MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Tvorba ilustrací pomocí rychlé distribuce kreslicích primitiv

BAKALÁŘSKÁ PRÁCE

Josef Sedlačík

Brno, jaro 2008

Prohlášení

Prohlašuji, že tato bakalářská práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Vedoucí práce: doc. Ing. Jiří Sochor, CSc.

Poděkování

Rád bych poděkoval vedoucímu práce doc. Ing. Jiřímu Sochorovi za cenné rady a pomoc při řešení práce.

Shrnutí

Cílem práce bylo nastudovat metodu nefotorealistického vykreslování a porovnat ji s jinými podobnými přístupy. V praktické části pak implementovat jednoduchou aplikaci, která by umožnila vytvářet ilustrace popsanou metodou.
V první části práce je popis několika metod pro nefotorealistické zobrazování a jejich srovnání. V druhé části práce je popsán realizovaný program.

Klíčová slova

grafická primitiva, hustota pravděpodobnosti, distribuční funkce, histogram

Obsah

Kapitola 1

Úvod do problematiky

1.1 Co je nefotorealistické zobrazování

Metody v počítačové grafice, které se snaží vytvářet fotorealistický obraz virtuální trojrozměrné scény, se označují jako „fotorealistické“. Myslí se tím míra podobnosti obrazu virtuální scény s fotografií.

Nefotorealistické zobrazování (angl. non-photorealistic rendering, zkratka NPR) je pak oblastí počítačové grafiky, která se o fotorealismus nesnaží. NPR je inspirováno uměleckými styly jako malba, kresba, technická ilustrace...

Fotorealismus je pro některé scény nevhodný, scény jsou zbytečně složité a těžko se v nich orientuje. Proto je vhodnější použít nefotorealistické zobrazení. NPR zobrazí jen důležité části a nepotřebné části zanedbá. Např. u manuálu k sestavení nějakého přístroje není zapotřebí mít do podrobna vykreslenou každou součástku, ale stačí zobrazit jen základní tvary součástek.

Současné NPR metody lze rozdělit do několika oblastí. Každá z těchto oblastí řeší odlišný okruh problémů – simulace tradičních malířských technik, simulace kreseb a rytin, metody pro názornější zobrazení scény...

1.2 Ilustrace se vzhledem perokresby

Metody snažící se napodobit vzhled perokresby, můžeme rozdělit do dvou kategorií: image-space a object-space metody. Někdy odkazované v literatuře jako image precision a object precision.

Metoda object-space provádí výpočty na jednotlivých objektech scény, porovnává jednotlivé objekty mezi sebou.

Metoda image-space bere v úvahu viditelnost jednotlivých pixelů každého objektu v scéně.

Problémem je tvorba animací kreseb. Pokud jsou grafická primitiva umístňována náhodně, při animování scény vzniká šum. Je tedy nutné zaručit koherenci umístění primitiv. Jednou možností je držet primitiva na místě relativně k obrazu. Je tak zaručena určitá koherence, ale tyto animace trpí tzv. „shower-door“ efektem – animace působí dojmem, že se odehrává za nepohyblivým reliéfním sklem. Pod pojmem grafická primitiva jsou považovány základní dvourozměrné objekty (úsečky,lomené čáry, kružnice, mnohoúhelníky, křivky...) Většinu z nich nalezneme jako základní útvary pro práci v kreslících programech.

Lepší dojem zanechávají techniky, které se snaží držet primitiva na místě relativně k jednotlivým objektům scény.

