PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA INFORMATIKY

BAKALÁŘSKÁ PRÁCE

Grafický editor s výstupem v MetaPostu

2012

Jan Tomeček

Anotace

Cílem práce je vytvoření jednoduchého nástroje pro vytváření kvalitních obrázků. Jde o grafický vektorový editor, jehož výstupem je kód v jazyku METAPOST.

Děkuji RNDr. Arnoštu Večerkovi za vedené mé bakalářské práce a jeho cenné rady. Dále bych rád poděkoval své dceři za psychickou podporu při psaní této práce.

Obsah

Seznam obrázků

1. Úvod

V této kapitole se čtenář seznámí krátce se systémem LaTeX a METAPOST. Je zde d uvedena cílová skupina uživatelů a stručný popis aplikace včetně srovnání s existujícími programy.

1.1. Krátce o MetaPostu

METAPOST je programovací jazyk určený ke kreslení vektorové grafiky. Je velmi podobný jazyku METAFONT, což je programovací jazyk vyvinutý prof. D.E. Knuthem na vytváření tzv. *pérovek*. Hlavním rozdílem je fakt, že výstupem METAPOSTu narozdíl od METAFONTu je vektorová grafika – buď PostScriptové programy nebo SVG grafika. Navíc obsahuje další prvky, které v METAFONTu nejsou podporované, jako třeba integrace textu a grafiky, zpřístupnění některých prvků PostScriptu (např. ořezávání, stínování, přerušované čáry). Více informací lze nalézt v [1].

1.2. Cílová skupina

Aplikace je určena pro osoby, které vytvářejí dokumenty v systému LaTeX a čas od času do nich potřebují vložit nějaký jednoduchý obrázek – zejména nějaký graf funkce, apod. Jde o lidi, kteří by rádi měli kvalitní obrázek, ovšem nemají čas ani chuť učit se syntaxy METAPOSTu. Tato aplikace je jednoduchým řešením tohoto problému. V několika minutách lze vytvořit soubor s příponou MP a ten pak jednoduše přeložit pomocí kompilátoru METAPOSTu. Výsledek pak lze použít přímo do LaTeXového souboru – viz Přílohu A.

1.3. Stručný popis činnosti aplikace

Aplikace, nazvaná MPEditor, je jednoduchý vektorový editor, jehož hlavní funkcí je generování kódu v METAPOSTu. Umožňuje vytvářet polygony, Bézierovy křivky, elipsy, textové popisky. Obsahuje standardní funce pro jejich úpravy - transformace, změna vlastností, a další.

1.4. Existující řešení a jejich srovnání

Existuje celá řada programů mající podobnou (a mnohem širší) funkčnost jako předkládaná aplikace. Následuje popis těch, o jejichž existenci autor této práce ví:

GNUplot je asi nejznámější vektorový editor po UNIXovými systémy. Jeho GUI je již poněkud zastaralé. Co se týče funkčnosti, nejsem schopen ji posoudit, protože jsem neměl možnost s ním pracovat.

Xfig je také celkem známý editor pod Windows, již s poněkud zastaralým ikdyž intuitivním ovládáním. Nicméně jedna z jeho velkých výhod je možnost exportu do velkého množství formátů.

WinFIG velmi pěkný vektorový editor pod Windows. Možnost exportu do obrovského množství grafických formátů, včetně METAPOST kódu. Při experimentování jsem přišel na množství chyb (chyby za běhu programu, chybný METAPOST kód). **Není** volně šiřitelný.

Metagraf je velmi povedený editor, mající mnoho funkcí. Pro mě bylo ovšem trochu nepohodlné seznamování s jeho GUI.

Program, jež je hlavním výsledkem této práce se nemůže ve funkčnosti vůbec srovnávat se zmiňovanými editory. Tato nevýhoda může být chápána jako výhoda, protože jeho ovládání je velmi jednoduché a uživatel se velmi rychle seznámí se všemi jeho funkcemi.

2. MetaPost

V této kapitole si ve stručnosti popíšeme jazyk METAPOST a to zejména ty jeho aspekty, které byly využity v prezentovaném programu. Další informace může čtenář nalézt v [1].

Jak již bylo řečeno, METAPOST je programovací jazyk, který slouží k vytváření vektorových obrázků. Je vhodný zejména pro technickou literaturu, především pro matematiku. Jeho výstupem je vysoce kvalitní grafika.

2.1. Proces vytváření obrázků

Na začátku stojí zdrojový text, který je zpravidla uložen v souborech s příponou `mp`. Tento soubor má předepsanou strukturu – zhruba lze říct, že obsahuje jeden nebo několik obrázků – viz dále. Zdrojový kód pak můžeme přeložit pomocí stejnojmeného překladače jazyka METAPOSTu – jde o soubor `mpost.exe`, tzn. na příkazovou řádku zadáme příkaz

`mpost` ⟨vstupní `mp` soubor⟩

Kompilátor vygeneruje několik souborů:

- soubor obsahující záznam o kompilaci (`*.log`),
- soubory s obrázky v PostScriptovém formátu `*.1`, `*.2`, ... (samozřejmě za předpokladu, že kompilace proběhla v pořádku).

Soubory s obrázky mohou být poté např. vloženy do dokumentů vytvořených systémem LaTeX.

2.2. Struktura mp souboru

Každý `mp`–soubor může obsahovat kód pro jeden či více obrázků. Jeho struktura je následující:

```
beginfig(1);
...
endfig;

beginfig(2);
...
endfig;

...

end
```

Mezi příkazy `beginfig` a `endfig` jsou pak samotné příkazy pro kreslení. Například výsledek příkazů umístěných mezi příkazy `beginfig(1)` a `endfig` je po úspěšném překladu uložen do souboru `*.1`, atd. Mimo tyto ohraničující příkazy mohou být (na začátku souboru) některé příkazy nastavující různé interní proměnné řídící překlad – pro podrobnosti viz [1].

2.3. Jednotky v MetaPostu

METAPOST pracuje se širokou škálou různých měrných jednotek – pracuje s centimetry (`cm`), palci (`inch`), postskriptovými body (`bp`), body (`pt`), atd. Platí mezi nimi následující vztahy:

$$1\ \texttt{inch} = 2.54\ \texttt{cm}, \quad 1\ \texttt{bp} = \frac{1}{72}\ \texttt{inch}, \quad 1\ \texttt{pt} = \frac{1}{72.27}\ \texttt{inch}.$$

2.4. Proměnné MetaPostu

METAPOST má deset základních datových typů: `numeric, pair, path, transform, (rgb)color, cmykcolor, string, boolean, picture, pen`. Budeme zde používat datové typy:

- `pair` – datový typ nesoucí x–ovou a y–ovou souřadnici bodu.
- `path` – (cesta) slouží k uložení různých křivek. Pomocí nich lze vykreslovat křivky, vyplňovat uzavřené křivky (např. elipsy), apod.
- `transform` – datový typ k uložení affiního zobrazení (rovinné transformace)
- `pen` – hlavní funkce pera spočívá v stanovení tloušťky a tvaru pera (umožnující kaligrafické efekty).

Například příkazy

```
z0 = (0,0);
z1 = (1cm,2cm);
z2 = (2cm,1cm);
z3 = (-1cm,0);
```

definují čtyři body v rovině. Příkaz

```
path p[];
```

deklaruje pole cest označené symbolem `p` a pak příkazy

```
p0 = z0--z1;
p1 = z0--z1--z2;
p2 = z0..controls z1 and z2..z3;
p3 = fullcircle;
```

představují inicializaci cest: `p0` představuje úsečku o vrcholech `z0` a `z1`; `p1` představuje polygon o vrcholech `z0`, `z1` a `z2`; `p2` představuje Bézierovu křivku začínající `z0` a končící `z3` s kontrolními body `z1`, `z2` a nakonec `p3` představuje jednotkovou kružnici.

Při psaní v jazyku METAPOST se nevyhneme afinním zobrazením. Transformovat můžeme jednotlivé body i celé cesty, např. kružnice o středu v bodě z0 a průměru 1cm je dána jako cesta daná vyhodnocením výrazu

```
fullcircle scaled 1cm shifted z0
```

Vysvětlení je následující: na jednotkovou kružnici (cesta `fullcircle`) se aplikovala rovinná transformace *změna měřítka* (`scaled`) s koeficientem rovným 1 `cm` a poté rovinná transformace *posunutí* (`shifted`) o vektor jenž je průvodičem bodu `z0`. Obecně lze deklarovat proměnnou reprezentující transformaci příkazem

```
transform T;
```

V METAPOSTu existuje řada předdefinovaných základních transformací, jako třeba již zmíněná *změna měřítka* `scaled` definovaná jako:

$$(x, y) \texttt{ scaled } a \quad = \quad (ax, ay)$$

nebo třeba *posunutí* `shifted` jako:

$$(x, y) \texttt{ shifted } (a, b) \quad = \quad (x + a, y + b)$$

a mnohé další. Obecně, je–li `q` cesta, pak

```
q transformed T
```

je cesta vzniklá z cesty `q` transformací `T`. Afinní zobrazení má šest parametrů, viz kapitolu o afinních zobrazeních. Máme–li transformaci `T`, její koeficienty lze identifikovat takto:

```
xpart T, ypart T, xxpart T, xypart T, yxpart T, yypart T,
```

znamenající koeficienty po řadě

$$t_{31}, t_{32}, t_{11}, t_{12}, t_{21}, t_{22}.$$

2.5. Kreslení jednoduchých grafických objektů

2.5.1. Kreslení křivek/cest

Křivky se kreslí poměrně jednoduše. Slouží k tomu příkaz `draw`, který vykreslí danou cestu příslušným perem příslušnou barvou příslušným způsobem (plná, čerchovaná, tečkovaná čára, atd.). Nejprve se vybere příslušné pero, např.

```
pickup pencircle scaled 2pt withcolor red;
```

čímž se nastaví právě používané pero na pero tvaru kruhu o průměru 2 `pt` s červenou barvou (pokud bychom místo `pencircle` napsali `pensquare`, mělo by pero tvar čtverce o délce strany 2 `pt`). Pak příkaz

```
draw p0;
```

kde `p0` je cesta z předchozí podkapitoly, vykreslí červenou úsečku. Příkaz

```
drawarrow p0;
```

vykreslí stejnou úsečku s tím rozdílem, že v bodě `z1` je šipka. Podobně příkaz

```
drawarrow reverse p0;
```

vykreslí stejnou úsečku, mající šipku v bodě `z0` (příkaz `reverze p0` vrátí obrácenou cestu k cestě `p0`). A nakonec příkaz

```
drawdblarrow p0;
```

vykreslí úsečku s šipkami na obou koncích. Tento postup platí nejen pro úsečku nebo polygon, ale jakoukoliv cestu.

