VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

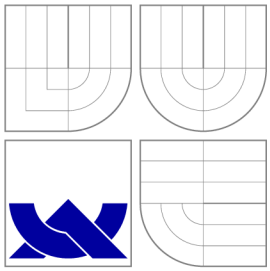
VÝUKOVÝ PROGRAM PRO DEMONSTRACI GENEROVÁNÍ 2D OBJEKTŮ V RASTRU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

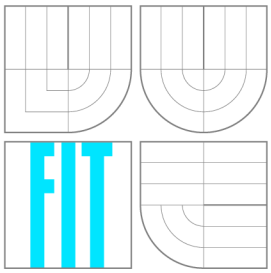
AUTOR PRÁCE
AUTHOR

MARTIN ŠŮS

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VÝUKOVÝ PROGRAM PRO DEMONSTRACI GENEROVÁNÍ 2D OBJEKTŮ V RASTRU

EDUCATION COMPUTER PROGRAM FOR DEMONSTRATION OF 2D OBJECTS GENERATION IN RASTER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN ŠŮS

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PŘEMYSL KRŠEK, Ph.D.

BRNO 2008

Zadání bakalářské práce/5357/2007/xsusma00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Šůs Martin**
Obor: Informační technologie
Téma: **Výukový program pro demonstraci generování 2D objektů v rastru**
Kategorie: Počítačová grafika

Pokyny:

1. Prostudujte problematiku generování 2D vektorových objektů v rastru
2. Prostudujte problematiku tvorby výukových a demonstračních programů
3. Navrhněte výukový program pro demonstraci generování 2D vektorových objektů v rastru
4. Implementujte navržený program ve vybraném jazyce (C/C++, Java, Python, C#)
5. Ve vytvořeném programu implementujte popsané principy a algoritmy
6. Zhodnoťte dosažené výsledky a stanovte další vývoj projektu

Literatura:

- Žara J., Beneš B., Felkel P.: Moderní počítačová grafika. 1. vyd. Praha, Computer press 1998, 448 s., ISBN 80-7226-049-9

Při obhajobě semestrální části projektu je požadováno:

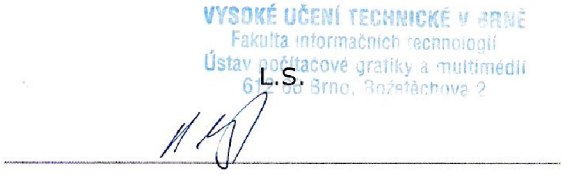
- Splnění prvních 3 bodů zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kršek Přemysl, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008



L.S.

doc. Dr. Ing. Pavel Zemčík
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Martin Šůs**
Id studenta: 78715
Bytem: Hálova 1081/8, 851 01 Bratislava
Narozen: 22. 01. 1986, Bratislava
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Výukový program pro demonstraci generování 2D objektů v rastru
Vedoucí/školitel VŠKP: Kršek Přemysl, Ing., Ph.D.
Ústav: Ústav počítačové grafiky a multimédií
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:

tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Abstrakt

V prvej, teoretickej časti sú diskutované rozdiely medzi rastrovou grafikov a vektorovou grafikou. Spomenuté sú ich klady a zápory a je vysvetlená potreba prevodu medzi týmito dvoma typmi grafík. Proces rasterizácie, prevodu z vektorovej do rastrovej grafiky, je dôkladne popísaný pre vybrané základné grafické prvky/entity. Druhá časť opisuje proces vývoju a implementáciu aplikácie.

Klíčová slova

vektorová grafika, rastrová grafika, rasterizácia, úsečka, kružnica, elipsa, Bézierova kubika

Abstract

There are discussed the differences between raster graphic and vector graphic in the first, the theoretical part of the thesis. Their pros and cons are mentioned and the importance of conversions between this two types of graphics is explained. Process of rasterization, conversion from vector graphic to raster graphic, for selected graphic primitives/entities is thoroughly described. The second part describes the process of development and implementation of the application.

Keywords

vector graphic, raster graphic, rasterization, line segment, circle, ellipse, Bezier cubic

Citace

Martin Šůs: Výukový program pro demonstraci generování 2D objektů v rastru, bakalářská práce, Brno, FIT VUT v Brně, 2008

Výukový program pro demonstraci generování 2D objektů v rastru

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Přemysla Krška, Ph.D.

.....
Martin Šůs
11. května 2008

Poděkování

Chtěl bych poděkovat vedoucímu své bakalářské práce, panu Ing. Přemyslu Krškovi, Ph.D., za odbornou pomoc, rady a podnětné připomínky při tvorbě tohoto projektu.

Obsah

Kapitola 1

Úvod

Táto práca sa zaoberá procesom, ktorý vedie k vytvoreniu výučbového program pre demonštráciu generovania 2D objektov v rastre. Jedná sa o problematiku uchovávania, zobrazovania a prevodu grafickej informácie, ktorá môže byť buď v rastrovej alebo vo vektorovej podobe. Výsledný program by mal byť schopný ponúknuť svojmu užívateľovi teoretické základy týkajúce sa danej témy a takisto aj nástroj na precvičenie a oboznámenie sa v praxi s rasterizačnými algoritmami.

V prvej kapitole si podrobne rozoberieme túto problematiku z teoretického hľadiska. Popíšeme si výhody a nevýhody jednotlivých typov grafík. Ďalej sa budeme venovať potrebe prevodu medzi týmito typmi grafík. Následne pristúpime priamo k rozboru vybraných grafických entít. Samotným cieľom je v tejto kapitole popis vytvorenia rasterizačných algoritmov pre tieto entity, až do ich finálnej podoby.

Ďalšia kapitola bude zameraná na dve podstatné časti tvorby aplikácie, ktorými sú návrh a implementácia. V prvej časti si popíšeme postup, ktorým sme dospeli k návrhu aplikácie a skutočnosti, ktoré nás pri tom ovplyvňovali a na ktoré sme museli reagovať. Druhá časť popisuje prostriedky a spôsob implementácie vychádzajúc z návrhu. Dôraz je pritom kladený na všeobecnejší postup tvorby komponentov aplikácie, než priamo na popis kódu.

V posledných dvoch kapitolách sa venujeme výsledkom dosiahnutým pri tvorbe aplikácie. Zhodnotíme jej prínos a prípadné ďalšie rozšírenia.

Kapitola 2

Rozbor problematiky

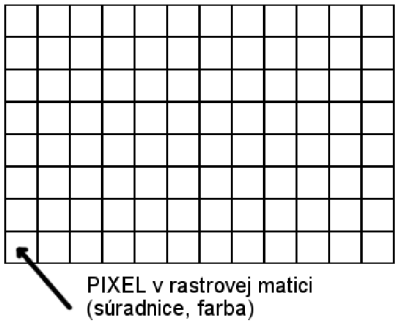
Neoddeliteľnou súčasťou návrhu a vývoju zadaného výučbového a demonštračného programu je teoretický rozbor látky, ktorou sa aplikácia zaoberá. V tejto kapitole budeme oboznámený s teoretickými základmi vektorovej a rastrovej grafiky, s dôležitými pojmami z týchto oblastí a s niektorými ich matematickými vlastnosťami a definíciami. Ďalej budú popísané postupy prevodov vybraných vektorových entít do rastru a bude vysvetlená potreba takýchto prevodov.

Počítačová grafika je dôležitou oblasťou v informatike. Zaoberá sa získavaním, analýzou, modelovaním a zobrazovaním grafických dát. V tejto práci sa zameriame na grafiku 2D. V počítačovej grafike rozlišujeme dva spôsoby uchovávania grafických dát. Každý z týchto prístupov má svoje klady, ale aj zápory. Týmito spôsobmi, ako už bolo naznačené, budeme rozumieť *rastrovú* a *vektorovú grafiku*.