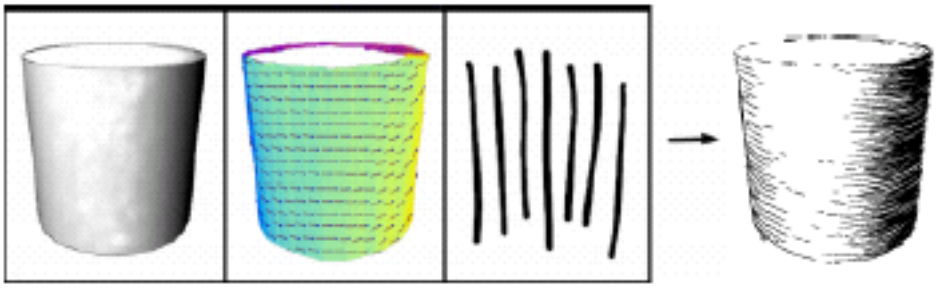
Kapitola 2

NPR metody

V této kapitole uvádím přehled několika NPR metod a jemný nástin na způsob, jakým přistupují ke zpracování obrazové informace. U každé z těchto metod jsem se snažil zobrazit ukázky ilustrující postupné zpracování jednotlivými metodami.

2.1 Image-space ilustrace se vzhledem perokresby pro textury

Systém se dělí dle formátu vstupu na 2 rozsáhlé kategorie: geometry-based systém, který má na vstupu informace ze 3D scény, a image-space systém – vstupem obraz v úrovních šedi.
Podle druhu vstupu se v této metodě zpracovává signál jiným způsobem. U 2D vstupu se

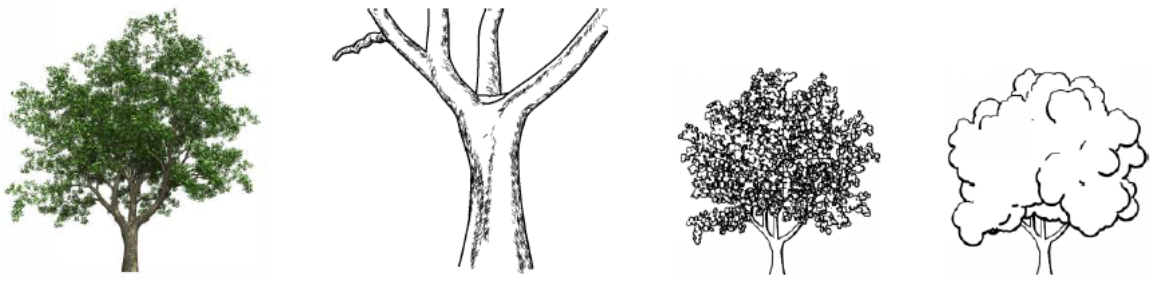


Obrázek 2.1: Jednotlivé vrstvy zleva stín, směr a množina vzorků. Vpravo výsledná ilustrace získáná složením komponent. (Přejato z [1])

výsledný produkt složí kombinací tří komponent. Šedotónní obraz – definuje stíny pro ilustraci. Směr – definuje orientaci textury v bodech. A množina vzorků, kterými se vykreslí konečná podoba ilustrace. Ukázka na obrázku 2.1. Uživatel si ale musí nadefinovat jednotlivé komponenty ručně. Narozdíl od 3D scény zvládá tato metoda u 2D vstupu ilustrovat i složitejší modely jako srst zvířat či lidský obličej. Podrobněji v [1].

Pro vstup lze použít jak 3D model tak i 2D obraz. Barevný obraz na vstupu můžeme použít jen za předpokladu, že bychom jej ještě před přímým zpracováním touto metodou převedli na šedotónní obraz.

Pro výsledné zpracování potřebujeme nadefinovat tři „obrazy“. Do tvorby těchto částí ale musí zasáhnout uživatel. Je to jediný systém, kde musí během vytváření ilustrace zasahovat uživatel. Všechny ostatní metody zde popsané generují ilustrace bez vnějšího zásahu. Kromě toho zde potřebujeme dva stejně velké buffery jako originální obraz a jeden menší, do kterých bychom mohli uložit jednotlivé vrstvy, z kterých se pak složí výsledná ilustrace.



(a) Model stromu (b) Ilustrace kmene (c) Ilustrace s prahováním 1000 (d) S prahováním 2000

Obrázek 2.2: Ukázka ilustrace metodou 2.2 (Přejato z [2])

2.2 Ilustrace stromů se vzhledem perokresby

Tato metoda byla navrhnutá pro ilustraci stromů. Ke zpracování potřebuje model stromu, vytvořeného např. systémem Xfrog[3]. Konečný model z tohoto systému je předzpracován a jsou vytvořeny dva soubory. V prvním je uložena geometrie kostry stromu. Ve druhém souboru jsou uloženy informace o listech – pozice a normálové vektory. V původním modelu stromu se většinou nachází velké množství listů, které je třeba zredukovat kvůli složitosti výpočtu.

