VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

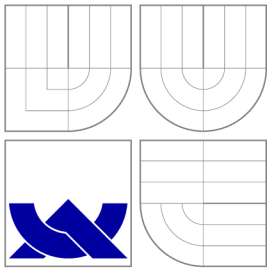
VÝUKOVÝ PROGRAM PRO DEMONSTRACI RASTERIZACE 2D OBJEKTŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

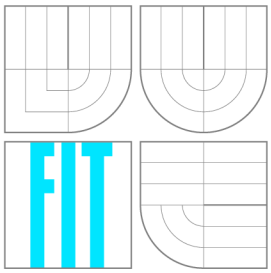
AUTOR PRÁCE
AUTHOR

VOJTĚCH KRUPIČKA

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VÝUKOVÝ PROGRAM PRO DEMONSTRACI RASTERIZACE 2D OBJEKTŮ

EDUCATION COMPUTER PROGRAM FOR DEMONSTRATION OF 2D SHAPES RASTERIZATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — VOJTĚCH KRUPIČKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. VÍT ŠTANCL
SUPERVISOR

BRNO 2009

Abstrakt

Tato bakalářská práce se zabývá problémem *rasterizace* vektorových objektů na zobrazovacích zařízeních (např. monitoru počítače). Rasterizace je jedním z nejdůležitějších jevů v počítačové grafice. Pro jeho pochopení je potřebná znalost matematických rovnic a vzorců jimiž jsou příslušné vektorové objekty popsány. Pro názornou demonstraci rasterizace vznikl výukový program, který jednoduchou cestou seznámí uživatele s principem rasterizace vybraného vektorového objektu.

Abstract

This bachelor thesis deals with the problem of *rasterization* of vector objects on display devices (e.g. computer screen). The rasterization is one of the most important phenomena in computer graphics. The knowledge of mathematical equations and formulas which describe relevant vector objects is necessary to understand this phenomenon. This education software for the visual demonstration of rasterization was created to familiarize users with the rasterization principles of selected vector objects.

Klíčová slova

Výukový program, rasterizace, vektorová grafika, 2D objekty, úsečka, elipsa, křivky, OOP, C++, wxWidgets

Keywords

Education computer program, rasterization, vector graphic, 2D shapes, abscissa, ellipse, curves, OOP, C++, wxWidgets

Citace

Vojtěch Krupička: Výukový program pro demonstraci rasterizace 2D objektů, bakalářská práce, Brno, FIT VUT v Brně, 2009

Výukový program pro demonstraci rasterizace 2D objektů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Víta Štancla. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Vojtěch Krupička
20. května 2009

Poděkování

Děkuji Ing. Vítu Štanclovi za cenné rady a připomínky při návrhu a implementaci programu a při psaní této práce. Rovněž děkuji všem, kteří mě při práci podporovali a pomáhali.

Obsah

1 Úvod **3**

2 Grafický výukový program **4**
2.1 Uživatelské rozhraní 4
2.2 Výběr vhodného vývojového prostředí 5
2.3 Výběr vhodných vektorových objektů 5
2.4 Grafické rozhraní programu 6
2.5 Vstupy a výstupy, ovládání aplikace 7
2.5.1 Vstupy a výstupy 7
2.5.2 Ovládání aplikace 8

3 Teorie rasterizace vektorových objektů **9**
3.1 Objekt úsečka 10
3.1.1 DDA algoritmus pro úsečku 12
3.1.2 DDA algoritmus s hlídáním chyby 13
3.1.3 Bresenhamův algoritmus pro úsečku 13
3.2 Objekt elipsa, kružnice 14
3.2.1 Midpoint algoritmus pro kružnici 14
3.2.2 Midpoint algoritmus pro elipsu 15
3.3 Křivky, interpolační křivky, aproximační křivky 16
3.3.1 Fergusonovy kubiky 17
3.3.2 Catmull-Rom spline 17
3.3.3 Beziérovy kubiky 18
3.3.4 Algoritmus DeCasteljau 19
3.3.5 B-Spline 19

4 Návrh aplikace **21**
4.1 Vývojové prostředí 21
4.2 Třívrstvá hierarchie programu 21
4.3 Realizace rastru 23
4.4 Simulační část a výpisy výsledků 23
4.4.1 Režimy pro vypisování výsledků 23
4.4.2 Rychlý režim pro výpis výsledků 24
4.5 Omezení programu 24
4.5.1 Výběr metody, typu křivky 24

5 Implementace **25**
5.1 Hlavní okno aplikace a *GUI* . . . 25
5.1.1 Inicializace aplikace, třída `vkApp` . . . 25
5.1.2 Hlavní okno, třída `vkMainFrame` . . . 26
5.1.3 AUI, *Advanced User Interface* . . . 26
5.1.4 Obsluha událostí . . . 26
5.2 Dceřiné okno, třída `vkChildFrame` . . . 27
5.2.1 Vytvoření a správa kreslícího plátna . . . 27
5.2.2 Komunikační protokol . . . 28
5.2.3 Editační vs. simulační část . . . 29
5.3 Kreslící plátno, třída `vkCanvas` . . . 29
5.3.1 Zavedení souřadnicového systému plátna . . . 29
5.3.2 Přibližování, oddalování, pohyb po plátně . . . 30
5.3.3 Správa kontrolních bodů . . . 31
5.3.4 Překreslování plátna . . . 31
5.3.5 Demonstrace reasterizace . . . 32
5.4 Simulátor a správce výsledků . . . 32
5.4.1 Třída `vkSimulator` . . . 32
5.4.2 Třída `vkResultHtmlHandler` . . . 32

6 Ovládání aplikace **34**
6.1 Hlavní prvky aplikace . . . 34
6.1.1 Systémové menu . . . 34
6.1.2 Panely nástrojů . . . 34
6.2 Editační část . . . 35
6.3 Simulační část . . . 36

7 Závěr **38**
7.1 Možnosti rozšíření aplikace . . . 38

Kapitola 1

Úvod

Během několika posledních let bylo dosaženo značného pokroku v oblasti počítačové grafiky a to jak v oblasti hardwaru tak i v oblasti softwaru. Tento pokrok je důsledkem zvyšování konkurence mezi jednotlivými firmami a rovněž vyššími nároky uživatelů. V dnešní době je snadné získat kvalitní grafický program a stát se grafikem nebo designérem, ať už amatérským či profesionálním.

Naučit se pracovat s těmito programy není nikterak složité a pokud je daná osoba alespoň trochu zručná, zvládne ovládání programu během několika týdnů. Zapomíná se však na elementární problémy, jejichž podstata by rozhodně neměla být zanedbávána a mezi ně patří především problém *rasterizace*.

Naprostá většina všech zobrazovacích zařízení je rastrová, diskrétní. Objekty, které se však na takovýchto zařízeních zobrazují jsou často popsány matematicky, pomocí rovnic, tedy *vektorově*. Ačkoliv je podpora rasterizace zakomponována přímo do jader grafických karet, a proto je vykreslování i většího množství složitých objektů velmi rychlé, mělo by být dobrým zvykem každého uživatele, který to s grafikou myslí alespoň trochu vážně, pochopit problematiku rasterizace v jejím celkovém rozsahu. Jelikož za tím stojí mnoho teorie a matematických rovnic a vzorců, naprostou většinu lidí takové studium spíše odradí.

Mým úkolem v tomto projektu bylo vytvořit jednoduchý, byť komplexní, uživatelsky příjemný výukový program, který by nenáročnou cestou dokázal osvětlit problém rasterizace jednoduchých vektorových objektů na zobrazovacích zařízeních, přesněji na monitoru počítače.

Kapitola 2

Grafický výukový program

Aby bylo možné vůbec takový program navrhnout, je nejprve nutné stanovit si cíle, které budou na vyvíjený program kladeny, a formulovat požadavky, které by měl program splňovat. Mezi základní požadavky na kvalitní výukový program jistě patří:

- Jednoduché, intuitivní ovládání,
- přehledné uspořádání prvků programu,
- rozsáhlé možnosti nastavení programu,
- detailní nápověda,
- podpora více jazyků programu,
- nezávislost na platformě, na které bude program používán.

Výčet těchto požadavků není kompletní a během vývoje ho bude možné rozšiřovat o další důležité vlastnosti.

Podobně, jako stanovení cílů a požadavků, které je třeba mít na paměti při vývoji nového programu, je neméně důležitým krokem specifikace cílové skupiny osob, pro níž bude program určen. Každý software je vyvíjený pro jistý účel a pro užší či širší část počítačové komunity. Přestože je rasterizace základním jevem nejen na monitoru, se kterým se setkává každý uživatel počítače denně, bylo by jistě bláhové tvrdit, že vyvíjený výukový program bude určen pro všechny, nebo že bude využíván všemi, kteří se aktivně zabývají počítačovou grafikou.

V první řadě je program vyvíjen jako školní pomůcka při výkladu látky týkající se rasterizace, nebo pro pochopení této problematiky studenty – uživateli, při samostudiu. Proto se od uživatele programu předpokládá jistá úroveň zkušeností práce s počítačem, stejně jako znalost jednotlivých rasterizačních metod, typů křivek, apod.

2.1 Uživatelské rozhraní

Jedním z hlavních požadavků na kvalitní program je jednoduché, intuitivní ovládání a přehlednost prvků programu. Už z tohoto bodu je patrné, že nebude možné si vystačit pouze s konzolovou aplikací, ale s aplikací okenní. Ačkoliv by nebylo nereálné vytvořit výukový program pro demonstraci rasterizace vektorových objektů pouze s textovým výstupem. Výsledek snažení by však pravděpodobně neměl ten správný efekt.

Jen pro představu, jako vstup programu by mohly být zadány souřadnice počátečního a koncového bodu úsečky a výstupem takovéto konzolové aplikace by byla sada čísel pro každý vypočítaný bod požadované úsečky.

Je tady jasné, a dnešní doba si to přímo žádá, vytvořit okenní aplikaci, jejíž ovládání je daleko jednodušší pomocí myši a klávesnice a vstup i výstup je prezentován v grafické podobě. Využití grafického uživatelského rozhraní, neboli GUI (*Graphical User Interface*), však s sebou kromě mnoha výhod přináší i několik problémů. Mezi ty nejzásadnější patří definice ideálního grafického rozhraní programu pro daný problém. Kam jaký prvek umístit, kam vykreslovat výstup, kde žádat vstup, apod. Tyto a další problémy budou rozebrány postupně v dalších kapitolách.

2.2 Výběr vhodného vývojového prostředí

Jelikož bylo zcela logicky vybráno řešení s grafickým uživatelským rozhraním, je rovněž na místě vybrat vhodný nástroj, ve kterém bude výsledný program, aplikace, vyvíjen. Při výběru takovéhoto nástroje je pochopitelně nutné brát v úvahu stanovené cíle a požadavky na nově vznikající software.

Vývojových prostředí, resp. programovacích jazyků a jejich grafických nadstaveb, je v dnešní době nepřeberné množství a to i takových, které by splňovaly všechny požadavky. Proto výběr jednoho z nich nebyl lehký úkol. Výsledný program by měl být nezávislý na platformě, na které bude používán (v úvaze byly pracovní stanice s operačním systémem Windows a Linux), dále by měl podporovat více jazyků a vývojové prostředí by mělo být s otevřenou licencí a volně šiřitelné. Jako vhodný vývojový nástroj byla zvolena grafická nadstavba (angl. *toolkit*) nad programovacím jazykem C++ jménem *wxWidgets*.

Tento výběr byl ovlivněn především faktem, že jsem se s touto grafickou knihovnou již setkal při práci na jiných projektech a programování v ní je relativně pohodlné díky rozsáhlé webové dokumentaci dostupné na [7], v níž jsou popsány veškeré konstanty, třídy i jejich metody. Zdrojové kódy včetně mnoha příkladů jsou volně k dispozici a kompilace do knihoven není obtížná. Pronikání do tajů *wxWidgets* je však velmi náročné hlavně z důvodu obsáhlosti této knihovny.

2.3 Výběr vhodných vektorových objektů

Nyní, když bylo vybráno vývojové prostředí pro vyvíjený program, je rovněž zapotřebí specifikovat základní funkce a chování výukového programu. Protože se jedná o program demonstrující rasterizaci vektorových objektů je nasnadě určit objekty, které budou dále v mém zájmu. Mezi základní vektorové objekty patří:

- Úsečka,
- elipsa/kružnice,
- křivky.

Tyto tři základní typy objektů byly vybrány jako výchozí a program bude umět zvládnout zobrazit průběh jejich rasterizace pomocí jedné z definovaných metod[1]. Úsečka je objektem nejjednodušším a proto by určitě chybět neměla. Mezi objekty však není i přímka.

[1] U křivek nelze mluvit přímo o metodách, ale o jednotlivých typech křivek.

Vysvětlení je snadné. Pro rasterizaci úsečky je zapotřebí definovat počáteční a koncový bod. Ty však přímka nemá a proto její rasterizace není možná. Z jiného úhlu pohledu je však přímka pouhým zobecněním úsečky. Dále by se mohlo zdát, že ve výčtu objektů chybí trojúhelník, čtverec nebo podobný složitější útvar (šestiúhelník apod.). Tyto objekty lze rozdělit na několik elementárních částí, které jsou tvořeny opět úsečkami nebo částmi kružnice, případně elipsy. Princip rasterizace je tudíž založen na stejných algoritmech a ve výsledku by zařazení těchto objektů nemělo žádný efekt.

Kružnice je speciálním případem elipsy a proto by bylo možné tyto objekty sloučit do jednoho. Jelikož se pro jejich rasterizaci používají rozdílné algoritmy a objekty samotné jsou popsány různými rovnicemi (viz 3.2.1 a 3.2.2), bylo by vhodné do programu začlenit objekty oba. Nejsložitějšími vektorovými útvary jsou zajisté křivky. Křivek existuje velké množství a proto je výběr zúžen pouze na ty nejznámější nebo jinak zajímavé.

