Editor pro mapové komponenty pro Investex Group, s.r.o.

Map editor for Investex Group, s.r.o.

Jozef Fekiač

Bakalářská práce
2007

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jozef FEKIAČ**
Studijní program:	**B 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Editor pro mapové komponenty pro Investex Group , s.r.o.**

Zásady pro vypracování:

Vypracujte editor pro mapové komponenty podle požadavků zadavatele (Investex Group , s.r.o.).
Pro tvorbu použijte J2EE, využijte knihovnu Standard Widget Toolkit (SWT).
Optimalizujte způsob uchování mapy a přenosu dat ze serveru.
Vezměte v úvahu velký objem dat u použitých map.
Mapa bude členěna do vrstev, které budou aktivní a schopné uchovávat i údaje, které nejsou kartografické.
Systém by měl být schopen pracovat s GPS údaji, včetně přesné kalibrace mapy.

Rozsah práce:

Rozsah příloh:

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

Seznam odborné literatury:

McNamara, J.: GPS for dummies. Wiley Publishing, 2004.
Pecinovský, R.: Myslíme objektově v jazyku Java 5.0. Grada.
Brůha, L.: Java-Hotová řešení. ComputerPress, 2003.
Kiszka, B.: 1001 tipů a triků pro programování v jazyce Java. ComputerPress, 2003.

Vedoucí bakalářské práce: **Ing. Martin Sysel, Ph.D.**
Ústav aplikované informatiky

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

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

Ve Zlíně dne 13. února 2007

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

prof. Ing. Vladimír Vašek, CSc.
děkan

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

ABSTRAKT

Editor máp je software vyrobený na požiadavku spoločnosti Investex Group s.r.o. napísaný v programovacom jazyku Java s využitím knižníc SWT. Jeho úlohou je umožniť kreslenie vektorových máp podľa bitmapových predlôh, ich kalibráciu, zobrazovanie zemepisných polôh na mape a priraďovanie rôznych druhov dát jednotlivým objektom. Táto práca popisuje technológie využité vo vývoji aplikácie, jej vnútornú štruktúru a končí užívateľským manuálom aktuálnej verzie.

Kľúčové slová: mapa, editor, GIS, kartografia, kalibrácia, Java, SWT

ABSTRACT

The Map editor is software made on demand of the Investex Group s.r.o. It is written in Java programming language and it uses the SWT libraries. Its purpose is to supply an ability of drawing vector based maps according to bitmap sources, their calibration, projection of geographic positions into the map and assigning of various types of data to individual objects. This thesis describes technologies used in the application development, its inner structure and ends with the user manual of current version.

Keywords: map, editor, GIS, cartography, mapping, calibration, Java, SWT

Rád by som poďakoval vedúcemu mojej bakalárskej práce pánovi Ing. Martinovi Syslovi Ph.D. za odborné vedenie, za pripomienky a čas venovaný mojej práci. Ďalej spoločnosti Investex Group s.r.o. za poskytnutie zadania bakalárskej práce, okrem toho hlavne pánovi Ing. Stanislavovi Pivarčimu a Michalovi Hanákovi za konzultácie, odbornú pomoc a dohľad nad projektom. Vďaka patrí aj Marošovi Sedliakovi za vytrvalosť pri testovaní aplikácie.

Prehlasujem, že som na bakalárskej práci pracoval samostatne a použitú literatúru som citoval. V prípade publikácie výsledkov, ak je to uvolnené na základe licenčnej zmluvy, budem uvedený ako spoluautor.

V Zlíne …….

Podpis diplomanta

OBSAH

ÚVOD

V logistike a dispečingu železničnej dopravy dispečeri vykonávajú mnoho automatizovateľných činností, po ktorých zautomatizovaní je možné niekoľkonásobne zvýšiť produktivitu. Preto spoločnosť Investex Group s.r.o. (*Investex*) vyvíja softvér Server Dopravných Informácií (*SDI*). Jeho súčasťou je aj spolupráca s mapou – zobrazovanie polohy, zobrazovanie trasy, atď. Preto je potrebné vyrobiť komponentu pre daný programovací jazyk schopnú pracovať s mapovými podkladmi. To zároveň vytvára nutnosť špecifikovať formát daných podkladov a vytvoriť aplikáciu na ich prípravu.

Existuje nespočetné množstvo aplikácií v kategórii GIS (Geografický Informačný Systém), bohužiaľ nevyhovujú požiadavkám. Väčšina týchto aplikácií nepracuje s požadovanými typmi dát, iné sú príliš komplexné na to, aby boli modifikované pre dané účely.

Preto bolo nutné vytvoriť aplikáciu ktorá by umožnila prekresľovanie vektorovývh máp z bitmapových predlôh a priraďovať dáta nakresleným objektom. Zadaný programovací jazyk bol Java s užívateľským rozhraním SWT z dôvodu vývoja celého projektu SDI v tomto jazyku. Časová náročnosť tvorby Editoru máp (*editor*) presahuje rámec bakalárskej práce, preto sa táto práca zaoberá poslednou stabilnou verziou programu ku dňu odovzdania.

I. TEORETICKÁ ČASŤ

1 KARTOGRAFIA

Kartografia (z gréckeho *chartis* – mapa, *graphein* – kresliť) je vedný odbor zaoberajúci sa znázorňovaním zemského povrchu, nebeských telies a objektov, javov na nich a ich vzájomných vzťahov. Mapy sa pôvodne tvorili pomocou pera a papiera, vynálezom a rozšírením počítačov došlo k revolúcii v kartografii. [13]

1.1 GIS

Geografický informačný systém (GIS) je počítačový nástroj pre mapovanie a analýzu vecí a javov reálneho sveta. GIS ukladá informácie o svete ako súbor tematických vrstiev, ktoré sa dajú prepojiť na základe svojej zemepisnej polohy.

Rozbor názvu GIS:

- **geo** znamená, že GIS pracuje s údajmi a informáciami vzťahujúcimi sa k Zemi, pre ktoré poznáme ich lokalizáciu v priestore
- **grafický** znamená, že GIS využíva prostriedky grafickej prezentácie dát a výsledkov analýz a grafickej komunikácie s užívateľom
- **informačný** znamená, že GIS vykonáva zber, ukladanie, analýzu a syntézu dát s cieľom získať nové informácie potrebné pre rozhodovanie, riadenie, plánovanie a modelovanie
- **systém** znamená, že GIS predstavuje integráciu technických a programových prostriedkov, dát, pracovných postupov, personálu, užívateľov apod., do jedného celku [7]

Špecializované geografické informačné systémy sa používajú približne od začiatku 90. rokov, keď niekoľko firiem uviedlo na trh software, ktorý spĺňa požiadavky na zber dát o území v grafickej podobe s možnosťou vstupného spracovania, udržiavania a ukladania grafických aj negrafických dát. [4]

2 GPS

Globálny polohový systém (GPS - Global Positioning System) je družicový systém pre určovanie polohy a času na zemskom povrchu a v priľahlom priestore. Je schopný poskytovať tieto údaje nezávisle na počasí 24 hodín denne. Oficiálny názov celého systému je *GPS NAVSTAR*. Ide o pasívny družicový dĺžkomerný systém. Cieľom prevádzkovateľa tohto systému, Ministerstva obrany USA, pôvodne bolo, aby vojenské jednotky mohli presne určovať polohu, rýchlosť a čas v jednotnom referenčnom systéme. Z uvedeného vyplýva, že systém bol vyvíjaný najmä pre vojenské účely, ale americký kongres neskôr schválil jeho využitie s určitými obmedzeniami aj pre civilný sektor. [12]

Dá sa povedať, že celý systém je založený na dvoch údajoch:

- *vzdialenosť* GPS prijímača od satelitu
- presná *poloha satelitu* na obežnej dráhe

Vzdialenosť prijímača od satelitu sa počíta podľa časového rozdielu medzi vyslaním signálu zo satelitu a jeho príjmom v GPS prijímači. Na určenie polohy je nutné, aby mal prijímač dostatočne silný signál aspoň z troch satelitov. Systém GPS dokáže určiť nie len polohu, ale aj nadmorskú výšku, to je však podmienené dostatočne silným signálom minimálne zo štyroch satelitov.

Ak sa prijímač ocitne na mieste nerušenom žiadnymi prekážkami (tzn. vysoké budovy, hory), môže sa pripojiť až na 12 satelitov, čo znamená určenie polohy s maximálnou odchýlkou ±5 metrov. Bohužiaľ, pri pohybe zariadenia v okolí spomínaných prekážok sa môže stať, že sa prijímač nedokáže pripojiť ani na jeden satelit, takže nedokáže vypočítať ani približnú polohu. Typicky nedostupnými oblasťami sú miesta pod vodnou hladinou, podzemné garáže, príliš husté lesné porasty a pre železničnú dopravu najproblematickejšie *tunely*. [5]

3 JAVA

Java je programovací jazyk pochádzajúci od firmy *Sun Microsystems*. Jedná sa o objektovo orientovaný jazyk vychádzajúci z C++, ku ktorému donedávna mala syntakticky najbližšie [5], kým spoločnosť *Microsoft* nevytvorila jazyk C#, ktorý Javu mnohými rysmi pripomína.

Oproti C# má však Java obrovskú výhodu – nezávislosť na platforme, pretože je prekladaná do špeciálneho medzikódu (*bytecode*), ktorý je na konkrétnom počítači alebo zariadení interpretovaný, príp. za behu prekladaný do *natívneho* kódu. Programátor môže teda program napísaný v *Jave* v operačnom systéme Windows spustiť na PC s Linuxom, na Macintoshi, na *PDA* – teda všade, kde je k dispozícii *Java runtime*.

3.1 SWT

Standard Widget Toolkit (**SWT**) je sada grafických ovládacích prvkov pre platformu Java, pôvodne vyvíjaná spoločnosťou *IBM* a momentálne podporovaná zoskupením *Eclipse Foundation* popri vývoji prostredia **Eclipse** IDE. SWT predstavuje alternatívu k vývojovým sadám grafických komponent jazyka Java *AWT* a *Swing*, ktoré sú spoločnosťou Sun Microsystems poskytované ako súčasť štandardu Java.

Knižnice SWT sú napísané v Jave. Na zobrazovanie prvkov grafického rozhrania, implementácia SWT využíva *natívne* knižnice ovládacích prvkov a teda zaručuje prirodzené správanie jednotlivých prvkov pre daný operačný systém. To zároveň objasňuje, prečo sú jednotlivé implementácie samotnej SWT rozdielne pre každý operačný systém a napriek tomu si program zachováva prenositeľnosť. [14]

3.2 JFace

JFace je knižnica obsahujúca triedy, ktoré zabezpečujú väčšinu úloh pri programovaní *grafického rozhrania*. Je postavená na SWT a navrhnutá na prácu s touto knižnicou bez jej skrývania. Okrem štandardných možností na prácu s obrázkami, textom, dialógmi, atď. ponúka dve dôležité výhody: *akcie* (actions - možnosť oddelenia príkazu od jeho grafickej reprezentácie v menu) a *prehliadače* (viewers – oddelenie dát od grafickej reprezentácie v ovládacom prvku). [2]

4 OBJEKTOVÉ PROGRAMOVANIE

Objektové programovanie (OOP) je typ programovania, pri ktorom programátori nedefinujú iba dátové typy a dátové štruktúry ale aj typy operácií (funkcie) použiteľné na danú štruktúru. Takto sa z dátovej štruktúry stáva *objekt*, ktorý obsahuje dáta aj funkcie. Okrem toho môže programátor vytvárať vzťahy medzi rôznymi objektmi (napr. jeden objekt môže zdediť vlastnosti iného). [11]

Objekty v zmysle OOP spĺňajú niekoľko výhodne využiteľných kritérií (zapúzdrenie, polymorfizmus a dedičnosť), ktorých popis však presahuje rámec tejto práce. Ďalšou výhodou OOP je rozdelenie programu do tried a zoskupenie tried do balíčkov (package), čím sa sprehľadní zdrojový kód a uľahčí sa orientácia v ňom.

4.1 Návrhové vzory

V softwarovom inžinierstve znamená návrhový vzor (**design pattern**) všeobecné opakovane použiteľné riešenie bežne sa vyskytujúceho problému pri návrhu aplikácií. Návrhový vzor nie je konečný postup, ktorý je možné priamo prepísať do zdrojového kódu. Je to popis (resp. šablóna) riešenia problému, ktoré môže byť použité v rôznych situáciách rôznym spôsobom. [10]

4.1.1 Proxy pattern

Návrhový vzor **proxy** je vo svojej najvšeobecnejšej forme trieda, ktorá funguje ako rozhranie k inej činnosti. Inou činnosťou môže byť čokoľvek – pripojenie k sieti, veľký objekt v pamäti, súbor, atď.