Chceme–li cestu `q` vykreslit třeba tečkovaně, provedeme to příkazem

```
draw q dashed withdots;
```

nebo čárkovaně, pak

```
draw q dashed evenly;
```

V METAPOSTu lze dokonce vytvořit přerušovanou čáru s vlastním vzorem. Přerušované čáry **nejdou** tvořit s jiným perem než `pencircle`.

2.5.2. Vyplňování uzavřených křivek/cest

Vyplňovat v METAPOSTu můžeme jen *uzavřené cesty*. K tomu lze použít klíčové slovo `cycle`. Např.

```
p5 = z0--z1--z2--cycle;
```

definuje trojúhelník o vrcholech `z0`, `z1` a `z2`. Příkaz

```
fill p5;
```

vyplní definovaný trojúhelník černou barvou. Pokud bychom si ho přáli vyplnit jinou barvou, slouží k tomu klíčové slovo `withcolor`, následované proměnnou typu *barva*. Např. kdybychom trojúhelník chtěli vyplnit červenou barvou, šlo by to udělat následujícím příkazem

```
fill p5 withcolor 1.red + 0.green + 0.blue;
```

2.5.3. Kreslení popisků

Pro vkládání popisků je určen příkaz `label`. Například příkaz

```
label("text",z0);
```

vykreslí popisek „text" u bodu `z0`. Většinou je ale potřeba vykreslit text o kousek posunutý od daného bodu. Například příkaz

```
label.bot("text", z0)
```

vykreslí popisek „text" **pod** bodem `z0` posunutý o 3 `bp`. Tuto vzdálenost lze změnit přenastavením proměnné `labeloffset`. Popisek může být orientován na všech 8 světových stran - dolů (`bot`), nahoru (`top`), doleva (`lft`), doprava (`rt`), nahoru–doleva (`ulft`), nahoru–doprava (`urt`), dolů–doleva (`llft`), dolů–doprava (`lrt`). Pokud navíc chceme spolu s popiskem vykreslit i tečku na daném referenčním bodě, stačí místo `label` napsat `dotlabel`. Průměr tečky je dán vnitřní proměnnou `dotlabeldiam` a je implicitně nastaven na 3 `bp`.

Font je nastaven pomocí proměnné `defaultfont`, což je řetězec a je implicitně nastaven na

```
defaultfont := "cmr10";
```

Písmo lze zvětšovat/zmenšovat pomocí proměnné `defaultscale`.

Většinou chceme ale vykreslovat text v TEXu. Toho lze dosáhnou tak, že text v `mp`–souboru, který chceme, aby byl zpracován TEXem, vložíme mezi klíčová slova `btex` a `etex`. Například

```
label.rt(btex $\sqrt x$ etex, z0);
```

umístí popisek $\sqrt{x}$ napravo od bodu `z0`. O tom, jak probíhá překlad, je možno se dočíst v citované literatuře.

2.6. Další možnosti MetaPostu

METAPOST toho umí samozřejmě mnohem více. Poskytuje spoustu pohodlných prostředků pro definování nejrůznějších cest, definování a vyplňování nejrůznějších oblastí, poskytuje základní programovací prostředky jako třeba cykly, umí řešit lineární rovnice, umožňuje definici maker a mnoho dalšího.

3. Použitý matematický aparát

Vzhledem k tomu, že jde o vektorový grafický editor, k jeho naprogramování je nutné znát základy analytické geometrie, a to zejména pojmy *afinních prostorů*, *afinních zobrazení*. V programu byla implementována funkce vytváření a transformace elipsy. K tomu účelu může posloužit *analytická teorie kuželoseček*. Nakonec je v této kapitole představena věta *o implicitní funkci* z matematické analýzy a její použití.

3.1. Afinní prostory a zobrazení

Tato kapitola vznikla výběrem a úpravou ze studijních textů [6, 7]. Vzhledem k rozsahu zde bude předpokládáno, že čtenář je seznámen s pojmy vektorového prostoru, jeho báze a dimenze, a dále s pojmem lineárního zobrazení. Uvedené definice budou téměř doslovně převzaty z těchto textů. Vzhledem k pozdější implementaci zde bude provedena malá kosmetická úprava, která spočívá v tom, že afinní souřadnice budou chápány jako řádky (v souladu se skriptem [7]) – to má za následek, že posléze definované matice afinních zobrazení budou transponované k těm definovaným v [6].

Nejprve je třeba nadefinovat pojem *afinního prostoru*.

Definice 1. Řekneme, že na množině A je dána struktura afinního prostoru, je–li dán vektorový prostor V a zobrazení $\mathrm{Add}_A : A \times V \to A$ takové, že pro všechna $a \in A$, u, $v \in V$ platí

$$\mathrm{Add}_A(\mathrm{Add}_A(a, u), v) = \mathrm{Add}_A(a, u + v)$$

a pro všechna a, $b \in A$ existuje právě jedno $v \in V$ tak, že

$$\mathrm{Add}_A(a, v) = b.$$

Množina A se strukturou afinního prostoru se nazývá afinní prostor, její prvky se nazývají body. Vektorový prostor V se nazývá zaměření afinního prostoru A. Dimenzi afinního prostoru A definujeme jako dimenzi jeho zaměření.

Nás bude zajímat zejména takový prostor, jehož zaměření je vektorový prostor všech uspořádaných dvojic reálných čísel (se standardně definovaným sčítáním prvků a násobením reálnými čísly) - budeme jej značit jako $(\mathbb{R}^2, +, \cdot)$ nebo krátce $\mathbb{R}^2$. Z vektorového prostoru se přirozeným způsobem může stát afinní prostor.

Poznámka 2. ([6], Příklad 2.2) Je–li V vektorový prostor, pak lze na něm definovat tzv. *kanonickou strukturu afinního prostoru*: Zaměřením prostoru V bude samotný prostor V; pro libovolné dva vektory u, $v \in V$ definujeme $\mathrm{Add}_V(u, v) = u + v$, kde součet na pravé straně je součet vektorů ve vektorovém prostoru V.

Nás tedy bude zajímat kanonická struktura afinního prostoru pro $V = \mathbb{R}^2$. Nebude–li hrozit nedorozumění, budeme psát v obou případech jen $\mathbb{R}^2$.

Zajímat nás budou zejména *afinní souřadnice*. K tomu je třeba definovat pojem afinní báze.

Definice 3. Nechť A je afinní prostor, V jeho zaměření. Nechť $a_0 \in A$ a $\alpha = (u_1, \ldots, u_n)$ je báze V. Pak dvojici $\varphi = (\alpha, a_0)$ nazýváme *afinní bází* afinního prostoru A. Bod a_0 nazýváme *počátkem báze* φ.
Pro libovolný bod $a \in A$ nazýváme *afinními souřadnicemi* bodu a vzhledem k afinní bázi $\varphi = (\alpha, a_0)$ $(n+1)$–tici

$$a_\varphi = ((a - a_0)^1_\alpha, \ldots, (a - a_0)^n_\alpha, 1).$$

Je třeba poznamenat, že každý bod afinního prostoru je svými afinními souřadnicemi jednoznačně určen. To nám umožnuje abstrahovat od podstaty prvku v afinním prostoru a chápat ho jako uspořádanou $(n+1)$–tici reálných čísel, přičemž poslední složka je rovna číslu 1. To bude výhodné mimojiné při práci s afinními zobrazeními.

Definice 4. *Nechť $a_1, \ldots, a_n$ jsou prvky afinního prostoru A, $c_1, \ldots, c_n$ jsou reálná čísla jejichž součet je roven jedné. Pak výraz*

$$c_1 a_1 + \ldots + c_n a_n$$

se nazývá afinní kombinací bodů $a_1, \ldots, a_n$ a jeho hodnota je definována výrazem

$$b + c_1(a_1 - b) + c_2(a_2 - b) + \ldots + c_n(a_n - b) \tag{1}$$

kde $b \in A$ je libovolný bod.

Definice afinní kombinace je korektní, protože se dá dokázat (viz [6], Věta 5.1), že hodnota výrazu (1) je vzhledem k prvku b konstantní.