2.4 Grafické rozhraní programu

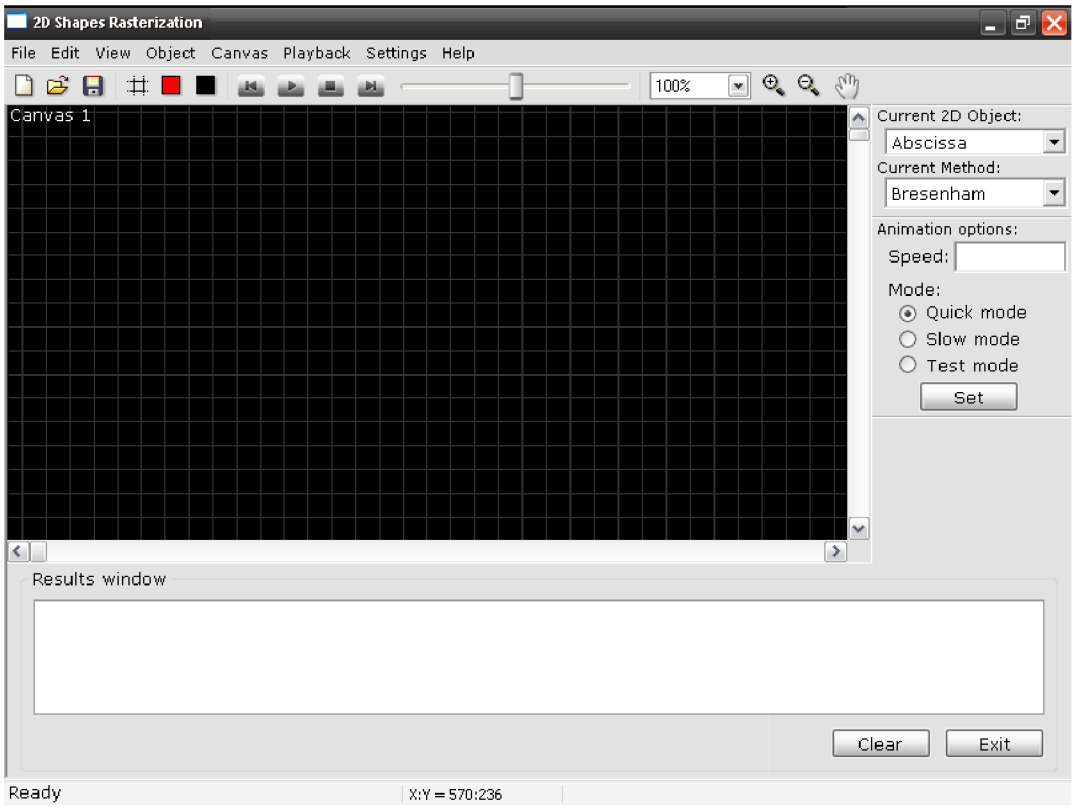
Nyní, když jsou specifikovány objekty, jenž mají být ukázkou rasterizace, je nutné se zaměřit na program jako celek. Jak bylo uvedeno při vytyčení cílů, program by měl být uživatelsky příjemný a snadno ovladatelný. Je proto zapotřebí věnovat jisté úsilí návrhu vlastního grafického rozhraní, tedy tomu, jak bude program vypadat a co bude obsahovat. V programu by neměly chybět tyto části:

- Systémové menu,
- vodorovný panel nástrojů pro rychlý přístup k jednotlivým prvkům programu,
- svislý panel nástrojů pro výběr vektorového objektu a metody pro rasterizaci,
- vlastní kreslící plocha,
- panel pro zobrazování výpočtů v jednotlivých krocích rasterizace,
- stavový řádek pro zobrazování právě prováděných operací.

Systémové menu v sobě ponese základní nabídky, pomocí nichž bude možné vykonávat všechny nezbytné akce a ovlivňovat nastavení a chování programu v jednotlivých částech jeho běhu. Pro zjednodušení ovládání bude k dispozici několik tematicky rozdělených panelů nástrojů. Každý panel nástrojů bude obsahovat prvky, které budou určeny pro ovládání jisté činnosti a budou korespondovat s jednotlivými částmi systémového menu. Mezi základní prvky lze zařadit např. nastavení barev pro kreslení (barva pera a barva pozadí kreslící plochy), nástroj *Lupa* nebo nástroj *Ručička* (více o nástrojích viz 6.1). Dále pak výběr jednotlivých objektů a jejich metod, zobrazení nápovědy, apod.

Nejdůležitější částí okna aplikace však bude vlastní kreslící plocha – plátno, nad kterým bude prováděn vstup od uživatele a rovněž zobrazován výstup. Definice vstupů a výstupů programu je popsána v 2.5.1. Dalším důležitým prvkem programu by měl být panel pro zobrazení výsledků rasterizace. Rasterizační metody v každém kroku provádějí několik výpočtů, jež vedou k nalezení právě vykreslovaného bodu. Aby uživatel měl přehled o těchto výpočtech, je nutné je zobrazit. Zobrazování dalších pomocných informací, jako je nápověda k aktuálně vybrané položce v systémovém menu nebo současná pozice myši nad kreslícím plátnem, bude realizováno pomocí stavového řádku rozděleného do několika částí.

Základní představa o grafickém rozhraní programu je znázorněna na obrázku 2.1. Nutno podotknout, že se nejedná o finální podobu a např. okno pro zobrazování výsledků jistě prodělá mnoho změn většího či menšího charakteru.



Obrázek 2.1: Návrh grafického rozhraní programu, *18. 1. 2009*.

2.5 Vstupy a výstupy, ovládání aplikace

Nyní, když je definováno grafické rozhraní programu, stejně jako vektorové objekty, na které se při vývoji aplikace zaměřím, je na místě v krátkosti navrhnout samotné chování a ovládání programu. Aby to však bylo možné, je třeba upřesnit vstupy, které program od uživatele očekává a výstupy, které poté uživateli zobrazí.

2.5.1 Vstupy a výstupy

Celý proces rasterizace vektorového objektu sestává ze dvou fází. V první fázi je nutné zadat kontrolní body objektu. Pro objekt úsečky to bude její počáteční a koncový bod. V případě elipsy střed a definice dvou poloos – hlavní a vedlejší. A konečně u křivek jejich řídicí body. Zadání kontrolních bodů bude chápáno jak vstup od uživatele. Ve druhé fázi dochází opakovaně k nalezení pozice aktuálního bodu pomocí výpočtů, které jsou specifické pro každou rasterizační metodu, resp. typ křivky. Každý nový bod představuje segment[2],

[2]U úsečky a elipsy/kružnice se jedná o bod – pixel, u křivek se pak jedná o úsečku, která končí v aktuálním bodě a začíná v bodě předchozím.

který je vykreslen na plátno. Provedení výpočtů a vykreslení nového segmentu potom bude chápáno jako výstup programu.

Samotný výpočet aktuálního segmentu objektu je netriviální záležitostí a je vždy závislý na předchozím segmentu, potažmo na kontrolních bodech objektu. Proto bude nutné tyto dvě fáze rasterizace od sebe oddělit i v samotném programu. Program se tedy bude skládat ze dvou hlavních částí:

- Editační část,
- simulační část.

Jak již bylo zmíněno výše, každá část představuje jednu fázi rasterizace. V editační části bude možné definovat kontrolní body objektu a další potřebné informace pro vlastní rasterizaci (např. u křivek lze definovat počet úseček, ze kterých je výsledná křivka tvořena). Při přechodu do simulační části budou provedeny a uloženy všechny výpočty, aby je bylo možné zobrazovat při simulaci rasterizace objektu.

2.5.2 Ovládání aplikace

Když byly definovány očekávané vstupy od uživatele a jejich výstupy, je možné uvést pregnantní ilustraci používání aplikace. Pro jednoduchost bude jako objekt použit objekt nejjednodušší – úsečka.

Po spuštění programu je nejprve zapotřebí vybrat požadovaný objekt a k němu příslušnou rasterizační metodu, které chce daný uživatel studovat. Poté je třeba definovat vstupy, tedy počáteční a koncový bod úsečky. Tím je praktická činnost uživatele ukončena a je možné program přepnout do simulační části. Simulační část v sobě nese přehrávání animace rasterizace. S každým krokem výpočtu dojde ke zobrazení aktuálně vykreslovaného segmentu na plátno a rovněž k poskytnutí všech důležitých výpočtů a jejich výsledků, které vedly k získání souřadnic daného bodu.

Při přehrávání animace je možné celou animaci spustit, zastavit či pozastavit. Dále pak krokovat dopředu a dozadu nebo měnit rychlost animace. Změna souřadnic, přidávání nových nebo mazání současných kontrolních bodů však možná není a to z důvodů, které byly popsány v kapitole 2.5.1.

Kapitola 3

Teorie rasterizace vektorových objektů

Po specifikaci vzhledu programu, jeho důležitých prvků a ovládání je na čase přejít k vlastní teorii rasterizace, probrat jednotlivé typy objektů, jenž byly vybrány a seznámit se s metodami používanými pro jejich rasterizaci. Jak bylo uvedeno v kapitole 2.3, jako vhodné objekty byly vybrány:

- Úsečka,
- elipsa/kružnice,
- interpolační křivky,
- aproximační křivky.

První dva objekty představují ucelené vektorové útvary, které jsou popsány rovnicemi a pro jejich rasterizaci existuje několik více či méně vhodných metod. Křivky jsou však zcela odlišné. Rasterizace křivek je realizována pomocí předem specifikovaného počtu úseček, ze kterých je výsledná křivka složená. U křivek proto nemluvíme o metodách pro jejich vykreslování, ale o typu křivky, jež má být vykreslen.

Jako ideální výukové metody pro rasterizaci úsečky byly vybrány tyto tři algoritmy:

- DDA Algoritmus (viz 3.1.1),
- rozšířený DDA algoritmus s hlídáním chyby (viz 3.1.2),
- Bresenhamův algoritmus (viz 3.1.3).

Již při výběru základních vektorových objektů v kapitole 2.3 bylo řečeno, že pro rasterizaci elipsy a kružnice se používá rozdílných algoritmů a i jejich matematický popis je odlišný. Jelikož je však kružnice speciálním případem elipsy, program bude implicitně pracovat pouze s elipsou. V případě, že by zadaný objekt mohl představovat kružnici (délky hlavní a vedlejší poloosy si budou rovny), bude uživateli zpřístupněn výběr metody pro rasterizaci kružnice. Toto chování programu bylo vybráno proto, aby uživateli poskytlo možnost sledovat rasterizaci dvou odlišných objektů, které pouze stejně vypadají, pomocí dvou rozdílných rasterizačních algoritmů. Jedná se o metody:

- Midpoint algoritmus pro elipsu (viz 3.2.2),

- Bresenhamův algoritmus pro kružnici (Midpoint algoritmus) (viz 3.2.1).

Křivek bylo popsáno nepřeberné množství a jejich rozdělení se dá provést mnoha způsoby. Jedním ze základních dělení křivek je právě rozdělení na *křivky interpolační* (křivka přímo prochází kontrolními body) a na *křivky aproximační* (křivka kontrolními body neprochází, kontrolní body však určují její tvar). Problémem však je, které druhy křivek vybrat jako vhodné pro demonstraci. Jelikož jde o výukový program, vycházel jsem především z literatury. Proto bylo vybráno pět pravděpodobně nejznámějších typů křivek, které jsou popsány v [5] stejně jako v [9]. Jedná se o křivky:

- Fergusonovy kubiky (viz 3.3.1),
- Catmull-Rom spline (viz 3.3.2),
- Beziérovy kubiky (viz 3.3.3),
- křivky pomocí algoritmu DeCasteljau (viz 3.3.4),
- uniformní kubický B-Spline (viz 3.3.5).

V následujícím textu budou probrány jednotlivé objekty a jejich rasterizační metody, resp. typy křivek. Většinu popisovaných algoritmů je možné nalézt v [9].

3.1 Objekt úsečka

Úsečka (angl. *abscissa*) patří mezi základní vektorové objekty. Rovněž je hned po bodu objektem nejjednodušším. K její definici je zapotřebí znát souřadnice počátečního a koncového bodu a její rovnici. Rovnice geometrických útvarů lze zpravidla vyjádřit ve třech tvarech:

- Obecná rovnice úsečky:

$$Ax + By + C = 0, A \ = (y_2 - y_1), B = (x_2 - x_1) \tag{3.1}$$

- Parametrické vyjádření:

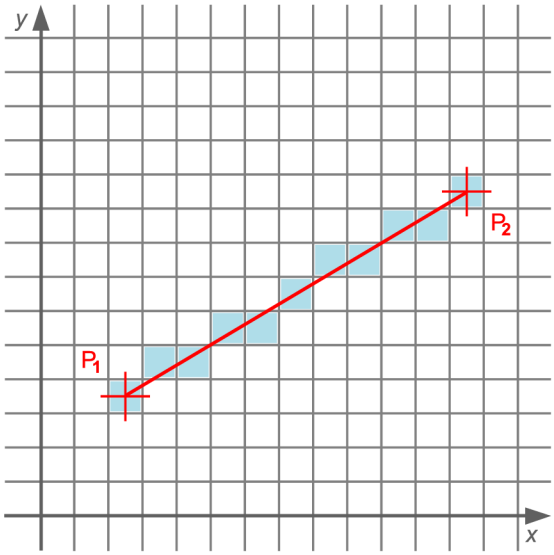
$$x = x_1 + t(x_2 - x_1), y = y_1 + t(y_2 - y_1), t \in \langle 0, 1 \rangle \tag{3.2}$$

- Směrnicový tvar:

$$y = kx + q, k \ = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \tag{3.3}$$

Pro výpočet souřadnic jednotlivých bodů úsečky a jejich zobrazení na diskrétních zařízeních (např. monitor) je nejvhodnější zvolit *směrnicový tvar*. Již z prvního pohledu je patrné, že tato rovnice obsahuje dělení. Hodnota podílu je však konstantní a lze ji vypočítat jen jednou před vlastním během algoritmu.