2.1 Rastrová grafika

Základným kameňom rastrovej grafiky je *pixel*. Grafická informácia je uložená v *rastrovej matici* zloženej práve z pixlov. Každý pixel predstavuje bod určený diskrétnymi súradnicami a farbou tohoto bodu.[5]

Rastrová grafika sa využíva predovšetkým na grafický výstup (zobrazovanie grafických dát). Na tomto princípe sú založené dnešné zobrazovacie prístroje (monitory, projektory, atď.), ale takisto aj prístroje na záznam (kamera, fotoaparát, atď.).
Takýto záznam má však svoje nevýhody, z ktorých najvýznamnejšou je maximálne rozlíšenie záznamu. Rastrová matica o určitých rozmeroch, obsahuje dáta o zobrazení objektov, nie však samotný popis týchto objektov. Preto je prakticky nemožné zvýšiť kvalitu zobrazenia. Je možné čiastočne upravovať fyzické rozlíšenie zobrazenia, to ale samotnú kvalitu záznamu nemení. Z tohoto dôvodu často vyplýva potreba predimenzovať veľkosť rozlíšenia záznamu, aby bolo možné uchovať čo najviac detailov.
Ďalej vzniká problém s modelovaním (úpravou) záznamu, pričom zmena zaznamenaných objektov je možná len na báze zmeny jednotlivých pixlov, nakoľko samotný popis objektov nie je uložený.

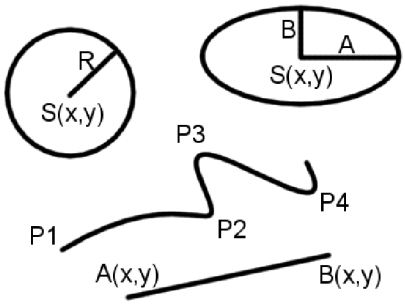


Obrázek 2.1: Rastrová grafika

2.2 Vektorová grafika

Vektorová grafika je postavená na vektorových entitách. Záznam je zložený z ľubovolného počtu takýchto entít s definovanými parametrami, teda z objektov. Z pohľadu 2D grafiky sú takýmito entitami (môžeme označiť aj ako geometrické entity) bod, úsečka, kružnica, elipsa, krivky a polygóny. Tieto objekty nazývame aj *základné grafické prvky* a sú obsiahnuté vo väčšine kresliacich programov v rovine [6].

Na rozdiel od rastrovej grafiky, kde nebolo možné zvýšiť kvalitu záznamu, vo vektorovej grafike je tento problém riešený už samotnými vektorovými entitami. Tie sú presne matematicky definované, a preto objekt, ktorý reprezentujú, je možné ľubovolne zväčšiť. Ďalej je možné meniť parametre tohoto objektu (farba, hrúbka a štýl čiar, atď.), takže modelovanie záznamu je takisto pomerne jednoduché.
K podrobnému popisu vybraných entít sa dostaneme ďalej v tejto kapitole.



Obrázek 2.2: Vektorová grafika

2.3 Dôvody potreby prevodov medzi vektorovou a rastrovou grafikou

Už sme si popísali oba spôsoby grafického záznamu a uviedli sme základné princípy a rozdiely týchto prístupov. Výsledkom ich vzájomného porovnania je konštatovanie, že vektorový grafický záznam je oveľa flexibilnejší, jednoduchšie modifikovateľný a z hľadiska

potrebného úložného priestoru šetrnejší. Vektorová grafika je preto veľmi často využívaná, avšak v niektorých prípadoch ju nie je možné použiť. Skutočný problém ale nastáva pri zobrazovaní takéhoto záznamu. Je to z dôvodu používania rastrových zobrazovacích zariadení, ako už bolo spomenuté. Aby teda bolo možné vektorový záznam zobraziť na rastrovom zariadení, je *rasterizácia* nezbytným krokom.

Rasterizácia je proces, pri ktorom sa vektorové objekty konvertujú (prevádzajú) do rastrovej podoby. Účelom je nájsť všetky pixly reprezentujúce prevádzaný objekt a určiť ich správnu farbu. K tomuto procesu sa využívajú algoritmy navrhnuté s ohľadom na čo najväčšiu efektivitu. Vektorový záznam pozostáva z množstva objektov. Tie ale v skutočnosti predstavuje len niekoľko entít, ktoré majú rôzne nastavené parametre. Stačí nám preto len niekoľko algoritmov na rasterizáciu práve týchto základných entít. Tým dostaneme spoľahlivý nástroj na prevod vektorového záznamu do rastrovej podoby.

2.4 Prevodové algoritmy

Základnými entitami sú už spomínané bod, úsečka, kružnica a elipsa. Z nich sa dajú jednoducho poskladať zložitejšie objekty, ktoré ale budú využívať k rasterizácii algoritmy týchto základných entít. O niečo zložitejšie sú algoritmy pre rasterizáciu kriviek, z ktorých si tiež jeden popíšeme.

V prvom rade je potrebná matematická definícia a predpis entity, s ktorou bude algoritmus pracovať. Ďalej sa zameriame na objasnenie podstaty algoritmu rasterizácie a následne na jeho prípadné zefektívnenie.

Vychádzajúc z poznatku, že rastrová grafika je diskrétna a faktu, že výpočty s celými číslami sú z hľadiska spracovania efektívnejšie ako práca s číslami reálnymi (tz. čísla s desatinnou čiarkou), bude snaha v týchto algoritmoch využívať práve celé čísla.

2.4.1 Úsečka

Úsečka je jedna z najjednoduchších entít v 2D grafike. Je ale aj jedna z najvyužívanejších, preto sa kladie veľký dôraz na efektívnosť jej rasterizačného algoritmu. Úsečka sa využíva napr. pri tvorbe lomených čiar, kriviek ale aj mnohých ďalších objektov.

Úsečka je definovaná dvoma bodmi v súradnicovej sústave. Súradnice sú reálne čísla, a preto sa pri prevode do rastrovej podoby zaokrúhľujú na celé. Úsečku tvorí priebeh priamky medzi dvoma bodmi na nej ležiacimi. Nasleduje niekoľko možných matematických zápisov priamky.

Obecná rovnica priamky

$$Ax + By + C = 0,\ A\ = (y_1 - y_2),\ B = (x_2 - x_1) \tag{2.1}$$

Parametrické vyjadrenie priamky

$$x = x_1 + t(x_2 - x_1),\ y = y_1 + t(y_2 - y_1) \tag{2.2}$$

Smernicový tvar priamky

$$y = kx + q,\ k\ = \frac{\Delta y}{\Delta x} = \frac{(y_2 - y_1)}{(x_2 - x_1)} \tag{2.3}$$

V rasterizačných algoritmoch pre úsečku sa môžeme oprieť o skutočnosť, že stačí ak sa v jednom smere (v jednej z osí x alebo y) neustále hodnota inkrementuje. Toto je dôvodom, vďaka ktorému je možné algoritmy zjednodušiť tým spôsobom, že úsečku budeme tvoriť len pre prvú polovicu prvého kvadrantu alebo druhú polovicu štvrtého kvadrantu. Tým dosiahneme v každom cykle algoritmu jednotkový posun na osi x, a to len v kladnom smere. Následne musíme riešiť vyhodnocovanie na osi y. Umiestnením úsečky len do prvej polovice prvého kvadrantu zaistíme inkrementáciu (nie však jednotkovú) v kladnom smere osi y. Teraz nám úsečka narastá rýchlejšie na osi x.
Pretože nie každá úsečka sa nachádza práve v prvej polovici prvého kvadrantu, budeme ju musieť pred aplikovaním algoritmu upraviť. Po výpočte súradníc nového bodu budú tieto zmeny napravené a tým bude dosiahnuté správne umiestnenie tohoto bodu. Prevod úsečky uskutočníme na základe zámeny jej súradníc, pričom využijeme osových súmerností. Tento proces bude popísaný bližšie pri popise samotného algoritmu.