Najdôležitejšie typy tohto návrhového vzoru využité pri vývoji tejto aplikácie sú:

- **virtual proxy** (virtuálne proxy) – umožňuje vytváranie objektov až v čase keď sú potrebné (tzv. *lazy* - lenivá inicializácia)
- **cache proxy** (skladovací proxy) – poskytuje dočasné úložisko výsledkov náročnej operácie, ktoré môžu využívať rôzni klienti (resp. triedy)
- **smart reference proxy** (inteligentný ukazovateľ) – poskytuje dodatočné akcie vždy keď sa kód odkazuje na cieľový objekt (napr. počítanie referencií na objekt)

4.1.2 Bridge pattern

Návrhový vzor **bridge** (most) je postup používaný v softwarovom inžinierstve na „*oddelenie abstrakcie od jej implementácie tak, aby sa mohli nezávisle meniť*“ [3]. To znamená zakrytie samotnej implementácie rozhraním, čo umožní zmenu implementácie bez nutnosti meniť spôsob prístupu k nej.

Podobným vzorom je **state** (stav) pattern. Tento vzor má definovanú množinu implementácii, ktoré sa zamieňajú v závislosti na stave objektu a tým menia jeho funkčnosť. Samozrejme, všetky implementácie majú spoločný abstraktný základ.

4.1.3 Factory pattern

Vzor **factory** (továreň) je objektovo orientovaný návrhový vzor, ktorý sa stará o vytváranie objektov bez špecifikovania konkrétnej triedy daných objektov. Vo všeobecnosti sa pojem factory používa pre každý mechanizmus, ktorý nepriamo vytvára inštancie objektov. V užšom pohľade ide napr. o metódu, ktorá na základe textového parametru (napr. časť názvu triedy) vytvorí inštanciu objektu z určeného balíčku. Ďalším príkladom môže byť vytvorenie rôzneho typu objektu na správu obrázku podľa veľkosti daného obrázku (ktorá je predaná ako parameter *factory metóde*).

II. PRAKTICKÁ ČASŤ

5 VNÚTORNÝ POPIS PROGRAMU

Vďak tomu, že je program napísaný v jazyku Java, je logicky rozčlenený do viacerých balíčkov (*package*). Rôzne balíčky riešia jednotlivé podproblémy s minimálnymi závislosťami na balíčkoch vyšších úrovní. Detaily implementácie jednotlivých tried (*javadoc*) sa nachádzajú v prílohe P2. Tento popis má slúžiť ako vodítko pre programátora akýmkoľvek spôsobom zasahujúceho do zdrojového kódu programu.

Spúšťacia funkcia *main* sa nachádza v triede `MapEditor`. Táto funkcia vytvorí hlavné okno a prenechá riadenie programu jemu.

5.1 Užívateľské rozhranie

Package: **`mapeditor.gui`**

Tento balík obsahuje jedinú triedu `MainWindow`, ktorá reprezentuje hlavné okno programu, odvodené od triedy `ApplicationWindow` nástrojovej sady JFace. Táto trieda sa stará o inicializáciu menu, panelu nástrojov (*toolbar*) a obrazového výstupu (*renderer*). Ďalej priradí jednotlivým položkám menu a toolbaru akcie, ktoré obsahujú kód vyvolaný po kliknutí na danú položku a zároveň kód nastavujúci titulok a ikonu tejto položky.

5.1.1 Akcie

Package: **`mapeditor.gui.actions`**

Hlavnou triedou tohto balíka je `ActionManager`. Správa sa podľa návrhového vzoru *factory*, pretože jej funkcia `getAction(String actionName)` na požiadanie vytvára inštancie žiadaných akcií z vnoreného balíku `custom`. Má tzv. *lazy* (lenivé) správanie, čiže žiadané objekty vytvára až keď je to potrebné. Ak je teda ako parameter `actionName` použitý reťazec „*Exit*“, táto funkcia vráti inštanciu žiadanej triedy `mapeditor.gui.actions.custom.ExitAction`. Nikdy sa nevytvára viac ako jedna inštancia, čiže pri viacnásobnom volaní tejto funkcie je vždy vrátená prvýkrát vytvorený objekt. Pokiaľ neexistuje požadovaná trieda, je vytvorená a vrátená inštancia triedy `DefaultAction`.

5.1.2 Komponenty

Package: **`mapeditor.gui.components`**

Balík `components` rozširuje toolkity SWT a JFace o ovládacie prvky, ktoré neobsahujú.

Trieda ***`DropDownToolItem`*** rozširuje položku toolbaru (JFace), takže po kliknutí na ňu sa rozbalí menu, ktoré je možné vytvoriť dynamicky po vytvorení tejto položky. Táto trieda v projekte momentálne nie je použitá.

Trieda ***`ImageCheckButton`*** je vlastný ovládací prvok, slúžiaci ako prepínač *zapnutý* / *vypnutý*. Vyžaduje nastavenie obrázku pre obidva stavy. Trieda je využitá v okne *Vrstvy* na prepínanie viditeľnosti a zároveň zobrazovanie ikony vrstvy.

Trieda ***`PickText`*** je jednoduchým spojením textového vstupu (*Text*) a tlačidla (*Button*), umožňujúca sledovať udalosti obidvoch týchto prvkov zároveň. Momentálne nie je využitá, je však určená na použitie pri výbere objektov z mapy napr. do plánu prepravy.

Trieda ***`SafeSaveDialog`*** je rozšírením štandardného dialógu na ukladanie súboru, naprogramovaná podľa návrhového vzoru *proxy* – obsahuje inštanciu FileDialog a ponúka všetky jej potrebné funkcie. Jediný rozdiel je, že táto trieda po výbere súboru kontroluje, či súbor existuje a ak áno, informuje o tom užívateľa a ponúkne mu možnosť výber zrušiť.

Trieda ***`ToolWindow`*** je základnou triedou od ktorej je odvodené okno vlastností objektov a okno vrstiev. Obsahuje vytvorenie nástrojového okna (tenký titulok, nezobrazuje sa na lište úloh atď.) a správne nastavenie posuvníkov pri zmene veľkosti okna.

5.2 Obrazový výstup

`Package:` **`mapeditor.renderer`**

Balíček `renderer` sa stará o grafický výstup celého programu. Priamo v ňom sa nachádzajú triedy ktoré ošetrujú vykresľovanie na najvyššej úrovni (poradie vrstiev, viditeľnosť vrstiev, štýly objektov) a riadia triedy vnoreného balíku graphics.

Trieda ***`EditorRenderer`*** je odvodená od rozhrania *`IRenderer`*, obsahuje preto funkcie na inicializáciu štýlov a pravidelne volané funkcie `beforeRender(...)`, `render(...)` a `afterRender(...)`. Funkcie `beforeRender` a `afterRender` nie sú určené na samotné kreslenie ale slúžia na doplnkové funkcie, momentálne napr. nastavenie správnej transformácie pre vykresľovanie a výpočet rýchlosti kreslenia. Oproti tomu funkcia `render` je počiatočným bodom prekreslenia obsahu okna. Spôsobuje vykreslenie všetkých

viditeľných vrstiev, vykreslenie pozadia pod práve zobrazovanou oblasťou a vykreslenie informácií na obrazovke.

Trieda ***Refresher*** je cyklicky bežiace vlákno (spustené pri otváraní hlavného okna), ktoré pri každom priebehu volá funkcie triedy *EditorRenderer* a tým spôsobuje opakujúce sa prekresľovanie obsahu okna. Iným možným riešením by bolo volať funkciu render po každej zmene obsahu obrazovky, čo je pri editore takmer neustála vec. Pri výrobe výslednej komponenty zobrazujúcej mapu bude táto trieda odstránená a nahradená prekresľovaním len v nutných okamžikoch.

5.2.1 Nízkoúrovňová grafika

Package: **mapeditor.renderer.graphics**

Balík graphics obsahuje iba rozhranie *IGraphicsCanvas2D*, ktoré je základom pre implementáciu „plátna“ na ktoré sa kreslí. Ďalšie triedy sú vo vnorených balíčkoch, pričom balíček swt obsahuje aktuálnu implementáciu rozhrania *IGraphicsCanvas2D*.

Celý balík graphics je budúcim cieľom optimalizácie, keďže navrhnuté triedy nemajú požadovanú efektivitu.

5.2.1.1 Štýly

Package: **mapeditor.renderer.graphics.style**

Balík style obsahuje výhradne dátové triedy (tzn. bez funkcií, len uchovávajú dáta) na identifikáciu štýlu kreslených objektov.

Trieda ***Color*** (*farba*) nesie informáciu o farbe využitú v nasledujúcich triedach.

Trieda ***Brush*** (*štetec*) nesie informáciu o farbe výplne kresleného plošného objektu.

Trieda ***Pen*** (*pero*) uchováva informácie o hrúbke farbe a štýle kreslených čiar alebo ohraničení plošných objektov.

Trieda ***Font*** (*písmo*) uchováva informácie o použitom písme, jeho farbe, veľkosti a štýle pri vykresľovaní textu.

5.2.1.2 Implementácia grafiky v SWT

Package: **mapeditor.renderer.graphics.swt**

Tento balík obsahuje aktuálnu implementáciu rozhrania *IGraphicsCanvas2D* z balíčku graphics a príslušné pomocné triedy. Táto implementácia využíva na kreslenie objektov možnosti poskytnuté toolkitom SWT. Samotné SWT používa na kreslenie funkcie poskytované operačným systémom v ktorom program beží, preto toto riešenie nie je najrýchlejšie, takže ho je tiež nutné v budúcnosti optimalizovať.

Trieda ***GraphicsCanvas2DSWT*** je horeuvedenou implementáciou, ktorá využíva ostatné triedy na pomocné účely.

Trieda ***HeapSorter*** je použitá na zoradenie objektov čakajúcich na vykreslenie podľa ich hĺbky (z-index). Na vykresľovanie sa používa tzv. maliarov algoritmus, takže sa kreslia všetky objekty postupne od najspodnejšieho (najmenší z-index) až po najvrchnejší (najväčší z-index).

Vnorený balík **renderobject** obsahuje rozhrania určené ako šablóny pre kreslené objekty a zároveň ich implementuje, čím vytvára škálu základných kresliteľných geometrických útvarov (napr. *ROLine* – úsečka, *ROCircle* – kruh) a niekoľkých zložených objektov pre zjednodušenie práce (napr. *ROCross* – kríž na označenie polohy, *ROSceneBackground* – obdĺžnik kreslený ako pozadie scény).

Balík **plugins** obsahuje rozhranie *ICanvasPlugin*, od ktorého je potrebné odvodiť rozšírenie plátna (tj. grafický element, ktorý sa bude na plátne zobrazovať bez závislosti na obsahu mapy a vykonávanej akcii napr. *pravítka*, *mierka* atď.). Rozšírenie plátna stačí pri štarte aplikácie zaregistrovať funkciou `IGraphicsCanvas2D.addPlugin(...)` a nastaviť viditeľnosť funkciou `ICanvasPlugin.setVisible(true)` a ďalej nie je potrebné sa o vykresľovanie tohto rozšírenia starať. Rozšírenia okrem toho môžu prijímať udalosti plátna, napr. pohyb ukazovateľa myši. Momentálne program obsahuje rozšírenia na zobrazenie GPS polohy v bode kde sa nachádza ukazovateľ (*GpsTracker*) a pravítka (*HorizontalRuler* a *VerticalRuler*).

5.3 Vrstvy

Package: **mapeditor.renderer.layers**

Tento balík umožňuje rozdelenie grafického výstupu do vrstiev a prácu s nimi.

Rozhranie ***ILayer*** musí implementovať každá vrstva. Deklaruje funkcie na identifikáciu samotnej vrstvy (ikony, popis vrstvy, identifikátor) a zároveň z rozhrania *ILayerActions*

dedí akcie, ktoré môžu byť vo vrstve vyvolané z iných častí programu (výber objektu alebo skupiny objektov, kreslenie atď.). Obsahuje okrem iného aj funkciu `render(...)` ktorá má na starosti vykreslenie všetkých objektov tejto vrstvy.

Trieda ***`LayerManager`*** je kombináciou návrhového vzoru *factory*, *proxy* a *bridge*. Umožňuje správu vrstiev, vytvára ich inštancie, a sprostredkúva prístup k funkciám aktuálnej pracovnej vrstvy. Z programu sa teda pristupuje len k inštancii triedy *`LayerManager`* bez nutnosti starať sa o aktívnu vrstvu. Všetky operácie špecifické pre vrstvy sú ošetrené ich vlastnými triedami. Okrem toho *`LayerManager`* umožňuje funkciou `addTempLayer(...)` pridať za behu programu dočasnú vrstvu (napr. zobrazenie trasy).