Definice 5. *Zobrazení $f : A \to B$ afinních prostorů A, B se nazývá afinní zobrazení, jestliže pro každé a_1, $a_2 \in A$ a každou afinní kombinaci $c_1 a_1 + c_2 a_2$ platí*

$$f(c_1 a_1 + c_2 a_2) = c_1 f(a_1) + c_2 f(a_2)$$

Nyní se konečně dostáváme k pojmu matice afinního zobrazení.

Věta 6. *Nechť jsou dány afinní prostory A, B s bázemi $\varphi = (\alpha, a_0)$, $\psi = (\beta, b_0)$. Pak ke každému afinnímu zobrazení $f : A \to B$ existuje právě jedna matice $M^f_{\varphi,\psi}$ taková, že*

$$f(a)_\psi = a_\varphi M^f_{\varphi,\psi}.$$

Tato matice je ve tvaru

$$M^f_{\varphi,\psi} = \left(\begin{array}{c|c} M^{f_0}_{\alpha,\beta} & \begin{matrix}0\\ \vdots \\ 0\end{matrix} \\ \hline f(a_0)_\psi & \end{array}\right)$$

kde $M^{f_0}_{\alpha,\beta}$ je matice lineárního zobrazení f_0 (zobrazující zaměření A do zaměření B) definovaného předpisem

$$f_0(u) = f(a+u) - f(a), \quad \forall u$$

(kde $a \in A$ je libovolný prvek) vzhledem k bázím α, β.

V práci jsou použité afinní zobrazení pro $A = B = \mathbb{R}^2$ – budeme jim říkat *rovinná zobrazení* (*tranformace*). V $\mathbb{R}^2$ uvažujeme jedinou afinní bázi $\varphi = (a_0, \alpha)$, kde

$$a_0 = (0,0), \alpha = \{(1,0),(0,1)\}.$$

Pak pro každé $(x,y) \in \mathbb{R}^2$ platí

$$(x,y)_\varphi = (x,y,1).$$

Dále pak každá rovinná transformace $f : \mathbb{R}^2 \to \mathbb{R}^2$ je podle Věty 6 jednoznačně daná maticí ve tvaru

$$T = \begin{pmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{pmatrix} \tag{2}$$

tedy je určena šesti parametry. Pak pro každý bod (x,y) (s homogenními souřadnicemi $(x,y,1)$) platí

$$f((x,y))_\psi = (x,y,1)T = (t_{11}x + t_{21}y + t_{31}, t_{12}x + t_{22}y + t_{32}, 1)$$

tj.

$$f((x,y)) = (t_{11}x + t_{21}y + t_{31}, t_{12}x + t_{22}y + t_{32}).$$

Zajímat nás budou transformace posunutí (translace), otočení (rotace) kolem počátku, změna měřítka (škálování), zkosení a transformace vzniklé jejich složením (např. rotace kolem určitého bodu).

Transformace **posunutí o vektor o souřadnicích** (t_x, t_y) je dána maticí (vzhledem ke smluvené bázi)

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{pmatrix}.$$

Transformace **rotace okolo počátku o orientovaný úhel** φ je dána maticí

$$\begin{pmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Transformace **škálování ve směru osy x s_x–krát a ve směru osy y s_y–krát** je dána maticí

$$\begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Transformace **zkosení ve směru osy x s_x–krát** je dána maticí

$$\begin{pmatrix} 1 & 0 & 0 \\ s_x & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Transformace **zkosení ve směru osy y s_y–krát** je dána maticí

$$\begin{pmatrix} 1 & s_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Matice složitejších transformací, které se dají napsat jako *složení* výše uvedených transformací lze určit pomocí následující věty – uvedeme jen speciální případ pro rovinné transformace.

Věta 7. *Nechť $f : \mathbb{R}^2 \to \mathbb{R}^2$ a $g : \mathbb{R}^2 \to \mathbb{R}^2$ jsou rovinné transformace s maticemi (vzhledem k dohodnuté afinní bázi) T a S. Pak zobrazení $h = f \circ g : \mathbb{R}^2 \to \mathbb{R}^2$ tj. zobrazení definované předpisem*

$$h(x) = f(g(x)) \quad \forall x \in \mathbb{R}^2$$

je rovinná transformace, jejíž matice (vzhledem k dohodnuté afinní bázi) je rovna součinu

$$S \cdot T.$$

Například transformace **rotace kolem bodu** (c_x, c_y) **o orientovaný úhel** φ je složením posunutí o vektor $(-c_x, -c_y)$, rotace kolem počátku a nakonec posunutím o vektor (c_x, c_y), tzn. matice této transformace je ve tvaru

$$\begin{pmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ -c_x\cos\varphi + c_y\sin\varphi + c_x & -c_x\sin\varphi - c_y\cos\varphi + c_y & 1 \end{pmatrix}.$$

Nakonec budeme potřebovat určit matici inverzní rovinné transformace.

Věta 8. *Nechť $f : \mathbb{R}^2 \to \mathbb{R}^2$ je prostá rovinná transformace s maticí T (vzhledem k dohodnuté afinní bázi). Pak matice T je regulární, existuje inverzní zobrazení k f^{-1}, toto zobrazení je také rovinná transformace a její matice (vzhledem k dohodnuté afinní bázi) je rovna T^{-1}. Konkrétně pro matici ve tvaru (2) platí*

$$T^{-1} = \begin{pmatrix} \frac{t_{22}}{\det T} & -\frac{t_{12}}{\det T} & 0 \\ -\frac{t_{21}}{\det T} & \frac{t_{11}}{\det T} & 0 \\ \frac{t_{21}t_{32}-t_{22}t_{31}}{\det T} & \frac{t_{12}t_{31}-t_{11}t_{32}}{\det T} & 1 \end{pmatrix}$$

3.2. Analytická teorie kuželoseček

V programu je možnost vytvoření a transformací elipsy. Elipsa je v něm reprezentována (prostou) rovinnou transformací, která převede jednotkovou kružnici na tuto elipsu. To znamená, že elipsa je zde chápána jako obraz jednotkové kružnice

$$\partial K((0,0),1) = \{(x,y) \in \mathbb{R}^2 : x^2 + y^2 = 1\}$$

v rovinném zobrazení f daného obecně maticí (2), přičemž toto zobrazení je prosté, což je ekvivalentní regularitě matice T. Označíme–li $S = T^{-1}$, $S = (s_{ij})$ (viz Větu 8), pak je elipsa dána jako množina

$$E = \{(x,y) \in \mathbb{R}^2 : (s_{11}x + s_{21}y + s_{31})^2 + (s_{12}x + s_{22}y + s_{32})^2 = 1\}. \qquad (3)$$

S touto reprezentací je spojeno několik problémů. První spočítá v určení délek poloos elipsy a úhlu, o který je elipsa rotována ze základního tvaru (tzn. poloosy elipsy jsou rovnoběžné s osami souřadnic). Je to potřeba při vykreslování elipsy.

Problém si můžeme zjednodušit předpokladem, že $s_{31} = s_{32} = 0$, protože tyto koeficienty nemají na počítané hodnoty vliv – určují jen souřadnice středu elipsy. Následující postup je inspirován obecnějším postupem prezentovaným ve skriptu [4].

Uvažujme tedy rovnici

$$(s_{11}x + s_{21}y)^2 + (s_{12}x + s_{22}y)^2 = 1,$$

která je ekvivalentní rovnici

$$(s_{11}^2 + s_{12}^2)x^2 + 2(s_{11}s_{21} + s_{12}s_{22})xy + (s_{21}^2 + s_{22}^2)y^2 = 1.$$

Pro jednoduchost položme

$$A_{11} = s_{11}^2 + s_{12}^2, \quad A_{12} = 2(s_{11}s_{21} + s_{12}s_{22}), \quad A_{22} = s_{21}^2 + s_{22}^2.$$

Provedeme nyní změnu souřadnic – půjde o rotaci o úhel α

$$x = u\cos\alpha - v\sin\alpha,$$
$$y = u\sin\alpha + v\cos\alpha.$$

Dosazením do naší rovnice získáme rovnici

$$A_{11}(u^2\cos^2\alpha - 2uv\sin\alpha\cos\alpha + v^2\sin^2\alpha) +$$
$$A_{12}(u^2\sin\alpha\cos\alpha + uv(\cos^2\alpha - \sin^2\alpha) - v^2\sin\alpha\cos\alpha) +$$
$$A_{22}(u^2\sin^2\alpha + 2uv\sin\alpha\cos\alpha + v^2\cos^2\alpha) = 1$$

a po úpravě dostáváme

$$u^2(A_{11}\cos^2\alpha + A_{12}\sin\alpha\cos\alpha + A_{22}\sin^2\alpha) +$$
$$uv(-2A_{11}\sin\alpha\cos\alpha + 2A_{22}\sin\alpha\cos\alpha + A_{12}\cos 2\alpha) +$$
$$v^2(A_{11}\sin^2\alpha - A_{12}\sin\alpha\cos\alpha + A_{22}\cos^2\alpha) = 1$$