V matematickom podaní je v predošlom odstavci popísaná priamka, pre ktorej smernicový predpis platí, že $0 < k < 1$. Prvý bod (od ktorého začíname v algoritme postupovať) má menšiu x-ovú súradnicu ako druhý bod. y-nová súradnica druhého bodu je väčšia ako v prvom bode.
Úsečka je vzorkovaná na ose x jednotkovým krokom, preto nazývame osu x *riadiacou osou* a osu y *osou vedľajšou*.

(a) (b)

Obrázek 2.3: (a) Kvadrant priebehu, (b) Rastrové zobrazenie

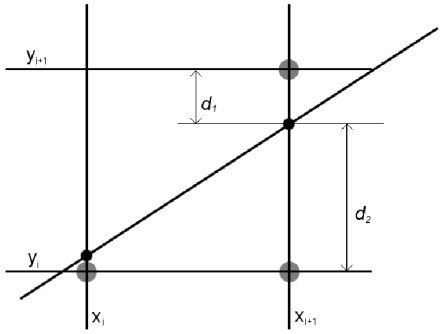
Bresenhamov algoritmus je založený práve na tomto princípe. Počiatočné súradnice sa zaokrúhľujú, pričom vzniknutá nepresnosť má len zanedbateľný charakter. Naopak pri práci zo smernicou by sa aj malé nepresnosti premietali nepriaznivo do výslednej podoby vypočítanej úsečky.
V každom kroku je predmetom skúmania pozícia nasledujúceho bodu na ose y. Sú vždy len dve možnosti, pričom sa y-nová súradnica nového bodu buď nemení od predchádzajúceho, alebo sa zvýši o 1. Matematicky zápis platný pre nové súradnice (predpokladáme predchádzajúce súradnice $[x_i, y_i]$): $[x_i + 1, y_i]$ alebo $[x_i + 1, y_i + 1]$.
Nová y-nová súradnica bude tá, ktorá je bližšie reálnemu prieniku úsečky s osou $x_i + 1$. Vzdialenosti oboch možných bodov od tohoto prieniku označíme a vyjadríme ako d_1 a d_2,

pričom vychádzame so smernicového zápisu priamky.

$$y = k(x_i + 1) + q \tag{2.4}$$

$$d_1 = y - y_i = k(x_i + 1) + q - y_i \tag{2.5}$$

$$d_2 = y_i + 1 - y = y_i + 1 - k(x_i + 1) - q \tag{2.6}$$



Obrázek 2.4: Nasledujúce súradnice

Rozdielom d_1 a d_2 získame hodnotu Δd, ktorej znamienko je určujúce pri rozhodnutí umiestnenia nového bodu. Pokiaľ je znamienko Δd záporné, nasledujúcemu bodu zostáva súradnica y_i, naopak pri kladnom znamienku to bude $y_i + 1$.

$$\Delta d = d_1 - d_2 = 2k(x_i + 1) - 2y_i + 2q - 1 \tag{2.7}$$

Výpočet je stále na báze reálneho čísla. Príčinou je výskyt smernice k, racionálneho čísla, v rovnici. k je podielom $\frac{\Delta y}{\Delta x}$ a preto je možné rovnicu previesť do čisto celočíselnej podoby. Výslednú hodnotu rovnice nazveme *predikátor* a označíme p_i. Platí rovnaké rozhodovanie podľa znamienka ako pre Δd.

$$\Delta d = 2\frac{\Delta y}{\Delta x}(x_i + 1) - 2y_i + 2q - 1 \tag{2.8}$$

$$p_i = \Delta d \Delta x = 2\Delta y x_i - 2\Delta x y_i - 2\Delta y + \Delta x(2q - 1) \tag{2.9}$$

Aby bolo možné v algoritme počítať iteratívnym spôsobom, musíme vyjadriť iterátor, ktorý sa bude pripočítavať k súčasnému predikátoru.

$$p_{i+1} = p_i - 2\Delta y x_i + 2\Delta x y_i + 2\Delta y x_{i+1} - 2\Delta x y_{i+1} \tag{2.10}$$

x_{i+1} nahradíme ako $(x_i + 1)$ (konštantná iterácia o 1 v každom cykle). y_{i+1}, v závislosti na znamienku p_i, nahradíme buď ako y_i (ak $p_i < 0$), alebo $y_i + 1$ (ak $p_i \geq 0$). Týmto nám vzniknú dva možné iterátory p_1 a p_2.

$$p_{i+1} = p_i + 2\Delta y - 2\Delta x(y_{i+1} - y_i) \tag{2.11}$$

$$p_i < 0 \qquad p_{i+1} = p_i + 2\Delta y \tag{2.12}$$

$$p_i \geq 0 \qquad p_{i+1} = p_i + 2\Delta y - 2\Delta x \tag{2.13}$$

Počiatočná hodnota p_i bude nastavená na hodnotu $2\Delta y + \Delta x$, kde x a y sú súradnicami počiatočného bodu.

Algoritmus Bresenham zapísaný v pseudokóde

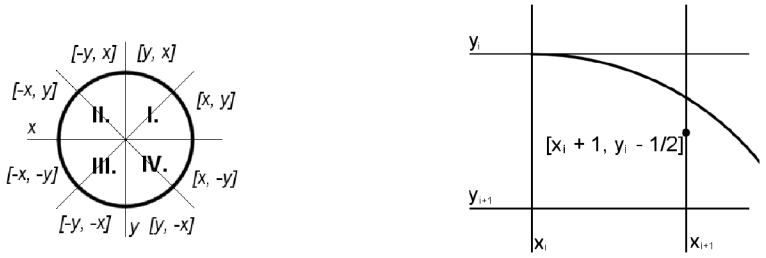
```
\\ ak úsečka rastie rýchlejšie na ose y, zameníme x-ové súradnice s y-novými
\\ túto skutočnosť nám reprezentuje premenná swap
bool swap = false;
if(abs(y2 - y1) > abs(x2 - x1)) {
  swap = true;
  SWAP(x1, x2);
  SWAP(y1, y2);
}

\\ kladný/záporný smer narastania úsečky na x-ovej osi
int step_x = 1;
if(x1 > x2)
  step_x = -1;

\\ kladný/záporný smer narastania úsečky na y-novej osi
int step_y = 1;
if(y1 > y2)
  step_y = -1;


\\ deklarácie ostatných premenných
int dx = abs(x2 - x1),          \\ delta x
    dy = abs(y2 - y1);          \\ delta y

int P = 2*dy - dx,              \\ predikátor
    P1 = 2*dy,                  \\ prvý iterátor
    P2 = 2*dy - 2*dx;           \\ druhý iterátor

\\ nastavenie počítaného bodu [x, y]
int y = y1,
    x;

\\ samotný cyklus algoritmu Bresenham
for(x = x1; x != x2; x = x + step_x) {
  if(swap)                      \\ ne/boli zamenené súradnice
    PutPixel(y, x, color);
  else
    PutPixel(x, y, color);

  if(P >= 0) {                  \\ posúvame sa na osi y
    P = P + P2;
    y = y + step_y;
  }
  else                          \\ neposúvane sa na osi y
    P = P + P1;
}
```