Trieda ***`LayerWindow`*** zapúzdruje okno vrstiev. Toto okno obsahuje len jeden ovládací prvok – inštanciu triedy *`LayerPage`*, ktorá vytvára zoznam všetkých vrstiev aj s príslušnými ikonami.

5.3.1 Aktuálne implementované vrstvy

`Package:` **`mapeditor.renderer.layers.custom`**

Každá vrstva implementuje špecifické úlohy a na ich realizáciu využíva k tomu určené balíky projektu. Kartografické vrstvy (*Hranice*, *Železnice*, *Stanice*, *Mestá*, *Kontrolné body*) používajú na správu vlastných dát ďalej popísaný *`DocumentManager`*.

5.3.1.1 Pozadie

Trieda ***`BackgroundLayer`*** implementuje vrstvu určenú na zobrazovanie a prácu s podkladovými bitmapami. Využíva vnorený balík `images` na správu obrázkov a v ňom triedu *`ImageObject`* a *`OptimizedImage`* na optimalizovanú reprezentáciu obrázkov. Obrázok je pri načítaní niekoľkokrát zmenšený, pri každej veľkosti je rozdelený na menšie štvorce a tie sú uložené v dočasnom adresári. Počet zmenšení a veľkosť menších štvorcov sa určujú v triede *`OptimizedImage`*.

5.3.1.2 Kalibrácia a pomocné čiary

Trieda ***`CalibrationLayer`*** umožňuje pracovať s *kalibráciou* mapy v samostatnej vrstve (kreslenie a editácia kalibračných bodov). Používa na to balík `calibration`.

Trieda ***GuideLineLayer*** umožňuje prácu s horizontálnymi a vertikálnymi pomocnými čiarami. Používa triedy vnoreného balíku `guidelines`.

5.3.1.3 Hranice

Trieda ***StateBorderLayer*** sprostredkúva prácu s lomenými čiarami reprezentujúcimi hranice. Implementuje funkcie na pridávanie, mazanie a posúvanie bodov, na dopĺňanie existujúcej krivky (lomenej čiary), rozdelenie segmentu hranice a uzavretie krivky.

5.3.1.4 Železnice

Trieda ***RailLayer*** obsahuje všetky funkcie vrstvy hraníc (okrem uzavretia krivky) a navyše pridáva možnosť *vetvenia*, tj. vytvárania uzlových bodov a nadväzovania ďalšími železnicami.

5.3.1.5 Mestá, stanice a kontrolné body

Vrstvy ***CityLayer***, ***StationLayer*** a ***ControlPointLayer*** pracujú s bodovými objektmi, ktoré sa líšia jedine grafickou reprezentáciou a niektorými vlastnosťami (*properties*). Nič z toho nesúvisí priamo s funkciami vrstvy, preto je tento rozdiel zanedbateľný. Sprostredkúvajú len vytváranie objektov a ich jednoduchú editáciu.

5.3.2 Výber

`Package:` **`mapeditor.renderer.layers.selection`**

Tento balík obsahuje triedy pracujúce s výberom (*selekciou*) objektov. Okrem nich obsahuje niekoľko rozhraní, ktoré musia implementovať objekty využívajúce možnosti výberu.

Trieda ***SelectableGroup*** uchováva vektor *označených* objektov implementujúcich rozhranie *`ISelectable`*.

Trieda ***Selection*** rozširuje triedu *`SelectableGroup`* o možnosť posúvania objektov (ktoré implementujú rozhranie *`IMovable`*), renderovanie posúvaných objektov a možnosť vnoreného výberu (*`SubSelection`*).

Trieda ***SubSelection*** je podobná triede *`Selection`* s tým, že neobsahuje vnorený výber. Táto trieda je využitá na uchovávanie a manipuláciu výberu vnorených objektov na mape, napr. vybraných bodov v rámci jednej vybranej železničnej trate.

5.3.3 Prichytávanie

Package: **mapeditor.renderer.layers.snapping**

Tento balík umožňuje prichytávanie objektov k iným objektom, napríklad pomocným čiaram. Každý objekt, ku ktorému môže byť nejaký bod prichytený (*snapper*) musí implementovať rozhranie *ISnapper* a počas behu programu musí byť zaregistrovaný funkciou `SnapManager.addSnapper(...)`.

Trieda ***SnapManager*** slúži na správu všetkých *snapperov* a vyhľadávanie prípadného cieľa prichytenia. Používa sa k tomu funkcia `snap(...)`, ktorá zadaný bod prichytí k najbližšiemu *snapperu* ak je nejaký v dosahu, preto je návratová hodnota typu *SnappedPoint* – obsahuje okrem novej (príp. nezmenenej) polohy bodu aj informáciu o počte prichytení, ktorý sa rovná nule v prípade že daný bod nie je možné z aktuálnej polohy prichytiť.

5.3.4 Rozdeľovanie kriviek

Package: **mapeditor.renderer.layers.splitline**

Tento balík je využiteľný len pre vektorové vrstvy. Obsahuje akciu pridávanú do kontextového menu týchto vrstiev (*SplitLineAction*), ktorej funkciou je rozdelenie vybranej úsečky vo vyznačenom bode a pridanie tohto bodu do krivky.

Trieda ***SplittingPoint*** reprezentuje bod, v ktorom sa má daná úsečka rozdeliť. Okrem polohy nesie aj informáciu o krivke, ktorej má tento bod patriť. Inštanciu triedy *SplittingPoint* vytvára priamo vektorová vrstva, ktorá ho predáva konštruktoru triedy *SplitLineAction*.

5.4 Zdroje

Package: **mapeditor.resources**

Tento balík obsahuje triedu **ResourceManager** (správca zdrojov) a vo vnorených balíkoch samotné zdroje (ikony, kurzory). *ResourceManager* obsahuje funkcie na získanie ikôn, ktoré podľa typu ikony (napr. ikona menu, kurzor, atď.) načítajú obrázok z potrebného adresára a vrátia referenciu naň. V podadresároch sa nachádzajú samotné kurzory, ikony vrstiev a ikony menu vo formáte *PNG*.

5.5 Stavy editora

Package: **mapeditor.states**

Balík `states` pracuje so stavmi editora. *Stav* predstavuje určité chovanie editora, resp. nástroj (kreslenie, úpravy, posun atď.).

`StateManager` je hlavnou triedou tohto balíku. Implementuje návrhový vzor *state*, čiže podľa potreby sa zamení inštancia stavu za inštanciu iného stavu a tým sa zmení chovanie programu. *`StateManager`* implementuje tak ako každý stav rozhranie *`IEditorState`* a po zavolaní príslušnej metódy predáva riadenie práve nastavenému stavu.

5.5.1 Aktuálne implementované stavy

Package: **mapeditor.states.custom**

Názov triedy implementujúcej stav editoru musí z formálnych dôvodov končiť slovom „*`State`*“, trieda musí byť umiestnená v tomto balíku a pridaná v konštruktore triedy *`StateManager`* funkciou `addState(nazovStavu)`, kde `nazovStavu` je názov triedy bez slova „*`State`*“. Všetky stavy sú odvodené od triedy *`DefaultEditorState`*, ktorá pre pohodlie poskytuje základné možnosti práce so stavmi, napr. prepočet polohy myši z jednotiek obrazovky na jednotky mapy, prácu s kolieskom myši (priblíženie, posuv) atď.

5.5.1.1 Výber objektu

Trieda ***`SelectState`*** predstavuje stav editora, v ktorom je možné vyberať objekty v danej vrstve. Zároveň má na starosti ošetrenie stlačenej klávesy *Shift* pri označovaní objektu a tým pádom pridanie alebo odobratie z označenej skupiny.

5.5.1.2 Výber bodu

Trieda ***`PointSelectState`*** poskytuje funkcie podobné ako *`SelectState`*, pracuje však s bodmi objektu vybraného v stave *`SelectState`*. V prípade že vybraný objekt neobsahuje žiadne podobjekty (body) alebo žiadny objekt nie je vybraný, tento stav nevykonáva žiadnu funkciu.

5.5.1.3 Ruka

Trieda ***`HandState`*** poskytuje možnosť posúvať pohľad ťahaním myši. Keďže *`DefaultEditorState`* toto umožňuje v akomkoľvek stave ťahaním stlačeného kolieska myši (príp. stredného tlačidla), má tento stav v podstate formálnu úlohu.

5.5.1.4 Lupa

Trieda ***`MagnifyState`*** umožňuje približovanie pohľadu klikaním ľavým tlačidlom myši a odďaľovanie pravým tlačidlom. Momentálne má aj táto trieda formálny význam, v konečnej komponente však bude nutné priblíženie priamo obdĺžnika vybraného ťahaním myši (tzn. zväčšenie vybraného obdĺžnika na celú plochu plátna).

5.5.1.5 Pero

Trieda ***`PenState`*** umožňuje kreslenie objektov do aktívnej vrstvy. Tento stav priamo objekty resp. body objektov nevytvára, volá však funkciu `add(...)` pracovnej vrstvy, ktorá sa stará o vytvorenie potrebného objektu podľa okolností.

5.6 Geometria

`Package:` **`mapeditor.geometry`**

Balík `geometry` obsahuje pomocné geometrické triedy, ktoré reprezentujú jednotlivé objekty (úsečka, kružnica) a slúžia na výpočty týkajúce sa práce s nimi (priesečníky, testy prienikov atď.). Okrem iného obsahuje aj základné rozhrania, od ktorých musia byť tieto objekty odvodené. Každý geometrický objekt musí implementovať rozhranie *`I2DObject`* a niektoré z nasledujúcich rozhraní (aj kombináciu):

- ***`IArea`*** – plošný objekt (zaberajúci určitú plochu),
- ***`ILine`*** – čiarový objekt (existuje priemet na tento objekt),
- ***`IPoly`*** – mnohouholníkový objekt (má konečné množstvo vrcholov > 2).

Trieda ***`Transform`*** predstavuje neúplnú afinnú transformáciu postačujúcu na účely zobrazenia. Obsahuje iba koeficient zmeny veľkosti (symetrická pre horizontálnu aj zvislú os) a posun v horizontálnom a vertikálnom smere. Je použitá na vyjadrenie vzťahu medzi jednotkami v ktorých je mapa kreslená a obrazovkovými bodmi (samozrejme aj opačne).

Trieda ***Algebra*** obsahuje momentálne jedinú využitú funkciu `determinant2x2(...)`, použitú na výpočet determinantu pri zisťovaní vzájomnej rotácie 2 vektorov.

5.7 Kalibrácia

`Package:` **`mapeditor.calibration`**

Balík `calibration` má za úlohu poskytnúť kalibráciu mapy, resp. správne vyladenie prepočtu medzi súradnicami GPS a jednotkami, v ktorých je kreslená mapa. Takže je zodpovedná za správne zistenie geografickej polohy bodu na mape a zároveň o zobrazenie zemepisnej polohy (získanej napr. z prijímača GPS) na mapu. *Kalibračné body* majú určenú polohu na mape a príslušnú GPS súradnicu. Tieto body určujú *triangluláciu*, ktorá vytvorí sieť trojuholníkov tak aby boli využité všetky kalibračné body, čiže je pokrytý celý konvexný obal týchto bodov. Mimo neho je oblasť nekalibrovaná, čiže nie je možné horeuvedené výpočty uskutočniť. Je to isté obmedzenie, pre momentálne potreby mapy je to však postačujúce. Pri určení geografickej polohy bodu sa teda najprv určí trojuholník, do ktorého bod patrí a podľa polohy v tomto trojuholníku sa interpoluje výsledná poloha z polôh vrcholov trojuholníka.

Trieda ***GpsPoint*** reprezentuje geografickú polohu. Členská premenná N predstavuje stupne severnej zemepisnej šírky a premenná E predstavuje stupne východnej dĺžky.

Trieda **`CalibrationPoint`** priraďuje pozícii na mape zemepisnú polohu. Zároveň je aj grafickou reprezentáciou tohto bodu, využitou vo vrstve *Kalibrácia*.

Trieda **`CalibrationManager`** slúži na komunikáciu medzi programom a triangluláciou. Umožňuje pridávanie kalibračných bodov aj samotné prepočty súradníc za využitia ostatných tried.

Trieda **`Interpolation`** obsahuje metódy na výpočet interpolácie medzi hodnotami troch bodov v rovine.