Rasterizace úsečky je implementována zpravidla tak, že se na *řídicí ose* postupuje s konstantním krokem dx a přírůstek na *vedlejší ose* je vypočítán podle zvoleného algoritmu a to tak, že se buď osa y nemění (přírůstek je pak roven 0) nebo se osa y upraví podle zvoleného kroku (přírůstek je pak roven dy). O tom, která osa bude *řídicí*, rozhoduje hodnota směrnice k. Tu lze vypočítat podle vzorce 3.3. Pokud je $|k| \leq 1$, úsečka přimyká k ose x, a proto



Obrázek 3.1: Úsečka jako vektor zobrazený na rastru.

bude *řídicí osou* právě osa x. Pokud je $|k| > 1$, bude obdobně úsečka přimykat k ose y a ta bude rovněž *osou řídicí*.

Základní problém studovaných algoritmů pro rasterizaci úsečky je ten, že dokáží pracovat pouze s úsečkou, která má jisté vlastnosti:

- Úsečka leží v prvním kvadrantu,
- je rostoucí od počátečního bodu P_1 ke koncovému bodu P_2,
- nejrychleji roste ve směru osy x.

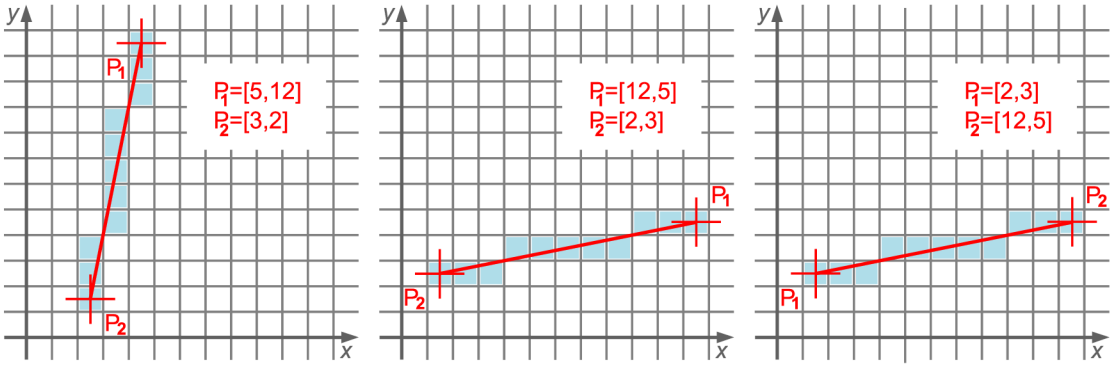
První bod předchozího výčtu je jasný. Druhý bod udává, že úsečka je orientovaná směrem doleva a nahoru, tedy že platí $x_1 < x_2$, $y_1 < y_2$. A konečně třetí bod udává, že $\Delta x > \Delta y, \Delta x = (x_2 - x_1), \Delta y = (y_2 - y_1)$. Je patrné, že pomocí algoritmů lze rasterizovat pouze úsečky ležící ve spodní polovině prvního kvadrantu jejichž sklon je nezáporný a nepřekračuje 45° vůči ose x. Osa x musí být *osou řídicí*.

Aby bylo možné rasterizovat libovolnou úsečku, je třeba před zahájením vlastního výpočtu provést několik kroků, které úsečku převedou na požadovaný tvar:

- Záměna os,
- záměna počátečního a koncového bodu.

Pomocí těchto jednoduchý kroků lze libovolnou úsečku modifikovat (nebo lépe transformovat) na úsečku, která bude splňovat všechny tři podmínky. Je třeba mít na paměti, že po vypočítání bodů takovéto úsečky je nutná zpětná transformace na původní hodnoty os, resp. souřadnic, jinak hrozí špatné vykreslení úsečky.

Po obecném úvodu do problematiky týkající se úseček jako geometrického útvaru a ujasnění si všech problémů, které vznikají při jejich rasterizaci je možné přistoupit k představení a popisu jednotlivých metod, které byly vybrány.

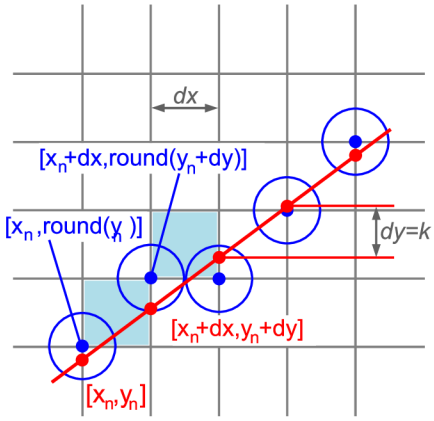


Obrázek 3.2: Záměna řídicí a vedlejší osy úsečky.

3.1.1 DDA algoritmus pro úsečku

DDA algoritmus (*Digital Differential Analyser*) je jedním z prvních algoritmů pro rasterizaci vůbec (popsán např. v [3]). Jeho základním rysem je používání aritmetiky v plovoucí desetinné čárce, proto byla jeho implementace do grafických karet náročná a efektivita algoritmu nízká. Díky tomu se od používání odstoupilo. Princip algoritmu je jednoduchý:

- Postupně se vykreslují body od počátečního P_1 ke koncovému P_2,
- v ose x se postupuje s konstantním přírůstkem $dx = 1$,
- přírůstek v ose y je dán velikostí směrnice úsečky k,
- souřadnice y se zaokrouhlí na nejbližší celé číslo.



Obrázek 3.3: DDA Algoritmus a nalezení nového bodu úsečky.

Při implementaci algoritmu se používá iterativního přístupu. Aktuální bod je odvozen z bodu předchozího. Algoritmus DDA je poté velmi jednoduchý a vychází z rovnic:

$$x_{i+1} = x_i + dx, dx = 1 \tag{3.4}$$

$$y_{i+1} = y_i + dy, dy = \frac{y_2 - y_1}{x_2 - x_1} \tag{3.5}$$

3.1.2 DDA algoritmus s hlídáním chyby

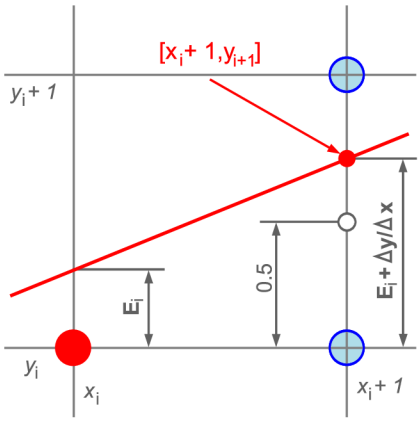
Algoritmus s hlídáním chyby je modifikací DDA algoritmu, který je rozšířen o tzv. *relativní odchylku*. Opět se postupuje od počátečního bodu P_1 ke koncovému bodu P_2 s přírůstkem $dx = 1$ pro osu x. Posun v ose y je určen podle hodnoty *relativní odchylky*. Celý algoritmus lze zapsat pomocí několika kroků:

- Nejprve je nutné vypočítat směrnici úsečky k,
- poté je vyjádřena relativní odchylka E, kde počáteční odchylka je $E = k$,
- pokud je odchylka $E > 0.5$, v ose y dojde k posunu o přírůstek,
- relativní odchylka je poté upravena podle vzorce $E = E - 1$.

3.1.3 Bresenhamův algoritmus pro úsečku

Bresenhamův algoritmus byl vyvinut J. E. Bresenhamem a je popsán v [1]. Tento algoritmus je nejpoužívanější metodou pro rasterizaci úsečky. Je to dáno především jeho snadnou implementací na grafických kartách a to proto, že pro určení nového bodu není zapotřebí aritmetiky v plovoucí desetinné čárce. Bresenhamův algoritmus si vystačí pouze s celočíselnou aritmetikou, sčítáním a porovnáním. Zavádí se zde pojem *Prediktor*, nebo-li rozhodovací člen. Na základě jeho znaménka se určuje posun v ose y. Princip činnosti je opět jednoduchý:

- Postupně se vykreslují body od počátečního P_1 ke koncovému P_2,
- v ose x se postupuje s přírůstkem $dx = 1$,
- posun v ose y je určen podle znaménka *prediktoru*.



Obrázek 3.4: Bresenhamův algoritmus a nalezení nového bodu.

- Rozhodování a výpočet chyby vykreslování E:

$$E_i + \frac{\Delta y}{\Delta x} \begin{cases} < 0,5 & \text{krok } (x_i + 1, y_i) & E_{i+1} = E_i + \dfrac{\Delta y}{\Delta x} \\ \geq 0,5 & \text{krok } (x_i + 1, y_i + 1) & E_{i+1} = E_i + \dfrac{\Delta y}{\Delta x} - 1 \end{cases} \tag{3.6}$$

- Nerovnice je násobena $2\Delta x$:

$$2\Delta x E_i + 2\Delta y - \Delta x \begin{cases} < 0 & E_{i+1} = E_i + 2\Delta y \\ \geq 0 & E_{i+1} = E_i + 2\Delta y - 2\Delta x \end{cases} \tag{3.7}$$

- Rozhodovací člen nazveme *prediktorem* P_i:

$$P_i = 2\Delta x E_i + 2\Delta y - \Delta x \begin{cases} < 0 & P_{i+1} = P_i + 2\Delta y \\ \geq 0 & P_{i+1} = P_i + 2\Delta y - 2\Delta x \end{cases} \tag{3.8}$$

- Nakonec je nutné definovat počáteční hodnotu predikace ($E_0 = 0$):

$$P_0 = 2\Delta y - \Delta x \tag{3.9}$$

3.2 Objekt elipsa, kružnice

Elipsa (angl. *Ellipse*) je dalším základním vektorovým objektem. Je definována svým středem v rámci souřadnicového systému, délkami hlavní a vedlejší poloosy a úhlem natočení hlavní poloosy elipsy vůči ose x. Rovnici elipsy se středem v počátku lze matematicky vyjádřit pomocí implicitní funkce:

$$F(x, y) : b^2x^2 + a^2y^2 - a^2b^2 \quad = \quad 0 \tag{3.10}$$

Kružnice (angl. *Circle*) je specifickým typem elipsy, kdy si jsou délky hlavní a vedlejší poloosy sobě rovny. K definici kružnice je potřeba znát souřadnice jejího středu a délku poloměru. Rovnici kružnice lze ze vztahu 3.10 odvodit dosazením poloměru r za obě poloosy. Po úpravách získáme rovnici $x^2 + y^2 - r^2 = 0$, kterou můžeme zapsat jako implicitní funkci:

$$F(x, y) : x^2 + y^2 - r^2 \quad = \quad 0 \tag{3.11}$$

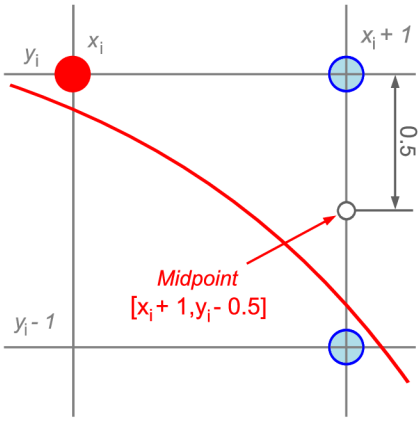
Při rasterizaci obou objektů se s výhodou využívá jejich symetričnosti. Kružnice je středově symetrická, díky čemuž je možné pro jeden vypočítaný bod odvodit dalších 7 bodů. Výpočet je potom nutné provádět pouze v jednom oktantu (jedné polovině kvadrantu). Elipsa je symetrická pouze $4\times$. To znamená, že pro každý vypočtený bod je možné odvodit tři další body a výpočet je nutné provádět pouze v jednom kvadrantu.

3.2.1 Midpoint algoritmus pro kružnici

Midpoint algoritmus pro kružnici byl publikován J. E. Bresenhamem v [2] a vychází z algoritmu pro úsečku, viz 3.1.3. Rovněž používá pouze celočíselnou aritmetiku, proto je jeho implementace výhodná. Navíc je kružnice středově souměrná. Po vypočítání jednoho bodu kružnice lze odvodit dalších sedm bodů a to prostou záměnou souřadnic nebo záměnou jejich znaménka. Pro rasterizaci kružnice stačí provést výpočty bodů ležících pouze v jednom oktantu.

Řídicí osou je osa x, vedlejší osou je pak osa y. Krok v ose x je konstantní a odpovídá zpravidla jedné jednotce (na zobrazovacích zařízeních je tomu jeden pixel). Algoritmus začíná v bodě $[0, r]$ a končí v průsečíku kružnice s hlavní diagonálou, kde $x = y$.

Podobně jako u Bresenhamova algoritmu pro úsečku (viz 3.1.3), i zde je zaveden rozhodovací člen, tzv. *Prediktor*. Z rovnice 3.11 lze určit polohu libovolného bodu $[x, y]$ vůči kružnici. Pokud je znaménko záporné, bod leží uvnitř kružnice, pokud je kladné, bod leží



Obrázek 3.5: Nalezení *Midpointu* a správné nastavení posuvu v ose y.

vně. Za tohoto předpokladu je možné definovat tzv. *Midpoint*. Z obrázku 3.5 je patrné že *Midpoint* leží v jedné polovině mezi možnými kandidáty pro následující bod (tedy body $[x_i, y_i]$ a $[x_i, y_i - 1]$). Dosazením polohy *Midpointu* (bod $\left[x_i, y_i - \frac{1}{2}\right]$) do vztahu 3.11 lze zjistit, zda tento bod leží uvnitř nebo vně kružnice a tím rozhodnout, jestli dojde k posunu na ose y nebo ne. Samotný princip algoritmu je následující:

- Inicializace pomocných proměnných $X2 = 3$ a $Y2 = 2r - 2$,
- inicializace prediktoru P na hodnotu $1 - r$,
- inicializace $[x, y]$ na $[0, r]$,
- postupně vykresluje body dokud je $x \leq y$ včetně jejich symetrických obrazů,
- před vykreslením každého bodu upraví jeho souřadnice vůči počátku,
- nastaví novou hodnotu *prediktoru* a případně provede posun v ose y.