2.4.2 Kružnica

Kružnica je ďalšou vektorovou entitou často využívanou na zápis objektu vo vektorovej grafike. Z pohľadu geometrie je presne určená stredom a polomerom. Stred kružnice sa označuje ako bod $S[s_x, s_y]$ a polomer ako R.
Matematický predpis funkcie kružnice je následovný:

$$F(x, y) : x^2 + y^2 - r^2 = 0 \tag{2.14}$$

Ako tomu bolo aj pri rasterizácii úsečky, bude snaha vypočítavať body patriace kružnici pomocou celočíselnej aritmetiky. Reálne čísla, ktorými je kružnica zadaná, sa zaokrúhlia bez výraznejšieho dopadu na presnosť vykreslenej kružnice.
Z pohľadu rastrovej grafiky je kružnica súmerná vo ôsmich oktantoch. Táto skutočnosť nám umožňuje aplikovať algoritmus na jeden oktant, pričom tak môžeme získavať súradnice kružnice pre všetky oktanty. Tieto body sa získajú zámenou súradníc a ich znamienok z vypočítaného bodu.
Ďalej pre všetky body kružnice so stredom $S[s_x, s_y]$ platí, že sú rovné bodom kružnice so stredom $S[0, 0]$, ku ktorým sa pripočítajú súradnice s_x a s_y (predpokladáme rovnaký polomer kružníc). Kružnicu preto prevádzame do rastrovej podoby so stredom $S[0, 0]$ a vypočítané body potom posúvame na správnu pozíciu.



(a) (b)

Obrázek 2.5: (a) Súmernosť kružnice, (b) Midpoint

Midpoint algoritmus pre kružnicu, tiež známy aj ako Bresenhamov, stavia na predošlých tvrdeniach. Samotné výpočty bodov sa uskutočňujú pre druhý oktant. Postupuje sa od bodu $[0, R]$ až do bodu, kde $x = y$. Týmto je v priebehu celého oktantu postup po osi x jednotkový. To znamená, že v každom cykle sa x-ová súradnica počítaného bodu zvýši o 1. Preto môžeme bod x_{i+1} uvažovať ako $x_i + 1$.
Predmetom rozhodovania bude opäť y-nová súradnica nového bodu. Možné sú dve pozície. V prvom prípade zostáva y-nová súradnica rovnaká ako pre predchádzajúci bod, v druhom prípade sa nový bod posunie o pixel nižšie. Toto rozhodnutie sa vyhodnocuje na základe bodu $\left[x_{i+1}, y_i + \frac{1}{2}\right]$, ktorý je stredom medzi dvoma možnými novými bodmi ($[x_{i+1}, y_i]$ a $[x_{i+1}, y_{i-1}]$). Dosadením tohoto bodu do funkcie kružnice [2.14] zistíme, či leží v kružnici, alebo mimo nej. Ak je znamienko výsledku kladné, to znamená, že tento bod nepatrí kružnici, je nasledujúcou y-novou súradnicou $y_i - 1$. V opačnom prípade zostáva súradnica rovná y_i. Výsledok označíme p_i a bude predikátorom.

$$p_i = (x_i + 1)^2 + \left(y_i - \frac{1}{2}\right)^2 - r^2 \tag{2.15}$$

Aby sme v algoritme mohli postupovať iteratívne, vyjadríme si rozdiel medzi p_i a p_{i+1} pre obidva prípady. p_i je buď záporné ($y_{i+1} = y_i$) alebo kladné ($y_{i+1} = y_i - 1$).

$$p_{i+1} = (x_i + 2)^2 + \left(y_{i+1} - \frac{1}{2}\right)^2 - r^2 \tag{2.16}$$

$$p_{i+1} = p_i + 2x_i + 3 - \left(y_i - \frac{1}{2}\right)^2 + \left(y_{i+1} - \frac{1}{2}\right)^2 \tag{2.17}$$

$$p_i < 0 \qquad p_{i+1} = p_i + 2x_i + 3 \tag{2.18}$$

$$p_i < 0 \qquad p_{i+1} = p_i + 2x_i + 5 - 2y_i \tag{2.19}$$

Predikátor inicializujeme na hodnotu $p_i = 1 - r$. Vytvoríme dva iterátory, $X2$ a $Y2$, ktoré budú aktualizovať predikátor. Ich hodnota bude inicializovaná, ako to vyplýva z rovníc [2.18] a [2.19]. $X2$ bude pripočítavaný k predikátoru v každom cykle. $Y2$ bude odpočítavané len za podmienky, že $p_i \geq 0$. V oboch iterátoroch sa nachádza násobenie, ktoré je možné odstrániť pripočítavaním/odpočítavaním dvojnásobnej hodnoty kroku. Pre $X2$ prepočítavame 2 a pre $Y2$ odpočítavame 2.

Algoritmus Midpoint pre kružnicu zapísaný v pseudokóde

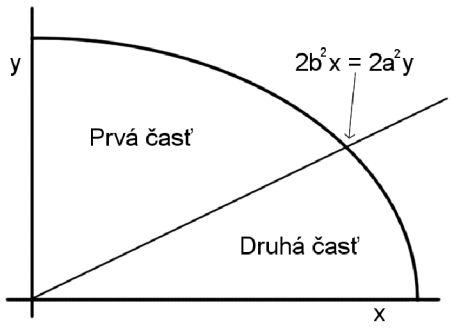
```
\\ inicializácia počítaného bodu
int x = 0,
    y = R;

\\ inicializácia predikátoru
int P = 1 - R;

\\ inicializácia iterátorov
int X2 = 3,
    Y2 = 2*R - 2;

\\ samotný cyklus algoritmu Midpoint pre kružnicu
while(x <= y) {                 \\ dosiahnutie hranice oktantu
  PutCirclePixel(x, y, color);  \\ vykreslenie bodu do všetkých oktantov
  if(P >=0) {                   \\ posun o pixel nižšie
    P = P - Y2;
    Y2 = Y2 -2;
    y = y - 1;
  }
  P = P + X2;                   \\ posun o pixel po osi x
  X2 = X2 + 2;
  x = x + 1;
}
```

2.4.3 Elipsa

Elipsa je entita v mnohých ohľadoch podobná kružnici, preto bude postup pri tvorbe algoritmu rasterizácie skoro rovnaký. Elipsa je súmerná v štyroch kvadrantoch. Zámenou súradníc alebo ich znamienok je možné dosiahnuť body v zvyšných troch kvadrantoch, ak

vypočítame bod na elipse napr. v prvom kvadrante.
Matematický predpis funkcie elipsy je následovný:

$$F(x, y) : b^2x^2 + a^2y^2 - a^2b^2 = 0 \tag{2.20}$$



Obrázek 2.6: Rozdelenie prvého kvadrantu elipsy

Pri výpočte bodov patriacich elipse, budeme stred elipsy brať ako $S[0, 0]$, podobne ako sme to robili pri kružnici. Vypočítaný bod následne posunieme na jeho skutočnú pozíciu. Prvý kvadrant, pre ktorý budeme body počítať, musíme rozdeliť na dve časti. A to v bode, ktorý má vlastnosť $2b^2x = 2a^2y$. V prvej časti (druhá polovica prvého kvadrantu) bude v každom cykle narastať x-ová súradnica o 1 a y-novú súradnicu budeme posúvať o 1 podľa predikátoru. V druhej časti (prvá polovica prvého kvadrantu) tomu bude naopak, pričom sa predikátor pri priechode medzi týmito časťami nanovo inicializuje. Rozhodovanie pri výbere medzi dvoma možnými bodmi bude opäť záležať na tom, či ich stred (midpoint) patrí do elipsy alebo nie.
V prvej časti, kde je riadiaca os x, teda $x_{i+1} = x_i + 1$, platí rozhodovanie podla predikátoru. Ak $p_i < 0$, potom $y_{i+1} = y_i$ a ak $p_i \geq 0$, potom $y_{i+1} = y_i - 1$.