5.7.1 Triangulácia

`Package:` **`mapeditor.calibration.triangulation`**

Trieda ***Delaunay*** implementuje rozhranie *`ITriangulation`* ako inkrementálnu *Delaunayovu triangluláciu* a umožňuje prácu s bodmi v nej.

Trieda ***Triangle*** reprezentuje trojuholník danej triangulácie a trieda *`Edge`* jeho hranu.

Trieda ***DoubleTriangle*** poskytuje funkciu na kontrolu susedných uhlov dvoch susedných trojuholníkov a funkciu na prehodenie uhlopriečky v štvoruholníku vytvorenom týmito trojuholníkmi podľa pravidiel *Delaunayovej triangulácie* (súčet uhlov, ktoré pretína uhlopriečka štvoruholníka, musí byť väčší ako súčet protiľahlých uhlov).

5.8 Kartografické dáta

Package: **mapeditor.mapdata**

Tento balík a jeho vnorené balíky poskytujú dátové typy (triedy) nesúce vektorové kartografické dáta a zároveň poskytujú triedy na ich uchovávanie, spracovanie, ukladanie a načítanie zo súboru.

Priamo balík `mapdata` obsahuje rozhrania, od ktorých musia byť geografické objekty odvodené a triedu ***ObjectData***, ktorá uchováva základné vlastnosti objektov a umožňuje ich jednoduché ukladanie a načítanie.

Rozhranie ***IMObject*** musí byť implementované každým kartografickým objektom.

IPointObject je rozhranie odvodené od *IMObject*, ktoré musí implementovať každý bodový objekt (mesto, stanica, atď.).

Rozhranie ***IMultiPointObject*** je tiež odvodené od *IMObject* a je určené pre objekty skladajúce sa z lomených čiar (teda obsahujúce viac bodov).

Rozhrania ***IPoint*** a ***INodePoint*** sú základom bodov, z ktorých sa vektorové objekty skladajú. Špeciálne *INodePoint* je základom uzlového bodu, v ktorom sa k danej krivke pripája iná.

5.8.1 Typy objektov

Package: **mapeditor.mapdata.objects**

Tento balík obsahuje momentálne implementované objekty, ktoré môže mapa obsahovať.

5.8.1.1 Mesto, Stanica a Kontrolný bod

Triedy ***City***, ***Station*** a ***ControlPoint*** implementujú rozhranie *IPointObject*. Líšia sa len niekoľkými vlastnosťami a grafickou reprezentáciou. Sú to najjednoduchšie objekty, ktoré sú definované len ich polohou.

5.8.1.2 Hranica a jej body

Trieda ***StateBorder*** reprezentuje hranicu štátu a používa inštancie triedy ***StateBorderPoint*** ako body, ktoré tvoria danú lomenú čiaru. Body sú spolu zviazané ako obojsmerne prepojený lineárny zoznam.

5.8.1.3 Železnica, jej body a uzly

Trieda ***RailRoute*** predstavuje časť železničnej trate. Jednotlivé body tejto trate sú reprezentované inštanciami triedy ***RailRoutePoint*** a sú rovnako ako v prípade hraníc zviazané obojsmerným lineárnym zoznamom. V prípade, že je niektorý bod trate bodom uzlovým, je inštancia *RailRoutePoint* v lineárnom zozname nahradená inštanciou triedy ***RailRouteNode***, ktorá obsahuje odkazy na nasledujúci a predchádzajúci bod v rámci ľubovoľného počtu tratí.

5.8.2 Vlastnosti objektov

Package: **mapeditor.mapdata.properties**

Balík properties obsahuje triedy zastupujúce vlastnosti objektov, triedy zobrazujúce ich v okne vlastností, rozhrania ktoré musia tieto vlastnosti implementovať a rozhrania ktoré musia implementovať objekty poskytujúce zoznam vlastností.

Vlastnosť je trieda zapúzdrujúca primitívny dátový typ jazyka Java alebo zložený dátový typ. Je odvodená od rozhrania ***IProperty***, ktoré ju núti implementovať funkcie na ukladanie resp. načítanie vlastných dát zo súboru a funkcie na nastavenie hodnoty aktuálnej, minimálnej a maximálnej, ako aj funkciu na nastavenie a validáciu aktuálnej hodnoty z textového reťazca (zvyčajne využívanú príslušným editovacím poľom na testovanie a nastavenie hodnoty).

Momentálne implementované vlastnosti zapúzdrujúce primitívne dátové typy sú:

- ***BooleanProperty*** – logická hodnota *true* / *false*
- ***FloatProperty*** – reálne číslo s plávajúcou desatinnou čiarkou
- ***IntegerProperty*** – celé číslo
- ***LongProperty*** – veľké celé číslo
- ***StringProperty*** – textový reťazec

Špeciálne dátové typy sú:

- ***PointProperty*** – súradnice bodu v jednotkách mapy
- ***GPSCoordProperty*** – súradnice bodu v stupňoch určujúce polohu objektu

Objekt, ktorý má obsahovať vlastnosti a teda ich poskytovať iným triedam, musí implementovať rozhranie ***IPropertiesProvider***. To vynucuje poskytnutie vektoru vlastností a zároveň po zmene každej vlastnosti je automaticky volaná jeho funkcia `setProperty(...)`.

Na zobrazenie vektoru vlastností sa používa trieda ***PropertyPageWindow***. Tá vytvorí ovládací prvok *PropertyPage*, ktorý sa naplní editovacími poliami patriacimi k jednotlivým vlastnostiam.

5.8.2.1 Editovacie polia pre vlastnosti

Package: **mapeditor.mapdata.properties.controls**

Pre každú vlastnosť by mal byť vytvorený ovládací prvok slúžiaci na zmenu hodnoty danej vlastnosti. Pre primitívne typy vlastností sú použité štandardné ovládacie prvky vhodné na ich editáciu. Pre špeciálne typy sú použité zložené ovládacie prvky.

Pre vlastnosť typu *PointProperty* bol navrhnutý ovládací prvok ***PointText***, ktorý obsahuje 2 textové polia na editáciu súradníc na zvislej a vertikálnej osi. Okrem toho obsahuje tlačidlo na výpočet týchto súradníc podľa zadanej *GPS polohy* a aktuálnej *kalibrácie*. Na zadanie GPS polohy sa zobrazí okno s editačným poľom pre vlastnosť *GPSCoordProperty*.

Pre vlastnosť *GPSCoordProperty* bol navrhnutý ovládací prvok ***GPSText***, ktorý obsahuje 2 textové polia na zadanie severnej zemepisnej šírky a východnej zemepisnej dĺžky v decimálnom formáte. Ďalej obsahuje tlačidlo na zobrazenie okna, v ktorom je možné zadať polohu vo formáte stupne, minúty, sekundy. Hodnoty z tohto okna sú prepočítané do decimálneho formátu, preto môže dôjsť k nepresnostiam spôsobeným zaokrúhľovaním dátového typu *float*.

5.8.3 Správa dokumentu

Package: **mapeditor.mapdata.document**

Tento balík spolu s vnorenými balíkmi sprostredkúva správu načítania a ukladania celej mapy, tak ako uchovávanie a poskytovanie dát práve načítanej mapy.

Triedy ***Input*** a ***Output*** užitočne rozširujú vstupno-výstupné triedy jazyka Java *DataInputStream* a *DataOutputStream* podľa návrhového vzoru *proxy*. Rozširujú tak ich možnosti o zápis a čítanie dvojrozmerných súradníc a každú vstupno-výstupnú operáciu rozširujú o výpis ladiaceho textu.

Trieda ***DocumentManager*** poskytuje funkcie na vytvorenie nového dokumentu, načítanie dokumentu zo súboru na disku a jeho uloženie. Súbor sa ukladá vždy v aktuálnej verzii programu. Pri načítaní sa rozpozná verzia dokumentu a podľa toho *DocumentManager* vyberie príslušnú triedu ***Loader***. Napr. pri načítaní dokumentu verzie 0.2 sa použije trieda *Loader* z vnoreného balíku `format_0_2`. Trieda *DocumentManager* zároveň obsahuje *hlavnú scénu*, ktorá uchováva všetky objekty dokumentu rozdelené do vrstiev.

Trieda ***DocumentDescriptor*** je použitá na identifikáciu typu a verzie dokumentu. Číta resp. zapisuje do súboru vždy niekoľko prvých bytov.

5.8.4 Scéna

Package: **mapeditor.mapdata.scene**

Pod *scénou* sa rozumie určitý súbor objektov mapy. V každom okamžiku behu programu existujú spravidla 2 scény. Jedna scéna je *hlavná*, ktorá zahŕňa všetky objekty v dokumente. Druhá scéna je *aktuálna*, ktorá je podmnožinou hlavnej scény a zahŕňa práve vykresľované objekty. Je to dôležité kvôli neustálemu prekresľovaniu obrazovky, aby nebol prechádzaný zoznam všetkých objektov v každom cykle.

Trieda ***Scene*** je reprezentáciou scény a trieda ***SceneManager*** je správca aktuálnej scény, ktorý sa okrem scény stará aj o obnovovanie prichytávaných objektov pri zmene pohľadu (resp. pri zmene vykresľovaných objektov).

5.9 Nastavenie

`Package:` **`mapeditor.setting`**

Balík `setting` obsahuje momentálne jedinú využitú triedu ***`SettingManager`***, ktorá sa stará o ukladanie, načítanie a poskytovanie parametrov programu. Pracuje so súborom vo formáte *XML*, ktorého názov je predávaný v konštruktore.

Táto trieda je použitá na ukladanie dvoch typov informácií do dvoch súborov. Jedným je posledný stav programu (poloha okna, aktívna vrstva...) a druhým je nastavenie programu (dosah prichytávania...). Pre nastavenie programu je pripravované nastavovacie okno vo vnorenom balíku `preferences`.

5.10 Správa a spojenie jednotlivých častí

`Package:` **`mapeditor.management`**

Trieda ***`EditorManager`*** umožňuje prístup ku všetkým hlavným častiam programu. Obsahuje referencie na jednotlivé časti a poskytuje ich pomocou klasických „get“ funkcií. Triedy všetkých objektov, ktoré poskytuje boli popísané vyššie.

5.11 Ladenie

`Package:` **`mapeditor.debug`**

Balík `debug` je pomôcka pri ladení programu. Obsahuje triedu ***`Debug`***, ktorá poskytuje funkciu `log(String file, String text)`, ktorá podľa nastavenia prepínačov vypíše (resp. zapíše) ladiaci text:

- **`writeToFiles`** – zapíše text do súboru s názvom [`file`].*log*
- **`writeToScreen`** – vypíše text do konzoly
- **`writeToSummary`** – zapíše text do spoločného ladiaceho súboru *_complete.log*

To umožňuje jednoduché zrušenie výpisu ladiacich textov nastavením premennej `writeToScreen = false`, príp. aj ostatných prepínačov.

Cesta k pomocným súborom sa nastavuje v triede `ResourceManager`, implicitne je to adresár *c:\sdi\mapeditor*.

6 UŽÍVATEĽSKÝ MANUÁL

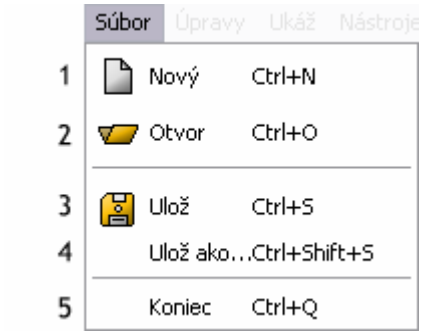
Táto časť je venovaná popisu a návodu na použitie vytvorenej aplikácie v momentálnom stave vývoja.

Ovládacími prvkami programu sú *hlavné menu*, *panel nástrojov* (ktorý je podmnožinou hlavného menu – nebude teda bližšie popísaný) a *kontextové menu* zobrazované v špecifických situáciách po kliknutí pravým tlačidlom myši na plochu mapy.

6.1 Hlavné menu

Hlavné menu obsahuje vnorené ponuky *Súbor*, *Úpravy*, *Ukáž* a *Nástroje*.

6.1.1 Ponuka Súbor



Obrázok 1 - Ponuka Súbor

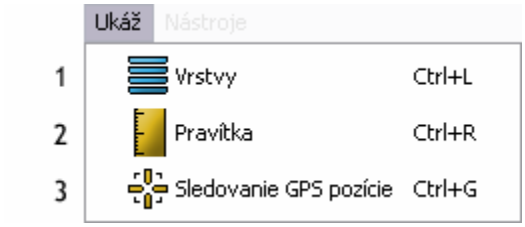
1. **Nový** – vytvorí nový prázdny dokument.
2. **Otvor** – otvorí existujúci dokument.
3. **Ulož** – zapíše práve upravovaný dokument na disk.
4. **Ulož ako** – zapíše práve upravovaný dokument na disk pod novým menom.
5. **Koniec** – ukončí aplikáciu.

