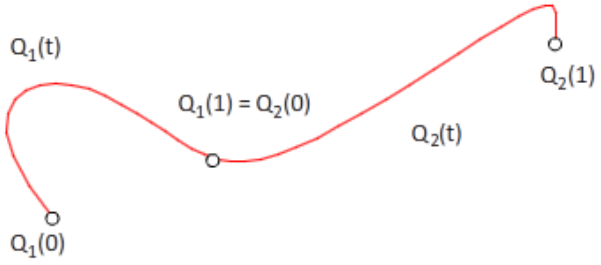
Ze základních vlastností zbývá už jen definovat inflexní bod I křivky $Q(t)$. Inflexní bod je takový bod, kde vektory zrychlení a okamžité rychlosti jsou rovnoběžné.

$$\vec{q}'(t_0) = k.\vec{q}''(t_0), \text{ kde } k \neq 0 \tag{2.11}$$

Některé informace v této podkapitole byly čerpány z [1] a [2].

2.3 Způsoby napojování křivek

V technické praxi se často setkáváme se složitými křivkami, které jsou složeny z několika na sebe napojených křivek. Díky možnosti napojování křivek jsme schopni vykreslovat i velmi složité obrazce. Ukázku napojení několika křivek na sebe můžeme vidět na obrázku 2.1, kde koncový bod jedné křivky je zároveň počátečním bodem křivky následující. Těmto bodům budeme říkat uzly.



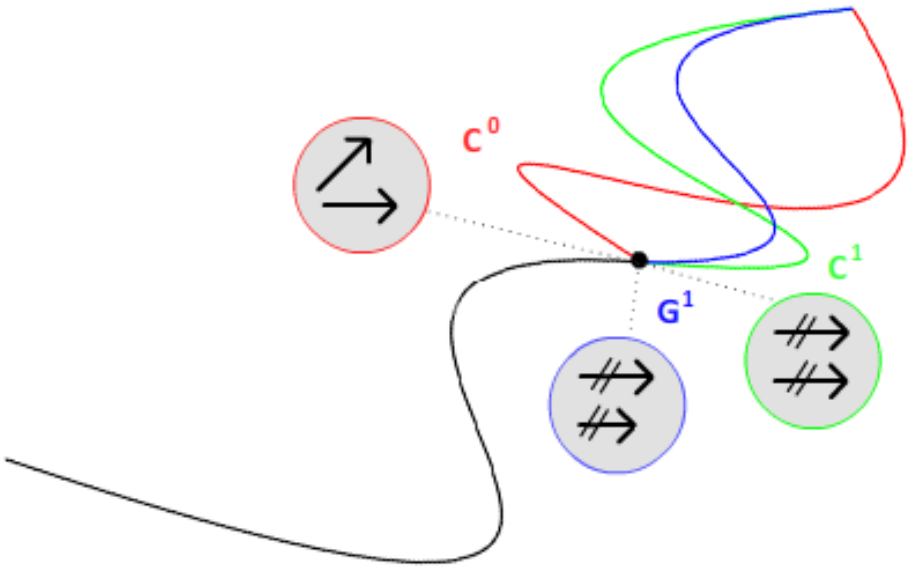
Obr. 2.1: Napojování křivek.

Křivky můžeme napojovat různými způsoby. Pokud chceme dosáhnout určitého stupně hladkosti napojení, nebude nás zajímat pouze rovnost počátečního a koncového bodu, ale především tzv. spojitost v uzlu. Říkáme, že dvě křivky napojené na sebe jsou parametricky nebo také C^k spojitě na sebe navázány, pokud v jejich společném uzlu jsou shodné vektory všech derivací až do řádu k, kde pod proměnnou k rozumíme první, druhou až k-tou derivaci polohového vektoru. To znamená, že k náleží do množiny přirozených čísel. Čím vyšší spojitost je požadována, tím delší dobu, ve smyslu parametru t, se obě křivky k sobě přimykají. Například ze spojitosti třídy C^0 plyne, že bod se pohybuje po křivce, ale v uzlu napojení může náhle změnit směr. Spojitost C^1 nám zase říká, že bod nemůže náhle změnit směr pohybu ani velikost okamžité rychlosti, ale zrychlení změnit může. Při spojitosti C^2 se nezmění směr, okamžitá rychlost ani zrychlení, tj. zakřivení. [2]

Mnohdy se při prvním pohledu na spojení dvou křivek nebo segmentů zdá, že křivky jsou spojeny parametricky, ale při bližší analýze zjistíme, že tečné vektory sice mají stejný směr, ale jejich rychlost a zrychlení se mění. Takové napojení nazýváme geometrická spojitost a značíme ji G^n. [2]

$$q_1(1) = kq_2(0), \text{ kde } k > 0 \tag{2.12}$$

Pro geometrickou spojitost třídy G^1, která se používá nejčastěji, je typická různá délka vektorů. Pohybující se bod po křivce tedy nemůže náhle změnit směr, ale rychlost a zrychlení ano. Příklad jednotlivých spojitostí vidíme na obrázku 2.2, kde můžeme vidět grafickou podobnost hladkosti napojení mezi G^1 a C^1. V praxi bývá často snazší realizovat geometrické napojení G^1 než parametrické C^1. Spojitost C^1 implikuje G^1, obráceně však nikoli. [1]



Obr. 2.2: Rozdíly mezi parametrickou spojitostí nultého stupně C^0, prvního stupně C^1 a geometrickou spojitostí prvního stupně G^1.

Spojitost G^2 je definována jako shoda prvních křivostí ${}^1k$ v uzlu obou segmentů, kde tzv. první křivost ${}^1k$ se spočítá podle vztahu (2.14). [1]

$$ {}^1k_{Q1}(1) = {}^1k_{Q2}(0) \tag{2.13} $$

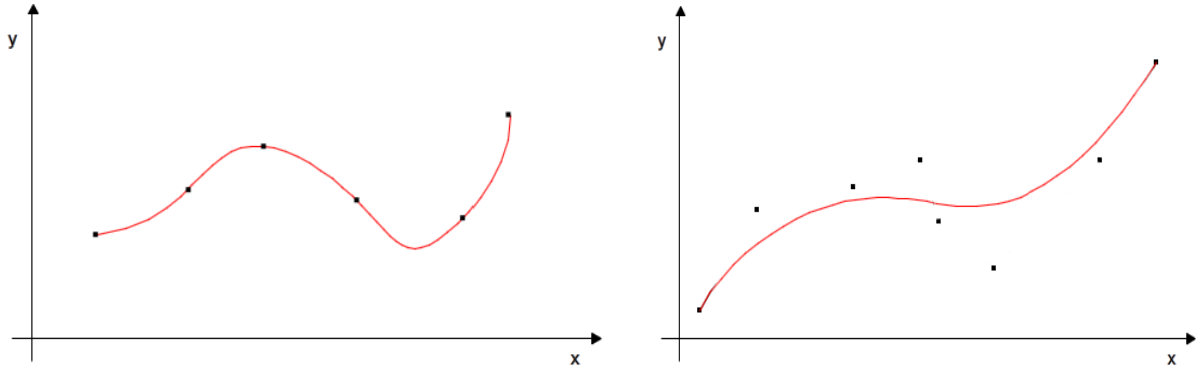
$$ {}^1k(t_0) = \sqrt{\frac{|\vec{q}'(t_0) \times \vec{q}''(t_0)|^2}{|\vec{q}'(t_0).\vec{q}'(t_0)|^3}} \tag{2.14} $$

2.4 Definice polynomiálních křivek

Základním a nejčastěji používaným druhem křivek v počítačové grafice jsou křivky polynomiální, které se dají velice rychle vypočítat a jsou snadno diferencovatelné. Pro snadnou manipulovatelnost, dostatečně širokou škálu tvarů, snadnost výpočtu a možnost zaručit parametrickou spojitost C^2 se používají polynomy 3. stupně, také nazývané kubiky. Křivky vyššího stupně se příliš nepoužívají z důvodu vzniku nežádoucího vlnění a složitosti jejich výpočtu.

Modelování křivek provádíme zvolením několika, čtyř v případě kubik, řídících bodů, na jejichž základě se pomocí matematického výpočtu provede vykreslení křivky. Jinou metodou je určení počátečního a koncového bodu křivky a poté zvolení tečných vektorů v těchto bodech. Při napojování křivek také máme možnost nastavovat hladkost navázání.

Existují dva základní způsoby interpretace řídících bodů, a to interpolace a aproximace. Při interpolaci generovaná křivka prochází danými body, zatímco při aproximaci je řídícími body tvar křivky určen, ale křivka jimi procházet nemusí.



Obr. 2.3: Křivky a jejich řídící body: a) interpolační, b) aproximační.

Ve (2.15) vidíme příklad matematicky zadané kubiky:

$$ Q(t) \quad \begin{cases} x = x(t) = a_x t^3 + b_x t^2 + c_x t + d_x \\ y = y(t) = a_y t^3 + b_y t^2 + c_y t + d_y \end{cases} \tag{2.15} $$

To lze zkráceně zapsat takto:

$$Q(t) = \boldsymbol{TC} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix}. \tag{2.16}$$

Derivací $\boldsymbol{T}$ získáme směrový vektor.

$$\vec{q}'(t) = \frac{\mathrm{d}}{\mathrm{d}t} Q(t) = \frac{\mathrm{d}}{\mathrm{d}t} \boldsymbol{TC} = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \boldsymbol{C} \tag{2.17}$$

Matice $\boldsymbol{C}$ obsahuje parametry, které ovlivňují tvar křivky. Jejich zadávání je však neintuitivní a neefektivní, proto ji rozepíšeme na součin $\boldsymbol{MG}$.

$$\boldsymbol{C} = \boldsymbol{MG} \Rightarrow Q(t) = \boldsymbol{TMG} \tag{2.18}$$

Nyní máme matici $\boldsymbol{M}$, také nazývanou bázová matice o rozměru 4x4, která je daná zvolenou metodou vykreslení křivek a určuje způsob jejich výpočtu. Pro Bézierovy kubiky je matice $\boldsymbol{M}$ daná takto:

$$\boldsymbol{M} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}. \tag{2.19}$$

Další maticí, která vznikla rozepsáním $\boldsymbol{C}$, je matice $\boldsymbol{G}$. Ta se nazývá matice geometrických podmínek nebo geometrický vektor. Reprezentuje vliv vnějších parametrů a obsahuje řídící body nebo tečné vektory. Velikost matice geometrických podmínek je pro kubiku 4x2.

Nakonec si můžeme vyjádřit obecný zápis rovnice pro výpočet *Q(t)*:

$$Q(t) = \boldsymbol{TMC} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}. \tag{2.20}$$

Součin $\boldsymbol{TM}$ definuje polynomiální bázi, tj. skupinu polynomů, která je společná pro všechny křivky určitého typu. Matice $\boldsymbol{G}$ potom obsahuje parametry ovlivňující tvar křivky, např. řídící body nebo kombinaci řídících bodů a tečných vektorů.