$$p_i = b^2(x_i + 1)^2 + a^2\left(y_i - \frac{1}{2}\right)^2 - a^2b^2 \tag{2.21}$$

$$p_{i+1} = b^2(x_i + 2)^2 + a^2\left(y_{i+1} - \frac{1}{2}\right)^2 - a^2b^2 \tag{2.22}$$

$$p_i < 0 \qquad p_{i+1} = p_i + b^2(2x_i + 3) \tag{2.23}$$

$$p_i \geq 0 \qquad p_{i+1} = p_i + b^2(2x_i + 3) + a^2(2 - 2y_i) \tag{2.24}$$

Naopak v druhej časti, kde je riadiaca os y, platí $y_{i+1} = y_i - 1$. A podľa predikátoru sa určí x_{i+1}: ak $p_i < 0$, potom $x_{i+1} = x_i + 1$ a ak $p_i \geq 0$, potom $x_{i+1} = x_i$. Vyjadríme si predikátor pre ďalší krok.

$$p_i = b^2\left(x_i + \frac{1}{2}\right)^2 + a^2(y_i - 1)^2 - a^2b^2 \tag{2.25}$$

$$p_{i+1} = b^2\left(x_{i+1} + \frac{1}{2}\right)^2 + a^2(y_i - 2)^2 - a^2b^2 \tag{2.26}$$

$$p_i < 0 \qquad p_{i+1} = p_i + b^2(2x_i + 2) + a^2(3 - 2y_i) \tag{2.27}$$

$$p_i \geq 0 \qquad p_{i+1} = p_i + a^2(3 - 2y_i) \tag{2.28}$$

Algoritmus začína s hodnotami $x = 0$ a $y = b$. Predikátor nastavíme na prvý midpoint, ktorým je $\left[1, b - \frac{1}{2}\right]$. Inicializácia p_i vyzerá následovne:

$$p_i = b^2 1^2 + a^2\left(b - \frac{1}{2}\right)^2 - a^2b^2 \tag{2.29}$$

$$p_i = b^2 + a^2(0.25 - b) \tag{2.30}$$

Po dosiahnutí bodu $2b^2x = 2a^2y$ sa predikátor musí znovu inicializovať. V tomto prípade sú hodnoty midpointu $\left[x + \frac{1}{2}, y - 1\right]$.

$$p_i = b^2\left(x + \frac{1}{2}\right)^2 + a^2(y - 1)^2 - a^2b^2 \tag{2.31}$$

2.4.4 Bézierova krivka

Bézierova krivka patrí medzi aproximačné krivky. Krivka n-tého stupňa je zadaná $n + 1$ riadiacimi bodmi P_i. Tie vytvárajú tzv. riadiaci polygón. Krivka leží v konvexnej obálke svojho riadiaceho polygónu. Matematický vzťah [2.32] popisuje tieto Bézierove krivky. Vo vzťahu sa nachádza B_i^n, ktorá predstavuje *Brensteinové polynómy* n-tého stupňa [2.33].

$$Q(t) = \sum_{i=0}^{n} P_i B_i^n(t) \tag{2.32}$$

$$B_i^n(t) = \binom{n}{i} t^i (1 - t)^{n-i}; \quad t \in \langle 0, 1\rangle; \quad i = 0, 1, \ldots n \tag{2.33}$$

Krivka prechádza koncovými bodmi riadiaceho polygónu. Pri zmene polohy niektorého z riadiacich bodov P_i je celá krivka pozmenená. Toto by malo neblahý dopad hlavne na krivky vyššieho stupňa. Preto je výhodnejšie vytvárať zložitejšie objekty spájaním (nadväzovaním) viacerých kriviek nižších stupňov. Najčastejšie je to Bézierova kubika, tzn. Bézierova krivka 3-tieho stupňa ($n = 3$). Tá je daná vzťahom odvodeným z [2.32]:

$$Q(t) = \sum_{i=0}^{3} P_i B_i(t) \tag{2.34}$$

[1]Pre zjednodušenie výpočtov zavedieme v algoritme premenné $AA = a^2$ a $BB = b^2$.

Algoritmus Midpoint pre elipsu zapísaný v pseudokóde[1]

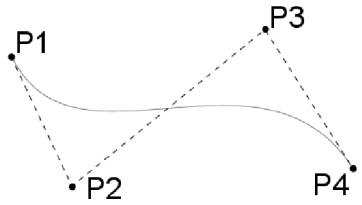
```
\\ inicializácia počítaného bodu
int x = 0,
    y = b;

\\ pomocné premenné
int AA = a*a,
    BB = b*b;

\\ inicializácia predikátoru
int P = BB - AA*b + AA/4;

\\ cyklus pre prvú časť algoritmu
while(AA*y > BB*x) {            \\ dosiahnutie bodu 2b^2x = 2a^2y
  PutEllipsePixel(x, y, color); \\ vykreslenie štyroch bodov elipsy
  if(P < 0) {
    P = P + BB*(2*x + 3);
  }
  else {
    P = P + BB*(2*x + 3) + AA*(2 - 2*y);
    y = y - 1;                  \\ midpoint nepatrí kružnici (y sa zmenší o jedna)
  }
  x = x + 1;                    \\ x sa zvyšuje v každom cykle
}

\\ prepočítanie predikátoru
P = BB*(x + 0.5)*(x + 0.5) + AA*(y - 1)*(y - 1) - AA*BB;

\\ cyklus pre druhú časť algoritmu
while(y >= 0) {                 \\ dosiahnutie hranice prvého kvadrantu
  PutEllipsePixel(x, y, color); \\ vykreslenie štyroch bodov elipsy
  if(P < 0) {
    P = P + BB*(2*x + 2) + AA*(3 - 2*y);
    x = x + 1;
  }
  else {
    P = P + AA*(3 - 2*y);
    x = x + 1;                  \\ midpoint nepatrí kružnici (x sa zvyšuje o jedna)
  }
  y = y - 1;                    \\ y sa zmenšuje v každom cykle
}
```

Brensteinové polynómy sú vyjadrené [2.36] a maticový zápis Bézierovej kubiky je [2.37].

$$\begin{aligned} B_0 &= (1-t)^3 \\ B_0 &= 3t(1-t)^2 \\ B_0 &= 3t^2(1-t) \\ B_0 &= t^3 \end{aligned} \tag{2.35}$$

$$\begin{aligned} Q(t) &= (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3 \\ Q(t) &= (-P_0 + 3P_1 - 3P_2 + P_3)t^3 + (3P_0 - 6P_1 + 3P_2)t^2 + (-3P_0 + 3P_1)t + P_0 \end{aligned} \tag{2.36}$$

$$Q(t) = [t^3 \; t^2 \; t \; 1] \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}, \quad t \in \langle 0, 1 \rangle \tag{2.37}$$



Obrázek 2.7: Bézierova kubika

Algoritmus, ktorý si popíšeme, nie je zameraný na efektivitu, ale na ukážku výpočtov bodov krivky podľa [2.37]. Princíp vytvárania rastrovej podoby tejto krivky je založený na výpočte len niekoľkých bodov krivky, ktoré sa spoja úsečkami. Ich počet je daný celým číslom označeným ako *quality* väčším ako nula. Body sú následne počítané pre každú hodnotu t od 0 až po 1 s krokom $\frac{1}{quality}$. Pritom platí, že počiatok krivky je v bode P_0 a koniec v bode P_3.