6.1.2 Ponuka Úpravy

Obrázok 2 - Ponuka Úpravy

1. **Vymazať** – vymaže označené objekty resp. ich označené časti.

6.1.3 Ponuka Ukáž



Obrázok 3 - Ponuka Ukáž

1. **Vrstvy** – ukáže okno zobrazujúce vrstvy.
2. **Pravítka** – ukáže horizontálne a vertikálne pravítko.
3. **Sledovanie GPS pozície** – ukáže okno zobrazujúce GPS pozíciu bodu, na ktorom sa nachádza ukazovateľ myši (len v kalibrovanej oblasti).

6.1.4 Ponuka Nástroje

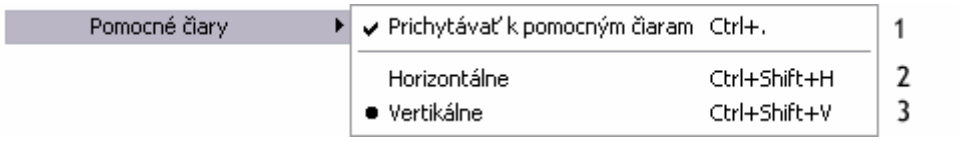
Obrázok 4 - Ponuka Nástroje

1. **Podkladová bitmapa**

Obrázok 5 - Ponuka Nástroje / Podkladová bitmapa

1.1. **Import podkladu** – otvorí okno na výber súboru a vybraný súbor vloží ako obrázok do vrstvy *Pozadie*.

2. **Pomocné čiary**



Obrázok 6 - Ponuka Nástroje / Pomocné čiary

2.1. **Prichytávať k pomocným čiaram** – *zapne* / *vypne* prichytávanie k nakresleným pomocným čiaram.

2.2. **Horizontálne** – aktivuje kreslenie vodorovných pomocných čiar.

2.3. **Vertikálne** – aktivuje kreslenie zvislých pomocných čiar.

3. **Výber objektu** – aktivuje nástroj Výber objektu.

4. **Výber bodu** – aktivuje nástroj Výber bodu.

5. **Lupa** – zmení nástroj na Lupu.

6. **Ruka** – prepne na nástroj Ruka.

7. **Pero** – aktivuje nástroj Pero.

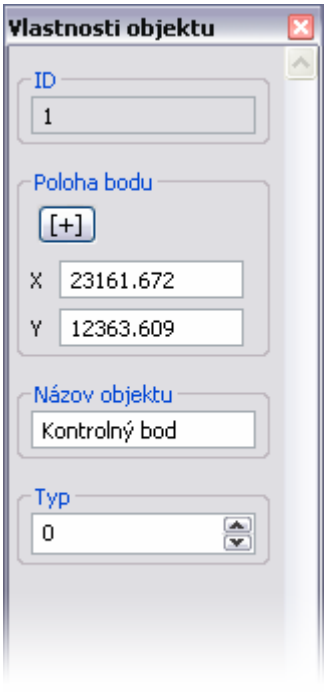
8. **Nastavenia** – táto ponuka je v aktuálnej verzii nefunkčná. V neskorších verziách bude otvárať okno s nastavením programu. K zmene nastavenia je v tejto dobe nutné ručne upraviť XML súbor *preferences.xml* umiestnený implicitne na ceste *c:\sdi\mapeditor*.

6.2 Módy programu (nástroje)

Mód (nástroj) programu predstavuje určitú funkciu (resp. súbor funkcií), umožňujúci pracovať s mapou, príp. s pohľadom na mapu.

6.2.1 Výber objektu

Nástroj Výber objektu slúži na označenie objektu kliknutím naň, resp. skupiny objektov nakreslením obdĺžnika zasahujúceho do všetkých týchto objektov. Vybraním jedného objektu sa automaticky zobrazí okno vlastností pre tento objekt.



Obrázok 7 – Príklad okna s vlastnosťami kontrolného bodu

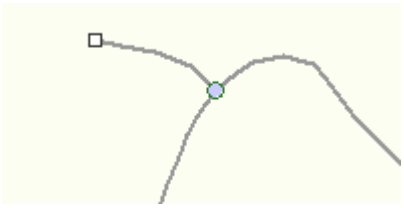
Po zatvorení okna vlastností je možné ho znovu otvoriť kliknutím pravým tlačidlom myši na označený objekt a vybraním položky Vlastnosti. Vstupné polia v okne vlastností sú intuitívne pomenované a ošetrené proti nesprávnemu vstupu, nebudú preto bližšie popisované.

6.2.2 Výber bodu

Nástroj výber bodu slúži na označenie bodu (resp. bodov) v rámci vybraného mnohobodového objektu. Prirodzene tento nástroj funguje len vo vrstvách Železnice a Hranice, ktoré tento typ objektov môžu obsahovať. Po označení jedného bodu a kliknutí pravým tlačidlom myši sa podobne ako pri výbere objektu zobrazí menu s položkou Vlastnosti, ktorá zobrazí okno vlastností daného bodu. Vo vrstve Železnice toto menu obsahuje ďalšie možnosti.

Obrázok 8 - Kontextové menu patriace k bodu železnice

1. **Povýš bod na uzol** – z obyčajného bodu vytvorí uzlový bod, ku ktorému je možné pripojiť iný úsek železničnej trate.



Obrázok 9 - Príklad uzlového bodu s nadväzujúcou železnicou

2. **Rozpojiť krivku** – rozdelí železnicu v danom bode. Obidve nové železnice majú koncové body na rovnakej pozícii ako bol vybraný bod, sú to však 2 rozdielne body.

Po kliknutí na segment označeného objektu mimo bodu zlomu sa na tomto mieste zobrazí prototyp nového bodu. Po kliknutím pravým tlačidlom myši sa zobrazí ponuka:

Obrázok 10 - Ponuka na vytvorenie nového bodu krivky

1. **Vytvor bod** – vytvorí nový bod krivky na mieste prototypu a vloží ho medzi susedné body.

6.2.3 Lupa

Nástroj Lupa slúži na približovanie resp. odďaľovanie pohľadu na mapu. Po kliknutí ľavým tlačidlom myši sa pohľad priblíži, po kliknutí pravým tlačidlom sa oddiali.

6.2.4 Ruka

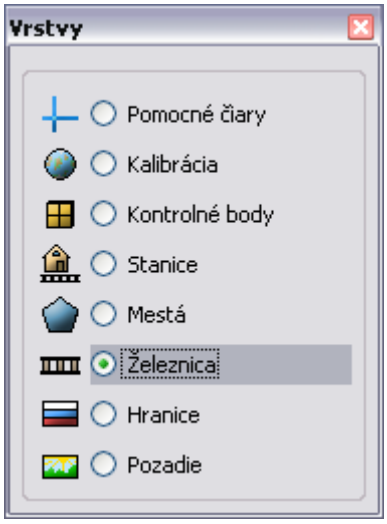
Nástroj Ruka slúži k posúvaniu pohľadu posúvaním myši so stlačeným ľavým tlačidlom.

6.2.5 Pero

Nástroj Pero slúži na kreslenie objektov do mapy podľa aktuálnej vrstvy.

6.3 Vrstvy

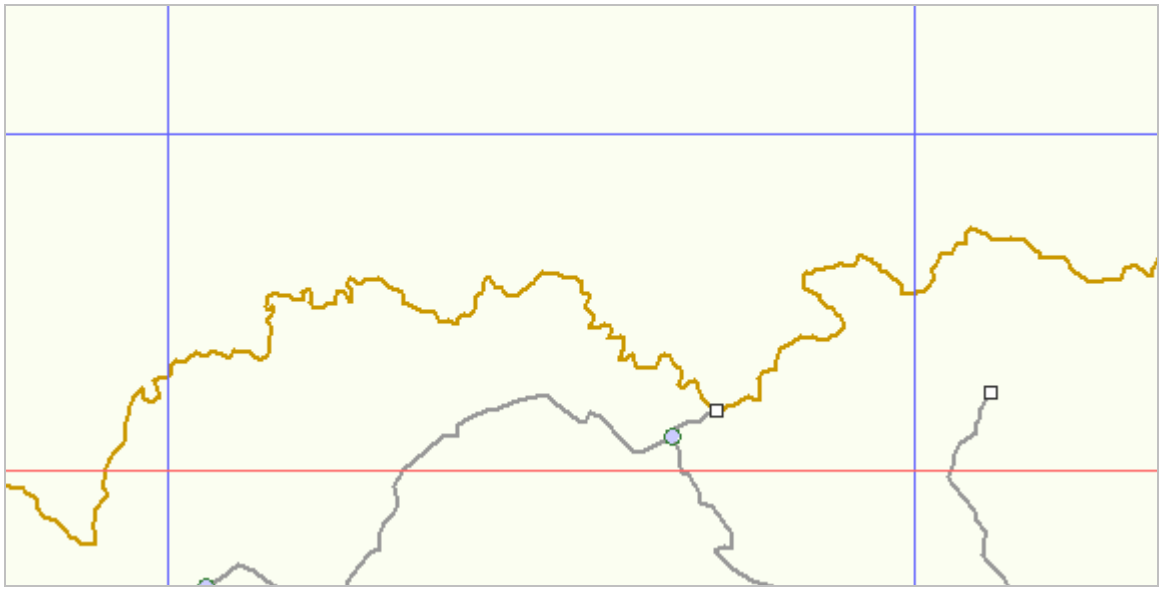
Jednotlivé vrstvy poskytujú rôzne funkcie a umožňujú kreslenie a editáciu rôznych druhov objektov. Prepínanie medzi vrstvami je možné v okne *Vrstvy*.



Obrázok 11 - Okno Vrstvy s aktívnou vrstvou Železnica

6.3.1 Pomocné čiary

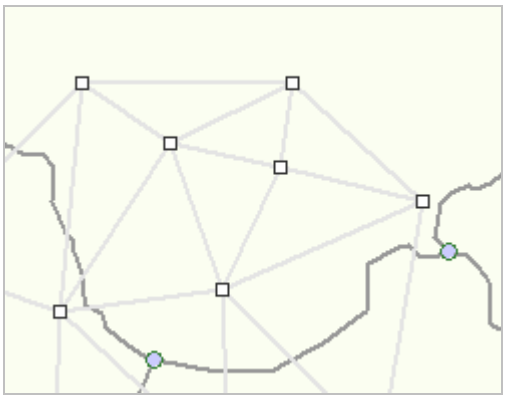
Táto vrstva slúži na kreslenie pomocných čiar. Orientácia pomocných čiar sa určuje v menu Nástroje / Pomocné čiary / *Horizontálne* (resp. *Vertikálne*).



Obrázok 12 - Výrez mapy s pomocnými čiarami

6.3.2 Kalibrácia

Vrstva *Kalibrácia* poskytuje možnosť editácie kalibračných bodov a tým celkovej kalibrácie mapy. Kalibračným bodom je nutné nastaviť príslušné *GPS súradnice*.



Obrázok 13 - Výrez mapy s kalibrovanou oblasťou

6.3.3 Kontrolné body

Táto vrstva slúži na editáciu kontrolných bodov na mape. Sú to dôležité miesta ako napr. *vstup do tunelu*, príp. iná dôležitá informácia.

Obrázok 14 - Ukážka kontrolného bodu

6.3.4 Stanice

Vrstva *Stanice* poskytuje možnosti editácie *železničných staníc* a železničných *hraničných prechodov*. Ich reprezentácia je rovnaká, líšia sa iba vlastnosťou *Typ*. Patričné hodnoty pre túto vlastnosť zatiaľ nie sú špecifikované, je však nutné rozlišovať stanicu, významnú stanicu a hraničný prechod.

Obrázok 15 - Ukážka železničnej stanice

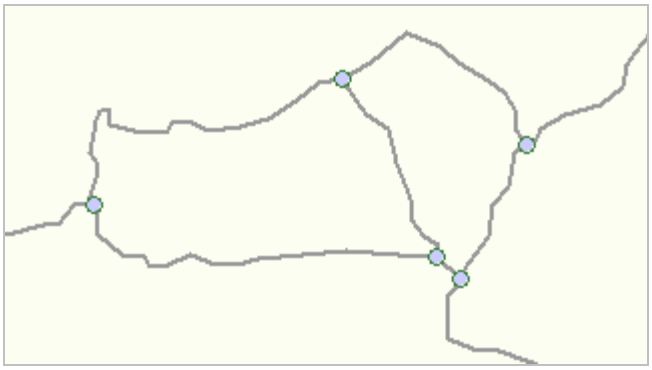
6.3.5 Mestá

Táto vrstva slúži na editáciu miest a obcí.