3.2.2 Midpoint algoritmus pro elipsu

Algoritmus předpokládá elipsu se středem v počátku systému souřadnic, s nulovým otočením (tzn. hlavní, resp. vedlejší, poloosa je rovnoběžná s osou x, resp. s osou y). Jak již bylo uvedeno, elipsa je $4\times$ symetrická, proto se výpočet jednotlivých bodů provádí pouze pro první kvadrant. Souřadnice bodů ve zbývajících kvadrantech jsou poté dopočítány posunutím nebo záměnou souřadnic. V průběhu rasterizace elipsy dochází ke změně řídicí osy. Tato změna nastává v bodě, v němž má tečna k elipse směrnici rovnu -1. V tomto bodě platí následující vztah:

$$\left[\frac{a^2}{\sqrt{a^2 + b^2}}, \frac{b^2}{\sqrt{a^2 + b^2}}\right] \tag{3.12}$$

To odpovídá vyjádření vztahu 3.10 jako funkce proměnné y, jejíž derivace podle x je rovna právě -1. Obdobně lze tento vztah vyjádřit pomocí parciálních derivací:

$$\frac{\partial F(x, y)}{\partial x} = 2b^2 x, \frac{\partial F(x, y)}{\partial y} = 2a^2 y \tag{3.13}$$

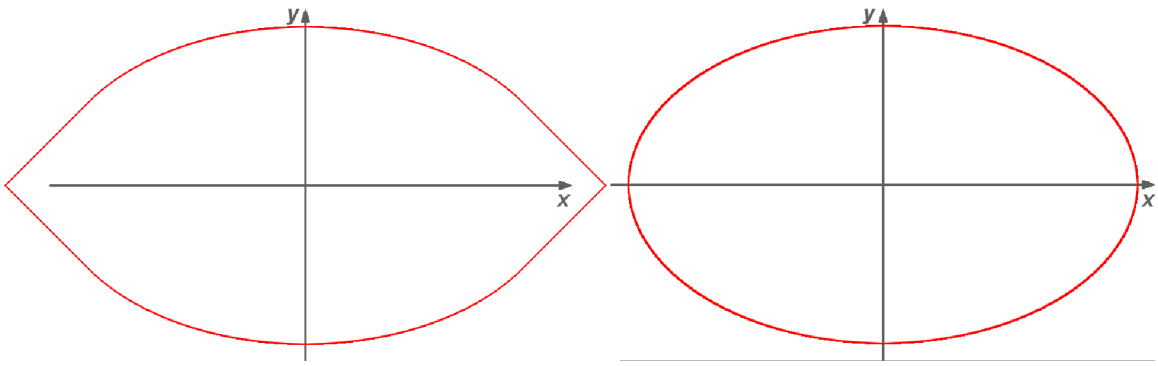
Rasterizace je tedy prováděna zvlášť pro dvě oblasti. V oblasti I je řídicí osou osa x a počáteční bod leží na souřadnici $[0, b]$. V oblasti II je poté řídicí osou osa y a vlastní výpočet je prováděn dokud je $y \geq 0$. V každém kroku dochází k posunu na řídicí ose o konstantní přírůstek. Přírůstek na vedlejší ose je dán polohou tzv. *Midpointu* (viz 3.2.1). Při výpočtech je rovněž použit rozhodovací člen, *Prediktor*. Pro oblast I je *prediktor* inicializován na:

$$p_I = b^2 + \frac{a^2(1-4b)-2}{4} \tag{3.14}$$

Pro oblast II je nutné nastavit novou hodnotu *prediktoru*. Jelikož celý algoritmus pracuje pouze s celými čísly, je nalezení vhodného vzorce pro nastavení *prediktoru* pro oblast II obtížné. Například vzorec uvedený v [5] není vhodný pro použití na počítačích s 32 bitovou architekturou. Při větších rozměrech elipsy je výsledná hodnota *prediktoru* mimo rozsah datového typu `Integer`[1], což ve výsledku způsobí deformaci jejího tvaru. Proto jako ideální vzorec pro nastavení hodnoty *prediktoru* pro druhou oblast jsem použil vzorec 3.15, který byl zveřejněn v [4].

$$p_{II} = p_i - \frac{a^2(4y-3)+b^2(4x+2)}{4} \tag{3.15}$$

Při použití tohoto vzorce nová hodnota *prediktoru* není tak vysoká a proto je elipsa vykreslena správně. Na obrázku 3.6 je vidět, jak se projeví použití obou vzorců při rasterizaci stejné elipsy.



Obrázek 3.6: Na obrázku vlevo je použit vzorec z [5], hodnota počátečního *prediktoru* pro oblast II potom vyšla $-2\,147\,483\,648$, kdežto na obrázku vpravo je použit vzorec 3.15 a hodnota *prediktoru* byla pouhých $-10\,111\,573$. Délka hlavní poloosy byla v obou případech 320 pixelů a délka vedlejší poloosy byla 200 pixelů.

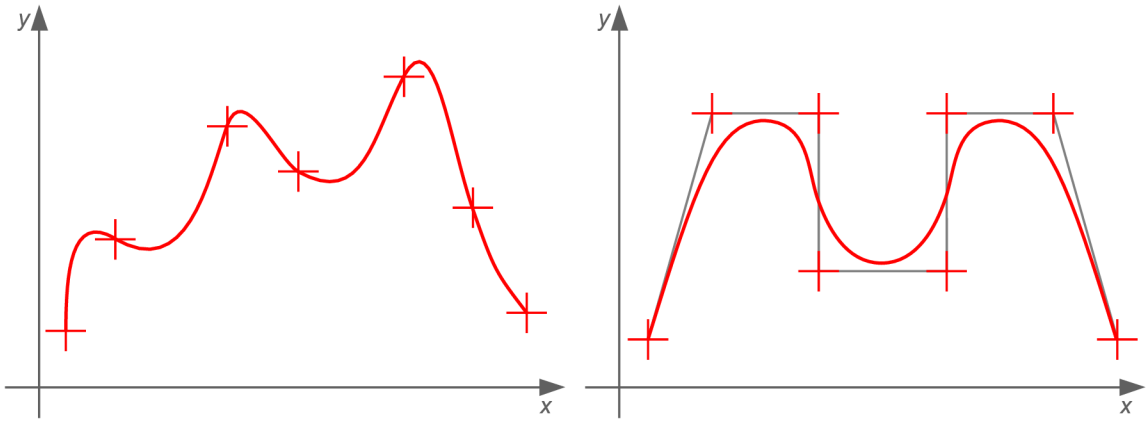
3.3 Křivky, interpolační křivky, aproximační křivky

V počítačové grafice se používají křivky *polynomiální*. Tyto křivky lze snadno vyčíslit a rovněž jsou jednoduše diferencovatelné. Polynomiální křivku můžeme zapsat pomocí rovnice:

$$Q_n(t) = a_0 + a_1 t + \ldots + a_n t \tag{3.16}$$

[1] Typ `Integer` je implementován na 32 bitech, tudíž je schopen správně zobrazit čísla v rozsahu $\langle -2^{31}, 2^{31}-1 \rangle$. Rozsah jakož i implementace typu `Integer` se může lišit v závislosti na používané architektuře.

Křivky je možné rozdělit na *interpolační křivky*, které přímo procházejí kontrolními body (*řídicí polygon*) a na *křivky aproximační*, kde kontrolní body určují výsledný tvar křivky. Ta však těmito body procházet nemusí.



Obrázek 3.7: Interpolační (vlevo) a aproximační (vpravo) křivka.

3.3.1 Fergusonovy kubiky

Jedná se o velmi často používaný druh interpolačních křivek. Někdy jsou též označovány jako *Hermitovské kubiky*. Tyto křivky byly popsány v [3]. Křivka je určena dvěma řídicími body P_0 a P_1 a jim příslušícími tečnými vektory $\overrightarrow{p'}_0$ a $\overrightarrow{p'}_1$. Kontrolní body určují polohu křivky a směr a velikost tečných vektorů určuje míru vyklenutí. Předpis pro Fergusonovy kubiky lze vyjádřit pomocí vztahu:

$$Q(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ \overrightarrow{p'}_0 \\ \overrightarrow{p'}_1 \end{bmatrix} \tag{3.17}$$

Po vynásobení matice vektorem lze získat hodnoty pro jednotlivé polynomy:

$$\begin{aligned} F_1(t) &= 2t^3 - 3t^2 + 1, \\ F_2(t) &= -2t^3 + 3t^2, \\ F_3(t) &= t^3 - 2t^2 + t, \\ F_4(t) &= t^3 - t^2. \end{aligned} \tag{3.18}$$

Při dosazení $t = 0$, resp. $t = 1$, lze ověřit, že křivka začíná, resp. končí v bodě P_0, resp. P_1. Výsledný vztah pro výpočet nového bodu křivky je potom definován rovnicí:

$$Q(t) = P_0F_1(t) + P_1F_2(t) + \overrightarrow{p'}_0F_3(t) + \overrightarrow{p'}_1F_4(t) \tag{3.19}$$

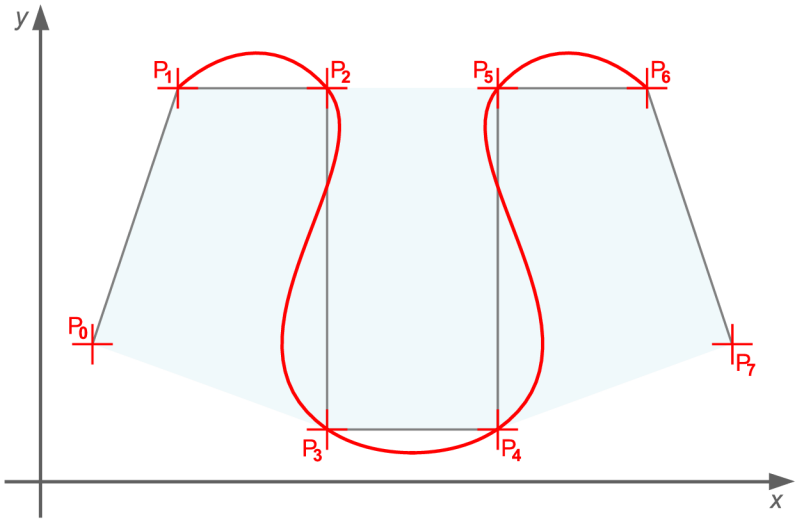
3.3.2 Catmull-Rom spline

Catmull-Rom spline je dalším typem interpolačních křivek. Vychází z *Kochanek-Bartels* křivek, které ochuzuje o nastavení tří parametrů pro každý bod. Tyto parametry dokáží

definovat napětí (*tension*), spojitost (*continuity*) a šikmost (*bias*) v každém řídicím bodě. Pro Catmull-Rom spline jsou pak všechny parametry rovny 0. Předpis lze vyjádřit pomocí maticového zápisu:

$$Q(t) = \frac{1}{2}\begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix}\begin{bmatrix} P_{i-3} \\ P_{i-2} \\ P_{i-1} \\ P_i \end{bmatrix} \tag{3.20}$$

Křivka nezačíná ve svém prvním řídicím bodě P_0, ale v bodě P_1 a stejně tak končí v předposledním řídicím bodě P_{n-2}. Pro dosažení interpolace v počátečním, resp. koncovém, řídicím bodě je třeba tyto body v seznamu bodů duplikovat. Další nepříjemnou vlastností Catmull-Rom křivek je, že neleží ve své konvexní obálce (viz 3.8). Přesto patří mezi oblíbené interpolační křivky, které se používají např. pro definici dráhy předmětu při animacích.



Obrázek 3.8: CatmullRom Spline a jeho konvexní obálka.

3.3.3 Beziérovy kubiky

První typem aproximačních křivek jsou *Beziérovy kubiky*. Jedná se o jedny z nejpopulárnějších křivek, které se používají např. pro definici typů písma nebo při šablonování. Beziérovy křivky n-tého stupně jsou obecně určeny $n+1$ řídicími body P_i. Pro zobrazení jedné kubiky je tedy zapotřebí minimálně čtyř bodů. Definice obecné Beziérovy křivky n-tého stupně je realizována vztahem

$$Q(t) = \sum_{i=0}^{n} P_i B_i^n(t), \tag{3.21}$$

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-1}; t \in \langle 0,1\rangle; i = 0, 1, \ldots n, \tag{3.22}$$

kde B_i^n jsou tzv. *Bernsteinovi polynomy* n-tého stupně. Pro křivku stupně tři lze odvodit

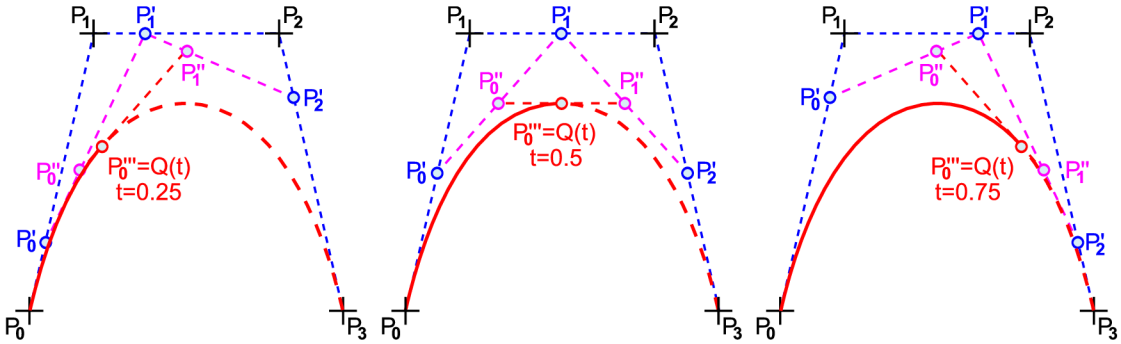
odpovídající vzorce pro výpočet Bernsteinových polynomů i nového bodu křivky:

$$\begin{aligned} B_1(t) &= (1-t)^3, \\ B_2(t) &= 3t(1-t)^2, \\ B_3(t) &= 3t^2(1-t), \\ B_4(t) &= t^3. \end{aligned} \tag{3.23}$$

Mezi základní vlastnosti těchto křivek lze zařadit například to, že křivka začíná v bodě P_0 a končí v bodě P_{n+1} a leží v konvexní obálce.