Kmen a větve jsou jsou v ilustraci vykresleny známými NPR technikami. Mohli bychom proto použít i algoritmus z kapitoly 2.5.3. Jelikož se listí výrazně liší od zbývajících povrchů, musí být zpracováno odděleně od zbytku scény.

Každý list je vykreslen grafickými primitivy. Seřazením primitiv podle hloubky ve scéně se pak určí, které se ve výsledku vykreslí. Pro zjištění hloubky umístění primitiv je používán hloubkový buffer. Pro každý pixel se spočítá rozdíl se všemi okolními pixely. Jako výsledek se bere největší pozitivní rozdíl. Tato hodnota určuje, jak daleko před okolními pixely je daný pixel. Pro konečné vykreslení se pak použije prahování[4]. Všechny pixely, které mají rozdíl hloubky větší než daný práh, jsou použity k záverečnému vykreslení. Okolní vegetace může být vykreslena stejným způsobem. Podrobnější popis této metody je popsán v [2].

Pro tvorbu ilustrací touto metodou je jediným možným vstupem 3D model. Což je podle mě celkem nevýhoda. Buď potřebujeme mít k dispozici soubory s 3D modely stromů nebo si je musíme vytvořit pomocí nějakého programu pro tvorbu rostlin.

Nevýhodou je podle mého názoru způsob zpracování listí, který celý algoritmus nejspíše hodně zpomaluje. I když se podaří množství listů zredukovat, jsou všechny zbývající listy vykresleny jako nějaký polygon. V takto vzniklém obraze pak navíc porovnávám každý pixel se všemi okolními pixely. U modelu s mnoha listy to může celý postup zpracování zpomalit.

2.3 Perokresba pro parametrické povrchy

Metoda pro automatické generování ilustrace se vhledem perokresby pro mnohostěnné modely. Moc se neliší od tradičních fotorealistických metod. Vstupem je 3D model, jedno nebo více světel a nastavení kamery. Výpočet se zahájí zjištěním viditelných povrchů a polygonů se stínem použitím 3D BSP stromu. Podrobnější informace o BSP stromech jsou v [5]. Výsledkem tohoto procesu je množina konvexních polygonů seřazených podle hloubky. Ty jsou

pak použity k sestavení 2D BSP stromu a rovinné mapy prezentující viditelné povrchy scény. Poté se můžou vykreslit jednotlivé oblasti rovinné mapy použitím textury tvořené čarami.

Jediným možným vstupem pro tuto metodu je 3D model. Pro 2D vstup by se tato metoda upravit nedala, protože bychom tak přišli o důležité informace, pomocí kterých se počítá výsledná ilustrace. Zjištění viditelných částí je řešeno jednoduchým způsobem za pomoci BSP stromu. Pokud modelem nemanipulujeme, je třídění pomocí BSP stromu rychlé.

Tuto metodu bych označil jako za nejsložitější ze všech, které v této práci zmiňuji. Složitostí bych ji nejspíše mohl srovnávat s tradičními fotorealistickými metodami.

Obrázek 2.3: Ukázka ilustrace metodou 2.3 (Přejato z [6])

2.4 Automatické NPR odstraněním měkkých stínů

Tato metoda narozdíl od ostatních zmiňovaných v této práci negeneruje binární obraz, ale barevný. Odstraňuje měkké stíny a přidává černé kontury podél objektů. První část zaměřená na odstranění stínů je založena na rozdělení původního obrazu. Na obraz s jasovou složku a dva barevné obrazy, které odpovídají barevným kanálům červená-zelená (RG) a modrá-žlutá(BY). Původní RGB obraz se převede na LMS prostor (kde L, M, S jsou dlouhé, střední a krátké vlnové délky). Pomocí LMS prostoru se původní barevný obraz převede na dané tři obrazy. Dále jsou nalezeny hrany v RG a BY obrazech a provedena jejich kombinace pomocí logického součtu. V dalším kroku se odvodí obraz klasifikovaný podle hran nalezených v předcházejícím kroku.