Obrázok 16 - Ukážka mesta

6.3.6 Železnica

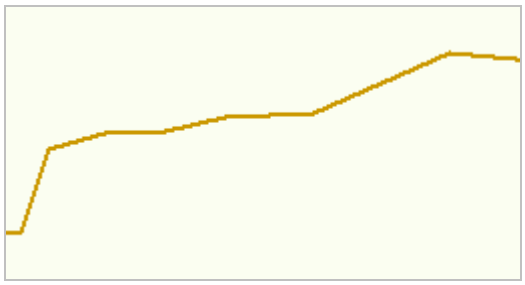
Vrstva *Železnica* slúži na kreslenie železničnej siete. Jednotlivé časti je najlepšie spájať a vetviť v koncových bodoch, aby sa predišlo vytvoreniu nekonečných slučiek (napr. sieť na obrázku Obrázok 17 je možné nakresliť takmer celú ako jeden úsek trate s viacerými uzlovými bodmi, je však výhodnejšie kresliť každú časť ako samostatný úsek).



Obrázok 17 - Ukážka železničnej siete

6.3.7 Hranice

Táto vrstva slúži na kreslenie štátnych hraníc.



Obrázok 18 - Ukážka hranice

6.3.8 Pozadie

Vrstva používaná na manipuláciu podkladových obrázkov. Okrem posunu obrázkov umožňuje nastavenie ich priehľadnosti, kvôli efektívnejšiemu prekresľovaniu.

Obrázok 19 - Ukážka obrázku použitého ako pozadie

7 BUDÚCNOSŤ

Program zatiaľ spĺňa výhradne kartografické požiadavky, je preto namieste spomenúť plán do budúcnosti, ktorý by mal viesť k úplnému dokončeniu aplikácie a naplneniu aj ostatných požiadaviek daných spoločnosťou Investex, tak ako aj ďalšie možnosti pohodlného rozšírenia programu v prípade zmeny požiadaviek.

7.1 Potrebné a požadované zlepšenia

Jedným z najväčších nedostatkov je momentálna absencia samostatnej komponenty, ktorá by mapu zobrazovala. Jej výroba však bude z väčšej časti upravovaním podmnožiny funkcií editora. Žiaľ, táto aplikácia zatiaľ neobsahuje niektoré funkcie potrebné pre výslednú komponentu, preto je nutné ich najskôr pridať do editora a potom z kompletnej aplikácie túto komponentu extrahovať.

Hlavnými chýbajúcimi funkciami sú rôzne optimalizácie. Najdôležitejšou je priestorové rozdelenie objektov na mape, aby sa pri zmene pohľadu na mapu a pri výbere objektov do scény neprechádzali všetky objekty dokumentu. Najvýhodnejším riešením tohto problému je tzv. *quadtree* (4-vetvový strom) a delenie objektov na podobjekty.

Ďalším nedostatkom je nízkoúrovňová grafika aplikácie. Na zobrazovanie sa používa štandardné *API* operačného systému, ktoré nepodporuje napr. orezávanie veľkých úsečiek a je pomerne pomalé pre zložitejšie vykresľovanie. Preto bude nutné prerobiť toto rozhranie napr. do *OpenGL* (pre jazyk Java je to knižnica *JOGL*).

Podstatnou časťou budúceho vývoja je možnosť priraďovania a spracovávania dát jednotlivým objektom mapy. Tento problém už prešiel hlbšou analýzou a je v budúcnosti prioritou danou firmou Investex, pretože po naprogramovaní týchto funkcií bude možné dokončiť fázu kreslenia mapy (hoci nepohodlne kvôli predchádzajúcim nedostatkom).

7.2 Ďalšie možnosti rozšírenia

Program bol koncipovaný tak, aby bola každá jeho časť rozšíriteľná o nové možnosti, v prípade nevyhovujúcich súčasných možností, prípadne pri zmene požiadaviek zadávateľa. Tieto rozšírenia je pri priemerných znalostiach jazyka Java možné urobiť intuitívne podľa práve implementovaných tried. Výhodne rozšíriteľné sú obzvlášť nasledujúce balíky.

7.2.1 Geometria

Package: **mapeditor.geometry**

V tomto balíku bude nutné doprogramovať geometrickú reprezentáciu nových tried, ktoré nebudú nahraditeľné žiadnym existujúcim geometrickým objektom.

7.2.2 Akcie menu

Package: **mapeditor.gui.actions.custom**

V tomto balíku je možné jednoducho pridať nové akcie menu odvodené od triedy *DefaultAction*. Potom ich stačí v inicializácii hlavného okna pridať do menu.

7.2.3 Vrstvy

Package: **mapeditor.layers.custom**

V tomto balíku sa nachádzajú jednotlivé vrstvy takže pri doprogramovaní novej stačí jej názov pridať do konštruktoru triedy *LayerManager*.

7.2.4 Kartografické objekty

Package: **mapeditor.mapdata.objects**

V prípade potreby nového typu objektov je nutné doplniť tento balík a vytvoriť inštanciu novej triedy pri príslušnej operácii buď existujúcej alebo novej vrstvy. Ďalej je nutné po vytvorení inštancie objekt pridať do dokumentu.

7.2.5 Vlastnosti objektov a ich editovacie polia

Package: **mapeditor.mapdata.properties**

Ak novovytvoreným typom objektov nepostačujú existujúce typy vlastností, je nutné doprogramovať novú odvodením od *IProperty* a vytvoriť pre ňu ovládací prvok v balíku `controls` a upraviť `PropertyPage.addControlForProperty(...)`.

7.2.6 Stavy editora

Package: **mapeditor.states.custom**

Pri absencii určitej funkcie je možné poskytnúť ju ako nový stav editora v tomto balíku. Stav je potrebné zaregistrovať v konštruktore triedy *StateManager*, ktorá zabezpečí pridanie do menu a panelu nástrojov.

ZÁVER

Podľa zadania spoločnosti Investex bol vypracovaný editor máp pre budúce mapové komponenty aplikácie SDI. V tejto práci bol popísaný spôsob tvorby editoru, spôsob ovládania a v poslednej časti boli zhrnuté nedostatky programu a zároveň poskytnuté možnosti jednoduchého rozšírenia programu bez hlbšej znalosti celej aplikácie.

Pre tvorbu programu bolo využité vývojové prostredie Eclipse 3.2. Program bol napísaný v jazyku Java (presnejšie J2EE), je teda spustiteľný na každom operačnom systéme s príslušným JRE. Pri vývoji grafického rozhrania bola použitá sada nástrojov SWT, takže na spustenie sú potrebné priložené knižnice. Kvôli dôvodom uvedeným v teoretickej časti obmedzujú tieto knižnice spustiteľnosť programu len na OS Windows (platforma *WIN32*), pri spúšťaní v inom operačnom systéme je teda nutné do adresára s programom nakopírovať SWT knižnice pre daný systém.

Editor v tejto dobe pracuje s dátami uloženými v pamäti sekvenčne, jedinou optimalizáciou je teda vykresľovanie len objektov ktoré zasahujú do zobrazovanej oblasti. Spôsob riešenia tohto problému bol navrhnutý v predchádzajúcich častiach. Pri momentálnej nutnosti spracovania mapy Slovenskej Republiky je však momentálna rýchlosť postačujúca. Nakreslené mapy sa ukladajú do vybraného súboru na lokálnom disku. Serverová časť aplikácie nebola spoločnosťou Investex dodaná, preto komunikácia so serverom a spôsob prenosu dát nie sú vyriešené.

Mapa je rozčlenená do tematických vrstiev, ktoré uchovávajú rozličné podstatné kartografické objekty popísané vyššie. Tieto objekty môžu uchovávať niektoré základné dáta (dôležité hlavne pre bodové objekty, napr. vlastnosť *Typ* objektu *Stanica*). Komplexnejšie formy dát (napr. zadávanie parametrov trate po kilometroch) sú zatiaľ vo vývoji, nie sú preto v práci popisované.

Pre prevod geografických polôh udávaných (resp. získavaných z GPS) v stupňoch do súradnicového systému aplikácie (a opačne) bol vytvorený kalibračný systém založený na *Delaunayovej triagulácii*. Ten je základom pre ďalšie funkcie na prácu s GPS údajmi, ktoré budú doplnené v prípade potreby.

ZÁVER V ANGLIČTINE

According to demands of the Investex Group s.r.o., a map editting application for planned map components of the SDI application was designed. This thesis describes approaches to develop the application, ways of using it and digests defects of the program in the last part. There were served possibilities of simple extension of the application without detailed information about it.

Eclipse 3.2 IDE was used as the environment for designing the program. The program was written in Java language (closely J2EE), so it is runnable on every opperating system with corresponding JRE. For *GUI* design there was used the SWT toolkit, so attached libraries are required to run the program. Because of reasons mentioned in the theoretical part, these libraries are constraining application runnability to OS Windows (*WIN32* platform). When running in other OS, it is needed to copy SWT libraries for that system to the application directory.

In current version, the editor is working with sequentially stored data, so the only optimalization is drawing only objects reaching the viewing area. The ways of solving this problem were offered in previous parts. Because of the necessity to draw only the map of the Slovak Republic at the moment, the current speed is satisfactory. Drawed maps can be saved into a file on the hard drive. The server part of the application was not delivered by the Investex company, so the communication with the server and the way of data transfers are not solved.

The map is divided into thematical layers, which are storing various important cartographic objects described thereinbefore. These objects can store some basic data (important mainly for point objects, e.g. the *Type* property of the *Station* object). More complex data forms (e.g. entering railway prameters by kilometers) are still in progress, so they are not mentioned in this work.

To transform geographic positions entered (or acquired from GPS) in degrees to the application coordinate system (and back), there was created a calibration system based on the *Delaunay triangulation*. It is the base for another functions for working with GPS data, which will be added in the case of need.

ZOZNAM POUŽITEJ LITERATÚRY

[1] BRŮHA, L. *Java - Hotová řešení*. [s.l.] : [s.n.], 2004. 328 s.

[2] *Eclipsedia : JFace* [online]. 15.5.2007 [cit. 2007-05-18]. Dostupný z WWW: <http://wiki.eclipse.org/index.php/JFace>.

[3] GAMMA, E, et al. *Design Patterns : Elements of Reusable Object-Oriented Software*. [s.l.] : [s.n.], 1995. 395 s.

[4] HORBAJ, P. Základné informácie o GIS a GPS a niektoré možnosti ich využívania (1). *AT&P journal*. 1.1.2005, č. 7/2005, s. 83-84.

[5] KIMIJAN, P. *Čo je to GPS?* [online]. 2005-2007 [cit. 2007-05-15]. Dostupný z WWW: <http://www.gps.favoriteam.sk/gps/>

[6] KISZKA, B. *Programování v jazyce Java : 1001 tipů a triků*. [s.l.] : [s.n.], 2003. 519 s.

[7] KUBÍČEK, J. *Geografické Informačné Systémy - GIS*. [s.l.], [2007]. 4 s. Odborová práca.

[8] MCNAMARA, J. *GPS for Dummies*. [s.l.] : [s.n.], 2004. 408 s.

[9] PECINOVSKÝ, R. *Myslíme objektově v jazyku Java 5.0*. [s.l.] : [s.n.], 2004. 604s.

[10] VLISSIDES, J., et al. *An Introduction To Design Patterns*. [s.l.] : [s.n.], 1999. 77 s.

[11] *Webopedia : Object-Oriented programming* [online]. 7.1.2003 [cit. 2007-05-18]. Dostupný z WWW: <http://www.webopedia.com/TERM/O/object_oriented_programming_OOP.html>.

[12] *Wikipédia : Global Positioning System* [online]., 14.5.2007 [cit. 2007-05-17]. Dostupný z WWW: <http://sk.wikipedia.org/wiki/Global_Positioning_System>.

[13] *Wikipédia : Kartografia* [online]., 10.5.2007 [cit. 2007-05-17]. Dostupný z WWW: <http://sk.wikipedia.org/wiki/Kartografia>.

[14] *Wikipedia : Standard Widget Toolkit* [online]. 16.5.2007 [cit. 2007-05-18]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Standard_Widget_Toolkit>.