3.3.4 Algoritmus DeCasteljau

Tento algoritmus je používaný v počítačové grafice pro rasterizaci Beziérových křivek n-tého stupně. Využívá rekurentní definice *Bernsteinových polynomů* (viz 3.21). Úseky řídicího polynomu jsou děleny v poměru hodnot t a $1-t$. Princip algoritmu spočívá v tom, že na základě hodnoty proměnné t jsou vypočítány řídicí body pomocného polygonu, který je dále dělen podle stupně křivky.



Obrázek 3.9: Princip činnosti algoritmu *DeCasteljau* pro různé hodnoty t.

3.3.5 B-Spline

V počítačové grafice jsou nejčastěji používány kubické spline křivky. Při návrhu programu byl vybrán jako ukázkový typ těchto křivek tzv. *uniformní kubický B-spline*. Tento typ křivky vychází z *Coonsových křivek* a vznikne jejich navázáním přes několik segmentů. B-spline je definován $n \geq 4$ body a je složen z $n-3$ segmentů. Křivka je pak definována rovnicí:

$$Q(t) = \sum_{i=0}^{n} P_i N_i^k(t); t \in \langle 0, t_m \rangle, \tag{3.24}$$

kde N_i^k lze vyjádřit rekurentní vztahem:

$$\begin{aligned} N_i^k(t) &= \frac{(t-t_i)}{(t_{i+k}-t_i)} N_i^{k-1}(t) + \frac{(t_{i+k+1}-t)}{(t_{i+k+1}-t_{i+1})} N_{i+1}^{k-1}(t) \\ N_i^0(t) &= 1 \qquad \text{pro } t \in \langle t_i, t_{i+1}) \\ N_i^0(t) &= 0 \qquad \text{jinak} \end{aligned} \tag{3.25}$$

Parametr t prochází intervalem $\langle t_3, t_{n+1} \rangle$ a hodnoty parametru t_i v uzlech definují tzv. *uzlový vektor*. Pokud je řeč o *uniformní B-spline* křivce, vzdálenost $t_{i+1} - t_i$ je konstantní. *B-spline* křivka nezačíná, resp. nekončí, v počátečním, resp. koncovém, řídicím bodě. Lze však zajistit, aby v těchto bodech začínala, resp. končila, a to pouhou modifikací uzlového vektoru (pro $k = 2$):

$$U_t \quad = \quad (0, 0, 0, t_{k+1}, \ldots, t_{m-k-1}, 1, 1, 1) \tag{3.26}$$

Mezi základní vlastnosti *B-spline* křivek patří:

- Křivka je určena $n + 1$ body,
- stupeň křivky k zajišťuje spojitost křivky $k + 1$,
- křivka leží v konvexní obálce,
- lokální změna tvaru křivky (projeví se mezi dvěma sousedními segmenty).

Kapitola 4

Návrh aplikace

Před samotným vývojem aplikace, tedy vlastním psaním zdrojových kódů programu, je nutné věnovat mnoho úsilí návrhu implementace. Tento proces bývá nejdůležitější částí celého vývojového cyklu. Podle kvality návrhu je možné dopředu odhadnout, jak bude výsledný program úspěšný. Pokud je návrh kvalitní, samotné programování většinou zabere méně času a program je možné jednoduše rozšiřovat a vylepšovat i po jeho vydání. Méně kvalitní nebo špatně provedený návrh implementace potom zpravidla vede k tomu, že psaní zdrojových kódů je časově náročné a rozšiřování programu po jeho vydání je zdlouhavé, ne-li nemožné. Takovéto programy většinou končí neúspěchem a upadají v zapomnění.

Mou snahou bylo věnovat návrhu implementace dostatek času a promyslet všechny možné varianty ještě před začátkem programování. A to hlavně proto, aby samotný vývoj programu zabral co nejméně času a jeho další rozšiřování bylo pokud možno co nejjednodušší.

4.1 Vývojové prostředí

Vývojové prostředí již bylo specifikováno v kapitole 2.2. Po nelehkém výběru jsem zvolil grafickou nadstavbu na programovacím jazykem C++ *wxWidgets*. Tato nadstavba je momentálně ve verzi *2.8.10*. Při zahájení vývoje však byla poslední stabilní verzí verze *2.8.9*, která nakonec byla použita. Pro eliminaci případných problémů spojených s kompilací nadstavby do jednotlivých knihoven, především pro rozsáhlé možnosti nastavení kompilátoru, byl použit tzv. *wxPack*. Tento program v sobě obsahuje všechny knihovny již zkompilované, stejně jako hlavičkové soubory pro jednotlivé třídy nadstavby *wxWidgets*.

Pro psaní zdrojových kódů programu byl použit program *CodeLite* ve verzi 1.0, revize 2785. Jedná se praktický textový editor, který má v sobě zabudovanou podporu nadstavby *wxWidgets*. Dokáže tudíž rozpoznat jednotlivé třídy a barevně je zvýraznit v kódu, umí celý program přeložit a sestavit do výsledného binárního souboru. Pro překlad a sestavení programu pak byl zvolen kompilátor *MinGW* ve verzi 5.1.4. Jedná se balík programů s GNU licencí pro systém Windows.

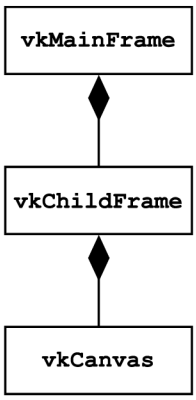
4.2 Třívrstvá hierarchie programu

Aplikace bude tvořena jedním hlavním oknem, které se uživateli zobrazí bezprostředně po spuštění programu. Toto okno v sobě ponese všechny hlavní prvky, které byly specifikovány

v kapitole 2.4. To znamená, že jednotlivé panely nástrojů, systémové menu i stavový řádek bude spravovat právě toto okno.

Dále je nutné, umožnit uživateli výběr objektu, který chce právě studovat. O to se bude rovněž starat hlavní okno. Jak však bylo nastíněno v kapitole 2.5.1, aplikace předpokládá dva hlavní režimy, v nichž se může nacházet. Editační a simulační část. Změna aktuálně vybraného objektu v simulační části však není možná, jelikož v této části běhu programu budou vypočítány všechny potřebné výsledky, jež se budou zobrazovat uživateli jako výstupy programu. A tyto výsledky jsou především závislé na aktuálně vybraném objektu (potažmo metodě, resp. typu křivky).

Takovéto chování programu je nutné, bohužel dosti omezující. Vhodné by bylo, aby si uživatel mohl vybírat objekt v jakékoliv části běhu programu. Proto je na místě přidat do programu další, dceřiné, okno. Tím se zamezí nutnosti spouštět více instancí programu zvlášť pro každý objekt. V jediné instanci aplikace pak bude možné spustit více dceřiných oken a pro každé nastavit požadovaný objekt. Mezi těmito okny lze jednoduše přepínat, zavírat je, nebo měnit jejich velikosti. Tímto opatřením je pak zátěž, která je kladena na hlavní okno, přenesena o úroveň níže. Přesto je patrné, že každé dceřiné okno bude určeno právě pro jeden vybraný objekt a ten bude možné měnit během editační části. Během simulační části však nikoliv. Toto chování je postačující.



Obrázek 4.1: Třívrstvá struktura programu v jazyce UML.

Jelikož byla hlavnímu oknu ponechána pouze obsluha událostí spojených s výběrem položky v systémovém menu nebo v jednom z panelů nástrojů, je jasné, že vlastní kreslící plátno bude potomkem dceřiného okna. Nyní je však problém, jak samotné plátno realizovat. Kreslící plátno, jak bylo uvedeno v kapitole 2.4, je nejdůležitější částí programu a právě pomocí něho dochází k zadávání vstupů od uživatele a k zobrazení výstupů (viz 2.5.1).

Jednou z prvních myšlenek při návrhu programu bylo ukázat uživatelům práci jednotlivých rasterizačních metod (resp. vykreslování různých typů křivek) najednou. Možné byly dva způsoby implementace:

1. plátno bude pouze jedno, uživatel definuje kontrolní body a vybere požadované metody, resp. typy křivek, a rasterizace se provede do tohoto plátna,

2. pláten bude více, do každého plátna uživatel definuje kontrolní body a každému plátnu nastaví požadovanou metodu, resp. typ křivky, rasterizace se poté provede v každém plátně zvlášť.

Oba způsoby jsou zajímavé, přesto druhý způsob nabízí více možností. Např. kdyby uživatel chtěl zadat dvě úsečky rozdílných délek, při použití prvního způsobu by to nebylo možné. Právě proto byl zvolen druhý způsob.

Pro rekapitulaci je tedy vhodné uvést, že aplikace se sestává z hlavního okna, které může obsahovat jedno či více zároveň otevřených dceřiných oken (pochopitelně, že je možné zavřít všechna dceřiná okna, potom však nelze zadávat vstupy ani sledovat výstupy programu). Každé dceřiné okno je přidruženo jednomu vybranému vektorovému objektu a obsahuje jedno nebo více kreslících pláten. Pro každé plátno je nastavena právě jedna metoda, resp. typ křivky, která se v daném plátně zobrazí. Tím vzniká třívrstvá struktura, hierarchie, programu jak je vidět na obrázku 4.1.

4.3 Realizace rastru

V počítačové grafice je rastr realizován jako matice, kde každý bod leží v určitém řádku a sloupci. Jednomu bodu zpravidla odpovídá jeden pixel. Kreslící plátno v této aplikaci bude simulovat právě takovouto matici, nebo-li mřížku. Pixel je jednotkou s proměnlivou velikostí v závislosti na aktuálně nastaveném rozlišení monitoru. Přesto je vždy příliš malý, aby byl vhodnou jednotkou pro znázornění průběhu rasterizace.

Z toho důvodu je nutné do kreslícího plátna implementovat funkci, jenž umožní jeho přibližování a oddalování. Tuto funkcionalitu bude mít na starosti jak hlavní okno (pomocí panelu nástrojů), tak i plátno samotné. Při přiblížení plátna bude nutné jeho celý obsah překreslit podle aktuální hodnoty přiblížení a jeden logický bod vzorkovat na odpovídající počet pixelů (fyzických bodů).

Jako výchozí hodnota přiblížení je 100%, která odpovídá skutečné velikosti. Jeden bod je pak roven jednomu pixelu. Při tomto přiblížení je téměř nemožné rozpoznat, který bod je aktuálně vykreslován, proto hodnoty menší než 100% (tedy další oddalování plátna) nebudou implementovány. Samotné přibližování pak bude realizováno po krocích, a sice na 200%, 400%, 800%, 1200%, 1600%, 2400% a 3200%. Každému přiblížení bude odpovídat velikost bodu na plátně (2 pixely, 4 pixely, ..., 32 pixelů).

S realizací rastru rovněž souvisí i vykreslování mřížky, která vnáší do plátna přehlednost a pořádek. Vykreslování mřížky je však závislé na aktuální hodnotě přiblížení. Při menších hodnotách není nutné, aby se mřížka vykreslovala okolo každého bodu, naopak by to působilo chaoticky a samotné body by mohly zaniknout ve směsi svislých a vodorovných čar. Naopak při větších hodnotách přiblížení je takové chování žádoucí.

4.4 Simulační část a výpisy výsledků

Výpisy výsledků jsou rovněž velmi důležitou částí programu, neboť uživateli podávají informaci o tom, které výpočty byly pro daný bod provedeny. Proto je nutné dosáhnout co největší přehlednosti při jejich zobrazení.

4.4.1 Režimy pro vypisování výsledků

Návrh počítá se třemi hlavními režimy pro zobrazování výsledků:

- Rychlý režim,
- normální (ladicí) režim,

- testovací režim.

První dva režimy se liší pouze při zobrazování výsledků. Zatímco *rychlý režim* zobrazuje všechny potřebné výsledky pro daný bod, *ladicí režim* pracuje podobným způsobem, jako ve známých vývojových prostředích tzv. *Debug mode*. To znamená, že se prochází přímo zdrojový kód vybraného algoritmu a pro každý řádek zobrazí aktuální hodnoty proměnných, které se na daném řádku nacházejí.

Testovací režim pak očekává jistou interakci s uživatelem i během simulační části běhu programu. Jde o jakési zkoušení uživatele ze znalostí algoritmů rasterizace. Po přechodu do simulační části uživatel nejprve zadá všechny body výsledného objektu, které považuje za správné. Program poté provede animaci rasterizace a opravu špatně zadaných bodů a uživateli zobrazí jeho hodnocení.

4.4.2 Rychlý režim pro výpis výsledků

Jak bylo uvedeno v 4.4.1, rychlý režim zobrazí v každém kroku simulace všechny důležité vzorce a výpočty, které bylo nutné provést. Aby bylo možné tyto výpisy dobře zobrazovat a spravovat je nutné najít nástroj se silnou vyjadřovací schopností.

Klasické zobrazování textu nestačí. Jako vhodný nástroj se přímo nabízí využít značkovacího jazyka HTML. Tento jazyk se používá k vytváření internetových stránek, proto je v něm největší důraz kladen především na vizuální stránku a sdělení. *wxWidgets* v sobě mají třídy, které dokáží s HTML pracovat a interpretovat ho. I když se jedná pouze o základní HTML – rozšíření jako CSS (*Cascading Style Sheet*) nebo *JavaScript* nelze použít – zobrazení výsledků bude snadné a především přehledné.

4.5 Omezení programu

Z výše popsaného návrhu nejdůležitějších částí programu vyplývá několik omezení, se kterými bude nutné počítat i při vývoji.

4.5.1 Výběr metody, typu křivky

Jelikož byla zvolena varianta s více plátny v jednom dceřiném okně, je nutné najít vhodný způsob jak realizovat výběr rasterizační metody, resp. typu křivky, pro dané plátno.

Výběr objektu, potažmo metody, je vhodné realizovat pomocí roletkového menu, tzv. *combo boxu* (ve *wxWidgets* realizován třídou *wxComboBox*). Pokud však bude otevřeno více pláten najednou, bude obtížné určit, pro které plátno je výběr požadován. Proto při otevření více pláten bude možnost výběru metody, resp. typu křivky, z roletkového menu (tedy z hlavního okna programu) znemožněna. Výběr pak lze realizovat pomocí tzv. *kontextového menu*, které se zobrazí při kliknutí pravým tlačítkem myši nad aktuálním plátnem. Nově nastavená metoda, resp. typ křivky, se zobrazí v hlavičce plátna.