Na křivky jsou často kladeny určité požadavky:

- Invariance vůči lineárním transformacím a projekcím.
- Křivka leží v konvexní obálce svých řídících bodu.
- Změna polohy jednoho řídícího bodu změní tvar křivky jen v okolí tohoto bodu a nezmění se křivka celá.
- Křivka má procházet krajními body svého řídícího polygonu.

Křivky, se kterými se v počítačové grafice setkáváme, mohou být buď racionální anebo neracionální. Racionální křivky jsou vyjádřeny v homogenních souřadnicích a vyskytují se u nich nenulové váhy řídících bodů. Neracionální vyjádření je tedy zvláštním případem racionálních křivek, kde jsou všechny váhy stejné a rovné jedné. Neracionální křivky nejsou invariantní k perspektivnímu promítání, zatímco racionální ano. [2]

Některé informace v této podkapitole byly čerpány z [1] a [2].

2.5 Bézierovy křivky

Mezi nejznámější aproximační křivky v počítačové grafice patří právě Bézierovy křivky. Teoretický základ těchto křivek vytvořil na přelomu 50. a 60. let P. E. Bézier, když vyvíjel programový nástroj UNISURF pro návrh křivek a ploch u francouzské firmy Renault. Jejich častým využitím je definice počítačových fontů. Nejdříve popíšeme obecné Bézierovy křivky *n*-tého stupně a poté si představíme nejčastěji používanou variantu kubiky.

2.5.1 Bernsteinovy polynomy

Bézierovy křivky mohou být definovány jako rekurzivní algoritmus, který vynalezl de Casteljau. Je však také nutné mít pro ně explicitní reprezentaci, což nám značně usnadní další teoretický rozvoj. [3] Proto si nyní představíme Bernsteinovy polynomy. Následujících pět rovnic nám definuje základní vlastnosti Bernsteinových polynomů:

$$B_{k,n}(0)=\begin{cases}1 & \text{pro } k=0\\ 0 & \text{jinak},\end{cases} \qquad B_{k,n}(1)=\begin{cases}1 & \text{pro } k=n\\ 0 & \text{jinak},\end{cases} \tag{2.21}$$

$$\forall k,n \in N \cup \{0\} \text{ a } t \in \langle 0,1\rangle \text{ je } B_{k,n}(t) \geq 0, \tag{2.22}$$

$$\sum_{k=0}^{n}(t)B_{k,n} = 1 \text{ pro } t \in [0,1], \tag{2.23}$$

$$B_{k,n}(t) = (1-t)B_{k,n-1}(t) + tB_{k-1,n-1}(t), \tag{2.24}$$

$$B_{k,n-1}(t) = \frac{n-k}{k} B_{k,n}(t) + \frac{k+1}{n} B_{k+1,n}(t), \tag{2.25}$$

$$\frac{\mathrm{d}}{\mathrm{d}t} B_{k,n}(t) = n\left[B_{k-1,n-1}(t) - B_{k,n-1}(t)\right]. \tag{2.26}$$

Druhý vztah zaručuje nezápornost Bernsteinových polynomů, tj. jejich funkční hodnoty jsou větší nebo rovny nule. Druhý a třetí pak zaručují, že výsledná křivka bude vždy ležet v konvexní obálce bodů řídícího polygonu. Čtvrtý vztah je rekurentní definicí Bernsteinova polynomu stupně n pomocí lineární kombinace dvou po sobě následujících Bernsteinových polynomů stupně $n-1$. [2]

Pro praktickou ukázku si definujeme Bernsteinovy polynomy pro kubiku, tj. pro $n = 3$. Jejich grafické vyjádření můžeme vidět v grafu na obrázku níže.

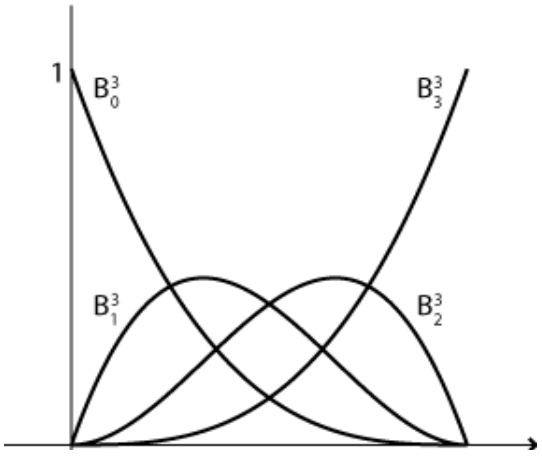
$$B_{0,3}(t) = (1-t)^3 \tag{2.27}$$

$$B_{1,3}(t) = 3t\,(1-t)^2 \tag{2.28}$$

$$B_{2,3}(t) = 3t^2\,(1-t) \tag{2.29}$$

$$B_{3,3}(t) = t^3 \tag{2.30}$$

Obrázek 2.4 ukazuje skupinu čtyř Bernsteinových polynomů pro třetí stupeň, kubiku. Je dobré poznamenat, že křivky Bernsteinových polynomů jsou v celém intervalu od $t_{\min}$ do $t_{\max}$ nenegativní, tj. nacházejí se v kladném kvadrantu a nenabývají záporných hodnot.



Obr. 2.4: Bernsteinovy polynomy pro kubiku. [9]

2.5.2 Bézierovy křivky obecně

Pro Bézierovu křivku n-tého stupně definujeme $n+1$ řídících bodů P_k. Bézierovy křivky jsou vystavěny na tzv. Bernsteinových polynomech, kde Bézierovu křivku definujeme takto:

$$Q^{BEZ}(t) = \sum_{k=0}^{n} P_k . B_{k,n}(t), \tag{2.31}$$

kde P_k je $n+1$ řídících bodů a $B_{k,n}(t)$ jsou Bernsteinovy polynomy n-tého stupně a vypočítáme je dle vztahu (2.32).

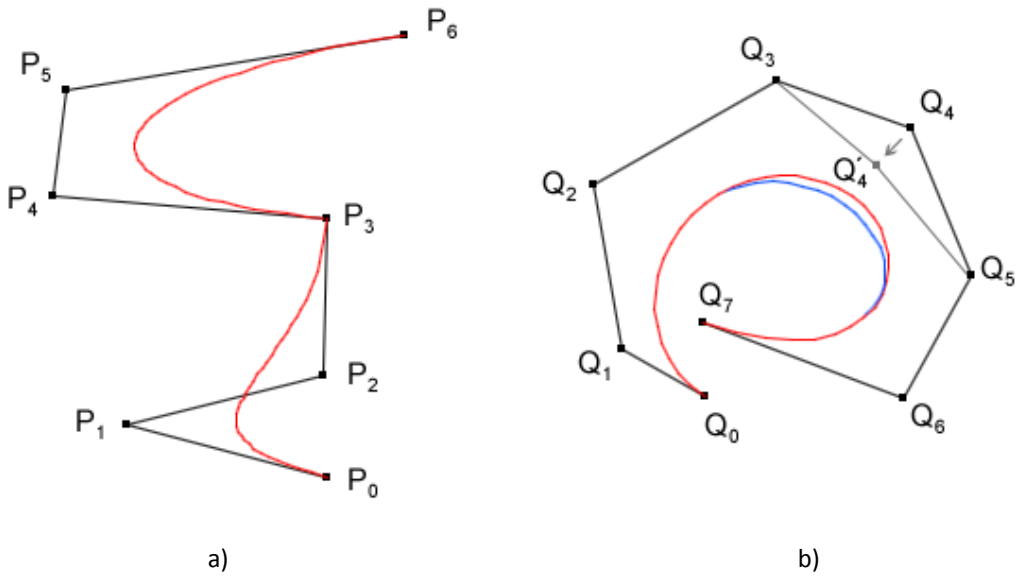
$$B_{k,n}(t) = \binom{n}{k} t^k \, (1-t)^{n-k}, \tag{2.32}$$

kde $t \in [0,1]$, $k = 0, 1, ..., n$. Při výpočtu tohoto vztahu je $\binom{n}{0} = 1$ a $0^0 = 1$.

$$p(0) = n(P_1 - P_0) \tag{2.33}$$

$$p(1) = n(P_n - P_{n-1}) \tag{2.34}$$

Vložíme-li do vztahu (2.31) parametr $t = 0$, resp. $t = 1$, snadno zjistíme, že křivka prochází prvním a posledním bodem řídícího polygonu, viz výrazy (2.33) a (2.34) pro tečné vektory v krajních bodech. To je jedna ze společných vlastností pro všechny Bézierovy křivky. Dále o nich můžeme říci, že výsledná křivka leží vždy v konvexní obálce bodů řídícího polygonu. [2]



a) b)

Obr. 2.5: Bézierovy křivky: a) 2 křivky 3. stupně napojené v bodě P_3, b) křivka 7. Stupně před editací a po editaci bodu Q_4.

Vlastností Bézierových křivek je, že při změně polohy jednoho řídícího bodu P_k dojde ke změně tvaru celé křivky a k nutnosti překreslit celý její průběh. Tato vlastnost je jedním z důvodů, proč se v praxi křivky dělí na segmenty vzájemně na sebe napojených Bézierových křivek nižšího stupně. [2] Praktický příklad můžeme vidět na obrázku výše. Vlevo vidíme dvě na sebe napojené křivky, které se vzájemně neovlivňují, zatímco u křivky 7. stupně vpravo je patrné, že změna polohy jednoho z bodů, změní tvar celé křivky.

2.5.3 Bézierovy kubiky

Vzájemně na sebe navázané segmenty složitějších křivek jsou nejčastěji tvořeny Bézierovými kubikami, tj. křivkami třetího stupně. Jsou zadány čtyřmi body P_0, P_1, P_2 a P_3 a začínají v prvním řídícím bodu P_0 a končí v posledním P_3. Při napojování kubik jsme schopni zaručit spojitost až do třídy stupně C^2, což je pro použití v počítačové grafice dostatečné.