Nyní nás zajímá pro jakou hodnotu úhlu α je poslední rovnice rovnicí elipsy v základním tvaru, tzn. pro jakou hodnotu je koeficient u členu uv roven nule. To je právě tehdy, když

$$A_{12}\cos 2\alpha + (A_{22} - A_{11})\sin 2\alpha = 0$$

nebo–li

$$A_{12}\cos 2\alpha = (A_{11} - A_{22})\sin 2\alpha.$$

Je–li $A_{11} = A_{22}$, pak je rovnice splněna např. pro

$$\alpha = \pi.$$

V opačném případě lze rovnici podělit výrazem $A_{11} - A_{22}$ a po jednoduché úpravě dostáváme

$$\alpha = \frac{1}{2}\operatorname{arctg}\frac{A_{12}}{A_{11} - A_{22}}$$

Pro takovou hodnotu úhlu α je pak naše rovnice v základním tvaru

$$B_{11}u^2 + B_{22}v^2 = 1,$$

kde

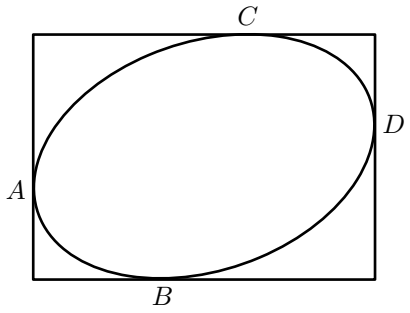
$$B_{11} = A_{11}\cos^2\alpha + A_{12}\sin\alpha\cos\alpha + A_{22}\sin^2\alpha,$$
$$B_{22} = A_{11}\sin^2\alpha - A_{12}\sin\alpha\cos\alpha + A_{22}\cos^2\alpha.$$

Odtud již snadno určíme délky poloos a a b, tzn.

$$a = \frac{1}{\sqrt{B_{11}}}, \quad b = \frac{1}{\sqrt{B_{22}}}.$$

3.3. Věta o implicitní funkci

Dalším problémem vyvstávajícím u elipsy je nalezení tzv. ohraničujícího obdélníka (*bounding rectangle*), což je elipse opsaný obdélník, jehož strany jsou rovnoběžné s osami souřadnic.
Naším úkolem bude nalézt souřadnice bodů A, B, C, D, vyznačené na obrázku 1. Konkrétně nás budou zajímat x–ové souřadnice bodů A a D a y–ové souřadnice



Obrázek 1. Elipsa a její *ohraničující obdélník*.

bodů B a C.

K určení ohraničujícího obdélníka využijeme větu o implicitní funkci, kterou lze najít téměř v každé učebnici diferenciálního počtu, např. viz [5].

Věta 9. *Nechť $F = F(x, y) : \mathbb{R}^2 \to \mathbb{R}$, $(x_0, y_0) \in \mathbb{R}^2$ tak, že $F(x_0, y_0) = 0$ a dále platí*

- *F je spojitá na okolí bodu (x_0, y_0),*
- *F má v bodě (x_0, y_0) spojitou parciální derivaci podle proměnné y,*
- $\dfrac{\partial F}{\partial y}(x_0, y_0) \neq 0.$

Pak existuje $\epsilon > 0$, $\delta > 0$ a funkce $f : (x_0 - \delta, x_0 + \delta) \to (y_0 - \epsilon, y_0 + \epsilon)$ taková, že

$$\Big(\mathrm{H}_0(F) \cap (x_0 - \delta, x_0 + \delta) \times (y_0 - \epsilon, y_0 + \epsilon)\Big) = \mathrm{graf}(f),$$

kde $\mathrm{H}_0(F) = \{(x, y) \in \mathbb{R}^2 : F(x, y) = 0\}$ je tzv. 0–hladina funkce F. Navíc, platí–li

- *existuje vlastní* $\dfrac{\partial F}{\partial x}(x_0, y_0)$,

pak funkce f má v bodě x_0 vlastní derivaci a platí

$$f'(x_0) = -\frac{\frac{\partial F}{\partial x}(x_0, y_0)}{\frac{\partial F}{\partial y}(x_0, y_0)}.$$

Poznámka 10. *O funkci f z předchozí věty říkáme, že je v okolí bodu (x_0, y_0) implicitně zadána (definována) rovnicí*

$$F(x, y) = 0.$$

Jak již bylo řečeno, elipsa je zde chápána jako množina (3), což není nic jiného, než 0–hladina funkce

$$F(x, y) = (s_{11}x + s_{21}y + s_{31})^2 + (s_{12}x + s_{22}y + s_{32})^2 - 1.$$

Tedy pro každý bod (x_0, y_0) ležící na elipse platí $F(x_0, y_0) = 0$. Z obrázku můžeme odhadnout, že podle věty 9 jsou v okolí bodů $B = (x_B, y_B)$ a $C = (x_C, y_C)$ definovány implicitně funkce, označme je f_B, f_C. Je vidět, že $f'_B(x_B) = 0$, $f'_C(x_C) = 0$. Podle věty 9 tedy platí

$$F(x_B, y_B) = F(x_C, y_C) = \frac{\partial F}{\partial x}(x_B, y_B) = \frac{\partial F}{\partial x}(x_C, y_C) = 0$$

přičemž

$$\frac{\partial F}{\partial y}(x_B, y_B) \neq 0, \quad \frac{\partial F}{\partial y}(x_C, y_C) \neq 0.$$

Nalezneme souřadnice bodů B a C. Budeme tedy řešit soustavu dvou rovnic o dvou neznámých

$$F(x, y) = 0, \quad \frac{\partial F}{\partial x}(x, y) = 0,$$

tedy v našem konkrétním případě soustavu

$$\begin{aligned}(s_{11}x + s_{21}y + s_{31})^2 + (s_{12}x + s_{22}y + s_{32})^2 &= 1,\\ 2s_{11}(s_{11}x + s_{21}y + s_{31}) + 2s_{12}(s_{12}x + s_{22}y + s_{32}) &= 0,\end{aligned}$$

kterou je asi nejlepší řešit substitucí

$$\begin{aligned}u &= s_{11}x + s_{21}y + s_{31},\\ v &= s_{12}x + s_{22}y + s_{32},\end{aligned}$$

(všimněme si, že vlastně $(u, v, 1) = (x, y, 1)S$) takže dostáváme soustavu

$$u^2 + v^2 = 1, \quad s_{11}u + s_{12}v = 0,$$

jež má dvě řešení

$$(u_{1,2}, v_{1,2}) = \left(\pm\frac{s_{12}}{\sqrt{b_{11}^2 + b_{12}^2}}, \mp\frac{s_{11}}{\sqrt{b_{11}^2 + b_{12}^2}}\right).$$

Řešení $(x_{1,2}, y_{1,2})$ již získáme jednoduše jako

$$(x_{1,2}, y_{1,2}, 1) = (u_{1,2}, v_{1,2}, 1)T$$

Dostáváme tak hodnoty y_1, y_2, což jsou y–ové souřadnice vrcholů ohraničujícího obdélníka.

Záměnou symbolů x a y dostáváme podobně x–ové souřadnice vrcholů ohraničujícího obdélníka.

4. Programátorská dokumentace

4.1. Použité programové prostředky

Aplikace je vytvořena pomocí Visual Studia 2010, v jazyce C++ s využitím knihovny *Microsoft Foundation Classes* (dále jen MFC). Zmiňme se v krátkosti o principech programování ve Windows.

Aplikace ve Windows používají událostmi řízený programovací model. Aplikace reagují na tzv. *události* zpracováváním zpráv, které jí zasílá většinou operační systém. Událostmi jsou např. stisk klávesy na klávesnici, pohyb myši, potřeba aktualizovat obsah okna aplikace, apod. Program ve Windows má svůj `WinMain()` (obdoba klasického `main()` z jazyka C). Nejvíce práce ovšem vykonává funkce `WndProc` – tzv. *procedura okna*, která provádí obsluhu *zpráv*. Procedura okna zpracovává zprávy zasílané příslušnému oknu, přitom volá buď lokální funkce příslušné aplikace nebo funkce rozhraní API, jež poskytuje systém Windows. `WinMain` vytvoří okno a následně vstoupí do *cyklu zpráv* nebo vyhledá zprávy a odešle je do procedury okna. Zprávy čekají na své zpracování ve frontě zpráv. Cyklus zpráv je ukončen zpracováním zprávy WM_QUIT, což má za následek i ukončení aplikace.

Nyní si můžeme krátce představit MFC. Je to objektově orientovaná knihovna zapouzdřující základní strukturu programů ve Windows a funkce rozhraní WinAPI.

Obsahuje obrovské množství tříd různé složitosti. Např. třída `CWnd` zapouzdřuje funkčnost okna. Až na výjimky provádí zapouzdření funkcí API. Kromě toho je MFC i *aplikačním rámcem* – napomáhá při definování struktury aplikace. Zejména jde o třídu `CWinApp`, která reprezentuje celou aplikaci – program v MFC obsahuje jedinou instanci objektu třídy jejímž předkem je právě `CWinApp`. Není třeba zmiňovat, že dalšími výhodami MFC jsou právě výhody objektově orientovanému programování. Zejména jde o dědičnost – díky níž programátor využije funkčnosti již existujících tříd zapouzdřující nejrůznější prvky aplikace – např. nabídky, lišty nástrojů, okna vlastností, práce se soubory, výstupními zařízeními jako třeba s monitorem či tiskárnu, kontejnery apod.

Posledním principem MFC jež stojí za zmínku, je architektura dokument/pohled, která definuje strukturu programu. Opírá se o objekt *dokument*, který uchovává data aplikace (tvoří vlastně datovou a částečně logickou vrstvu aplikace) a objekt *pohled*, který zprostředkovává pohledy na tyto data a většinou také zajišťuje komunikaci s uživatelem (tvoří vlastně prezentační a částečně logickou vrstvu aplikace). Výhoda tohoto přístupu je zřejmá – jde oddělení datové části od prezentační.

Tento text byl zpracován z [3], kde může čtenář najít téměř všechny odpovědi na základní otázky týkající se MFC.