Obrázek 2.4: Originální obrázek a obrázek upravený metodou 2.4

Následuje vrácení barev do obrazu, ale už s odstraněnými měkkými stíny. Jako poslední krok se k tomuto obrazu přidají kontury objektů. Výsledný efekt je lépe vidět na obrázku 2.4.

Je to jediná metoda v této práci, která generuje barevný obraz. Ke zpracování používá 2D obraz, ale mohli bychom použít i 3D model a vyrenderovat z něj 2D obraz. Jelikož se jedná o NPR metodu pro generování barevného obrazu, tak oproti všem zde zmiňovaným přístupům používá všechny barevné kanály. Vzdáleně bych ji mohl přirovnat k metodě z kapitoly 2.1, která výsledný obraz také skládá z několika rozdílných vrstev získaných z původní předlohy. Metoda je podrobněji popsána v [7]

2.5 Ilustrace pomocí rychlé distribuce kreslících primitiv

Jednou z NPR metod je i metoda ilustrace pomocí rychlé distribuce kreslících primitiv, kterou jsem měl v rámci této práce nastudovat a vytvořit podle ní jednoduchou funkční aplikaci. Základní popis metody je převzat z [8] a je doplněn textem, který podrobněji popisuje způsob, jak algoritmus funguje.

2.5.1 Odvození algoritmu pro vytváření ilustrace

Kreslící proces by měl vytvořit binární obraz, kde pravděpodobnost, že pixel x_i je vykreslen, je identická k intensitě odpovídajícímu pixelu v originálním obraze, tj. $p(x_i = 1) = I(x_i)$. Navíc bychom chtěli nezávisle umístit vzorky pro generování tohoto binárního obrazu. Potřebujeme najít PDF (hustotu pravděpodobnosti - v angl. probability density function) $q(x)$, která by mohla být použita algoritmem z podkapitoly 2.5.3.[8]

Po umístění N vzorků podle PDF $q(x)$ je pravděpodobnost, že pixel x_i není vykreslen, $(1 - q(x_i))^N$. Využijeme ale opačného jevu, kdy pixel x_i vykreslen je.

$$\begin{aligned}(1 - q(x_i))^N &= p(x_i = 1) \\ q(x_i) &= 1 - \sqrt[N]{1 - p(x_i = 1)}\end{aligned} \tag{1}$$

pro všechna $i = 1...n$, kde n je počet pixelů v obraze.
Potřebujeme, abychom integrací $q(x_i)$ získali jedničku a dodrželi tak definici CDF (distribuční funkce - v angl. cumulative density function). Kombinací těchto omezení a předchozí rovnice získáme

$$\sum_{i=1}^{n} q(x_i) = 1 \Leftrightarrow ... \Leftrightarrow \sum_{i=1}^{n} \sqrt[N]{1 - p(x_i = 1)} = n - 1 \tag{2}$$

Pro obecné obrazy nemůže být tento systém rovnic řešen přímo, ale je zapotřebí použít nějakou numerickou metodu, např. binární hledání. Binární hledání nemusí být počítáno přes celou sumu. Každý term v sumě závisí pouze na pravděpodobnosti $p(x_i = 1)$ každého pixelu, která závisí na intensitě $I(x_i)$ vstupního obrazu, která je obvykle kvantifikována, např. pro 256 odstínů šedé. Jestliže označíme kvantifikované odstíny jako $I_j, j = 1...K$ a histogram vstupního obrazu jako $H(I_j)$, můžeme rovnice 1 a 2 přepsat jako

$$q(I_j) = 1 - \sqrt[N]{1 - I_j} \tag{3}$$

a

$$\sum_{j=1}^{K} H(I_j) \sqrt[N]{1 - I_j} = n - 1 \tag{4}$$

které zahrnují sumu jen přes K odstínů a nikoliv přes n pixelů.[8]
Výsledný program můžeme použitím histogramu zrychlit v řádu několika sekund.