Pro vykreslení Bézierovy kubiky můžeme využít tří způsobů: výpočet přes maticový zápis, pomocí algoritmu de Casteljau s konstantním krokem k anebo využití dělení křivky pomocí algoritmu de Casteljau. První způsob je popsán rovnicí (2.35) a navazuje na předchozí pochopení teorie o polynomiálních křivkách, která je popsána v kapitole 2.4. Maticový zápis pro Bézierovy kubiky vypadá následovně:

$$Q(t) = TMC = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}. \tag{2.35}$$

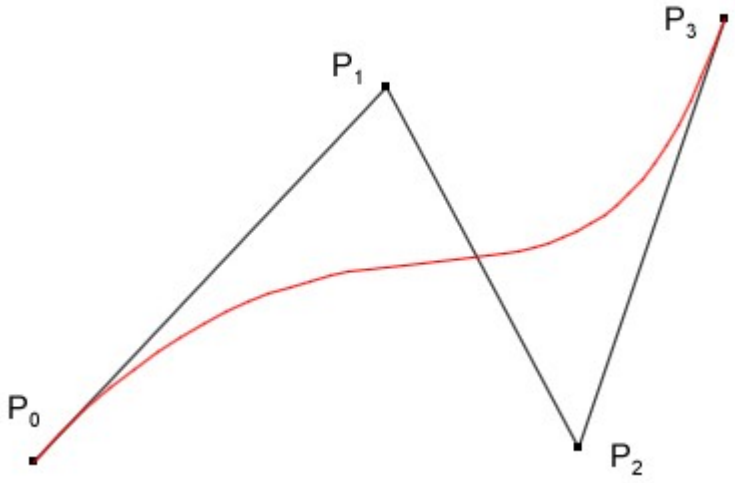
Tečné vektory v prvním a posledním bodě mají tvar:

$$p(0) = 3(P_1 - P_0), \tag{2.36}$$

$$p(1) = 3(P_3 - P_2). \tag{2.37}$$

Druhým a třetím způsobem se budeme zabývat v kapitole 2.6 nazvané příkladně Algoritmus de Casteljau. Na závěr kapitoly si ukážeme praktickou ukázku Bézierovy křivky třetího stupně. Na obrázku 2.6 je červenou čarou vykreslena kubika a černými čarami znázorněna konvexní obálka.

Některé informace v této podkapitole byly čerpány z [1], [2] a [3].



Obr. 2.6: Bézierova kubika se svými řídícími body.

2.6 Algoritmus de Casteljau

Algoritmus popsaný v této kapitole patří mezi nejzákladnější výpočetní techniky pro křivky. Jeho hlavní přínos spočívá v jednoduchosti a velice intuitivní souhře mezi geometrickou konstrukcí a algebraickými výpočty. Díky tomuto algoritmu jsme schopni určit pozici bodu v určitém časovém okamžiku v intervalu od t_{min} do t_{max}. Praktické využití nacházíme například při vykreslování Bézierových křivek. [3]

2.6.1 Praktický příklad: vykreslení paraboly

Jako příklad vytvoříme jednoduchou konstrukci pro vykreslení paraboly. Zvolíme si tři řídící body P_0, P_1, P_2 a parametr t, který náleží do množiny reálných čísel. Následující tři rovnice nám pomůžou k určení pozice bodu P_0^2, která je závislá na čase t.

$$P_0^1(t) = (1-t)P_0 + tP_1 \tag{2.38}$$

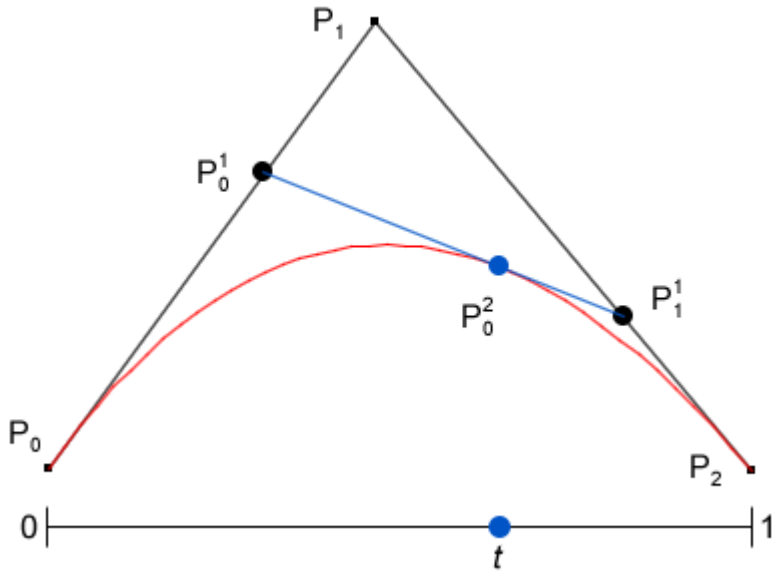
$$P_1^1(t) = (1-t)P_1 + tP_2 \tag{2.39}$$

$$P_0^2(t) = (1-t)P_0^1(t) + tP_1^1(t) \tag{2.40}$$

Vložením rovnic (2.38) a (2.39) do (2.40) dostaneme následující rovnici pro okamžitý výpočet pozice P_0^2:

$$P_0^2(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2. \tag{2.41}$$

Opakováním lineární interpolace, ilustrované na obrázku 2.7 níže dostaneme parabolu vykreslenou pohybujícím se bodem P_0^2 po dráze mezi body P_0^1 a P_1^1.



Obr. 2.7: Parabola sestrojená pomocí konstrukce algoritmu de Casteljau.

Na tomto obrázku vidíme princip vykreslení paraboly. Na úsečce mezi dvěma řídícími body vytvoříme nový bod, který spojíme úsečkou s dalším nově vytvořeným bodem mezi dalšími dvěma řídícími body. Tento cyklus probíhá tak dlouho, dokud nevytvoříme jednu jedinou úsečku, která je na obrázku vyznačena modře a na níž leží hledaný bod. V případě tří řídících bodů, tedy pro $n=2$, vytvoříme poslední úsečku už ve druhém kroku, nicméně můžeme si představit, že stejným principem vykreslujeme křivku s mnohem více řídícími body.

Příklad a některé informace v této podkapitole byly čerpány z [3].

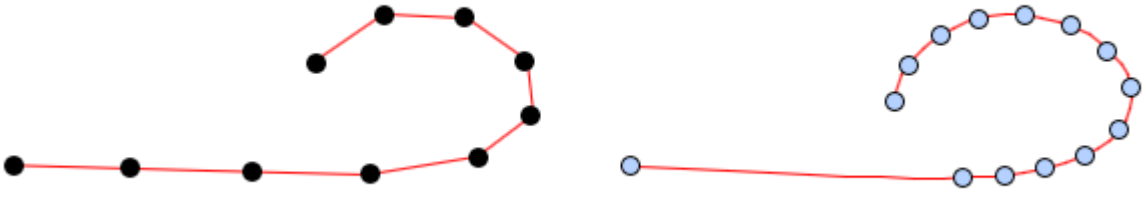
2.6.2 Algoritmus de Casteljau obecně

Algoritmus de Casteljau lze zapsat v rekurentním vztahu takto:

$$P_i^j(t) = (1-t)P_i^{j-1} + tP_{i+1}^{j-1}, \tag{2.42}$$

kde j nese hodnotu cyklu a i hodnotu pořadí bodu v daném cyklu,
volíme: $j = i, i+1, i+2, \ldots, n$ a $i = 1, 2, \ldots, n$.

Při vykreslování bodu na křivce můžeme použít dva postupy. Jedním z nich je rozdělení intervalu času od $t_{\min}$ do $t_{\max}$ s konstantním krokem k, např. $k = 0{,}05$. Nevýhodou tohoto způsobu je, že nehledě na rovnost či zakřivení křivky je krok k vždy stejný, a proto dochází k nahrazení rovné části mnoha úsečkami, zatímco zakřivená část je aproximována příliš hrubě.



a) b)

Obr. 2.8: Ukázka vykreslení křivky: a) po krocích, b) metodou postupného dělení.

Na obrázku 2.8 je zřejmé, že vykreslení křivky s konstantním krokem k není nejefektivnější způsob. Obrázek vlevo ukazuje vykreslení křivky konstantním krokem k, který je znázorněn černými body. Křivka je aproximována stejně v zakulacené části jako v části rovné. Pravý obrázek ilustruje efektivní způsob vykreslení, kde zakulacená část je aproximována více úsečkami než její rovná část. Výhodou je, že tato metoda generuje menší množství dat než metoda s konstantním krokem k.

2.6.3 Vykreslování křivky postupným dělením

Jiným způsobem vykreslování je postupné dělení křivky na dvě poloviny. Pokud je jedna z polovin dostatečně rovná, tj. vyhovuje předdefinovaným mezím, pak se aproximuje jednou úsečkou. Pokud není, pokračuje se na ní v dělení. Kritériem pro dostatečně rovnou část může být tloušťka čáry nebo délka úhlopříčky pixelu. V takovém případě je již další dělení zbytečné.

V počítačové grafice můžeme dělit křivku až do velikosti jednoho pixelu, další dělení už nemá smysl, protože pixel je nejjemnější rastr, který je možné zobrazit. Vstupem pro algoritmus dělení jsou čtyři řídící body P_0, P_1, P_2 a P_3 a výstupem pak dvě čtveřice L_0, L_1, L_2, L_3 a R_0, R_1, R_2, R_3, které můžeme vyjádřit následujícími rovnicemi:

$$L_0 = P_0, \tag{2.43}$$

$$L_1 = \frac{(P_0 + P_1)}{2}, \tag{2.44}$$

$$tmp = \frac{(P_1 + P_2)}{2}, \tag{2.45}$$

$$L_2 = \frac{(L_1 + tmp)}{2}, \tag{2.46}$$

$$R_3 = P_3, \tag{2.47}$$

$$R_2 = \frac{(P_2 + P_3)}{2}, \tag{2.48}$$

$$R_1 = \frac{(tmp + R_2)}{2}, \tag{2.49}$$

$$L_3 = R_0 = \frac{(L_2 + R_1)}{2}. \tag{2.50}$$

2.6.4 Rozdělení křivky na dvě vzájemně napojené

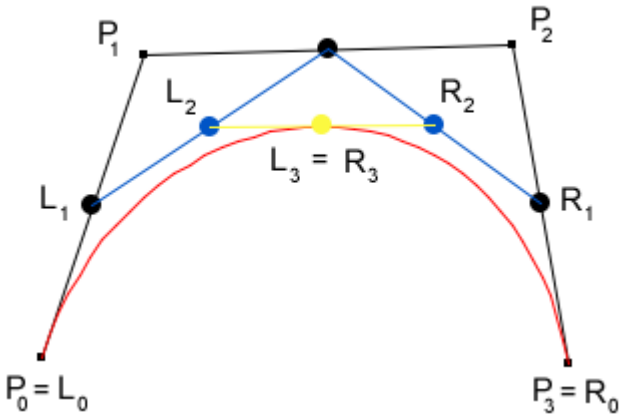
Algoritmus de Casteljau může sloužit také k rozdělení křivky na dvě na sebe napojené části v libovolném místě, tj. pro zvolenou hodnotu parametru $t \in \langle 0, 1 \rangle$, aniž by přitom došlo ke změně původního tvaru.