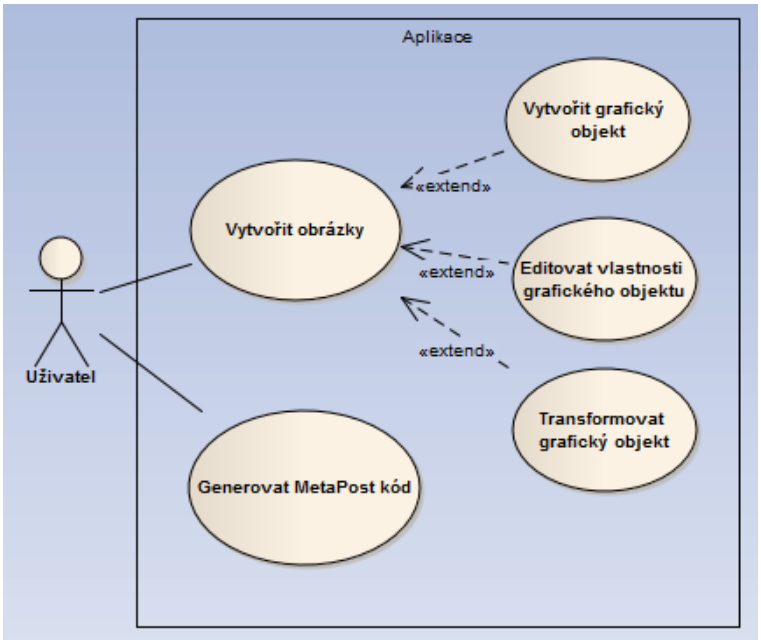
4.2. Návrh architektury aplikace

V této kapitole je prezentován diagram případů užití a diagram tříd.

Nejprve popišme funkční požadavky na systém. Aktér je jediný – jde o *uživatele* aplikace. Případy užití jsou následující:

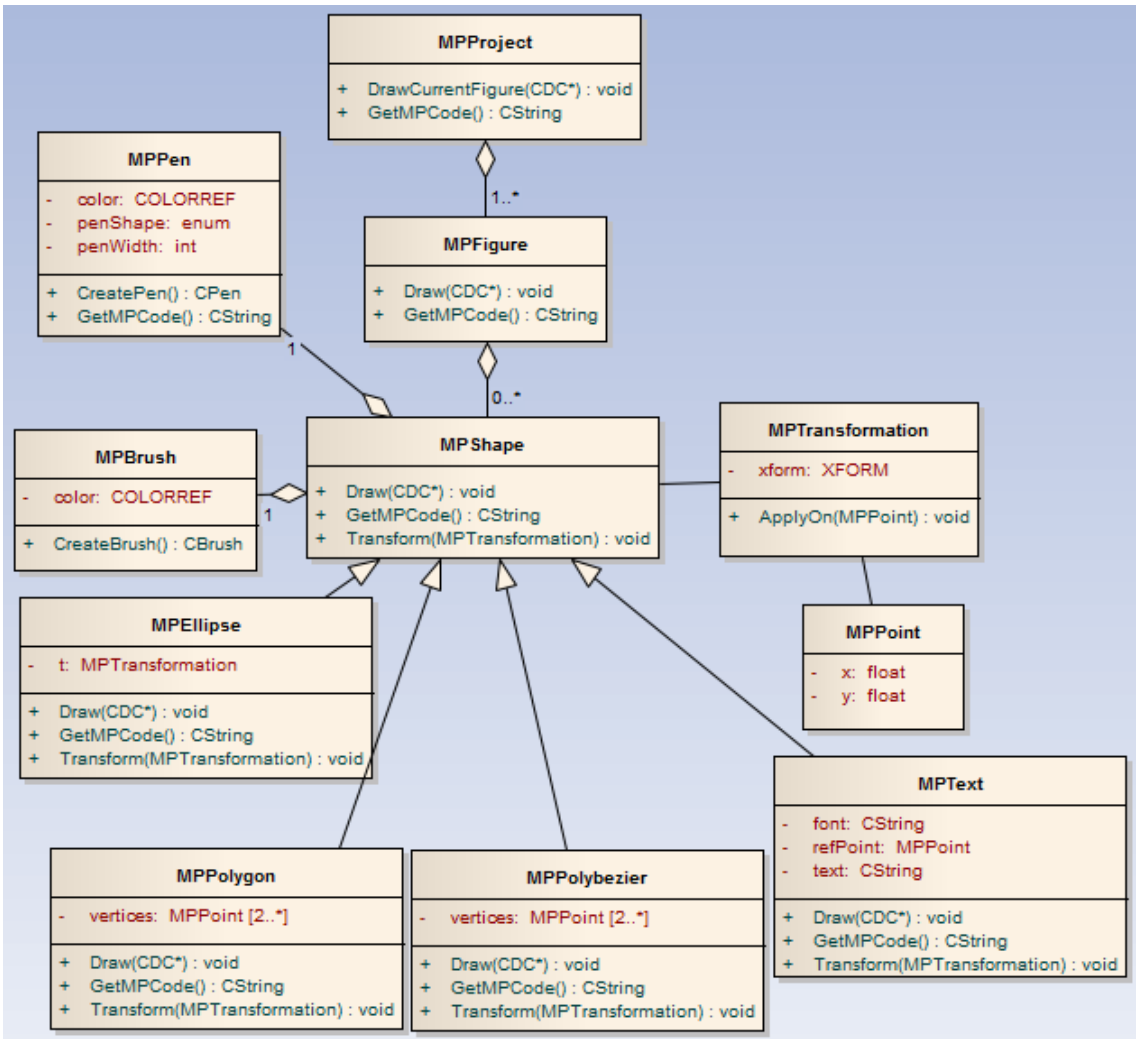
- Vytvořit obrázky. Tento případ užití rozšiřují
  - Vytvořit grafický objekt.
  - Editovat vlastnosti graf. objektu.
  - Transformovat graf. objekty.
- Generovat kód v METAPOSTu.

Diagram případů použití je zobrazen na obrázku 2.



Obrázek 2. Diagram případů užití

V následujícím diagramu tříd je zobrazena grafická knihovna aplikace – z důvodu velkého množství atributů a metod jsou zobrazeny jen ty nejdůležitější - viz obrázek 3.



Obrázek 3. Diagram tříd grafické knihovny.

4.3. Popis důležitých tříd

V této kapitole popíšeme nejdůležitější třídy a jejich nejdůležitejší atributy a metody. Objevují se zde třídy a struktury specifické pro WinAPI a MFC – výběr potřebných struktur a tříd pro potřeby tohoto textu je umístěn v Příloze B.

4.3.1. Třída MPPoint

Jednoduchá třída mající dva veřejně přístupné atributy, čímž je x–ová a y–ová souřadnice bodu typu FLOAT (číslo v plovoucí řádové čárce).

4.3.2. Třída MPPen

Zapouzdřuje atributy pera, kterými jsou vykreslovány body, elipsy, polygony

a Bézierovy křivky.

4.3.3. Třída MPBrush

Zapouzdřuje atributy štětce, který se používá k vyplnění uzavřených polygonů, Bézierových křivek a elips.

4.3.4. Třída MPTransformation

Představuje objekt afinní transformace. Její jediný atribut je struktura typu `XFORM`. Ty reprezentují koeficienty afinní transformace.

4.3.5. Třída MPShape

Přestavuje abstrakci nejobecnějšího grafického objektu. Je dědící třídou třídy `CObject` – a to kvůli *serializaci*, tj. způsobu ukládání dokumentu s využitím knihoven v MFC.

- Atributy
  - `MPPen pen` – pero použité k vykreslování objektu,
  - `MPBrush brush` – štětec použitý k vyplnění objetku – jde–li o vyplnitelný objekt
- Veřejné metody
  - `virtual void Draw(CDC*)` – vykreslí objekt do kontextu zařízení na které ukazuje argument funkce.
  - `virtual CString GetMPCode()` – vygeneruje kód METAPOSTu.
  - `virtual void Transform(MPTransformation&)` – transformuje daný objekt pomocí afinní transformace.
  - `virtual MPShape* Clone()` – vytvoří na haldě kopii grafického objektu a vrátí ji ve své návratové hodnotě.

4.3.6. Třída MPEllipse

Je potomkem třídy `MPShape`. Reprezentuje elipsu. Jejím hlavním parametrem je atribut třídy `MPTransformation`, který vyjadřuje afinní transformaci, která je deformuje jednotkouvou kružnici na elipsu.

4.3.7. Třída MPPolyBezier

Je potomkem třídy `MPShape`. Jejím hlavním atributem je pole objektů typu `MPPoint` reprezentující řídící body.

4.3.8. Třída MPPolygon

Je potomkem třídy `MPShape`. Jejím hlavním atributem je pole objektů třídy typu `MPPoint` reprezentující vrcholy polygonu.

4.3.9. Třída MPText

Je potomkem třídy `MPShape`. Jejími hlavními atributy jsou řetězce `font`, `text`, které udávají font a obsah textového popisku.

4.3.10. Třída CMPEditorDoc

Tato třída zapouzdřuje datovou a částečně logickou vrstvu aplikace. Jejím hlavním objektem je ukazatel na objekt třídy `MPProject`. Vzhledem k velkému množství veřejných metod zde uvedeme pouze *odpovědnosti* této třídy:

- správa dat aplikace: podávání informací o stavu dat, rozhraní pro manipulaci s daty,
- uložení/načtení dat do/ze souboru,
- správa *undo*/*redo* funkcí,
- správa interní schránky (clipboard).