2.5.2 Generování náhodných bodů

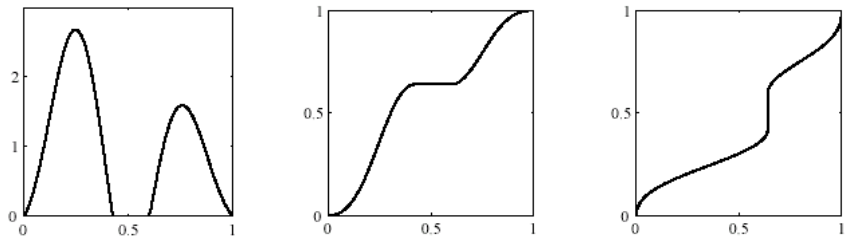
Pomocí 1D PDF

Je dána jednodimenzionální PDF $p(x), x \in [0, 1]$ a množina rovnoměrně distribuovaných náhodných vzorků x_i nad $[0, 1]$. Vzorky můžeme rozdistribuovat podle $p(x)$. Ukázka 1D PDF je na obrázku 2.6(a).[8]

Spočítáme distribuční funkci

$$C(x) = \int_0^x p(t)dt$$

jak ukazuje obrázek 2.6(b). Poté můžeme $C(x)$ invertovat a transformovat jednotlivé vzorky x_i na $x_i' = C^{-1}(x_i)$. Graficky je to znázorněno na obrázku 2.6(c) jako mapování množiny vzorků na ose y na osu x. Množina x_i znázorněná jako kolečka na ose x na obrázku 2.6(c) jsou distribuovány podle původní PDF $p(x)$. Počítání inverze CDF není úplně přímé, jsou-



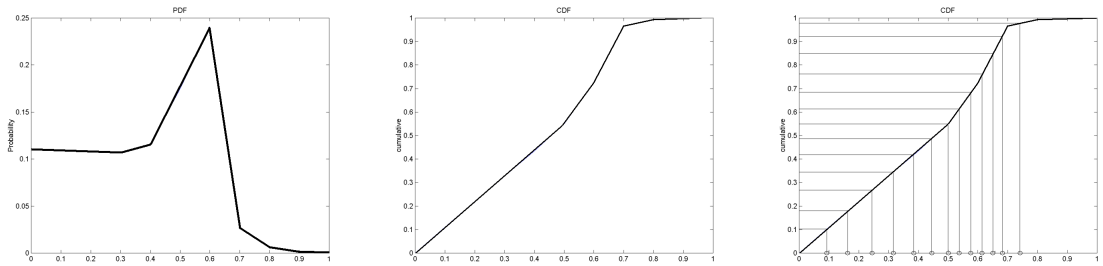
(a) PDF (b) CDF (c) Inverse 2.5(b)

Obrázek 2.5: Ukázka transformace s nulovými oblastmi v PDF. Obr. 2.5(c) není funkce. (Přejato z [9])

li totiž v PDF místa s nulovou pravděpodobností, integrují se v CDF na stejnou hodnotu. Obrázek 2.5.2 ukazuje daný problém. V této části pak neexistuje inverze. Daná funkce se může rozložit na několik částí, u nichž inverze neporušují podmínky pro funkce. Jednotlivé části se pak vyřeší samostatně. Další možností je daný úsek vynechat, ale v takovém případě se ve výsledném obraze nemusí daný řádek(sloupec) vůbec vykreslit.[8]