Původní křivka je vykreslena v konvexní obálce řídících bodů $P_0, ..., P_n$. Při dělení křivky vytvoříme dvě skupiny řídících bodů $L_0, ..., L_n$ a $R_0, ..., R_n$. Jak můžeme předpokládat, bod L_0 bude roven bodu P_0 a bod R_n bodu P_n. Vzájemné napojení křivek je v bodech L_n a R_n. To si můžeme ověřit na obrázku 2.9. Řídící body dvou nových křivek $L_0, ..., L_n$ a $R_0, ..., R_n$ získáme pomocí těchto vztahů:

$$L_i = \sum_{j=0}^{i} \binom{i}{j} \frac{P_j}{2^i}, \text{ pro } i = 0,1,\ldots,n, \tag{2.51}$$

$$R_i = \sum_{j=i}^{ni} \binom{n-i}{n-j} \frac{P_j}{2^{n-i}}, \text{ pro } i = 0,1,\ldots,n. \tag{2.52}$$

Výše uvedený výpočetní postup pro vytvoření nových řídících bodů je realizován půlením úseček. Pokud tento postup opakujeme, konverguje polygon určený řídícími body k Bézierově křivce. Každé další dělení generuje dva nové řídící polygony, které jsou přesnějšími aproximacemi této křivky. [2]



Obr. 2.9: Rozdělení křivky pomocí algoritmu de Casteljau.

Využití nacházíme v situacích, kdy potřebujeme editovat jen část křivky a přitom zachovat ostatní části nezměněny.

2.6.5 Zvýšení stupně křivky

Další metodou, jak se vypořádat s nedostatečným počtem řídících bodů, je zvýšení stupně křivky o jednu. Cílem je vytvoření nového řídícího bodu tak, aby nedošlo ke změně tvaru původní křivky. Tím dosahujeme vetší flexibility při modelování.

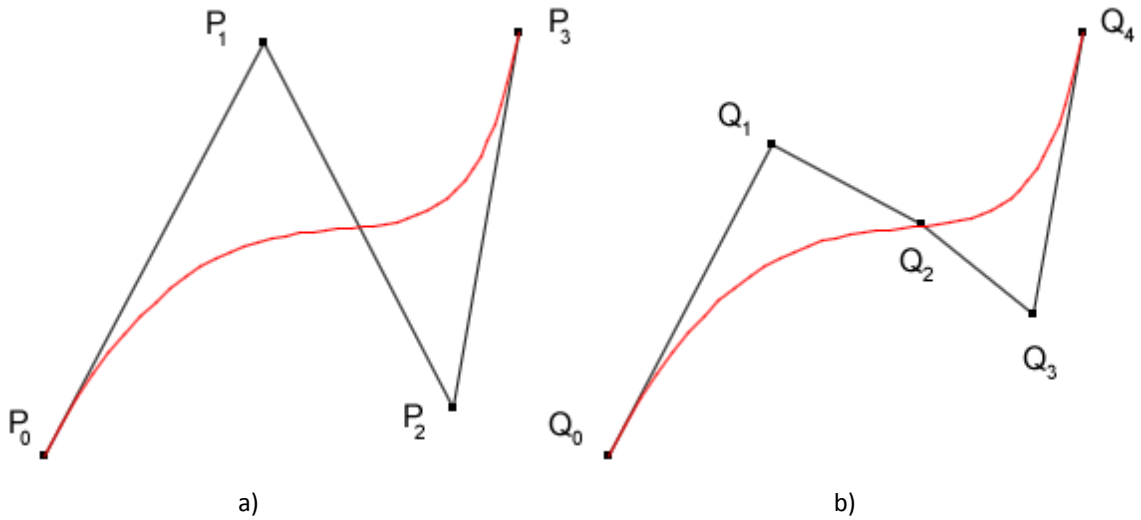
Hledáme křivku s řídícími body P_0 až P_{n+1}, která opisuje stejnou dráhu jako křivka výchozí s řídícími body P_0 až P_n. Toho dosáhneme přepočtením původních bodů a přidáním dalšího. Aby byla zaručena identita křivek, musí pro body nového polygonu platit: [2]

$$Q_i = \eta_i P_{i-1} + (1-\eta_i)P_i, \tag{2.53}$$

$$\eta_i = \frac{i}{n+1}, i = 1, 2, ..., n. \tag{2.54}$$

Přepočítané body jsou označené Q_0, Q_1, Q_3 a Q_4. Bod Q_2 je nově vytvořený bod řídícího polygonu. Vlastnost Bézierových křivek, popsaná v kapitole 2.5.2, zajišťuje, že křivka prochází počátečním a koncovým bodem řídícího polygonu. Tedy pokud máme zajištěnou identitu křivek, nemá smysl přepočítávat první a poslední bod.

Zvýšení stupně křivky hraje důležitou roli v povrchovém designu. Je nezbytné, aby při vytváření povrchů pomocí vkládání křivek byly tyto vkládané křivky stejného stupně. Toho dosáhneme zvýšením stupně všech vkládaných křivek až k té s nejvyšším stupněm. Jedním z dalších využití jsou situace při převodu dat mezi různými CAD/CAM nebo jinými grafickými systémy. Předpokládejme, že máme parabolu, tj. Bézierovu křivku 2. stupně, a chceme ji převést do systému, který zná jen kubiky. Vše, co musíme udělat, je zvýšit stupeň naší paraboly. [3]



a) b)

Obr. 2.10: zvýšení stupně Bézierovy křivky: a) křivka 3. stupně, b) identická křivka 4. stupně.

2.6.6 Snížení stupně křivky

Na zvyšování stupně můžeme nahlížet jako na proces, který představuje nadbytek informací. Křivka je popsána více informacemi, než je vlastně nutné. Opačný proces by se mohl zdát zajímavější, možný nadbytek při vyjádření křivky by bylo možné snížit. Přesněji řečeno, můžeme vykreslit danou křivku stupně $n+1$ jako tu stupně n. Tento proces budeme nazývat snížení stupně křivky.

Jak můžeme předpokládat, přesné snížení stupně není možné. Například pokud máme kubiku se čtyřmi řídícími body, nemůžeme tuto křivku zapsat kvadraticky. Na snížení stupně křivky tedy můžeme nahlížet jako na metodu, jak přiblížit danou křivku nižšímu stupni. Pokud máme danou Bézierovu křivku s řídícími body $P_i, i = 0, ..., n+1$, můžeme se pokusit najít křivku $D_i, i = 0, ..., n$, která se přibližuje té původní rozumným způsobem.

Rovnice pro snížení stupně se mohou zapsat do jedné maticové rovnice takto:

$$\begin{bmatrix} 1 & a_{12} & \cdots & a_{1n} & a_{1(n+1)} \\ a_{21} & a_{22} & \cdots & a_{2n} & a_{2(n+1)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{(n+1)1} & a_{(n+1)2} & \cdots & a_{(n+1)n} & a_{(n+1)(n+1)} \\ a_{(n+2)1} & a_{(n+2)2} & \cdots & a_{(n+2)n} & 1 \end{bmatrix} \begin{bmatrix} P_0 \\ \vdots \\ P_n \end{bmatrix} = \begin{bmatrix} D_0 \\ \vdots \\ D_{n+1} \end{bmatrix}, \tag{2.55}$$

$$D = \boldsymbol{M}P, \tag{2.56}$$

kde $\boldsymbol{M}$ je matice s $n+2$ řádky a $n+1$ sloupci.

Při snižování stupně se snažíme přiblížit křivku stupně $n+1$ křivce stupně n. V rámci (2.56) to znamená, že bychom dostali D a přáli bychom si nalézt B. Je jasné, že to není možné pomocí vyřešení lineárního systému, protože $\boldsymbol{M}$ není čtvercová matice.

Pomůžeme si jednoduchým trikem. Obě strany rovnice (2.56) vynásobíme transponovanou maticí $\boldsymbol{M}^T$ a dostaneme:

$$\boldsymbol{M}^T\boldsymbol{M}B = \boldsymbol{M}^T\boldsymbol{M}D. \tag{2.57}$$

Nyní máme lineární systém pro neznámou D se čtvercovou maticí $\boldsymbol{M}^T\boldsymbol{M}$. Řešení je již jednoduché.

3 Java

Java je objektově orientovaný programovací jazyk, který vyvinula firma Sun Microsystems. Poprvé byl představen 23. května 1995 a postupně se stal jedním z nejpoužívanějších programovacích jazyků na světě. Především díky svým vlastnostem, kterými jsou nezávislost na platformě a přenositelnost, je hojně využíván při programování internetových aplikací, a to jak na straně klienta, tak na straně serveru. Dále je hojně využíván pro mobilní telefony, čipové karty a jiná zabudovaná zařízení, pro něž je prioritou pracovat na různých systémech.

Mezi hlavní výhody jazyka Java nepochybně patří velké množství již vytvořených knihoven, které usnadňují programování tak, že nemusíme řešit, jak danou problematiku realizovat, nýbrž to, co chceme zrealizovat. Při programování se tak dostáváme na vyšší úroveň.

3.1 Grafické uživatelské rozhraní (GUI - Graphical User Interface)

Už od svého vzniku umožňuje jazyk Java práci s grafickým rozhraním, které je nezbytné pro pohodlnou a intuitivní práci s programy i applety. První knihovnou, která poskytla možnost práce s okny, byla Java AWT (Abstract Windowing Toolkit). Poskytovala jen velmi základní komponenty pro komunikaci s uživatelem a velmi jednoduchou práci s 2D grafikou. Její hlavní nevýhodou byla multiplatformní stránka Javy, což znamenalo, že žádná nová vlastnost nemohla být přidána, dokud nebyla implementována na všech hlavních platformách (MS Windows, Linux, Mac OS X). Její dnešní nástupkyní je knihovna Java Swing, uvolněná v roce 1998 jako součást JFC (Java Foundation Classes). Swing je plně funkční, profesionální grafické uživatelské rozhraní, které bylo vytvořeno tak, aby umožnilo téměř všechny formy komunikace s uživatelem. Tato knihovna je v dnešní době (2009) široce používaná, často aktualizovaná a přenositelná. Existují i alternativní knihovny, z nichž můžeme uvést například SWT (Standart Window Toolkit) od společnosti IBM. [4]

Dobrým stylem je používat rady a zásady od společnosti Sun o vzhledu aplikací. Ian F. Darwin, autor knihy *Java kuchařka programátora* (Java Cookbook), doporučuje nahlédnout do knihy *Java Look and Feel Design Guidelines*, která nabízí pohled celé řady specialistů na psychologii a praktický vzhled aplikací. Tito specialisté pracují s knihovnou Java Swing od jejího vzniku a poradí nám, jak s ní nejlépe pracovat. [4]

3.2 Návrhy rozvržení okna GUI

Abychom mohli rozšířit naše programy a aplikace do světa a zajistit jejich stejné nebo alespoň podobně použitelné zobrazení, jsou zapotřebí nástroje, které se postarají o případné odchylky na různých platformách, tedy na PC nebo Apple a jejich operačních systémech. Tuto náročnou práci můžeme přenechat knihovně Java Swing, která podporuje mnoho způsobů rozvržení našich komponent.