4.3.11. Třída CMPEditorView

Tato třída zapouzdřuje prezentační a částečně logickou vrstvu aplikace. Jejím úkolem je zobrazování dat *dokumentu* a interakce s uživatelem (s operačním systémem) – obsluha zpráv. Opět vzhledem k rozsáhlosti této třídy zde uvedeme pouze její *odpovědnosti*:

- komunikace s uživatelem: obsluha zpráv vyvolaných akcí uživatele, zobrazení výsledku na pracovní ploše,
- správa a vykreslování označení objektů (pomocí chráněných metod),
- komunikace s ovládacími prvky: *Figure View* a *Properties*.

4.4. Typ dat a způsob jejich uložení

Program pracuje se soubory s příponou `mped`, které obsahují hodnoty atributů příslušné instance třídy `MPProject`. Tato třída vlastně představuje typ dat aplikace. Odtud plyne, že každý soubor typu `mped` se skládá z několika obrázků.

Data jsou ukládána v binární formě mechanismem *serializace* podporovaným MFC. Pomocí něj lze jednoduše ukládat a načítat data objektů nejrůznějších tříd. K tomu je jen potřeba, aby *serializovaná* třída měla jako svého předka třídu `CObject` a přepsala její virtuální metodu `Serialize`.

5. Uživatelská příručka

5.1. Instalace

Aplikace je určena pro PC s operačními systémy Windows XP nebo Windows 7. Instalace spočívá ve spuštění instalačního souboru `MPEditorSetup.exe`. Ten vytvoří v příslušné složce dva soubory:

- `MPEditor.exe` aplikace,
- `MPEditor.chm` soubor nápovědy k aplikaci.

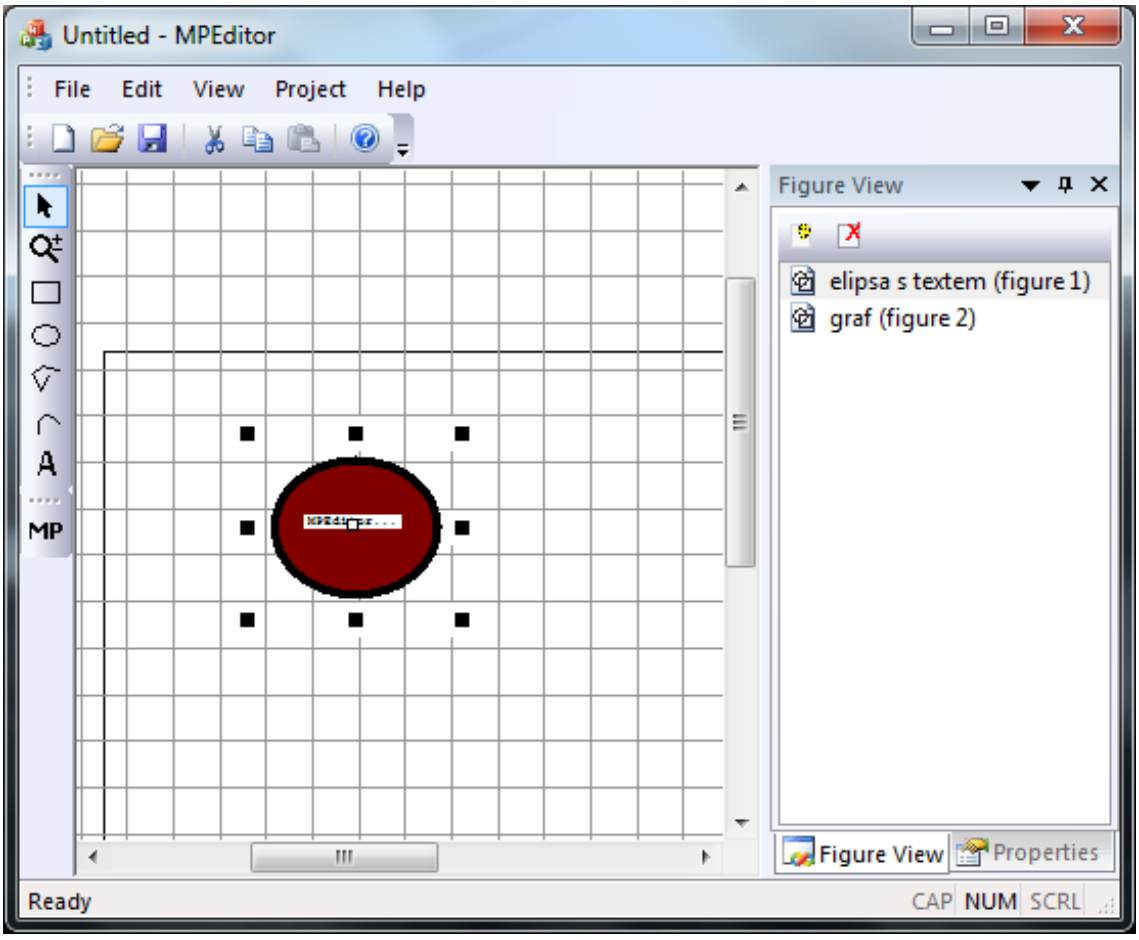
5.2. Popis GUI

Grafické rozhraní je standardní – snaží se napodobovat klasické grafické editory. Skládá se z hlavní nabídky (*File*, *Edit*, ...), tří nabídek nástrojů (standardní nabídka, nabídka grafických nástrojů, nabídka pro generování `mp`–souborů). Dále obsahuje pracovní okno, okno vlastností objektů a okno obsahující seznam obrázků. Příklad vzhledu aplikace je na obrázku 4. Vzhled se dá dosti podstatným způsobem jednoduše měnit podle potřeby.

5.3. Funkčnost aplikace

Program poskytuje základní funkce grafického vektorového editoru. Zejména jde o

- otevření souboru s obrázky a jejich uložení,
- standardní nástroje pro manipulaci – funkce *undo*, *redo*, *copy*, *cut*, *paste*, *remove*,
- možnost přizpůsobení vzhledu svým potřebám, zobrazení pomocné mřížky, stavového řádku,
- možnost přidávání a mazání obrázků,
- vytváření grafických objektů (obdélník, elipsa, polygon, Bézierova křivka, popisky/text), jejich transformace a editace vlastností,
- generování souborů zdrojového kódu jazyka METAPOST,
- nápovědu.



Obrázek 4. Vzhled okna aplikace

5.4. Práce s aplikací

Okno aplikace se skládá z několika částí, jejichž rozložení lze podle potřeby poměrně dost měnit. Na obrázku 4. je příklad vzhledu okna aplikace. Skládá se z hlavního menu, nabídek, pracovního okna a dokovacích oken.

GUI obsahuje následující ovládací prvky dosažitelné pomocí položky *Menu→View→Toolbars and Docking Windows*:

Lišta *Standard* obsahuje standardní tlačítka: Nový, Otevřít, Uložit, Vyjmout, Kopírovat, Vložit, O programu…, viz obrázek 5.

Lišta Graphical obsahuje tlačítka pro určení módu: mód výběr/transformace, lupa, vytvoření obdélníku, elipsy, polygonu, Bézierovy křivky a popisku (text), viz obrázek 6.

Lišta *MetaPost Toolbar* obsahuje jediné tlačítko pro generování MetaPost kódu,

Obrázek 5. Lišta *Standard*.

Obrázek 6. Lišta *Graphical*.

viz obrázek 7.

Dokovací okno *Properties* určené k zobrazení a editaci vlastností jednotlivých grafických objektů, viz obrázek 8.

Dokovací okno *Figure View* sloužící k zobrazení seznamu obrázků daného souboru, jejich vytváření, mazání a editaci popisku, viz obrázek 9.

Dokument aplikace MPEditor se skládá z obrázků. Nový dokument obsahuje pouze jeden obrázek. Mezi jednotlivými obrázky je možno se přepínat pomocí ovládacího prvku *Figure View*.

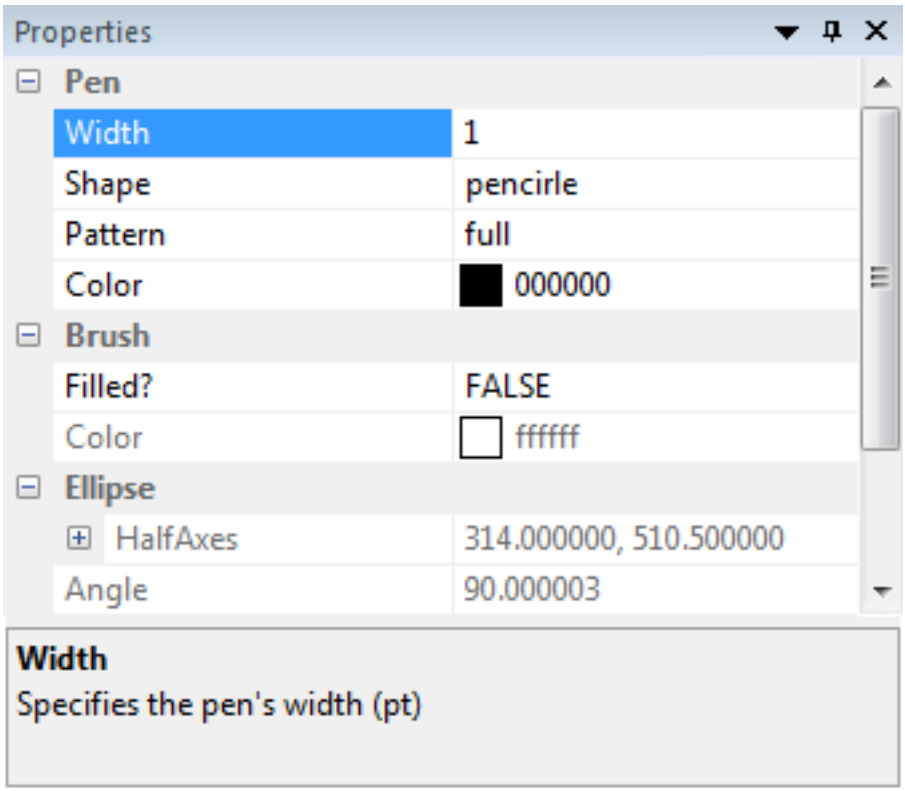
5.4.1. Práce s obrázky

Jak již bylo řečeno, dokument aplikace MPEditor se sestává z několika obrázků (`figures`). Je to dáno strukturou `mp`–souboru. Hlavním nástrojem k manipulaci s obrázky je dokovací okno *Figure View*, které lze nalézt buď na pracovní ploše aplikace, nebo pokud je vypnuté, lze jej zapnout v *Menu→View→Toolbars and Docking Windows*. Toto okno se skládá ze dvou tlačítek: vytvořit nový obrázek, vymazat aktuálně zobrazený obrázek, přičemž největší část okna zabírá samotný seznam obrázků, viz obrázek 9.