Algoritmus výpočtu bodov Bézierovej kubiky zapísaný v pseudokóde

```
// Predpokladáme pole o štyroch prvkoch (point): P
// a pole LP, do ktorého sa ukladajú vypočítané body (point).
// hodnota quality je daná

point XY;                       // deklarácia počítaného bodu
double step = 1.0/quality;      // krokový iterátor pre "t"
double t = 0.0;

for(int i = 0; i <= quality; i++) {
  XY.x = (-   P[0].x + 3*P[1].x - 3*P[2].x + P[3].x )*t^3
       + (  3*P[0].x - 6*P[1].x + 3*P[2].x          )*t^2
       + (- 3*P[0].x + 3*P[1].x                     )*t
       + (    P[0].x                                );

  XY.y = (-   P[0].y + 3*P[1].y - 3*P[2].y + P[3].y )*t^3
       + (  3*P[0].y - 6*P[1].y + 3*P[2].y          )*t^2
       + (- 3*P[0].y + 3*P[1].y                     )*t
       + (    P[0].y                                );

  t = t + step;
  LP[i] = XY;
}
```

Algoritmus spojenie vypočítaných bodov Bézierovej kubiky zapísaný v pseudokóde

```
// Predpokladáme pole LP, ktoré obsahuje vypočítané body krivky

int sum = LP.size() - 1;        // počet úsečiek

for(int i = 0; i < sum; i++) {
  DrawLine( LP[i], LP[i + 1] ); // spojenie susedných bodov úsečkou
}
```

Kapitola 3

Návrh a implementácia aplikácie

Pred samotnou implementáciou aplikácie je potrebné určiť ciele, ktoré by mal výsledný program spĺňať z užívateľského hľadiska. Tie vyplývajú hlavne zo zadania práce. Na základe týchto požiadaviek bude zostavený návrh aplikácie a jej predbežné zloženie z programátorského hľadiska. Takisto je potrebné zvoliť vývojové nástroje a postup implementácie.

3.1 Požiadavky na program

Ako už bolo naznačené, aplikácia je zameraná na výučbu a demonštráciu vybraných rasterizačných algoritmov. Je preto nutné zhodnotiť požiadavky z pohľadu užívateľa, ktorý tento program bude používať.
Prvým bodom je teoretická časť, ktorá bude užívateľovi prístupná v nápovede programu. Každý algoritmus bude podrobne popísaný a užívateľ tak získa predstavu o jeho podstate. Aby si nadobudnuté teoretické poznatky mohol prakticky overiť, bude aplikácia poskytovať aj nástroje na ich demonštráciu.

GUI programu by nemalo byť zložité. Program by mal byť jednoducho a intuitívne ovládateľný. Grafický výstup programu, konkrétne rastrová plocha zobrazujúca výsledok rasterizácie, je nutnosťou. Výhodou bude možnosť priblíženia alebo oddialenia pohľadu.
Ďalej je potrebná možnosť voľby konkrétneho algoritmu. Každý z týchto algoritmov je v princípe zložený z troch častí. **Prvá časť** je výber parametrov (definícia) pre danú entitu. **Druhou časťou** je príprava algoritmu na samotný výpočet hladaných bodov. Týmto sa myslia inicializácie riadiacich premenných algoritmu a prípadné použité heuristiky. **Tretia časť** je samotný priebeh algoritmu, ktorý už je realizovaný v cykle.
Na základe tohoto rozdelenia je možné navrhnúť spoločnú funkcionalitu ovládania demonštrovaných algoritmov. Body budú volené priamo z rastrovej plochy. Ostatné parametre entít budú nastaviteľné samostatne. Po zadaní všetkých potrebných parametrov bude možné pristúpiť priamo k priebehu algoritmu. Každý algoritmus bude rozdelený na kroky (logické časti algoritmu). Tieto kroky budú základom demonštrácie procesu rasterizácie pre každú z vybraných entít. V každom kroku budú premenné algoritmu, ktoré sú relevantné pre užívatela, aktualizované. Takisto bude každý krok komentovaný výpisom s aktuálnymi hodnotami (napr. vyhodnocovanie podmienok, výpočty nových bodov, atď). Na rastrovej ploche budú zobrazované všetky zmeny, ktoré algoritmus vypočíta. Takto dosiahneme možnosť podrobného preskúmania algoritmu a celkový prehľad jeho priebehu.

3.2 Návrh aplikácie

Aplikácia bude objektovo orientovaná. Pri návrhu vychádzame z požiadaviek, ktoré sme popísali v predošlej časti. Aplikáciu rozdelíme na časti (triedy), ktoré budú zaisťovať potrebnú funkčnosť. Samotná implementácia bude popísaná až neskôr.

V prvom rade budeme potrebovať udržiavať informácie o parametroch a rastrovej podobe počítanej entity. Za týmto účelom navrhneme triedu `cField`. Rastrová podoba jedinej entity spravidla zaberá len malú časť z rastrového pola, ktoré je zobrazované. Preto bude výhodným riešením použiť hash tabuľku, ktorá bude udržiavať informácie len o vybraných bodoch a nie o celej ploche, ktorej veľká časť nebude využitá. Základným prvkom v tejto hash tabuľke bude jedna bunka predstavujúca jeden pixel a jeho aktuálne nastavenie. Nenastavené pixle sa v tabuľke udržiavať nebudú. Týmto nám vznikli dve nové triedy `cHashTable` a `cCell`.
Dáta entity vo vektorovej podobe budú zaznamenávané v triede `cEntity`.
Trieda `cField` teda bude obsahovať primárne dva objekty [2] (z tried `cHashTable` a `cEntity`), nad ktorými bude poskytovať kontrolu a metódy na manipuláciu s nimi.

O grafické zobrazenie dát objektov triedy `cField` sa bude starať trieda `cCanvas`. Bude poskytovať prostriedky na vykreslenie rastrového pola s rôznym priblížením. A takisto aj možnosť zobraziť entitu nad týmto rastrovým pohľadom v jej "reálnej"[3] podobe.

Algoritmy, ktoré majú veľa spoločných znakov a spoločný princíp ovládania, budú vychádzať z triedy `cAlgorithm`. Tá bude zabezpečovať spoločnú formu pre objekty všetkých algoritmov. Jej primárnou úlohou bude spravovať objekt `cField`, ktorého dáta budú reprezentovať výsledky algoritmu (vypočítané body, riadiace body, atď). Ďalej vytvoríme triedu `cMessageBox`, ktorá bude poskytovať možnosť textového výpisu. Jej objekt bude takisto ovládaný z triedy `cAlgorithm`.
Každý odvodený algoritmus bude mať implementované vlastné správanie pre vybrané metódy zdedené z triedy `cAlgorithm`. Tými bude napríklad implementácia samotného rasterizačného algoritmu a panel s menami a hodnotami premenných pre tento algoritmus.

Objekty týchto tried spolu vytvoria celú aplikáciu, ktorá bude komunikovať s užívateľom. Vzájomná komunikácia, usporiadanie objektov a celková správa aplikácie bude implementovaná v triede `cFrame`, hlavnom okne aplikácie. V nej budú umiestnené jednotlivé objekty a ovládacie prvky.
S touto triedou bude spojená trieda `cSetFrame`, ktorá bude spravovať nastavenie celej aplikácie. Metódy na záznam a prístup k uloženým nastaveniam budú implementované v triede `cXmlSettings`.

Na záver spomenieme triedu `cXmlLanguage`, ktorá implementuje metódy na prístup k textom, ktoré sú používané v celej aplikácii. Texty môžu byť v rôznych jazykoch. Na preloženie aplikácie pritom stačí vytvorenie nového jazykového súboru a preloženie textov, ten je potom možné v aplikácii použiť a tak zmeniť jej jazyk.

[2] Objektom nazývame jednu inštanciu triedy.
[3] Reálnou podobou sa myslí presné zobrazenie entity (nakoľko to len rastrový monitor dovoľuje).