Kapitola 5

Implementace

Aplikace se skládá ze tří částí:

1. Hlavní okno aplikace definující uživatelské rozhraní (*GUI*),
2. dceřiné okno obsahující kreslící plátna,
3. simulátor a správce výsledků.

Při vývoji bylo využíváno zvyklostí a návrhových vzorů popsaných v [6]. Knihovna *wxWidgets* je čistě objektová, proto každý prvek je objektem definovaný třídou. Ve zdrojových kódech byla dále zavedena jednotnost názvů tříd, proměnných a názvů metod podle tabulky 5.1.

`vkMainFrame`	název třídy
`CalcLogicalPosition()`	název proměnné
`main_frame`	název metody

Tabulka 5.1: Zavedení konvence jmen ve zdrojových kódech.

Jak je vidět, názvy tříd začínají prefixem `vk` a jejich vlastní jméno začíná vždy velkými písmeny. Jména jednotlivých metod rovněž začínají velikými písmeny, avšak narozdíl od tříd nemají žádný prefix. A konečně názvy proměnných jsou psány malými písmeny. Pro oddělení slov v názvech proměnných je použit znak _ (podtržítko).

V následujícím textu bude popsána implementace všech tří částí aplikace a jejich nejdůležitějších nebo jinak zajímavých částí.

5.1 Hlavní okno aplikace a *GUI*

5.1.1 Inicializace aplikace, třída `vkApp`

Při spuštění programu, který používá nadstavbu *wxWidgets*, je vždy nejprve volána instance třídy, která je zděděná od třídy `wxApp`. V této aplikaci to je třída `vkApp`. Tato třída představuje hlavní třídu celé aplikace, podobně jako funkce `main()` v programovacím jazyce C. Bezprostředně po inicializaci třídy `vkApp` je volána její metoda `OnInit()`, která se postará o alokaci potřebné paměti, zpřístupnění ovladačů pro jednotlivé typy obrázků a dalších nezbytných systémových zdrojů. Hlavním úkolem metody však je vytvoření instance třídy hlavního okna a jeho zobrazení.

5.1.2 Hlavní okno, třída `vkMainFrame`

Hlavní okno aplikace bývá zpravidla zděděno od třídy `wxFrame`. Pro tuto aplikaci však použití této třídy bylo nedostačující, a to z důvodu možnosti správy a vytváření několika dceřiných oken. Tato hierarchie je zajišťována dvojicí tříd nazvaných `wxMDIParentFrame` a `wxMDIChildFrame` (*Multiple Document Interface*). První třída představuje hlavní, rodičovské okno a druhá pak jeho potomka, tedy okno dceřiné.

Hlavní okno v sobě nese definici položek systémového menu a rovněž všech panelů nástrojů. Při jeho vytváření je nejprve volána metoda `InitDefaults()`. Ta nastaví implicitní hodnoty všech důležitých třídních proměnných, které je rovněž možné nastavovat i z externích konfiguračních souborů. Tato funkcionalita však implementována nebyla. Po inicializaci je možné vytvořit systémové menu (metoda `InitMenuBar()`), panely nástrojů (metoda `InitToolBar()`) a konečně i stavový řádek (metoda `InitStatusBar()`).

5.1.3 AUI, *Advanced User Interface*

Jak bylo stanoveno při vytyčení cílů, které budou od programu očekávány (viz 2), je od grafického rozhraní očekáváno intuitivní ovládání a přehledné rozmístění prvků programu. Tyto prvky jsou zobrazeny v několika tématicky oddělených panelech nástrojů, které korespondují s položkami v systémovém menu. Aby však bylo možné měnit vzhled aplikace při jejím běhu, bylo nutné sáhnout po knihovně, která nese jméno AUI (*Advanced User Interface*) publikované na [8].

Pomocí této knihovny, lze docílit možnosti přeskupování panelů nástrojů, podle potřeb uživatele. Dále je možné jednotlivé panely vyjmout z celkového rozložení aplikace a nechat je volně „plavat“ nad aplikací jako samostatná okna. Rovněž je možné některé panely skrýt a znovu zobrazit.

O tuto činnost se stará instance třídy `wxAuiManager` pojmenovaná `toolbar_mgr`, která v sobě uchovává informace o všech panelech, jež má spravovat. Při změně některého z panelů či jeho prvků je nutné volat metodu `Update()`, která se postará o překreslení celého panelu. Bohužel to však v prostředí Windows nefunguje úplně správně. Po zavolání metody `Update()` nedojde k překreslení celého panelu nástrojů. Dotyčný panel se překreslí až při najetí myši nad něj. Toho chování bohužel nelze ovlivnit bez zásahu do zdrojových kódu samotné knihovny.

5.1.4 Obsluha událostí

Aby bylo možné program ovládat a reagovat na asynchronní události přicházející od uživatele, ať už prostřednictvím klávesnice nebo myši, bylo nutné pro každý prvek nacházející se v okně programu, stejně tak pro každou položku v systémovém menu definovat sadu událostí a implementovat jim odpovídající množinu metod. Každá z těchto metod se stará o správné chování programu. Ať už se jedná o provedení nějaké události (např.: zavření programu nebo zobrazení nápovědy) nebo pouze o nastavení vnitřních proměnných (např. výběr objektu a jeho metody pro rasterizaci apod.).

O obsluhu těchto událostí se stará tzv. *tabulka událostí*, jejíž část je znázorněna v 5.2. Pomocí této tabulky je možné asociovat všechny prvky aplikace (resp. dané třídy) s požadovanou událostí. Ta je pak ošetřena příslušnou funkcí. *Tabulka událostí* má obecně tvar viz 5.3.

Klíčové slovo `EVENT` označuje typ události, která má být obsloužena, klíčové slovo `func` potom definuje metodu, která danou událost obslouží. Pro asociaci události s nějakým

```
BEGIN_EVENT_TABLE(vkMainFrame, wxMDIParentFrame)
     EVT_CLOSE          (vkMainFrame::OnClose)
     EVT_MENU           (MENU_ID_QUIT, vkMainFrame::OnQuit)
     EVT_COMBOBOX       (CTRL_ID_METHOD, vkMainFrame::OnSelectMethod)
     EVT_MENU           (MENU_ID_METHOD_0, vkMainFrame::OnSelectMethod)
     EVT_MENU           (MENU_ID_SIM_BUILD, vkMainFrame::OnSimBuild)
     EVT_BUTTON         (MENU_ID_SIM_BUILD, vkMainFrame::OnSimBuild)
     EVT_SPINCTRL       (CTRL_ID_SPEED_SPIN, vkMainFrame::OnSpeedChange)
     EVT_AUI_PANE_CLOSE (vkMainFrame::OnToolbarClose)
     ...
END_EVENT_TABLE()
```

Tabulka 5.2: tabulka událostí podle konvencí *wxWidgets* pro třídu `vkMainFrame`.

prvkem systémového menu nebo panelu nástrojů je nutné zadat i jeho unikátní identifikátor (klíčové slovo `ID` v tabulce 5.3) v rámci aplikace. Tento identifikátor je zpravidla realizován výčtovým typem a nabývá kladné celočíselné hodnoty.

```
BEGIN_EVENT_TABLE(class, parent_class)
     EVENT      (func)
     EVENT      (ID, func)
END_EVENT_TABLE()
```

Tabulka 5.3: Obecná struktura *tabulky událostí* ve *wxWidgets*.

5.2 Dceřiné okno, třída `vkChildFrame`

Jak bylo uvedeno výše, pro implementaci hlavního okna byla použita knihovna *MDI* a hlavní okno je zděděno od třídy `wxMDIParentFrame`. Jak vyžaduje samotné používání této třídy, je nutné, aby dceřiné okno bylo zděděno od třídy `wxMDIChildFrame`. Dceřiné okno je implementováno jako hlavní prostředník mezi vrstvou nejvyšší (tedy hlavním oknem aplikace) a vrstvami na nejnižší úrovni hierarchie (kreslící plátno, simulátor a správce výsledků).

5.2.1 Vytvoření a správa kreslícího plátna

Jak bylo nastíněno při návrhu aplikace v kapitole 4.2, každé dceřiné okno bude obsahovat jedno nebo více kreslících pláten. Díky tomu lze pro každé plátno definovat jinou metodu rasterizace, resp. typ křivky. Také je možné pro všechna plátna zadat stejnou rasterizační metodu, ale v každém plátně definovat požadovaný objekt různými kontrolními body. Potom lze sledovat jak vypadá rasterizace např. pro objekt úsečky, jejichž délky se liší pouze o několik pixelů. Toto chování je jistě zajímavé a pro výukové účely přínosné.

Pro realizaci vhodného rozvržení pláten v dceřiném okně je opět, podobně jako pro správu panelů nástrojů, použita vestavěná knihovna *AUI*. Pomocí ní lze docílit správného

rozložení všech pláten v dceřiném okně. Dále je možné, díky možnosti zobrazení záhlaví pro každé pole (angl. *pane*), rychle informovat uživatele o metodě, resp. typu křivky, pro dané plátno prostým textem v tomto záhlaví. Text je možné libovolně měnit v závislosti na zvolené metodě. Kromě samotného titulku okna je povoleno zobrazit tlačítko pro zavření (přesněji řečeno pro schování) příslušného pole i s jeho obsahem a také tlačítko pro maximalizaci daného pole.

O správu všech polí, resp. kreslících ploch se stará instance třídy `wxAuiManager` pojmenovaná v tomto případě `canvas_mgr`. Vytvoření nového, resp. zavření stávajícího, pole, má na starosti metoda `CreateNewCanvas()`, resp. metoda `OnCanvasClose()`. Samotné vytvoření nového pole však není triviální záležitostí. Je nutné vždy upravit rozložení stávajících polí. Jinak by hrozilo špatné zobrazení a to by v celkovém důsledku způsobilo nevzhlednost aplikace.

5.2.2 Komunikační protokol

Jelikož je dceřiné okno pouhým prostředníkem mezi nejvyšší vrstvou (hlavní okno aplikace) a vrstvou nejnižší (kreslící plátno), bylo nutné implementovat zjednodušený komunikační protokol, pomocí něhož lze z hlavního okna sdělit nastanuvší událost do všech otevřených pláten aktivního okna.

Celá komunikace sestává ze tří částí:

1. Po události v hlavním okně je volána veřejná metoda dceřiného okna `OnChangeValue()`, která jako jediný parametr obsahuje identifikátor události,

2. podle identifikátoru události je poté zpětně v těle metody `OnChangeValue()` volána příslušná veřejná metoda hlavního okna, která pouze vrací nově nastavenou hodnotu nějakého prvku okna,

3. nová hodnota je uložena do třídní proměnné dceřiného okna a poslána všem otevřeným plátnům pomocí příslušné veřejné metody kreslícího plátna.

Díky tomu je možné zajistit nastavování všech možných parametrů a přímou změnu ihned promítnout do všech pláten. Nová hodnota je rovněž uložena v dceřiném okně. Důvod pro toto chování je nejlepší vysvětlit na příkladě:

> *Mějme spuštěnou aplikaci, která bude mít otevřena dvě dceřiná okna. V každém okně je možné nastavit různý objekt nebo různou barvu pozadí. Nehledě na to, že každé okno se může nacházet v různém režimu (editační nebo simulační část běhu programu). Jelikož je vyžadováno přepínání mezi dceřinými okny, je nutné zpětně nastavit jednotlivé položky v panelech nástrojů v hlavním okně na aktuální hodnoty podle aktivního okna.*

Při události `EVT_ACTIVATE` (dceřiné okno se stalo aktivním), je v její obslužné metodě `OnActivate()` informováno hlavní okno a jsou mu poslány hodnoty všech proměnných dceřiného okna. To zajistí správné překreslení hlavního okna. O příjem těchto hodnot a překreslení se stará kolekce veřejných metod hlavního okna uvozená prefixem `Set` (z angl. *Setter*).

5.2.3 Editační vs. simulační část

Program je implicitně spouštěn v editační části a přechod do části simulační je umožněn až tehdy, kdy každé otevřené kreslící plátno obsahuje dostatek kontrolních bodů pro daný objekt nebo vybranou metodu, resp. typ křivky. Pokud by tomu tak nebylo, přechod do simulační části by mohl skončit chybou a pádem programu.

O vlastní přechod do simulační části se stará tlačítko **Build**, nebo jemu příslušná položka v systémovém menu. Toto tlačítko je implicitně neaktivní. Aktivním se stává po splnění všech podmínek uvedených v minulém odstavci. Při obsluze události je předáno řízení aktivnímu dceřinému oknu, přesněji jeho metodě `StartSimulators()`. Tato metoda reprezentuje svým chováním kompilaci programu. Nejprve je zobrazeno modální okno, ve kterém je zobrazen průběh kompilace. Poté se pro každé otevřené plátno vytváří instance třídy `vkSimulator`. Tato třída je zodpovědná za provedení všech výpočtů a uložení potřebných výsledků pro demonstraci rasterizace. Její funkcionalita je popsána v kapitole 5.4.1.

Aby byl uživateli umožněn návrat zpět do editační části programu, je k dispozici tlačítko **Clean**. Toto tlačítko je po spuštění rovněž neaktivní a aktivním se stává po úspěšné kompilaci a spuštění simulační části. Příkaz `clean` zpravidla slouží k vymazání binárních souborů vzniklých při překladu programu. A i zde slouží tlačítko **Clean** ke zrušení všech pomocných instancí třídy `vkSimulator` a správce výsledků (viz 5.4.2). Pro obsluhu této události je volána metoda dceřiného okna `CleanSimulators()`. Po zpětném návratu do editační části je opět možné přidávat nové, resp. mazat stávající, kontrolní body, měnit typ objektu nebo metody, resp. typy křivek.