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

API	*Application Programming Interface* - súbor funkcií poskytovaný určitou knižnicou resp. operačným systémom
AWT	*Abstract Window Toolkit* - staršia kolekcia tried na tvorbu užívateľského rozhrania v jazyku Java
GIS	*Geografický Informačný Systém* - technické (v tejto práci výhradne sofwarové) vybavenie na prácu s geografickými údajmi
GPS NAVSTAR	GPS - *Global Positioning System* - družicový systém pre určovanie polohy. NAVSTAR - *NAVigation Signal for Timing And Ranging*
GUI	*Graphical User Interface* - grafické užívateľské rozhranie
IDE	*Integrated Development Environment* - integrované prostredie pre vývoj aplikácií
Javadoc	Priemyselný štandard na dokumentovanie tried napísaných v Jave
J2EE	Časť platformy Java s najrozsiahlejšou množinou funkcií.
JOGL	Knižnica poskytujúca funkcie OpenGL jazyku Java.
JRE	*Java Runtime Environment* - sofwarový balík umožňujúci beh programov napísaných v jazyku Java
OOP	*Object-Oriented Programming* - objektovo orientované programovanie
PNG	*Portable Network Graphics* - často používaný grafický formát
SDI	*Server Dopravných Informácií* - software vyvíjaný spoločnosťou Investex Group s.r.o. na automatizovanie činností bežných v logistike
SWT	*Standard Widget Toolkit* - knižnice jazyka Java poskytujúce štandardné ovládacie prvky operačného systému
WIN32	*32-bitové API* systémov *Windows* (od Windows 95 až po najnovšie 64-bitové verzie)
XML	*eXtensible Markup Language* - obecný značkovací jazyk, ktorý predstavuje štandardný formát na výmenu informácií

ZOZNAM OBRÁZKOV

ZOZNAM PRÍLOH

P1 *Zdrojový kód programu* (vrátane celého projektu pre prostredie Eclipse). Príloha sa nachádza na priloženom CD v adresári *P1_workspace*. Po spustení programu Eclipse je nutné nastaviť ako *workspace* tento adresár.

P2 *Dokumentácia* k zdrojovému kódu (*javadoc*) vo formáte *HTML*. Nachádza sa na priloženom CD v adresári *P2_doc*. Pre prezeranie je nutné otvoriť súbor *index.html*.

P3 Vývojové prostredie *Eclipse 3.2*. Nachádza sa na CD v adresári *P3_eclipse*.

P4 *Skompilovaný program* s knižnicami pre platformu *WIN32*. Nachádza sa na CD v adresári *P4_runnable*. Program sa spúšťa dávkovým súborom *!start.bat*.

P5 *Java Runtime Environment*. Softwarový balík, ktorý je nutné nainštalovať pre spustenie aplikácie. Nachádza sa na priloženom CD v adresári *P5_jre*.

P6 *J2EE SDK*. Knižnice, ktoré je nutné nainštalovať v prípade úpravy zdrojového kódu a následnej kompilácie. Nachádzajú sa na CD v adresári *P6_j2ee*. Po inštaláci nie je nutné inštalovať JRE, pretože balík SDK ho obsahuje.

P7 *Dokumentácia J2SE* (Java 2 Standard Edition). Nachádza sa na CD v adresári *P7_doc_j2se*. Kompletná online dokumentácia k J2EE sa nachádza na adrese http://java.sun.com/j2ee/docs.html.

Algoritmy a rasterizace 2D grafických objektů

Algorithms and rasterization of 2D graphical objects

Jan Sečkař

Bakalářská práce
2007

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

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

Zásady pro vypracování:

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

Rozsah práce:

Rozsah příloh:

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

Seznam odborné literatury:

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

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

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

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

Ve Zlíně dne 13. února 2007

prof. Ing. Vladimír Vašek, CSc.
děkan

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

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

ABSTRAKT

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

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

ABSTRACT

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

Keywords: rasterization, vector, circle, ellipse, curve

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

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

Ve Zlíně …….

Podpis diplomanta

OBSAH

ÚVOD

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

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

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

I. TEORETICKÁ ČÁST

1 JEDNODUCHÉ OBJEKTY

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

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

1.1 Úsečka

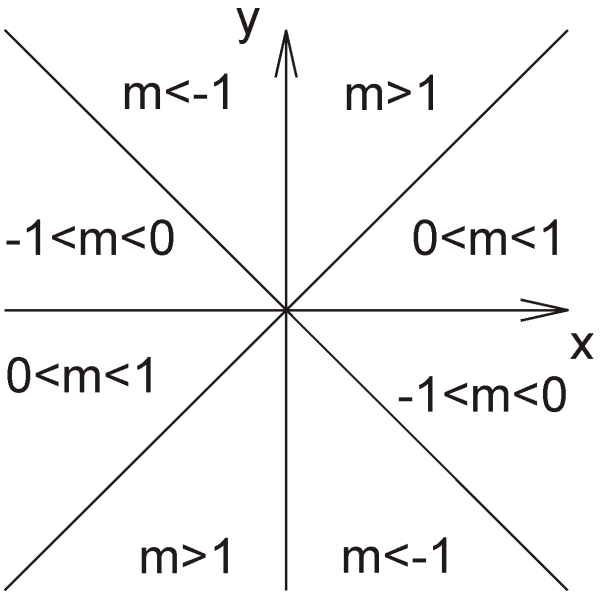
Úsečka je nejjednodušší grafický prvek. V analytické geometrii obsahuje úsečka nekonečně mnoho bodů. Na rastrovém monitoru se však skládá z konečného počtu bodů, které získáme rasterizací. Navzdory tomu, že je poměrně jednoduchá, její rasterizaci je věnována velká pozornost. Pomocí úseček se totiž vykreslují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

1.1.1 Rasterizace úsečky

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

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

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



Obr. 1: Velikost směrnice m pro různé sklony úsečky

1.1.2 DDA algoritmus

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

Z parametrického popisu úsečky

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

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

je možné odvodit pro osu *x* iterační zápis

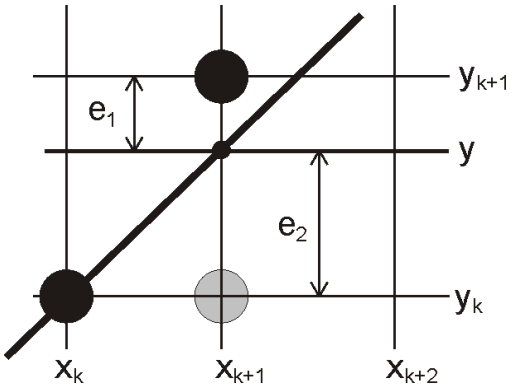
$$x_{k+1} = x_k + 1, \tag{4}$$

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

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

1.1.3 Bresenhamův algoritmus

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



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

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

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

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

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

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

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

Pro rychlejší výpočet je odvozena rovnice

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

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

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

Pro výpočet následujícího bodu je vzorec upraven do tvaru

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

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

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

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

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

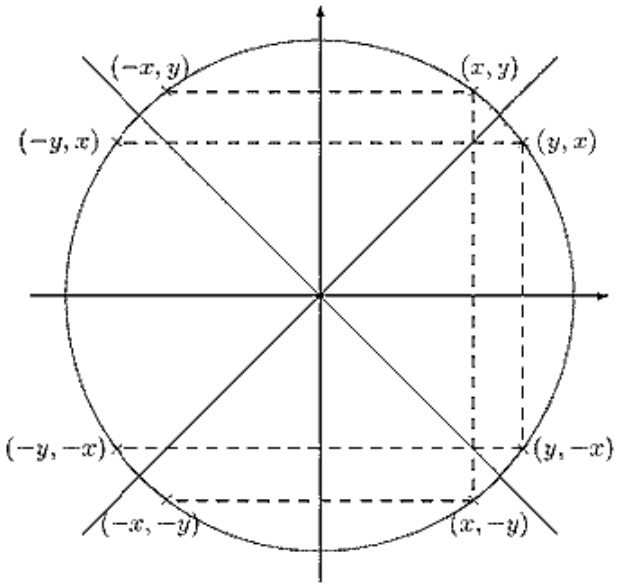
1.2 Kružnice

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

1.2.1 Rasterizace kružnice

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

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



Obr. 3: Symetrické rozdělení kružnice

Další možností rasterizace kružnice je rasterizace pomocí polárních souřadnic.

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

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

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

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

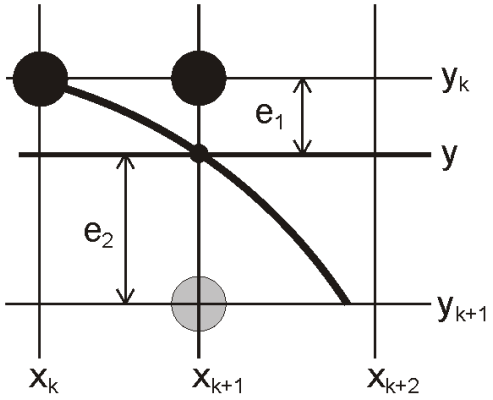
1.2.2 Bresenhamův algoritmus

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

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

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

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



Obr. 4: Výběr pixelu kružnice na základě velikosti odchylky

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

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

Po výpočtu zjistíme podle znaménka polohu následujícího pixelu pomocí kritérií

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

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

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

1.3 Elipsa

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

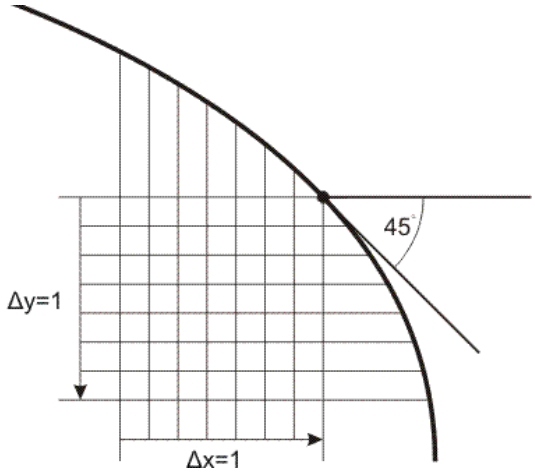
1.3.1 Bresenhamův algoritmus

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

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

Implicitní rovnice elipsy se středem v počátku je

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



Obr. 5 – Změna řídící osy při rasterizaci elipsy

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

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

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

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

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

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

2 KŘIVKY

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

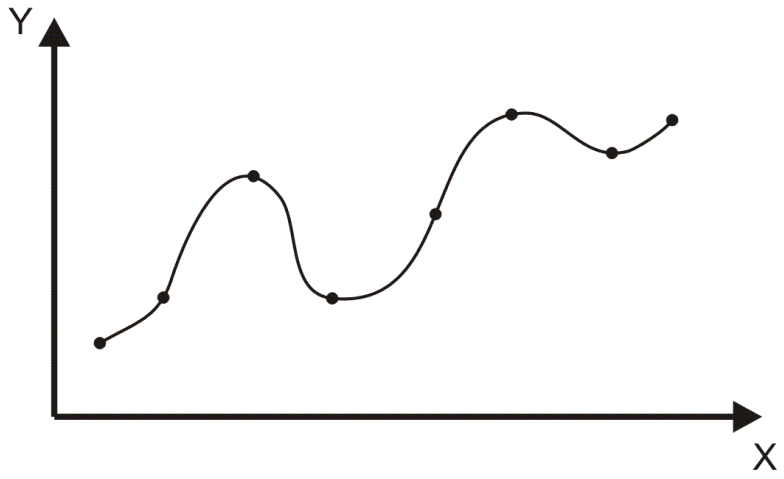
Křivky, které poskytují dostatečně širokou škálu tvarů jsou křivky třetího stupně (kubiky). Tyto křivky jsou také nejčastěji využívány. Jsou jednoduše manipulovatelné a je možné u nich zaručit takzvanou spojitost C^2, která vytváří plynulou návaznost křivek. Výpočet těchto křivek bývá nenáročný.

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

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

2.1 Křivky interpolační

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



Obr. 6 – Interpolační křivka

2.1.1 Interpolace jediným polynomem

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

Interpolace polynomem n -1 řádu znamená najít řešení rovnic

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

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

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

2.1.2 Interpolace po částech

Častější interpolační technikou je interpolace křivky po částech.