Představme si, že vytváříme program pro vykreslení grafů. Navrhneme velikost okna, která je vyhovující pro nás, náš monitor a platformu, na které program vyvíjíme. Na světě ale existují i jiné monitory a jiná rozlišení, než jaké používáme my. Proto by bylo chybou zadávat velikosti komponent v pixelech nebo jiné rozměrové jednotce. Předvedeme si několik způsobů návrhu GUI tak, aby bylo přizpůsobitelné změnám velikostí okna atd.

a) BorderLayout

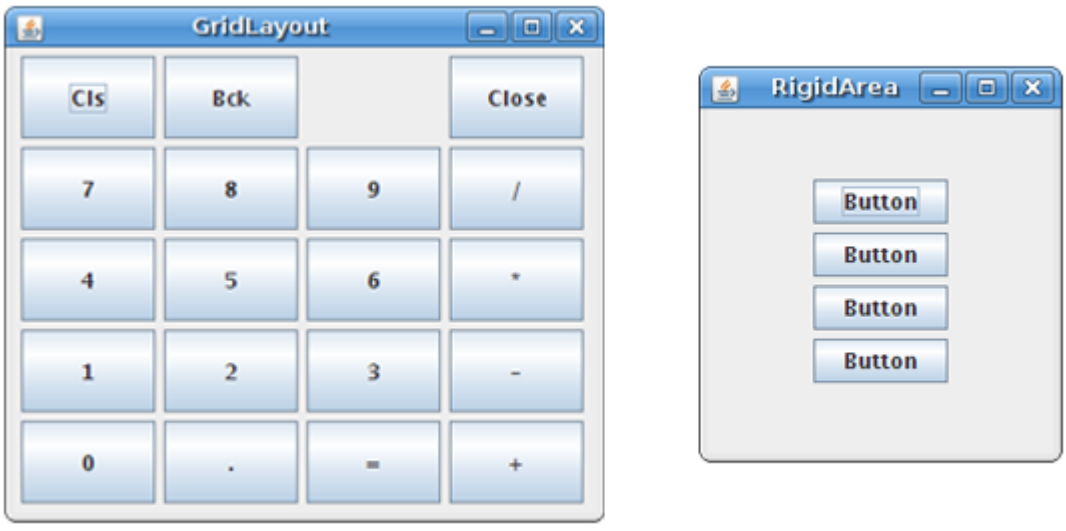
BorderLayout je výchozím rozvržením (layoutem) v komponentách JFrame, JWindow, JApplet. Pomocí Border Layoutu rozdělíme naši grafickou plochu v programu na 5 oblastí: sever (North), jih (South), západ (West), východ (East) a střed (center). Tyto oblasti přizpůsobují své rozměry velikosti okna tak, že vyplní celou plochu. Praktickým příkladem je třeba ovládání pro web kameru nebo obrázek či video, které můžeme vložit do komponenty. [4]

b) FlowLayout

Rozvržení Flow Layout nám umožní vkládat komponenty napříč kontejnerem v osách *x* nebo *y*. Opět přizpůsobí velikost komponent velikosti okna. Je hlavním layoutem komponent JPanel a Applet. Používá se k vytvoření seznamu tlačítek například ve vertikálním směru. Směr zarovnání stejně jako mezery mezi komponentami lze nastavit. [4]

c) GridLayout

Tento typ rozložení vytvoří neviditelnou mřížku, kterou naplní vloženými komponentami. Lze nastavit počet řádků a sloupců mřížky, přičemž počet řádků, je-li nastaven jako nenulový, má přednost. [5]



a) b)

Obr. 3.1: Základní rozvržení: a) Grid Layout [10], b) Flow Layout. [8]

d) Další layouty

Existuje řada dalších rozvržení než ty, které jsme právě popsali. Jejich kompletní výpis najdete v tabulce níže.

Tab. 3.1: Přehled vybraných správců rozvržení v Javě

Název	Popis	Výchozí komponenta
FlowLayout	Umístí komponenty napříč kontejnerem.	JPanel, Applet
BorderLayout	Rozdělí okno na pět oblastí.	JFrame, JWindow, JApplet
GridLayout	Pravidelná mřížka, všechny prvky mají stejnou velikost.	žádná
CardLayout	V jednom okamžiku zobrazí pouze jednu z mnoha komponent; užitečné pro vytváření průvodců.	žádná
gridBagLayout	Velmi flexibilní, ale velice složité rozvržení.	žádná
BoxLayout	Umístí komponenty do jednoho řádku nebo sloupce.	žádná

3.3 Grafické komponenty v Javě

V obou knihovnách, starší AWT a novější Swing, máme k dispozici mnoho grafických komponent. Mezi nejdůležitější a zároveň nejpoužívanější patří: JPanel, pomocí kterého jsme schopni vnořovat komponenty vzájemně do sebe, klasické tlačítko JButton, popisek JLabel a JTextField pro komunikaci s uživatelem pomocí formulářového pole. Pro kreslení ve 2D i 3D je důležitá komponenta JPanel, neboť je široce požívána jako virtuální plátno. V knihovně AWT existuje komponenta JCanvas, ale dnes se již příliš nepoužívá a není obsažena ve Swing.

3.3.1 Těžké a lehké komponenty

Již od začátku byla knihovna AWT koncipována jako rozhraní mezi Javou a grafickými API platformami. Všechny grafické komponenty byly tedy kresleny přímo systémem a spoléhaly se na něj. Tyto nativně vykreslené prvky se nazývají těžké, protože měly vlastní neprůhledné okno systému, a to, jak komponenty vypadají, záleželo na systému.

Knihovna Swing přináší nový přístup, a to ten, že každá komponenta je sama zodpovědná za svůj vzhled a není závislá na systému, což je základní myšlenka jazyka Java jako takového. Každá komponenta sama definuje, jak má vypadat, a při zavolání metody paintComponent() se vykreslí na monitor. Samotné kreslení se provádí přímo v Javě a po jeho dokončení je výsledný obrázek předán operačnímu systému k vykreslení. Tento přístup je tzv. lehký. [6]

3.3.2 Java a 2D Kreslení

Kreslení v Javě provádíme pomocí metody paintComponent(). Při volání metody vkládáme jako parametr objekt Graphics, který následně převádíme na Graphics2D. Tento krok se provádí kvůli zpětné kompatibilitě. K dispozici máme dva druhy počítačové grafiky: rastrovou grafiku, např. obrázek nebo fotka, a vektorovou grafiku, tj. body, čáry, křivky atd. Oba typy mají své výhody a nevýhody. Výhody vektorové grafiky oproti rastrové jsou např. menší velikost dat, možnost zvětšení a zmenšení, pohyby a animace. V Javě máme několik předdefinovaných typů: body, čáry, polygony, kruhy, elipsy a křivky. [7]

a) Metody pro kreslení

Komponentu, která nám simuluje virtuální plátno, vytvoříme přetížením JPanelu. Poté můžeme uvnitř metody paintComponent() začít s kreslením. Stěžejní metodou je draw(), ta nám zajistí vykreslení námi předem definovaných tvarů, křivek a čar. Před touto metodou ovšem voláme metody pro nastavení barev, stylů a nastavení objektů. Jednou z těchto metod je setColor(), která na naše zavolání změní barvu pro vykreslení. Nastavení stylu, síly a zakončení čáry provádíme vytvořením objektu typu BasicStroke, kterému předáme příslušné parametry. Podobně jako se tvoří styl čáry, se také dá vytvořit objekt pro vykreslení, např. čtverec nebo elipsa. Vytvořením příslušných objektů z knihovny Swing Geom jsme připraveni k jejich nakreslení. Pokud chceme nejenom kreslit obrysy prvků a objektů, použijeme metodu fill(). Jednotlivé grafické prvky se vykreslí při spuštění a inicializaci appletu. Pokud kdykoli v průběhu programu chceme překreslit tyto prvky, zavoláme metodu repaint() příslušného objektu. Při změně velikosti okna nebo komponenty se tato metoda volá sama a dojde k překreslení.

b) Akce a události

Další úlohou appletu je reakce na uživatele. Pro interaktivní používání appletu je potřeba definovat pro jednotlivé komponenty tzv. listener. Tento objekt se stará o naslouchání, tj. sledování komponenty, zda na ni nebyla aplikována nějaká definovaná událost. Těmito událostmi mohou být: stisknutí tlačítka, přejetí nebo kliknutí myší, změna výběru atd. V našem appletu toto využíváme při výběru řídících bodů pro vykreslení Bézierovy kubiky.

4 Praktická část

V praktické části bakalářské práce se zaměříme na realizaci Java appletů jako vizuálních pomůcek pro výuku počítačové grafiky. Budou představeny jeden po druhém společně s návodem k jejich ovládání. Pro tuto bakalářskou práci byly vytvořeny tři applety nazvané: Bézierovy kubiky, Algoritmus de Casteljau pro Bézierovy kubiky a Zvýšení stupně křivky. Jsou umístěné na webových stránkách pod hlavičkou Fakulty elektrotechniky a komunikačních technologií jako soubor vizuálních pomůcek pro intuitivnější pochopení problematiky Bézierových křivek.

Ještě před tím, než si představíme první z nich, popíšeme si nezbytné prvky pro jejich funkčnost. Jsou jimi především třída Graf, vytvořená speciálně pro tuto práci.

a) Třída Graf

Třída Graf je vytvořená jako přetížená třída JPanel s implementovaným listenerem událostí. Je naprogramovaná tak, že ji lze velice jednoduše přizpůsobit potřebným podmínkám daného appletu. Například můžeme změnit velikost rastru nebo plochy grafu. Také lze jednoduše vypínat některé vlastnosti, což je využito v prvním appletu, kde pravý horní graf Q(t) je plnohodnotnou instancí třídy Graf, zatímco další dva grafy x(t) a y(t) slouží pouze k zobrazení průběhu, a proto nedisponují uživatelskou interakcí.