Kliknutím myši na jednotlivé řádky se příslušné obrázky stávají aktivními – tzn. jejich obsah se zobrazí na pracovní plochu aplikace a lze jej měnit. Obrázky lze popisovat – stačí na danou **již vybranou** položku v seznamu kliknout a pomocí klávesnice editovat název obrázku. Konec editace se provede stisknutím tlačítka `Enter`. Každý nově vytvořený obrázek je nepojmenovaný. Jméno obrázku slouží pouze k lepší orientaci uživatele.

Kromě toho lze vytvářet nový a mazat existující obrázek v *Menu→Project*.

Obrázek 7. Lišta *MetaPost Toolbar*.



Obrázek 8. Okno *Properties*.

5.4.2. Vytváření grafických objektů

V aplikaci je možno vytvářet následující grafické objekty: obdélníky, elipsy, polygony, Bézierovy křivky, textové popisky. Na začátku vytváření je potřeba kliknout na příslušnou ikonu v liště *Graphical*. Po vytvoření se aktualizuje okno *Properties*, ve kterém lze editovat jednotlivé vlastnosti daného objektu.

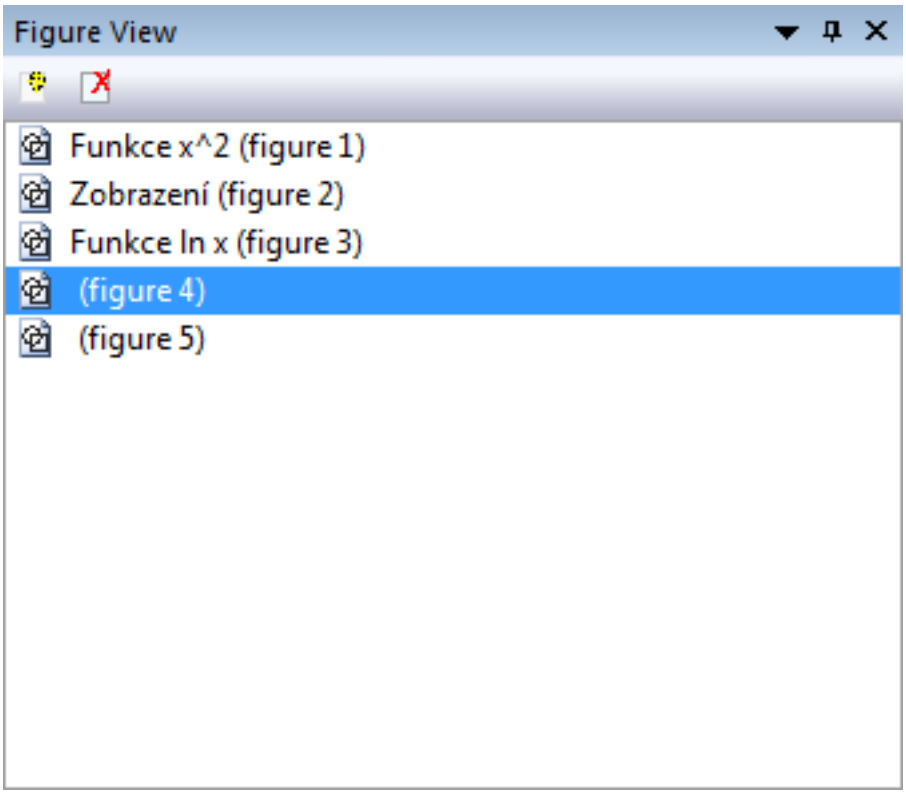
Obdélník Tvorba obdélníků se skládá ze tří fází:

- Stisknutí levého tlačítka na místě, kde má být jeden vrchol obdélníka,
- při stisknutém levém tlačítku pohyb myší,
- na místě, kde má být umístěn diagonální vrchol obdélníka je levé tlačítko uvolněno.

Elipsa Tvorba elipsy se skládá ze stejných fází jako vytvoření obdélníka.

Polygon je tvořen následujícím postupem:

1. Kliknutí levého tlačítka na místo prvního vrcholu polygonu,



Obrázek 9. Okno *Figure View*.

2. pohyb myší na k dalšímu vrcholu,

3. opakují se kroky 1 a 2, přičemž poslední vrchol se zadá dvojklikem levého tlačítka.

Bézierova křivka Ve skutečnosti jde o křivku, která je složena z Bézierových křivek (budeme se na ně zde odkazovat jako na *segmenty*. Její tvorba je trochu složitější na popis, ale jinak velmi intuitivní:

1. Vytvoření prvního bodu a prvního kontrolního bodu prvního segmentu křivky:

   - Stisknutí levého tlačítka myši na místě prvního bodu křivky,
   - při stisknutém levém tlačítku pohyb myší na místo, kde má být první kontrolní bod Bézierovy křivky,
   - uvolnění levého tlačítka.

   V této chvíli je vytvořen první bod a první kontrolní bod prvního segmentu.

2. Dokončení i–tého segmentu křivky a začátek $(i+1)$–ního segmentu křivky:

- Stisknutí levého tlačítka myši na místě koncového bodu i–tého segmentu,
- při stisknutém levém tlačítku pohyb myší na místo, kde má být první kontrolní bod $(i+1)$–ního segmentu,
- uvolnění levého tlačítka (tím je dán první kontrolní bod $(i+1)$–ního segmentu a z něho i druhý kontrolní bod i–tého segmentu.

3. Ukončení tvorby křivky se provede dvojklikem levého tlačítka – tím se také zruší právě vytvářený segment.

Text Vytvoření se sestává pouze z kliknutí na místo, kde má být umístěn text (přesněji řečeno: na referenční bod). Vytvoří se popisek s nápisem „Text". Ten je možno upravit v okně vlastností *Properties*.

5.4.3. Označování grafických objektů

V této kapitole se setkáme s pojmem *ohraničující obdélník* daného grafického objektu. Jde o nejmenší možný obdélník (se stranami rovnoběžnými se „souřadnými osami obrazovky"), který pokrývá tento objekt.

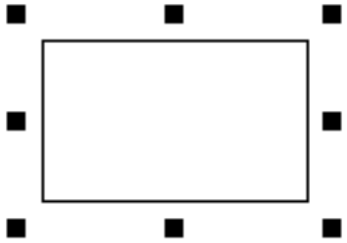
Označit objekty lze celkem třemi způsoby:

- právě vytvořený objekt se automaticky označí,
- v režimu výběru (tzn. při stisknutém tlačítku *výběr/transformace*) kliknutím na objekt (lépe řečeno: na jeho ohraničující obdélník) se označí pouze **jeden** objekt. Pokud je více objektů „na sobě", označí se ten úplně „nahoře",
- v režimu výběru (tzn. při stisknutém tlačítku *výběr/transformace*) označením obdélníkové části pracovní plochy okna (stejným postupem jako vytvoření obdélníka) se označí všechny objekty, které s touto částí mají neprázdný průnik.

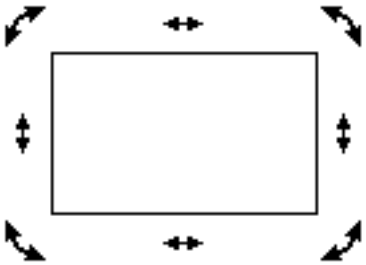
Je vidět, že v programu je způsob označování poněkud těžkopádný. Odtud vyvstává problém, že pokud máme dva objekty „na sobě", můžeme označit **pouze** ten horní. K tomu je určen příkaz *Push back*, který označený objekt „umístí úplně dozadu". Je dostupný z *Menu→Edit* nebo z kontextového menu, které se zobrazí kliknutím pravého tlačíta myši na pracovní plochu.

5.4.4. Transformace grafických objektů

K tomu je určen nástroj *výběr/transformace*. Tlačítko je umístěno na liště *Graphical* na prvním místě. Transformovat lze pouze označené objekty. Jakmile je objekt, nebo skupina objektů označená, můžeme s ní provádět různé afinní



Obrázek 10. Výběr v módu škálování (změna měřítka).