3.3 Implementačné prostredie a použité nástroje

Ako implementačný jazyk celej aplikácie sme si z ponuky možných jazykov zvolili `C++`. Jazyk je objektovo orientovaný, s čím počíta aj návrh aplikácie.
Ďalším nástrojom bude *cross-platform GUI toolkit* ***wxWidgets*** [3]. Ide o nástroj poskytujúci implementačné prostriedky pre grafické rozhranie programu. Jeho výhodou je prenositeľnosť kódu medzi jednotlivými platformami. Túto výhodu využijeme pri vývoji našej aplikácie, ktorá bude preložiteľná a spustiteľná minimálne na platformách MS Windows a Linux.
Na správu dát popísaných v XML použijeme *C++ XML Parser* [4], ktorý je multiplatformový.
Programová dokumentácia bude generovaná programom *Doxygen*. Je to nástroj na automatické zostavovanie dokumentácie zo zdrojového kódu, ktorý je komentovaný požadovanou syntaxou.
Aplikácia bude vyvíjaná v prostredí *wxDev-C++* a testovaná bude na oboch platformách - Windows a Linux.

3.4 Implementácia

Naším cieľom bude vytvoriť čo najlepšiu aplikáciu, spĺňajúcu všetky požiadavky uvedené v návrhu. Návrh aplikácie bude realizovaný postupne, v etapách. Pri implementácii budeme využívať hlavne prostriedky wxWidgets. S pomocou týchto komponentov, ktoré sú nám k dispozícii, vytvoríme vlastné triedy popísané v návrhu aplikácie. Z nich následne zostavíme aplikáciu.
Názvy funkcií/metód, premenných a tried je snaha pomenovať výstižne, v anglickom jazyku tak, aby čo najviac vypovedali o svojom význame.
V tejto kapitole si popíšeme implementáciu niekoľkých najdôležitejších tried. Spomenuté a popísané budú len najzásadnejšie metódy a dáta týchto tried. Programovú dokumentáciu je možné nájsť na CD uvedenom v prílohách.

3.4.1 Zoznam implementovaných tried

Nasleduje celkový zoznam tried so stručným popisom. Po tomto zozname budú v ďalšej časti podrobnejšie popísané najdôležitejšie z nich.

App2D - hlavná trieda aplikácie, odvodená z triedy `wxApp`.

cFrame - hlavné okno aplikácie.

cCanvas - okno na vykresľovanie dát z triedy `cField`.

cAlgorithm - východzia trieda pre všetky implementované rastrové algoritmy.

[cAlg_Bresenham] - implementácia algoritmu Bresenham pre úsečku.

[cAlg_Cirmid] - implementácia algoritmu Midpoint pre kružnicu.

[cAlg_Ellmid] - implementácia algoritmu Midpoint pre elipsu.

[cAlg_Beziercubic] - implementácia algoritmu Bézierovej kubiky.

[4] C++ XML Parser http://iridia.ulb.ac.be/ fvandenb/tools/xmlParser.html

cField - trieda implementuje uloženie a manipuláciu s grafickými dátami.

cHashTable - hash tabuľka pre objekty triedy `cCell`.

cCell - predstavuje základný objekt v rastrovom poli - pixel a jeho nastavenia.

cEntity - východzia trieda pre záznam predpisu entity.

[cEntLine] - uchováva predpis úsečky.

[cEntCircle] - uchováva predpis kružnice.

[cEntEllipse] - uchováva predpis elipsy.

[cEntBezierCubic] - uchováva predpis Bézierovej kubiky.

cMessageBox - okno na textový výpis, odvodené od triedy `wxTextCtrl`.

cSetFrame - grafické rozhranie pre nastavovanie vlastností programu.

cXmlLanguage - implementuje správu jazykových dát v aplikácii.

cXmlSettings - implementuje správu nastavení aplikácie.

3.4.2 cField

Trieda `cField` je jedným zo základných kameňov aplikácie, pretože v nej sa udržiavajú všetky dáta, ktoré reprezentujú rastrový a aj skutočný obraz entity. Objekty tejto triedy sú ovládané dvoma ďalšími triedami - `cCanvas` a `cAlgorithm`.
Trieda obsahuje jeden objekt triedy `cHashTable`. Na manipuláciu s týmto objektom sú v tejto triede implementované metódy: `SetCell`, `UnsetCell`, `GetCell` a `Clear`. Takto sa vykonávajú zmeny s dátami rastrového záznamu objektu `cField`.
Druhý objekt, `cEntity`, je priamo prístupný a na jeho správu je možné využívať metódy implementované v jeho triede. Trieda `cEntity`, obsahuje metódy `SetInit`(stav objektu) a `SetShow`(ne/zobrazovať objekt). Každá trieda od nej odvodená má navyše aj metódu `Init`, ktorá zabezpečuje správnu inicializáciu dát entity v danom objekte.

3.4.3 cCanvas

`cCanvas` je trieda odvodená z triedy `wxScrolledWindow`, ktorá je implementovaná vo wxWidgets. Využíva ju na vykreslenie dát z objektu triedy `cField`, na ktorý má ukazovateľ. Vykresľovanie týchto dát je dosiahnuté naimplementovaním správania zdedenej metódy `OnPaint` a niekoľkých ďalších metód na vykresľovanie a obnovovanie obrazu. Tými sú napr.: `DrawCell`, `UpdateCell` (vykreslenie/obnovenie jednej bunky - pixlu), alebo `DrawEntity` (vykreslenie skutočnej entity nad rastrovým obrazom). Ďalej spomenieme dôležitú metódu `Zoom`, ktorá nastavuje veľkosť priblíženia pohľadu.

Trieda `cCanvas` obsahuje už spomínaný ukazovateľ na objekt triedy `cField`, ktorého zmenou a následným zavolaním metód vykreslenia, dosiahneme grafické zobrazenie jeho dát. Takto je možné používať jedno okno na vykresľovanie viacerých objektov triedy `cField`, pričom prechod medzi nimi je jednoduchý.

Užívateľský vstup je zabezpečený metódou `OnMouse`, kde sú zachytávané a ošetrené akcie vykonané myšou. Prechod myši nad rastrovým polom spôsobuje zobrazenie súradníc

aktuálneho pixlu (za predpokladu, že je táto funkcia zapnutá). Ďalej je tu ošetrené volenie si riadiacich bodov/pixlov pre aktuálny algoritmus. Pri kliknutí je volaná metóda objektu `cAlgorithm` (`GetPoint`) s príslušnými parametrami (súradnice zvoleného pixlu), ktorá tento podnet spracuje.

3.4.4 cAlgorithm

Uchovávanie a zobrazovanie dát sme si už popísali a implementovali. Teraz pristúpime k samotnému jadru aplikácie, ktorým je využitie týchto prostriedkov na prezentáciu priebehu rasterizačného algoritmu. Na tento účel nám slúži trieda `cAlgorithm` a od nej odvodené triedy so špecifikovaným správaním zdedených abstraktných metód.

Samotná trieda `cAlgorithm` je odvodená z triedy `wxPanel`. Je základom na zobrazenie spoločných ovládacích prvkov pre všetky algoritmy. Tým sa myslí resetovanie celého algoritmu, vykonanie jedného kroku algoritmu, spustenie/zastavenie priebehu algoritmu po krokoch, nastavovanie rýchlosti tohoto priebehu a možnosť vykreslenia skutočnej entity. V tejto triede sa vytvára objekt triedy `cField`, takže pre každý rasterizačný algoritmus existuje samostatný záznam s dátami pre zobrazenie. Preto môžeme pracovať s viacerými algoritmami naraz, a pritom nestratíme ich doterajšie výsledky. Ďalej je v tejto triede implementovaná metóda na prevzatie vykresľovacieho okna (`TakeCanvas`) a uloženie/načítanie výpisu (metódy `Save`, `Load`).