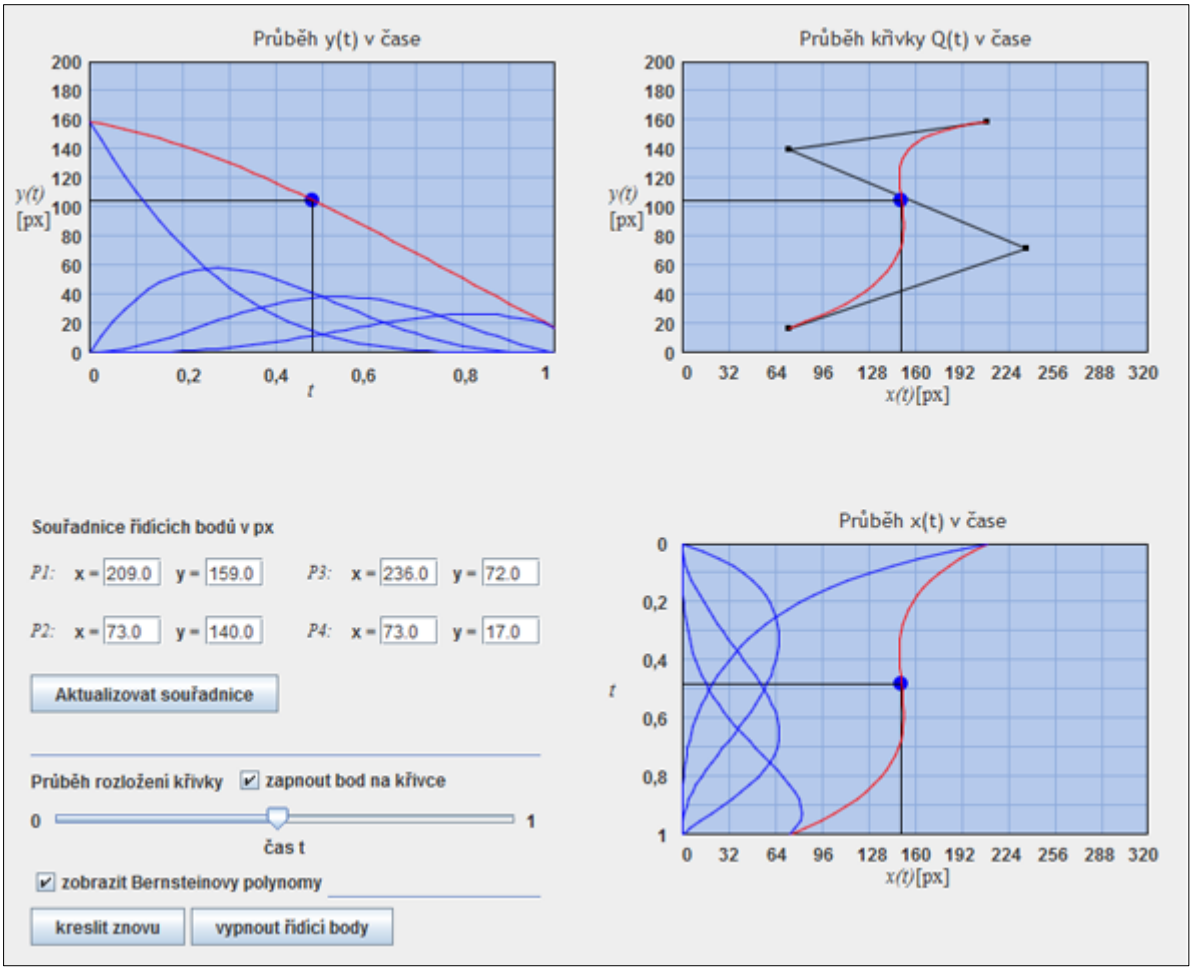
b) Třída Šipka

Tato třída je opět přetížená třída JPanel. Slouží ke grafickému znázornění výpočtu bodů ve 2. appletu Algoritmus de Casteljau pro Bézierovy kubiky. Při její implementaci můžeme měnit velikost, tvar i barvu šipky.

4.1 Applet Bézierovy kubiky

Applet Bézierovy kubiky byl představen již v semestrálním projektu. Nyní ho rozšíříme o další funkce. Do grafu, který je umístěn v pravém horním rohu, přidáváme 4 řídící body pro Bézierovu křivku postupným klikáním kdekoli na ploše grafu. Při určení posledního bodu, tj. čtvrtého, se vykreslí Bézierova kubika s námi zadanými body. Další dva grafy slouží k rozložení křivky na průběh x(t) a y(t) v čase. Zaškrtnutím „zapnout bod na křivce“ zobrazíme bod na všech třech grafech. Táhnutím jezdce simulujeme čas t a můžeme vidět, jak je průběh Q(t) složen z rozložených částí průběhů x(t) a y(t).

Můžeme si zobrazit Bernsteinovy polynomy, jejichž součtem dostáváme funkční hodnoty průběhů x(t) a y(t). Jsou znázorněny modrými čarami pod křivkou. Průběh x(t) je z našeho pohledu otočen, aby intuitivně znázornil skládání Bézierovy kubiky.



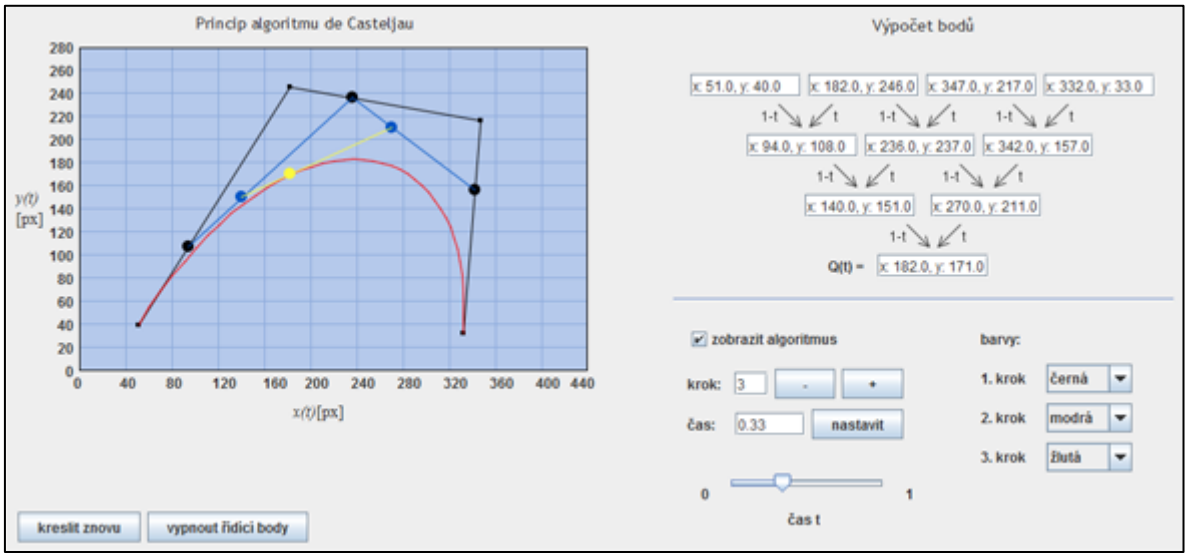
Obr. 4.1: Applet Bézierovy kubiky.

Dalšími intuitivními funkcemi jsou změna polohy řídících bodů a tlačítka „kreslit znovu“ a „vypnout řídící body“. Při změně poloh bodů volíme čísla v mezích os, jinak obdržíme chybovou zprávu. Totéž se stane při pokusu vložit jiné znaky než číslice. Tlačítko „vypnout řídící body“ skryje řídící polygon, abychom mohli lépe pozorovat průběh křivky. Na obrázku 4.1 vidíme tento applet se zapnutým řídícím polygonem i bodem znázorňujícím rozložení křivky.

4.2 Applet Algoritmus de Casteljau pro Bézierovy kubiky

Tento applet graficky znázorňuje postup při vykreslení Bézierovy křivky pomocí algoritmu de Casteljau. Čtyři řídící body zvolíme stejným způsobem jako u předchozího appletu. Grafické znázornění algoritmu de Casteljau zobrazíme zaškrtnutím „zobrazit algoritmus“. Defaultně je navolen 3. krok algoritmu. V případě Bézierových kubik, tj. aproximačních křivek 3. stupně, při 3. kroku vykreslíme bod kopírující naši křivku. Tlačítky „+“ a „–“ můžeme zvyšovat či snižovat krok a sledovat postup při konstrukci algoritmu. Roletkové nabídky vpravo nám umožňují měnit barvy jednotlivých kroků a lépe tak pochopit, jak se kroky postupně konstruují. Posuvník opět slouží k simulaci času t v intervalu od 0 do 1. Pomocí něho překreslujeme konstrukci algoritmu pro zvolený čas t.

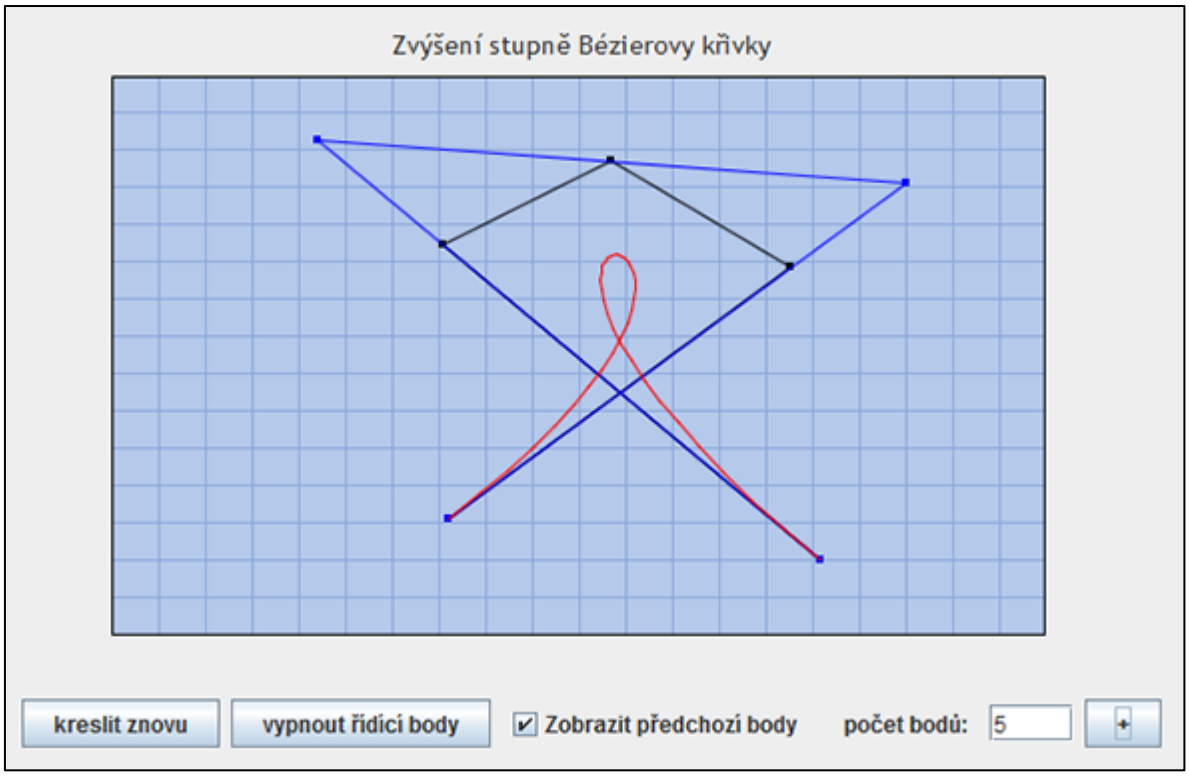
Poslední částí, kterou je třeba vysvětlit, je schéma výpočtu bodů vpravo nahoře. Textová pole jsou zarovnána do převrácené pyramidy a zobrazují souřadnice nově vypočtených bodů z těch námi zadaných. Ty jsou zobrazeny ve čtyřech textových polích úplně nahoře. V každé další vrstvě pak ze dvou vrchních souřadnic bodů počítáme dle (2.39) bod nový. Směr výpočtu je znázorněn šipkami s příslušným násobkem vedle. Již z úvahy je zřejmé, že pokud nastavíme jezdce na nulu, bude mít poslední vypočtený bod, tedy bod vykreslující Bézierovu křivku, stejné souřadnice jako první bod řídícího polygonu. Stejně tak při nastavení času *t* na hodnotu 1, bude poslední bod shodný s posledním řídícím bodem. To je vlastnost Bézierových křivek popsaná v kapitole Bézierovy křivky obecně. Na obrázku 4.2 si můžeme prohlédnout applet se zapnutou konstrukcí algoritmu de Casteljau a vypočtenými souřadnicemi bodů.