Obrázek 11. Výběr v módu rotace/zkosení.

transformace. Jsou celkem dva módy transformací: škálování a rotace/zkosení – viz obrázek 10. a 11.

Mezi těmito módy se lze přepínat kliknutím na označené objekty.

Mód škálování Používá ke změně měřítka objektu v jedné nebo obou směrech souřadných os. A to pomocí aktivních čtverců – pomocí čtverců ve vrcholech se provádí změna měřítka v obou směrech, pomocí ostatních jen v jednom směru.

Mód rotace/zkosení V tomto módu lze pomocí aktivních „šipek" ve vrcholech označení rotovat objekty okolo jejich společného středu. Ostatní aktivní šipky slouží ke zkosení označených objektů.

5.4.5. Mazání grafických objektů

Označené objekty se smažou pomocí tlačítka `Delete`.

5.4.6. Vlastnosti grafických objektů

Objekty vytvářené v MPEditoru mají různé vlastnosti zejména jde o vlastnosti *pera*, kterým je objekt vykreslován a *štětce*, kterým je objekt vyplňován (pokud je tento objekt uzavřený) – to platí pro všechny objekty s výjimkou

textu. Následuje seznam vlastností pera, štětce a specifických vlastností příslušných objektů, které lze editovat:

Pero (Pen)

- *Šířka (Width)* – šířka pera v `pt`. Celočíselná hodnota mezi 1 a 30.
- *Tvar (Shape)* – tvar pera. Hodnoty jsou buď `pencircle` nebo `pensquare`.
- *Pattern (Vzor)* – přerušovaná čára. Hodnoty jsou `full` (plná čára), `evenly` (přerušovaná čára), `withdots` (tečkovaná čára).
- *Barva (Color)* – barva pera.

Štětec (Brush)

- *Vyplněn? (Filled?)* – zda objekt je či není vyplněn.
- *Barva (Color)* – barva výplně.

Polygon, Bézierova křivka

- *Uzavřený? (Closed?)* – zda je polygon, křivka uzavřený. Pokud je hodnota `TRUE` (pravda), je ve vlastnostech zobrazena vlastnost *Štětec*.
- *Počáteční šipka? (Beginning arrow?)* – zda křivka začíná šipkou, nebo ne.
- *Koncová šipka? (Ending arrow?)* – zda křivka končí šipkou, nebo ne.

Text

- *Text (Text)* – řetězec, udávající zobrazovaný text.
- *Font (Font)* – použitý font (implicitně je hodnota nastavena na `"cmr10"`).
- *Směr posunutí (Direction of shift)* – směr posunutí text od referenčního bodu (možné hodnoty jsou: žádné posunutí (`Center`), doleva (`Left`), doprava, nahoru, dolů, doleva–nahoru, doleva–dolů, doprava–nahoru, doprava–dolů).
- *Vzdálenost (Distance)* – velikost posunutí textu od referenčního bodu v 0.1 `mm`.
- *Nakreslit tečku? (Draw dot?)* – zda kreslit tečku na místě referenčního bodu.

Vlastnosti lze editovat v okně *Properties*.

5.4.7. Generování kódu v Metapostu

Jakmile máme objekty nakreslené, stačí zmáčknout tlačítko *MetaPost*, které je jediným tlačítkem lišty *Metapost Toolbar*. Poté se zobrazí standardní okno pro uložení souboru, přičemž jako implicitní přípona souboru je dána nastavena na `mp`.

5.4.8. Nástroj lupa

Druhou položkou na liště *Graphical* je nástroj *Lupa*. Po výběru tohoto nástroje lze „přibližovat" objekty pracovního okna kliknutím levého tlačítka myši a „oddalovat" kliknutím pravým tlačítkem myši na pracovní plochu.

5.4.9. Ukládání/načítání dokumentu

Jednotlivé dokumenty aplikace MPEditor jsou uloženy v souborech s příponou `mped`. Ukládání a načítání je naprosto standardní jako u většiny podobných aplikací pod Windows.

Závěr

Cílem této práce bylo vytvořit grafický vektorový editor k vytváření jednoduchých obrázků pomocí systému METAPOST. Hlavní funkcí tohoto programu je generování zdrojového textu pro zpracování překladačem jazyka METAPOST. Obsahuje standardní nástroje, jako je vytváření obdélníků, elips, polygonů, Bézierových křivek, textu, jejich transformace a editaci jejich vlastností (pero, výplně). Aplikace samozřejmě podporuje základní funkce jako např. *copy*, *paste*, *undo*, *redo*.

Jak již bylo zmíněno, program neobsahuje některé speciality, které by umožnily pohodlnější vytváření obrázků. Nicméně se domnívám, že předložená aplikace tvoří slušný základ pro případná další rozšíření funkčnosti (např. editace vrcholů polygonů, další grafické objekty, další vlastnosti grafických objektů, přímé vytváření obrázků).

Conclusions

The main aim of this work is graphical vector editor for creation of simple figures using of system METAPOST. The main function of this application is the generation of input text for processing by compiler of the programming language METAPOST.
The application includes standard tools for creation of rectangles, ellipses, polygons, Bézier curves, text, their transformations and editing of their properties (pen, brush). It supports basic functions as *copy*, *paste*, *undo* or *redo* actions.

As mentioned previously, the application doesn't contain certain specialities, which could enable more comfortable tools for creating of figures. Althought, I suppose that the submitted program is a solid base for additional extensions of functionality (e.g. editing of vertices in polygons, mor graphical types of objects, more properties of graphical objects, direct creation of figures).

Reference

[1] Hobby, John D. *Metapost - a User's Manual.*

[2] Eckel, Bruce. *Myslíme v jazyku C++ : knihovna programátora.* Grada, Praha, 2006.

[3] Prosise, Jeff. *Programování ve Windows pomocí MFC.* Computer Press, Praha, 2000.

[4] Janyška, Josef, Sekaninová, Anna. *Analytická teorie kuželoseček a kvadrik.* Masarykova univerzita, Přírodovědecká fakulta, Brno, 2001.

[5] Jarník, Vojtěch. *Diferenciální počet I.*, ČSAV, Praha, 1955.

[6] Krupka, Michal. *Geometrie pro informatiky.* Elektronické skriptum, 2008.

[7] Drdla, Josef. *Počítačová grafika.* Univerzita Palackého, Olomouc, 1991.

A. Příloha

Nyní ukážeme jeden ze způsobů jak vkládat vytvořené obrázky do LaTeXovských dokumentů.

- Pomocí editoru vytvoříme `mp`–soubor, např. `obrazky.mp`. Na příkazovou řádku napíšeme

  ```
  mpost obrazky.mp
  ```

  V pracovním adresáři se tímto vytvoří soubory `obrazky.1`, `obrazky.2`, ...

- Používáme–li např. `cslatex.exe`, v LaTeXovském dokumentu bude následující: Hlavička bude obsahovat řádky

  ```
  \usepackage{epsf}

  \DeclareGraphicsRule {*} {mps} {*} {}
  ```

  a pro vložení prvního obrázku, pak stačí na příslušném místě napsat třeba:

  ```
  \begin{figure}
  \begin{center}\epsfbox{obrazky.1}\end{center}
  \caption{Můj první obrázek.} \label{fig:obr1}
  \end{figure}
  ```

  Přitom soubory s obrázky by měly být ve stejném adresáři jako LaTeXovský soubor.

B. Vybrané struktury a třídy knihovny MFC

V této části přílohy zhruba popíšeme některé sturktury a objekty z WinAPI a MFC, o kterých byla zmínka v textu.

CString zapouzdřuje *řetězec*. Má spoustu užitečných metod k manipulaci s řetězci.

COLORREF struktura nesoucí informaci o barvě.

XFORM struktura nesoucí informaci o afinní transformaci.

CDC je třída MFC. Jde o zkratku *Device Context* (česky: *kontext zařízení*) zapouzdřující zařízení, do kterého lze kreslit (obrazovka, okno, tiskárna).

CPen je třída MFC. Třída zapouzdřující pero, kterým se vykreslují čáry a jiné grafické objekty.

CBrush je třída MFC. Zapouzdřuje štětec, kterým se vyplňují uzavřené oblasti.

C. Obsah přiloženého CD

V samotném závěru práce je uveden stručný popis obsahu přiloženého CD/DVD, tj. závazné adresářové struktury, důležitých souborů apod.

`bin/`
: Adresář obsahuje instalační soubor `MPEditorSetup.exe`.

`doc/`
: Dokumentace práce ve formátu PDF, vytvořená dle závazného stylu KI PřF pro diplomové práce, včetně všech příloh, a všechny soubory nutné pro bezproblémové vygenerování PDF souboru dokumentace (v ZIP archivu), tj. zdrojový text dokumentace, vložené obrázky, apod.

`src/`
: Kompletní zdrojové texty programu MPEditor se všemi potřebnými (převzatými) zdrojovými texty, knihovnami a dalšími soubory pro bezproblémové vytvoření spustitelných verzí programu (v ZIP archivu).

`readme.txt`
: Instrukce pro instalaci a spuštění programu MPEditor, včetně požadavků pro jeho provoz.

Navíc CD/DVD obsahuje:

`data/`
: Ukázková a testovací data použitá v práci a pro potřeby obhajoby práce.

U veškerých odjinud převzatých materiálů obsažených na CD/DVD jejich zahrnutí dovolují podmínky pro jejich šíření nebo přiložený souhlas držitele copyrightu. Pro materiály, u kterých toto není splněno, je uveden jejich zdroj (webová adresa) v textu dokumentace práce nebo v souboru `readme.txt`.