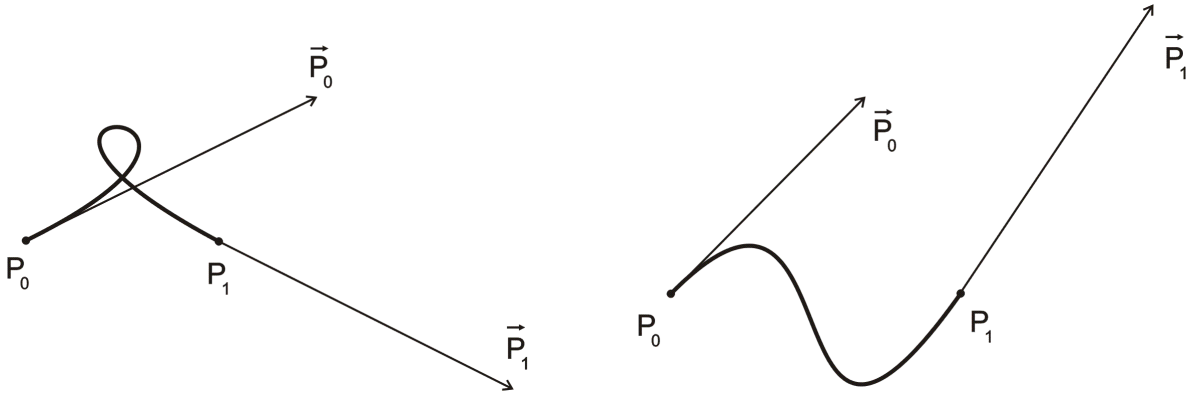
Aproximací polynomem třetího řádu potom rozumíme aproximaci jejích každých čtyř bodů polynomem třetího stupně. Tímto však vzniká problém návaznosti křivek v místě jejich propojení. Existují však metody, pomocí kterých je řešena návaznost v těchto bodech automaticky.

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

2.1.3 Fergusonovy kubiky

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

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



Obr. 7 – Příklad Fergusonových kubik

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

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

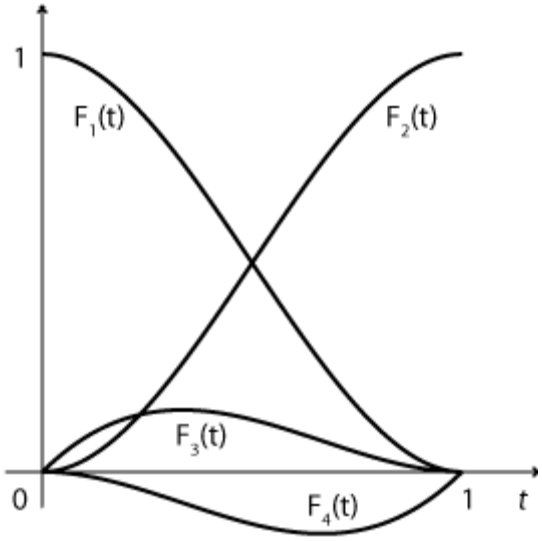
kde F_1, F_2, F_3, F_4 jsou kubické Hermitovské polynomy tvaru

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

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

Maticový zápis:

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



Obr. 8 - Hermitovské polynomy

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

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

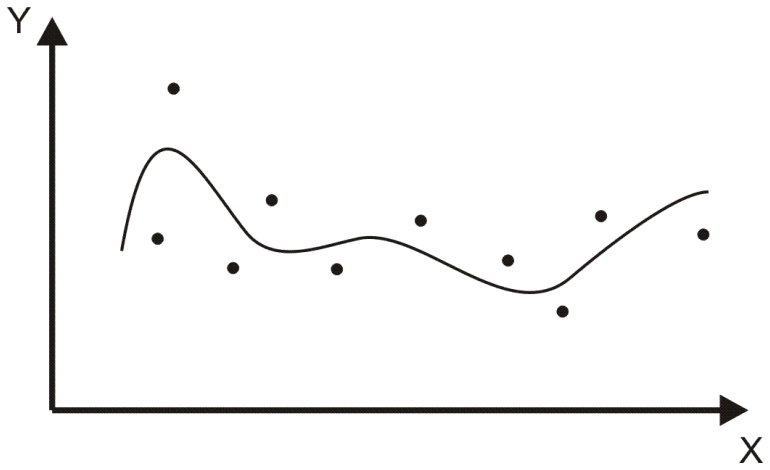
Spojitost při navazování oblouků je zaručena, v případě že je roven poslední bod předchozího oblouku prvnímu bodu oblouku následujícího. Spojitost C^2 je zaručena, pokud jsou velikosti vektorů P_1' předchozího a P_0' následujícího oblouku shodné.

2.2 Křivky aproximační

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

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

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



Obr. 9 – Aproximační křivka

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

Parametricky lze danou kubiku Q(t) vyjádřit ve tvaru

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

Zkráceně můžeme zapsat v maticovém tvaru

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

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

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

Konstantní matici *C* můžeme rozepsat do součinu

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

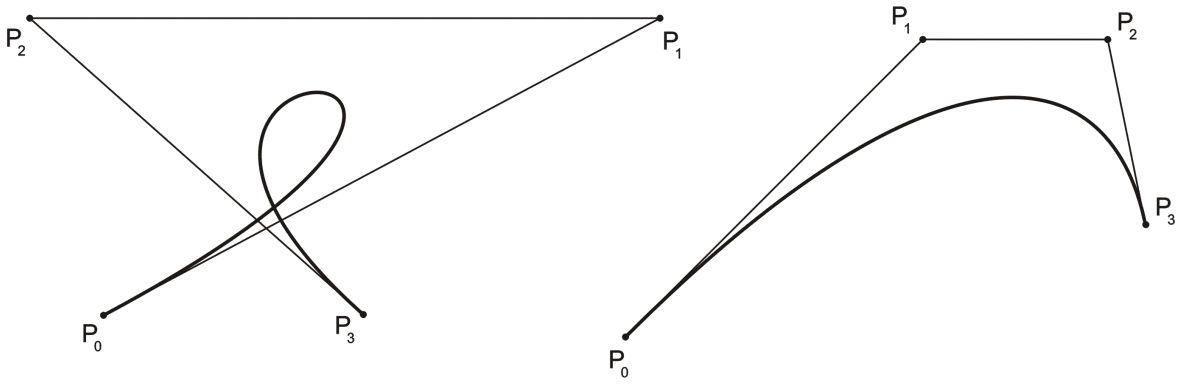
kde matice *M* je typu 4x4 a nazývá se bázová matice. Čtyřprvkový vektor *G* se nazývá geometrický vektor. Geometrický vektor reprezentuje vliv vnějších parametrů. Obsahuje řídicí body, nebo řídicí body a tečné vektory atp. Bázová matice, která je dána použitou metodou,pak určuje výpočet křivky podle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

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

2.2.1 Beziérovy kubiky

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

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



Obr. 10 – Příklad Bezpérových kubik

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

Beziérova kubika je určena vztahem

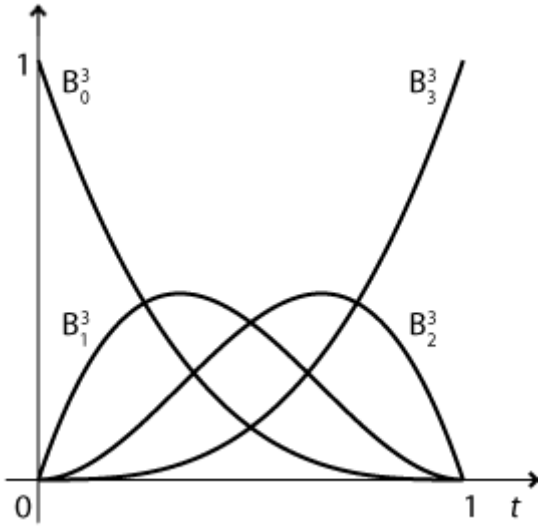
$$P(t) = P_0 B_0(t) + P_1 B_1(t) + P_2 B_2(t) + P_3 B_3(t) = \sum_{i=0}^{3} P_i B_i(t), \tag{32}$$

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

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

Maticový zápis:

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



Obr. 11 - Kubické Bernsteinovy polynomy

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

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

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

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

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

3 JAVA

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

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

3.1 Základní vlastnosti

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

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

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

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

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

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

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

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

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

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

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

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

- **víceúlohový** – podporuje zpracování vícevláknových aplikací

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

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

3.2 Nevýhody jazyka Java

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

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

3.3 Eclipse

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

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

II. PRAKTICKÁ ČÁST

4 PROGRAMOVÁ ČÁST

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

4.1 Struktura programu

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

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

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

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

4.2 Popis jednotlivých tříd

4.2.1 Rasterize

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

4.2.2 Canvas

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

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

4.2.3 IntInput

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

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

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

4.2.4 Writer

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

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

4.2.5 Scene

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

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

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

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

4.2.6 IRasterizer

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

Třída obsahuje funkce:

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

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

4.2.7 IRasterizerDetail

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

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

4.2.8 RasterizerBasicCircle

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

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

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

4.2.9 RasterizerBezier

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

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

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

4.2.10 RasterizerBresenhamCircle

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

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

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

4.2.11 RasterizerBresenhamEllipse

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

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

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

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

4.2.12 RasterizerBresenhamLine

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

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

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

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

4.2.13 RasterizerDDALine

Třída byla vytvořena pro rasterizaci úsečky pomocí DDA algoritmu.

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

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

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

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

4.2.14 RasterizerFerguson

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

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

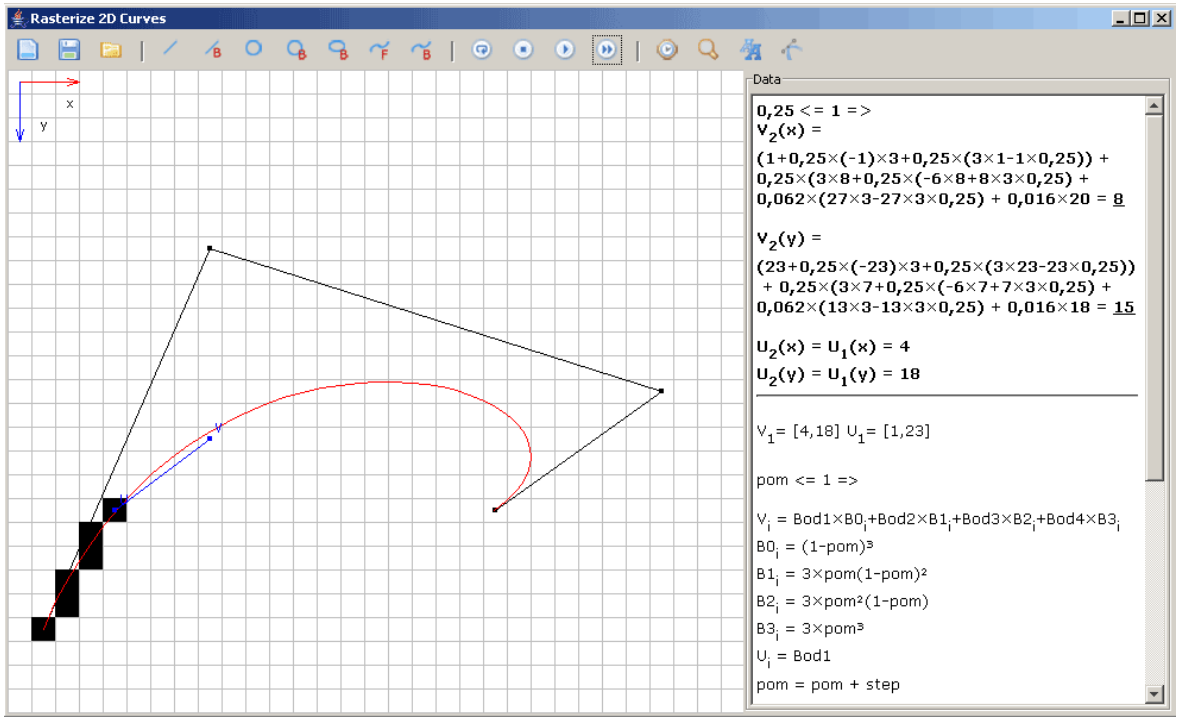
Proces rasterizace je obdobný jako u rasterizace Beziérovy kubiky. Po zadání všech bodů je volána funkce *rasterizeMain()*. Ta vypočte body první přímky a zobrazí užité vzorce do panelu *data*.

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

5 UŽIVATELSKÉ ROZHRANÍ

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

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



Obr. 12 – Uživatelské rozhraní

5.1 Parametry

Uživatelsky je možné nastavovat některé parametry programu v určeném rozsahu:

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

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

5.2 Ovládání

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

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

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

ZÁVĚR

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

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

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

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

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

CONCLUSION

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

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

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

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

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

SEZNAM POUŽITÉ LITERATURY

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

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

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

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

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

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

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

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

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

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

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

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

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

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

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

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

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