Obr. 4.2: Applet Algoritmus de Casteljau pro Bézierovy kubiky.

4.3 Applet Zvýšení stupně Bézierovy křivky

Zvýšení stupně křivky uplatníme, pokud při editaci křivek je řídící polygon utvořen nedostatečným počtem bodů. Tímto algoritmem zvýšíme počet řídících bodů při zachování původní křivky, viz kapitola Zvýšení stupně křivky. Na obrázku 4.3 je applet znázorňující praktickou ukázku tohoto algoritmu. Tlačítkem „+“ vpravo dole můžeme přidávat libovolný počet bodů. V našem případě následkem zaokrouhlování dojde při nadmíře bodů k viditelné deformaci křivky. Standardně zaškrtnuté políčko „zobrazit předchozí body“ nám zachová předešlé řídící body vyznačené modře. Můžeme vidět, že první a poslední bod se nemění, viz teoretický popis v kapitole Bézierovy křivky obecně.



Obr. 4.3: Applet Zvýšení stupně Bézierovy křivky.

4.4 Webové stránky pro applety

Všechny applety jsou dostupné přes libovolný webový prohlížeč s podporou platformy Java, tj. je třeba stáhnout a nainstalovat Java SE Development Kit (JDK). Stránky jsou rozdělené na úvodní stranu a další tři odkazy nesoucí název výše popsaných appletů. Ty jsou dostupné na adrese http://www.simplicityandspace.cz/bc/, kde kromě appletů najdeme i krátkou teorii, popis jejich funkce a návod k ovládání.

Hlavním cílem těchto stránek je seskupit applety a krátce popsat principy Bézierových křivek, algoritmu de Casteljau a algoritmu pro zvýšení stupně, aby byla snadno pochopitelná i pro nezasvěcené návštěvníky.

5 Závěr

V teoretické části bakalářské práce jsme si popsali a vysvětlili pojmy a témata nezbytná k pochopení principů probraných algoritmů, z nichž některé jsou součástí Java appletů.

Výsledkem této bakalářské práce je soubor appletů pro výuku počítačové grafiky na VUT v Brně. Applety si každý může prohlédnout na již zmíněných webových stránkách, kde je možné nalézt i krátký teoretický základ ke znázorněné problematice. Každý z appletů je navržen tak, aby pochopitelně a intuitivně vysvětloval dané téma.

První z appletů, nazvaný Bézierovy kubiky, vysvětluje téma nejznámějších aproximačních křivek třetího stupně v počítačové grafice, Bézierových kubik. Můžeme si prohlédnout, jakým způsobem vytvořit křivku ze dvou různých funkcí závislých na čase a vztah Bézierových křivek k Bernsteinovým polynomům.

Applet Algoritmus de Casteljau pro Bézierovy kubiky nám slouží k pochopení principu algoritmu de Casteljau, který patří mezi nejjednodušší a zároveň nejintuitivnější způsoby výpočtu průběhů Bézierových křivek. Pomocí snadného ovládání můžeme pozorovat vytvoření konstrukce pro námi libovolně zadaný čas a sledovat tak vykreslení křivky. Pro snazší pochopení můžeme postupovat po krocích a sledovat, jak se vytváří jednotlivé body, jejichž souřadnice vidíme ve výpočetním schématu.

Třetí a poslední ze souboru appletů, pojmenovaný Zvýšení stupně Bézierovy křivky, znázorňuje možnost zvýšit stupeň Bézierových křivek pro jejich snazší editaci. Při navýšení stupně je možné si zobrazit původní řídící polygon a lépe tak porozumět, jak tento algoritmus funguje.

Všechny tyto applety svou funkčností poslouží ke zlepšení úrovně výuky počítačové grafiky tak, že si student bude moci ověřit funkčnost matematických rovnic v praxi.

Literatura

1. **Žára, Jiří, Beneš, Bedřich a Felkel, Petr.** *Moderní počítačová grafika.* Brno: Computer Press, 1998. ISBN 80-7226-049-9.

2. **Žára, Jiří, Beneš, Bedřich, Sochor, Jiří a Felkel, Petr** *Moderní počítačová grafika.* Brno: Computer Press, 2004. ISBN 80-251-0454-0.

3. **Farin, Gerald.** *Curves and surfaces for GAGD.* místo neznámé: Academic Press, 2002. ISBN 1-55860-737-4.

4. **Darwin, Ian F.** *Java kuchařka programátora.* místo neznámé: Computer Press, 2006. ISBN 80-251-0944-5.

5. **Jelínek, Lukáš.** Java (29) - správci rozložení. *Linuxsoft.cz.* [Online] 9. 11. 2006. [Citace: 13. 12 2009.] http://www.linuxsoft.cz/article.php?id_article=1353.

6. **neznámý.** Swing (Java). *Wikipedie.* [Online] 6. 6 2009. [Citace: 13. 12. 2009.] http://cs.wikipedia.org/wiki/Swing_(Java).

7. **Bodnar, Jan.** Painting in Swing. *ZetCode.* [Online] 12. 9. 2007. [Citace: 13. 12. 2009.] http://zetcode.com/tutorials/javaswingtutorial/drawing/.

8. **neznámý.** The Swing layout management. *ZetCode.* [Online] 12. 11. 2007. [Citace: 14. 12. 2009.] http://zetcode.com/tutorials/javaswingtutorial/swinglayoutmanagement/.

9. **Kvasnička, Ondřej.** Křivky v počítačové grafice. *Centre of Computer Graphics and Data Visualisation.* [Online] 10. 11 2009. [Citace: 16. 12. 2009.] http://herakles.zcu.cz/education/ZPG/cviceni.php?no=8.

10. **Chapman, Stephen J.** *Začínáme programovat v jazyce Java.* místo neznámé: Computer Press, 2001. ISBN 80-7226-472-9.

1 Seznam přiloh

2 Obsah CD

Na přiloženém CD můžeme nalézt offline verzi internetových stránek s Java applety a stručnou teorií pro detailnější pochopení dané problematiky. Stránky nalezneme ve složce „stránky“ a jsou spustitelné pomocí souboru „index.html“. Stránky jsou rozčleněné do podstránek nesoucí název appletů.

Applety, společně s třídami, které jsou ke spuštění a překladu zapotřebí, jsou zabalené do tzv. JAR archivu. Ten lze rozpakovat stejným způsobem jako běžný soubor ZIP. Všechny applety jsou dostupné přes libovolný webový prohlížeč s podporou platformy Java, tj. je třeba stáhnout a nainstalovat Java SE Development Kit (JDK). Popis stránek viz kapitola 4.4.

Dále na přiloženém CD můžeme nalézt elektronickou verzi bakalářské práce, která je shodná s vytištěnou a svázanou verzí. Najdete ji v kořenovém adresáři ve formátu pdf nazvanou „bakalářská práce“.

Součástí přílohy jsou i zdrojové kódy obou tříd „Graf“ a „Šipka“. Ty lze najít v adresáři „zdrojové kódy“.