Objekty predstavujúce vybrané algoritmy budú vytvárané z tried, ktoré sú odvodené z tejto triedy a majú implementované správanie pre zdedené, abstraktné metódy: `Step`, `Reset` a `GetPoint`. Tieto triedy takisto obsahujú dáta (premenné) potrebné práve pre daný rasterizačný algoritmus. Niektoré z týchto premenných, ktoré sú dôležité pre užívateľa, sú pomenované a zobrazené na paneli.
Metóda `Step` je samotný rasterizačný algoritmus, rozdelený na logické časti. Pri každom volaní tejto metódy sa vykoná jedna časť tohoto algoritmu. Toto krokovanie je komentované, zmeny, ktoré vykonáva sa ihneď premietajú do rastrového zobrazenia a zároveň sa na paneli aktualizujú hodnoty viditeľných premenných.
`Reset` je metóda, ktorá nuluje alebo nastaví premenné pre daný algoritmus na prednastavené hodnoty, vymaže výpis a vyčistí rastrové pole.
Metódu `GetPoint` sme spomínali už skôr v tejto kapitole. Pri jej zavolaní, ktoré vyvoláva užívateľ kliknutím myši na rastrovú plochu, je jej úlohou vyhodnotiť, či sa má brať ohľad na zadaný pixel. Ak je už rasterizačný algoritmus zahájený, nie je možné meniť predpis počítanej entity. Na druhej strane, ak je bod akceptovaný, vyhodnotí sa jeho úloha v predpise entity. Platí pravidlo, že ak bol zadaný potrebný počet riadiacich bodov entity, nový bod nahrádza najstarší zvolený bod. Týmto spôsobom je možné upravovať entitu.

3.4.5 cFrame

Z vyššie popísaných tried vytvárame objekty a tie umiestnime do objektu triedy `cFrame`. Tento objekt je hlavným oknom celej aplikácie. Jeho úlohou je spravovať rozloženie viditeľných objektov a poskytovať užívateľovi niektoré ovládacie prvky. Tými sú napr. menu, ovládanie priblíženia rastrovej plochy a výber algoritmu zo zoznamu. Ďalej spomeňme, že táto trieda takisto zabezpečuje prístup k textom vo vybranom jazyku a zmeny nastavení aplikácie.
Objekt tejto triedy je vytváraný pri inicializácii aplikácie. Obsahuje ukazovatele na objekt

`cCanvas`, `cMessageBox`, `cXmlLanguage`, `cXmlSettings` a na pole objektov tried odvodených od `cAlgorithm` (rasterizačné algoritmy k dispozícii).

Kapitola 4

Výsledky

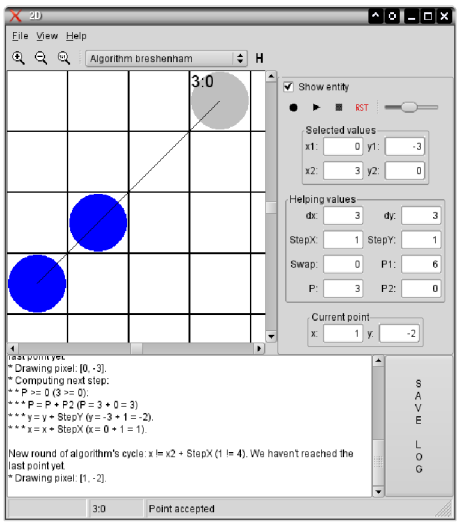
Aplikácia bola implementovaná podľa návrhu. Výsledný spustiteľný program je použiteľný v praxi na plnenie účelov svojho vzniku. Nie je potrebné žiadne špeciálne softwarové vybavenie platforiem, na ktorých tento program beží. Testovanie programu prebiehalo priebežne s jeho vývojom a všetky nájdené nedostatky boli odstránené.

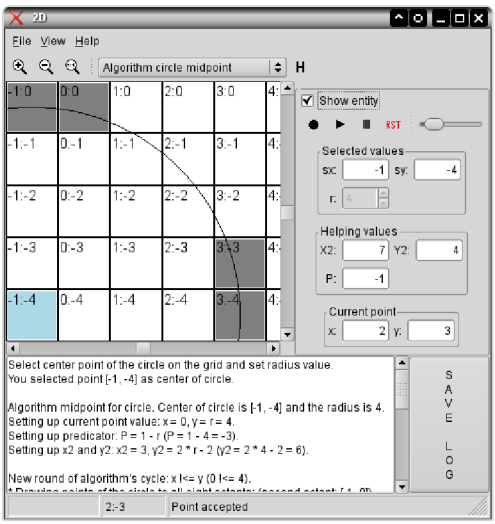
4.1 Ovládanie

Ovládanie programu je jednoduché a celková kompozícia je dobre navrhnutá. Programové menu poskytuje prístup k nastaveniam programu, niektorým ovládacím prvkom a nápovede. Kliknutím myši si užívateľ volí riadiace body z rastrovej plochy a na paneli vpravo nastavuje zvyšné parametre. Ovládacie prvky algoritmu sú umiestnené na tom istom paneli hore. Po zahájení algoritmu už nie je možné upravovať predpis entity.

4.2 Ukážky

Pre detailnejšiu predstavu o aplikácii nasleduje niekoľko snímkov samotného programu (na platforme Linux).





Kapitola 5

Záver

Projekt, ktorého úlohou bolo vytvorenie aplikácie použitelnej na výučbu a demonštráciu, je dokončený. Táto práca popisuje teoretické základy týkajúce sa danej problematiky, a aj postup návrhu a implementácie samotnej aplikácie. Aplikácia splnila ciele, ktoré si na začiatku kládla. Je jednoducho a intuitívne ovládateľná a poskytuje dobré prostriedky na praktické oboznámenie sa s vybranými rasterizačnými algoritmami. Výhodou aplikácie je jej viacjazyčnosť, pričom doplnenie iného jazyka je pomerne jednoduché, bez potreby zásahu do samotného programu.

Táto práca by mala byť prínosom pre študentov, ktorí sa oboznamujú zo základmi počítačovej grafiky, ale aj pre laikov. Osobným prínosom pre mňa bola skúsenosť so samostatnou prácou na rozsiahlejšom projekte a praktické používanie rozličných nástrojov pri jeho tvorbe.

Budúcnosť tohoto projektu je v jeho ďalšom vývoji. Smer, ktorým by sa mal uberať, je postupné dopĺňanie ďalších rasterizačných algoritmov na výber a rozširovaním užívateľských možností. Program je možné vďaka multiplatformovým nástrojom, s ktorých pomocou bol vyvíjaný, rozšíriť aj na iné platformy než len MS Windows a Linux.

Literatura

[1] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: principles and practice.* Addison-Wesley Publishing Company, Inc., 2003. ISBN 0-201-84840-6.

[2] Přemysl Kršek. *Základy počítačové grafiky, Studijní opora.* 2006.

[3] Julian Smart, Robert Roebling, Vadim Zeitlin, and Robin Dunn. Wxwidgets documentation. `http://www.wxwidgets.org/docs/`, 2007.

[4] WWW stránky. Bézierova křivka. `http://cs.wikipedia.org/wiki/Bézierova_křivka`

[5] WWW stránky. Raster graphics. `http://en.wikipedia.org/wiki/Raster_graphics/`

[6] Jiří Žára, Bedřich Beneš, Jiří Sochor, and Petr Felkel. *Moderní počítačová grafika.* Computer Press, 2004. ISBN 80-251-0454-0.

Prílohy

1. Diagram tried
2. CD nosič
    - súbory so zdrojovými kódmi
    - dokumentácia zdrojových kódov
    - manuál k programu
    - technická správa vo formáte pdf

Príloha číslo 1.: Diagram tried