5.3 Kreslící plátno, třída `vkCanvas`

Třída `vkCanvas` realizuje samotný rastr. Tedy mřížku, do které se zadávají, resp. vykreslují, kontrolní body, resp. body, pixely, výsledného objektu. Třída je zděděná z vestavěné třídy `wxScrolledWindow`. Nad plátnem jsou definovány události, které mohou nastat v jedné z částí běhu programu nebo v obou částech. V tabulce 5.4 je znázorněno, na které události se reaguje v jednotlivých částech běhu programu.

přiblížení/oddálení plátna	editační i simulační část
pohyb po plátně	editační i simulační část
změna barev popředí a pozadí plátna	editační i simulační část
zobrazení/skrytí kontrolních bodů	editační i simulační část
zobrazení/skrytí pomocných objektů	editační i simulační část
zadávání/mazání/změna pozice kontrolních bodů	editační část
spuštění/zastavení/krokování/změna rychlosti animace	simulační část

Tabulka 5.4: Události nad kreslícím plátnem podle aktuálního režimu běhu programu.

5.3.1 Zavedení souřadnicového systému plátna

Aby bylo možné realizovat přiblížení, resp. oddálení plátna, a poté správně vykreslovat prvky (kontrolní body, pomocné objekty, jednotlivé body rasterizovaného objektu), je nejprve nutné zavést souřadnicový systém. Pro třídu `wxScrolledWindow`, resp. pro plochu

určenou pro kreslení objektů, tzv. *Device Context*, je charakteristické to, že bod o souřadnicích $[0, 0]$ leží v levém horním rohu okna. Osa x roste zleva doprava a osa y roste z hora dolů. Takováto reprezentace souřadnicového systému však není přirozená, proto je nutné provést překlopení osy y. K tomu lze použít vzorec:

$$y = |y - h|\,, \tag{5.1}$$

kde h je výška okna. Aby byla orientace na plátně snadnější jsou vždy vykreslovány obě hlavní osy. Celé plátno je tedy rozděleno na 4 kvadranty a souřadnice bodů jsou přepočítány na logické souřadnice na plátně, které jsou běžnému uživateli již přirozené. O přepočet fyzických souřadnic (tedy pozice myši na plátně) na souřadnice logické se stará metoda `CalcLogicalPos()`.

Při zobrazení výsledků v jednotlivých krocích však nastává problém při použití těchto souřadnic. A to z důvodu, že objekt může být definován kontrolními body, které leží v různých kvadrantech. Při operacích jako je sčítání nebo odčítání by tudíž nastal chaos a rozpor se zobrazovanými výsledky. Proto se při zobrazování výsledků používají absolutní souřadnice plátna, kde bod $[1, 1]$ leží v levém dolním rohu plátna. Bod $[0, 0]$ na plátně neexistuje.

5.3.2 Přibližování, oddalování, pohyb po plátně

Velikost plátna je implicitně nastavena na 640×640 pixelů a není možné ji měnit. Tato velikost je dostatečná pro demonstraci rasterizace všech objektů popsaných v kapitole 2.3. Velikost plátna je při rozlišení obrazovek používaných v dnešní době dostačující na to, aby se celé plátno vešlo do okna aplikace. Jelikož je nutné implementovat funkci přiblížení a oddálení (nástroj *Lupa*), bylo potřeba věnovat značné úsilí realizaci pohybu po plátně. Tento problém by šel vyřešit pouhým zvětšováním plátna podle aktuální hodnoty přiblížení. To by ovšem mohlo způsobit pád programu v důsledku nedostatku paměti. Kreslící plátno by totiž při přiblížení 3200% mělo velikost 20480×20480 pixelů, přičemž reálně viditelná oblast by tvořila pouhých 0,25% plátna.

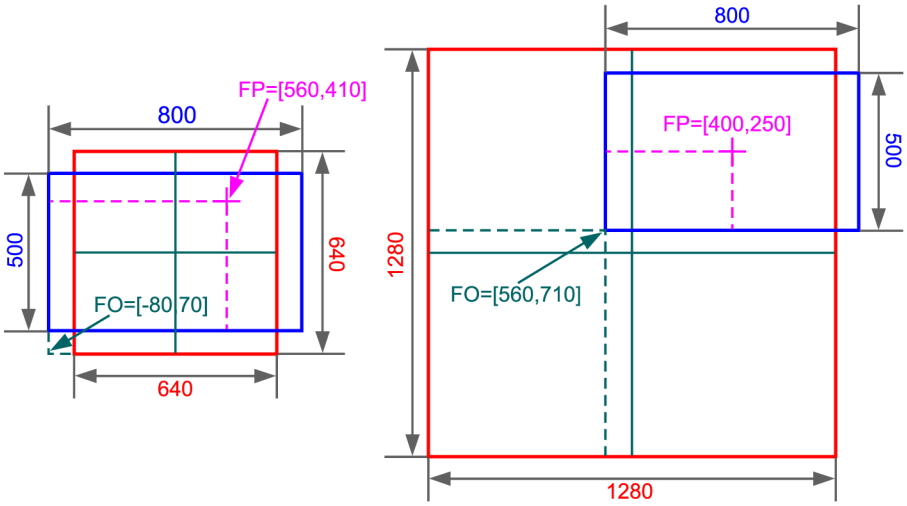
Ve výsledku je posouvání plátna realizováno opačně. Plátno zůstává staticky na jednom místě, mění se však pozice okna nad plátnem. Tato pozice je uložena v proměnné `focus_offset` a její přepočet je dán vzorci:

$$\texttt{focus\_offset} = (\texttt{focus\_point} \cdot \texttt{ratio}) - \texttt{size}/2, \tag{5.2}$$

$$\texttt{focus\_offset} = \texttt{focus\_offset} + (\texttt{m\_last\_drag} - \texttt{m\_drag}) \tag{5.3}$$

pro každou z os. Vzorec 5.2 je použit při změně hodnoty přiblížení plátna, kde proměnná `focus_point` představuje pozici myši nad plátnem, kde došlo k požadované události (pokud je tato událost volána z panelu nástrojů hlavního okna, je hodnota proměnné nastavena na jednu polovinu velikosti okna), proměnná `ratio` pak značí podíl velikostí jednoho logického bodu pro aktuální přiblížení a pro předchozí přiblížení a konečně proměnná `size` udává velikost okna. Vzorec 5.3 je používán při přesouvání plátna pomocí nástroje *Ručička*. Proměnná `m_last_drag`, resp. `m_drag`, uchovává pozici myši při události kliknutí levým tlačítkem myši, resp. uvolnění levého tlačítka myši. Lepší představu posuvu okna nad plátnem lze získat z obrázku 5.1.

Díky těmto opatřením je možné do okna vykreslit pouze skutečně viditelnou oblast plátna, čímž lze ušetřit paměť a zrychlit běh samotného programu hlavně při událostech spojených s plátnem.



Obrázek 5.1: Na obrázku je zobrazeno plátno (červeně) a vlastní okno aplikace (modře), `focus_offset` je znázorněn jako *FO* a `focus_point` jako *FP*. Obrázek demonstruje přiblížení ze 100% (vlevo) na 200% (vpravo). Tato akce byla vyvolána kolečkem myši na pozici 560×410 nad oknem.

5.3.3 Správa kontrolních bodů

Kromě činností popsaných výše je možné nad plátnem provádět další operace. Tou hlavní je zadávání kontrolních bodů objektu - vstupů od uživatele. Zadání kontrolního bodu je realizováno pouhým kliknutím nad plátno. Poté je bod přidán do seznamu kontrolních bodů a na plátno vykreslen. Vložené kontrolní body je pak možné přemisťovat a to pouhým uchycením bodu a tažením na jinou pozici. Dále je možné body mazat a to buď dvojklikem myši nad bodem nebo pomocí položky z kontextového menu.

5.3.4 Překreslování plátna

Aby bylo možné zobrazovat uživateli vždy aktuální obsah plátna, je nutné plátno překreslovat. O to se stará metoda `RedrawCanvas()`, která volá další pomocné metody, jež se starají o vykreslení skupiny objektů plátna podle aktuální části, v níž se program nachází a podle nastavení globálních parametrů programu.

Tyto činnosti jsou zpravidla tři. V první části je vykreslena mřížka rastru podle aktuálně nastavené hodnoty přiblížení. Pokud je zobrazování mřížky zakázané, vykreslí se pouze osy x a y. Tato činnost je definována v metodě `DrawGrid()`. Další fází je vykreslení kontrolních bodů jako křížků vždy ve středu pixelu, na kterém leží. Poslední fází, alespoň v editační části, je vykreslení pomocného objektu, neboli skutečné podoby objektu jako vektoru. Tento objekt je nezávislý na aktuální hodnotě přiblížení. Při vykreslování úsečky a elipsy jsou použity vestavěné metody `DrawLine()` a `DrawEllipse()`. Při vykreslování křivek jsou volány pomocné instance tříd, podle aktuálního typu křivky, které provedou výpočet všech bodů, ze kterých je křivka složena. Výsledné pole bodů je poté vykresleno jako polygon pomocí vestavěné metody `DrawLines()`. O tyto činnosti se stará metoda `DrawControlPoints()`.

Pokud se aktuální plátno nachází v simulační části, je nutné vykreslovat jednotlivé body, resp. úsečky, výsledného objektu, jež se rasterizuje. K tomu slouží metody `DrawPixel()`, resp. `DrawCanvasLine()`.

5.3.5 Demonstrace reasterizace

Hlavním účelem programu je demonstrovat rasterizaci vektorového objektu. Ta je prováděna také na kreslícím plátně. Demonstrace probíhá přehráváním animace, kdy je v každém kroku vykreslen jeden segment objektu (bod v případě úsečky a elipsy, nebo přímka v případě křivek). Ovládání animace je realizováno v hlavním okně aplikace, animace samotná je však implementována přímo ve třídě `vkCanvas`.

Při spuštění animace je volána metoda `Start()` instance objektu `wxTimer`, která jako jediný parametr předpokládá rychlost animace v milisekundách. Pro zdůraznění, který segment je v daném kroku vypočítán a vykreslován, je celá animace rozdělena do mezikroků, kdy změnou barvy vykreslovaného segmentu je docíleno jeho blikání. O to se stará metoda `OnTimer()`, která v závislosti na hodnotě proměnné `subframe_state` nastaví správnou barvu a nechá překreslit plátno.

5.4 Simulátor a správce výsledků

Nedílnou součástí programu je simulační část, ve které je uživateli demonstrována rasterizace požadovaného objektu. Kromě vlastního vykreslování objektu je nutné vypočítat a zobrazit výsledky jednotlivých kroků simulace.

5.4.1 Třída `vkSimulator`

O výpočet všech potřebných výsledků se stará třída `vkSimulator`. Při přechodu do simulační části (po stisknu tlačítka **Build**) je pro každé plátno vytvořena jedna instance této třídy. Každá instance ví, kterému plátnu náleží a ihned po inicializaci si od plátna vyžádá všechny potřebné hodnoty, tj. seznam kontrolních bodů, aktuální objekt a aktuální metodu, resp. typ křivky.

Na základě těchto hodnot je poté zavolána privátní metoda této třídy, která obsahuje algoritmus pro výpočet všech bodů daného objektu. Tyto body jsou ukládány do speciálního seznamu snímků `frames_list`, který je následně předán zpět příslušnému plátnu pro vykreslování objektu při demonstraci rasterizace.

Kromě seznamu snímků je uložen druhý seznam, `results_list`, do něhož jsou ukládány výsledky jednotlivých kroků výpočtu. Tento seznam je tvořen jednou ze tříd pro uchování výsledků, které představují pouze pomocné úložiště. Seznam je poté předán třídě `vkResultHtmlHandler`, která se stará o zpracování a zobrazení výsledků v HTML podobě (*Rychlý režim* - viz 4.4.1).

5.4.2 Třída `vkResultHtmlHandler`

Třída `vkResultHtmlHandler` je rovněž inicializována při přechodu do simulační části. Inicializace se však provádí v metodě hlavního okna `CreateResultBar()`, kde rovněž dochází k vytvoření nového pole a přidání tohoto pole pod správu *AUI manažera* (viz 5.1.3). Dceřinému oknu je poté předán ukazatel na nového správce výsledků a další komunikace je prováděna pouze z tohoto okna.

Poté, co dceřiné okno získá ukazatele na správce výsledků, zavolá jeho hlavní metodu `SetSimulators()`, jejímž parametrem je seznam instancí tříd `vkSimulator`. Správce výsledků si vyžádá od každé instance `vkSimulator` seznam výsledků, `results_list`, a provede vygenerování HTML kódu pro každý krok zvlášť. Jednotlivé kroky odpovídají vždy jednomu

řádku tabulky a jejich HTML kód je uložen do seznamu `frames_list`. První krok je krokem inicializačním a je zobrazován vždy. Každý další krok je zobrazován podle aktuálního snímku animace. O zobrazení aktuálních výsledků se stará metoda `RegenerateHtmlContent()`, která má jako jediný parametr číslo aktuálního snímku. Tato metoda vygeneruje nový HTML kód včetně nového snímku a zobrazí ho.

Při zobrazování výsledků je možné použít tři režimy zobrazování:

- Aktuální snímek je přidán na začátek tabulky,
- aktuální snímek je přidán na konec tabulky,
- je zobrazen pouze aktuální snímek a inicializační řádek.

Změnu režimů je možné provádět pomocí kontextového menu přímo ve správci výsledků. Na závěr je vhodné ještě uvést, že správce výsledků je pouze jeden pro každé dceřiné okno. Pokud má okno otevřeno více pláten, každému plátnu poté odpovídá jeden sloupec tabulky ve výpisu výsledků.

Kapitola 6

Ovládání aplikace

Tato kapitola ve stručnosti popisuje základní prvky programu a jeho ovládání.

6.1 Hlavní prvky aplikace

6.1.1 Systémové menu

Systémové menu obsahuje tyto hlavní položky:

- **File** – hlavní menu pro otevření nového dceřiného okna, nebo zavření aplikace,
- **View** – v tomto menu je možné zobrazovat nebo skrývat jednotlivé panely nástrojů, okno výsledků (pouze v simulační části) a stavový řádek,
- **Object** – slouží pro výběr vektorového objektu a jeho metody, resp. typu křivky,
- **Canvas** – zde je možná přidat nové kreslící plátno do aktivního dceřiného okna, nastavovat barvy pozadí a popředí, hodnotu přiblížení a zobrazování pomocných obrazů vektorového objektu a jeho kontrolních bodů,
- **Playback** – pomocí položek v tomto menu je možné přepínání mezi jednotlivými částmi aplikace, definice režimu zobrazování výsledků, přehrávání simulace nebo zvyšování či snižování její rychlosti,
- **Window** – pomocné menu pro tzv. *MDI* schéma aplikace, správa dceřiných oken,
- **Help** – pomocí tohoto menu lze zobrazit informace o autorovi programu, online nápovědu a nápovědu k implementovaným metodám.