3 Zdrojový kód třídy Graf

```
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.Polygon;
import java.awt.RenderingHints;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.geom.AffineTransform;
import java.awt.geom.Line2D;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;
import javax.swing.JLabel;
import javax.swing.JPanel;

class Graf extends JPanel implements MouseListener {

    //odkazy na instance trid
    private Bezier be;
    private GrafXt xt;
    private GrafYt yt;

    //configuracni nastaveni
    public Dimension size;
    public int border = 0;
    private int x = border, y = border, caraX1, caraX2, caraY1, caraY2;
    //x,y souradnice polohy grafu, caraX1,caraY1,caraX2,caraY2 souradnice
    car pri cyklusu
    // bod 0 grafu xNullposition, yNullposition
    public int xNullposition;
    public int yNullposition;
    public int stepVertical;
    public int stepHorizontal;
    private BasicStroke bs;

    //nastaveni bodu
    public Point2D.Double[][] bod;
    public int numpoints = 0;
    public int setnumpoints = 4;
    private double k = 0.025;
    private double t;
    public double t_rozkladu = 0;

    //graficke nastaveni
    public boolean polynom = true;
    public boolean rozklad = false;
    private Color color = new Color(181,201,235);
```

```
public JLabel name;
public boolean paint = true;
public boolean text = false;
public boolean bernstein = false;
public boolean graf_x = false;

Body bodyPrubehu = new Body((int)(1/k));
Graphics2D g2;

Graf() {
    bod = new Point2D.Double[setnumpoints+1][setnumpoints+1];
    bernstein = true;
}

Graf(Bezier input, GrafXt xt, GrafYt yt) {

    addMouseListener(this); //prida se MouseListener -
    osetreni udalosti mysi
    be = input; //odkaz na instanci Bezier
    this.xt = xt; //odkaz na graf X
    this.yt = yt; //odkaz na graf Y

    bod = new Point2D.Double[setnumpoints+1][setnumpoints+1];
    text = true;
}

@Override
public void paintComponent( Graphics g ) {

    g2 = (Graphics2D) g; //konverze Graphics na Graphics2D
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                    RenderingHints.VALUE_ANTIALIAS_ON);

    //výpočet x,y,šířky a výšky
    size = getSize(); //velikost obalu grafu
    //nastaveni velikosti a pozice grafu
    size.width = size.width - 1;
    size.height = size.height - 1;
    x = border;
    y = border;
    xNullposition = border;
    yNullposition = size.height + border;

    //vykreslení grafu

    //nasteveni stylu cary
    bs = new BasicStroke (1.0f, BasicStroke.CAP_SQUARE,
    BasicStroke.JOIN_MITER);
    g2.setStroke(bs);

    //vykresleni ctverecku s obrajem
    Rectangle2D.Double graf = new
    Rectangle2D.Double(x,y,size.width,size.height);
```

```
g2.setColor(color);
//g2.setColor( new Color(255,255,255) ); //nastaveni bileho pozadi
g2.fill( graf );
g2.setColor(Color.black);
g2.draw( graf );

//vykresleni horizomtalnich car;
g2.setColor( new Color(141,171,220) );

stepVertical = 32;
stepHorizontal = 20;

//cyklus car horizontalnich
for(int i = 1; i <= (size.height/stepHorizontal); i++) {
    caraX1 = border + 1; // 1 sila ramecku
    caraY1 = caraY2 = yNullposition - (stepHorizontal*i);
    caraX2 = size.width + border - 1;
    //y2 = yNullposition - (border+(step*i));
    Line2D cara = new Line2D.Double(caraX1,caraY1,caraX2,caraY2);
    g2.draw(cara);
}
//cyklus car vertikalnich
for(int i = 1; i <= (size.width/stepVertical); i++) {
    caraY1 = border + 1 ;
    caraX1 = caraX2 = xNullposition + (stepVertical*i);
    caraY2 = size.height + border - 1 ;
    //y2 = yNullposition - (border+(step*i));
    Line2D cara = new Line2D.Double(caraX1,caraY1,caraX2,caraY2);
    g2.draw(cara);
}

if(text) {
    g2.setColor(new Color(0,0,0));
    String str = "Začněte kreslit zde";
    g2.setFont( new Font("SansSerif", Font.PLAIN, 18) );
    g2.drawString(str, 90, 105);
}

//pokud je zapnuto zobrazeni bodu
if(polynom) {

    //nastaveni barvy cary
    g2.setColor(Color.black);
    //cyklus pro vykresleni bodu v grafu
    for(int i = 1; i <= numpoints; i++) {
        //bod 4x4 ctverec
        //Rectangle2D.Double point = new
         Rectangle2D.Double(coordlist[i].x-2,coordlist[i].y-2,4,4);
        Rectangle2D.Double point = new
         Rectangle2D.Double(bod[0][i].x-2,bod[0][i].y-2,4,4);
        g2.fill(point);
        //pokud existuji alespon 2 body, vykresli se cara mezi nimi
        if(numpoints > 1 && i < (numpoints)) {
```

```
                    Line2D cara = new
                    Line2D.Double(bod[0][i].x,bod[0][i].y,
                            bod[0][i+1].x,bod[0][i+1].y);
                    g2.draw(cara);
                }
            }

        }

        //zaskrtnute rozjladani krivky
        if(rozklad && numpoints == setnumpoints) {

            Point bod_rozkladu = new Point();
            bod_rozkladu = getFcePoints(t_rozkladu);

            if(bod[0][1] != null) {
                //nastaveni barvy cary
                g2.setColor(Color.blue);
                //System.out.println(bod[0][1]);

                //Rectangle2D.Double point = new
Rectangle2D.Double(bod_rozkladu.x-5,bod_rozkladu.y-5,10,10);
                Line2D cara_y = new
Line2D.Double(bod_rozkladu.x,bod_rozkladu.y,
                            0,bod_rozkladu.y);
                Line2D cara_x = new
Line2D.Double(bod_rozkladu.x,bod_rozkladu.y,
                            bod_rozkladu.x,200);

                g2.fillOval(bod_rozkladu.x-5, bod_rozkladu.y-5, 10, 10);
                g2.setColor(Color.black);
                g2.draw(cara_y);
                g2.draw(cara_x);
            }

        }

        if(numpoints == setnumpoints) {

            //double x1,x2,y1,y2;
            int countPoints = ((int)(1/k));
            double x[] = new double[countPoints+1];
            double y[] = new double[countPoints+1];
            double bern_x[] = new double[countPoints+100];
            double bern_y[] = new double[countPoints+100];
            int i = 0;

            //Point2D.Double point = new Point2D.Double();
            Point point = new Point();

            //nastaveni barvy cary
            g2.setColor(Color.red);
```

```
for(t=0;t<=1;t=t+k) {

    point = getFcePoints(t);
    //bodyPrubehu.setBod(point);
    t = Round(t,3);
    //System.out.println(t);
    //System.out.println(point);
    x[i] = point.x;
    y[i] = point.y;

    if(t >= k) {
        //System.out.println(i);
        Line2D line = new Line2D.Double(x[i-1],
        y[i-1],x[i],y[i]);
        g2.draw(line);
        //Line2D line = new Line2D.Double(bernstein_x[i-1],
        y[i-1],x[i],y[i]);

    }
    i++;
}

if(bernstein) {

double vzdalenost = 320;
if(graf_x) {
    vzdalenost = 200;
}

for(int b=1;b<=4;b++) {
    int j = 1;

    bern_x[0] = 0;
    bern_y[0] = y[0] + ((200 - y[0]) *
    (1 - getBernstein(0, b)));
    if(graf_x) {
        bern_y[0] = 0;
        bern_x[0] = getBernstein(0, b) * x[0];
    }


    t=k;

    for(t=k;t<=1;t=t+k) {
        System.out.println(j);
        t = Round(t,3);

        bern_x[j] = t* vzdalenost;
        bern_y[j] = y[j] + ((200 - y[j]) *
        (1 - getBernstein(t, b)));
        if(graf_x) {
            bern_y[j] = t * vzdalenost;
            bern_x[j] = getBernstein(t, b) * x[j];
```

```
                }

                    g2.setColor(Color.blue);
                    Line2D line = new Line2D.Double
                    (bern_x[j-1],bern_y[j-1],bern_x[j],bern_y[j]);
                    g2.draw(line);

                    j++;

            }
        }

        }

    }

}

public Point getFcePoints(double t) {

    double x,y;

    if(t == 0) {
       x = bod[0][1].x;
       y = bod[0][1].y;
    }
    else {
        for(int k = 1; k <= (setnumpoints-1); k++)
            for(int n = 1; n <= (setnumpoints-k); n++)
                bod[k][n] = new Point2D.Double(((1-t) * bod[k-1][n].x +
                t * bod[k-1][n+1].x), ((1-t) * bod[k-1][n].y + t *
                bod[k-1][n+1].y));

        x = bod[setnumpoints-1][1].x;
        y = bod[setnumpoints-1][1].y;
    }

    return ( new Point((int)x,(int)y) );

}

public void setPoint(Point2D.Double point) {
    bod[0][numpoints+1] = point;
    ++numpoints;
    polynom = false;
    repaint();
}

public double getBernstein(double t, int bernstein) {
    double B = 0;
    switch(bernstein) {
        case 1:
            B = (1-t)*(1-t)*(1-t);
```

```
            break;
        case 2:
            B = (3*t)*(1-t)*(1-t);
            break;
        case 3:
            B = (3*t*t)*(1-t);
            break;
        case 4:
            B = t*t*t;
            break;
    }
    return B;
}

public void mousePressed(MouseEvent e) {

    if(numpoints < setnumpoints) {

        text = false;

        bod[0][numpoints+1] = new Point2D.Double(e.getX(),e.getY());
        xt.setPoint(new Point2D.Double(e.getX(),e.getY()));
        yt.setPoint(new Point2D.Double(e.getX(),e.getY()));

        double x, y;

        x = bod[0][numpoints+1].x;
        y = 200 - bod[0][numpoints+1].y;

        switch (numpoints+1) {
            case 1:
                be.jTextField1.setText(""+x);
                be.jTextField2.setText(""+y);
                break;
            case 2:
                be.jTextField3.setText(""+x);
                be.jTextField4.setText(""+y);
                break;
            case 3:
                be.jTextField5.setText(""+x);
                be.jTextField6.setText(""+y);
                break;
            case 4:
                be.jTextField7.setText(""+x);
                be.jTextField8.setText(""+y);
                break;
        }

        numpoints++;
        repaint();

    }
}
```

```
    public void mouseEntered(MouseEvent e) {
        color = new Color(195,214,245);
        repaint();
    }

    public void mouseExited(MouseEvent e) {
        color = new Color(181,201,235);
        repaint();
    }

    public static double Round(double Rval, int Rpl) {
        double p = (double)Math.pow(10,Rpl);
        Rval = Rval * p;
        double tmp = Math.round(Rval);
        return (double)tmp/p;
    }

}

//pretizena trida GrafXt - neinteraktivni graf
class GrafXt extends Graf {

    GrafXt() {
        bod = new Point2D.Double[setnumpoints+1][setnumpoints+1];
        graf_x = true;
    }

    public Point getFcePoints(double t) {

        double x,y;

        if(t == 0) {
           y = 0;
           x = bod[0][1].x;
        }
        else {
            for(int k = 1; k <= (setnumpoints-1); k++)
                for(int n = 1; n <= (setnumpoints-k); n++)
                    bod[k][n] = new Point2D.Double(((1-t) * bod[k-1][n].x +
                    t * bod[k-1][n+1].x), ((1-t) * bod[k-1][n].y + t *
                    bod[k-1][n+1].y));

            y = t*200;
            x = bod[setnumpoints-1][1].x;
        }

        return ( new Point((int)x,(int)y) );

    }

}
```

```
//pretizena trida GrafYt – neinteraktivni graf
class GrafYt extends Graf {

    GrafYt() {
        bod = new Point2D.Double[setnumpoints+1][setnumpoints+1];
    }

    public Point getFcePoints(double t) {

        double x,y;

        if(t == 0) {
           x = 0;
           y = bod[0][1].y;
        }
        else {
            for(int k = 1; k <= (setnumpoints-1); k++)
                for(int n = 1; n <= (setnumpoints-k); n++)
                    bod[k][n] = new Point2D.Double(((1-t) * bod[k-1][n].x +
                    t * bod[k-1][n+1].x), ((1-t) * bod[k-1][n].y + t *
                    bod[k-1][n+1].y));

            x = t*320;
            y = bod[setnumpoints-1][1].y;
        }

        return ( new Point((int)x,(int)y) );

    }

}
```

4 Zdrojový kód třídy Šipka

```
package decasteljau;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.RenderingHints;
import java.awt.Toolkit;
import java.awt.geom.Line2D;
import java.io.InputStream;
import javax.swing.JPanel;

public class Sipka extends JPanel {

    Image m_image;
    Graphics2D g2;
    public int smer;
    public String ret;

    Sipka(int i, String s) {
        this.smer = i;
        this.ret = s;
    }

    @Override
    public void paintComponent( Graphics g ) {
        g2 = (Graphics2D) g; //konverze Graphics na Graphics2D
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                            RenderingHints.VALUE_ANTIALIAS_ON);
        if(smer == 0) {
            /*Line2D cara1 = new Line2D.Double(0,0,19,19);
            Line2D cara2 = new Line2D.Double(19,19,10,16);
            Line2D cara3 = new Line2D.Double(19,19,16,10);*/
            Line2D cara1 = new Line2D.Double(25,0,44,19);
            Line2D cara2 = new Line2D.Double(44,19,35,16);
            Line2D cara3 = new Line2D.Double(44,19,41,10);
            g2.draw(cara1); g2.draw(cara2); g2.draw(cara3);
            g2.drawString(ret, 10, 15);
        }
        else if(smer == 1) {
            Line2D cara1 = new Line2D.Double(20,0,1,19);
            Line2D cara2 = new Line2D.Double(1,19,4,10);
            Line2D cara3 = new Line2D.Double(1,19,10,16);
            g2.draw(cara1); g2.draw(cara2); g2.draw(cara3);
            g2.drawString(ret, 22, 15);
        }

    }
}
```