6.1.2 Panely nástrojů

Panelů nástrojů bylo implementováno celkem šest. První čtyři jsou horizontální a lze je umístit buď v horní nebo spodní části aplikace. Každý panel nástrojů nese nejdůležitější prvky menu:

- **Standardní panel** – odpovídá menu **File**,
- **Plátno** – odpovídá položce menu **Canvas**,

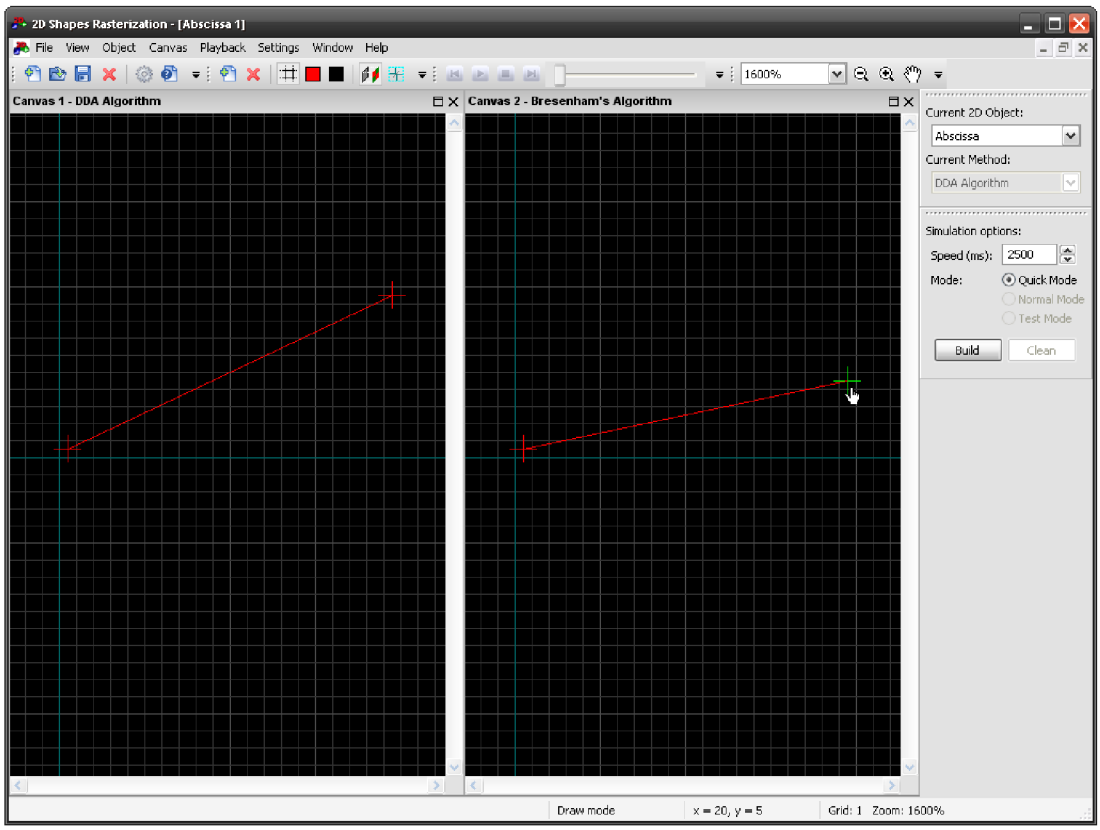
- **Přehrávač** – odpovídá položce **Playback**, navíc obsahuje posuvník (angl. *Slider*), který slouží k nastavení aktuálního snímku během simulace,
- **Lupa** – odpovídá menu **Canvas**, ale obsahuje pouze položky pro nastavení aktuální hodnoty přiblížení a nástroj *Ručička*, pomocí něhož lze pohybovat s plátnem metodou *„Táhni a pusť“* (angl. *Drag and Drop*).

Zbylé dva panely nástrojů jsou tvořeny boxy a lze je umístit do levé nebo pravé části aplikace.

- **Objekt** – odpovídá položce **Object** v systémovém menu, obsahuje dvě roletková menu pro výběr objektu a jeho metody, resp. typu křivky, pokud je otevřeno více kreslících pláten najednou, je výběr metody znemožněn,
- **Simulace** – odpovídá položce **Playback** a obsahuje nastavení režimu zobrazování výsledků, rychlost simulace a tlačítka pro přepínání mezi jednotlivými částmi programu.

6.2 Editační část

Po spuštění se aplikace nachází v editační části svého běhu. V této části je možné definovat vektorový objekt a jeho metodu, resp. typ křivky. Aplikace je znázorněna na obrázku 6.1.



Obrázek 6.1: Okno aplikace v editační části.

Zadávání kontrolních bodů je realizováno pomocí myši prostým kliknutím levým tlačítkem nad kreslícím plátnem. Přiblížením kurzoru myši k již vloženému body je tento bod

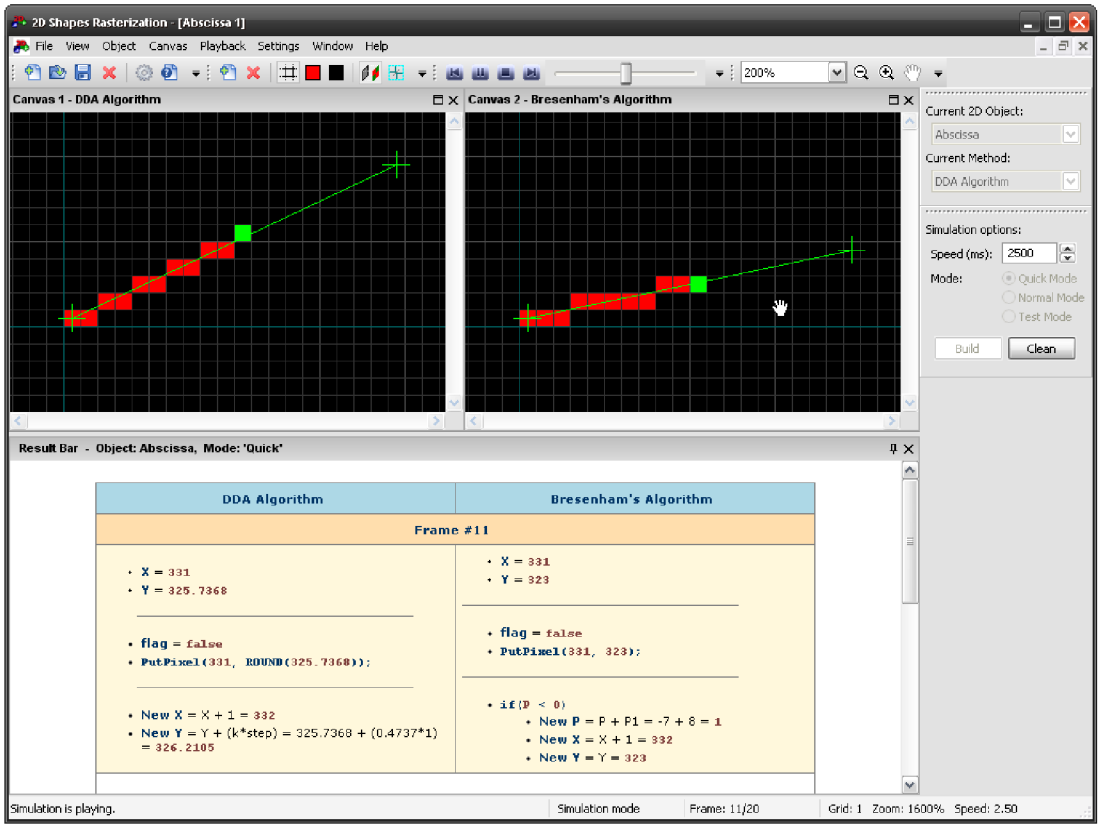
nastaven jako aktivní, je vykreslen jinou barvou (implicitně čistá zelená) a je možné ho smazat, nebo změnit jeho pozici uchycením a přetažením bodu.

Přibližování, resp. oddalování, lze provádět dvěma způsoby. Buď pomocí příslušné položky v systémovém menu nebo panelu nástrojů, nebo pomocí kolečka myši. Pootočením kolečka myši od sebe, resp. k sobě, je plátno přiblíženo, resp. oddáleno. Pohled je vystředěn podle pozice myši nad plátnem při pootočení kolečka myši.

Pomocí nástroje *Ručička* je možné plátno libovolně posouvat po okně. Pokud je tento nástroj aktivní dojde k posunutí plátna prostým uchycením a tažením na novou pozici.

6.3 Simulační část

Pro přechod do simulační části slouží tlačítko **Build**. Toto tlačítko se stane aktivním jakmile bude definován dostatečný počet kontrolních bodů potřebných pro rasterizaci aktuálně zvoleného objektu. V simulační části je nástroj *Ručička* aktivní a nelze ho deaktivovat.



Obrázek 6.2: Simulační část, část již vykreslené úsečky, která se rasterizuje a okno výsledků.

Pomocí panelu nástrojů *Playback* (viz kapitola 6.1.2) je možné spustit, resp. pozastavit nebo úplně zastavit animaci simulace. Dále je možné simulaci krokovat vpřed a zpět nebo se přesouvat o libovolný počet kroků pomocí posuvníku. Pro pohodlné přehrávání simulace je možné využít gest myši. Tato gesta jsou čtyři a jsou prováděna pohybem myši nad plátnem se stisknutým pravým tlačítkem:

- Pohyb myši doleva - krok zpět,

- pohyb myši doprava - krok vpřed,
- pohyb myši nahoru - zvýší rychlost simulace jednoho kroku,
- pohyb myši dolu - sníží rychlost simulace jednoho kroku.

Rasterizace je znázorněna opět do kreslícího plátna. Výsledný objekt je vykreslován barvou popředí, aktuálně vypočítaný segment je potom zvýrazněn blikáním. Při simulační části je rovněž k dispozici okno výsledků, ve kterém jsou zobrazeny všechny potřebné výpočty pro právě vykreslovaný segment.

Kapitola 7

Závěr

Práce na této zprávě i na projektu samotném pro mě byla velmi přínosná z hlediska celkového pochopení jednotlivých rasterizačních metod pro úsečku a elipsu a hlavně pak algoritmů pro rasterizaci typů křivek, které bývají často složité. Přesto, že všechny popsané a implementované algoritmy byly navrženy před mnoha lety, většina z nich je stále aktivně využívána buď při rasterizaci samotné – jsou implementovány v jádrech počítačových karet, nebo v animačních a jiných grafických programech. To svědčí o tom, že se jedná o skutečně kvalitní algoritmy.

Neméně důležitým přínosem pak pro mě bylo zdokonalení se v objektově orientovaném programování a programovacím jazyce C++ a pochopitelně i v samotné nadstavbě *wxWidgets*, kterou jsem si během práce na projektu velmi oblíbil.

7.1 Možnosti rozšíření aplikace

Návrh aplikace byl proveden tak, aby bylo možné ve vývoji pokračovat i po jejím vydání. Hlavní možností rozšíření aplikace je samozřejmě přidávání dalších metod, resp. typů křivek, kterých může být pro každý objekt implementováno až 10. Aplikace je navržena tak, aby zásahů do kódu bylo co nejméně, přesto se však zásahům nelze úplně vyhnout.

Mezi další možné rozšíření lze zařadit například možnost exportu kreslícího plátna do obrázku nebo uložení kreslícího plátna do formátu XML a jeho opětovné načtení, stejně jako ukládání výsledků buď do HTML nebo XML. Tato funkcionalita byla při vývoji považována za méně důležitou a proto nebyla implementována.

Kromě ukládání a načítání souborů je dále možné implementovat nastavení programu, které by bylo opět vhodné ukládat do formátu XML a při opětovném spuštění aplikace tento soubor načíst a podle něho nastavit globální parametry aplikace. Poslední možností rozšíření je pak implementovat další režimy pro zobrazování výsledků (viz 4.4.1). Z časových důvodů byl implementován pouze tzv. *Rychlý režim*.

Literatura

[1] Bresenham, J. E.: Algorithm for Computer Control of Digital Plotter. *IBM System Journal*, ročník 4, 1965: s. 25–30.

[2] Bresenham, J. E.: A Linear Algorithm for Incremental Display of Circular Arcs. *Communications of the ACM*, ročník 20, č. 2, 1977: s. 100–106.

[3] Foley, J. D.; van Dam, A.; Feiner, S. K.; aj.: *Computer Graphics - Principles and Practice*. Reading, Massachusetts: Addison-Wesley, druhé vydání, 1995, ISBN 0-201-84840-6.

[4] Ho, W.: Midpoint Ellipse Algorithm. 1995-2007 [cit. 2009-05-08].
URL http://www.winnyefanho.net/

[5] Kršek, P.: *Základy počítačové grafiky*. FIT VUT v Brně, 2007/08 [cit. 2009-05-08].
URL http://www.fit.vutbr.cz/

[6] Smart, J.; Hock, K.: *Cross-Platform GUI Programming with wxWidgets*. Pearson Education, Inc., 2006, ISBN 0-13-147381-6.

[7] Smart, J.; Roebling, R.; Zeitlin, V.; aj.: wxWidgets 2.8.10: A portable C++ and Python GUI toolkit [online]. 2009-02-01 [cit. 2009-05-08].
URL http://docs.wxwidgets.org/stable/

[8] Williams, N.; Williams, A.; Kaczmarek, K.; aj.: Welcome to the wxAUI Project. 2006-2009 [cit. 2009-05-08].
URL http://www.kirix.com/labs/wxaui.html

[9] Žára, J.; Beneš, B.; Sochor, J.; aj.: *Moderní počítačová grafika*. Computer Press, 2004, ISBN 80-251-0454-0.