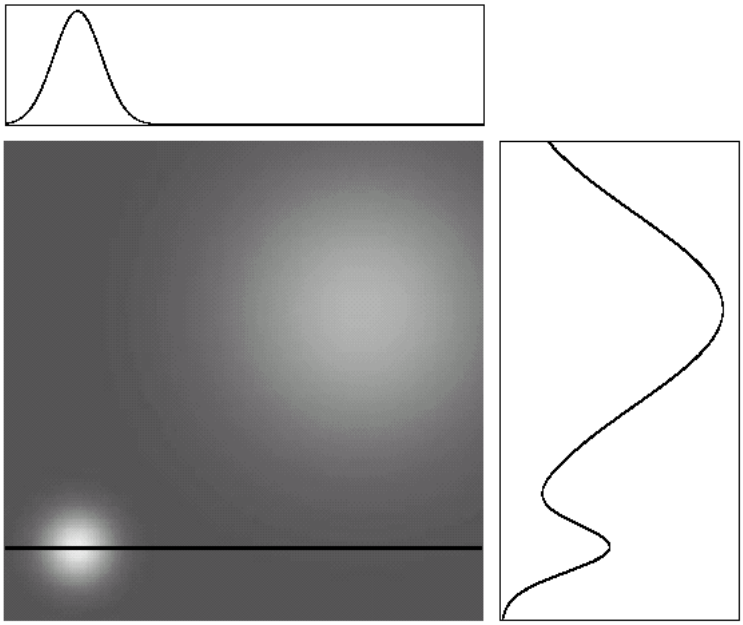
Pomocí 2D PDF

Je dána dvoudimenzionální PDF $p(x), x \in [0,1]^2$ a množina rovnoměrně distribuovaných náhodných bodů p_i nad $[0,1]^2$. Body můžeme rozdistribuovat podle $p(x)$ užitím transformační metody.[8]



(a) 1D PDF (b) 1D CDF (c) 1D CDF a její vzorkování

Obrázek 2.6: Ukázka redistribuce bodů podle 1D PDF.



Obrázek 2.7: 2D PDF a pravděpodobnost pro jednotlivé řádky (vpravo) a 1D PDF řádku (nahoře) (Přejato z [9])

K nalezení y-souřadnice q_i^y redistribuovaného bodu q_i spočítáme distribuční funkci

$$M(y) = \int_0^y m(t)dt, \text{ kde } m(y) = \int_0^1 p(x)dx$$

a získáme $q_i = M^{-1}(p_i^y)$. Funkce $M(y)$ je monotónní, ale ne striktně monotónní, jestliže některé řádky mají nulovou intenzitu. $M^{-1}(y)$ tedy existuje téměř vždy až na místa isolovaných bodů.[8]

Máme-li spočteno q_i^y, můžeme určit x-ové souřadnice q_i^x podle PDF, respektivě podle řádky. Matematicky můžeme daný vztah vyjádřit jako podmíněnou PDF

$$c(x|q_{i,y}) = \frac{p(x, q_{i,y})}{m(q_{i,y})}$$

a její CDF

$$C(x|q_{i,y}) = \int_0^x c(s|q_{i,y})ds$$

Stejně jako předtím, x-ová souřadnice nového bodu je dána inverzní funkcí $q_i^x = C^{-1}(p_i^x|q_{i,y})$. Pro diskrétní PDF $q(x)$, stejně jako při odvození z obrazu, můžeme $M(y)$ a $C(x|y)$ jednoduše předpočítat jako 1D a 2D pole. Jednoduchým hledáním v poli pak můžeme najít hledané souřadnice. Obrázek 2.5.2 ukazuje 2D PDF, kde tmavé oblasti jsou mapovány na nízké pravděpodobnosti a světlé oblasti na vysoké pravděpodobnosti. Vpravo je marginalní funkce $m(y)$ získaná integrací jednotlivých řádků. Nahoře je jedna z mnoha řádkových PDF, $c(x|y)$ pro nějaké y znázorněné černou čarou v obrázku.[8]

2.5.3 Algoritmus

Shrnutím předchozí teorie můžeme přepsat algoritmus do 7 základních kroků.

1. Načtení vstupního obrazu.

2. Vytvoření histogramu intezit obrazu.

3. Nalezení N, počtu primitiv, které se budou vykreslovat, použitím rovnice 4.
   Pro spočtení N je zapotřebí použít některou z numerických metod (půlení intervalu, metoda sečen...)

4. Spočtení PDF $q(I_j), j = 1...K$, kde K je počet úrovní intenzit, použitím rovnice 3.
   Za N se dosadí hodnota spočtená v předcházejícím kroku.

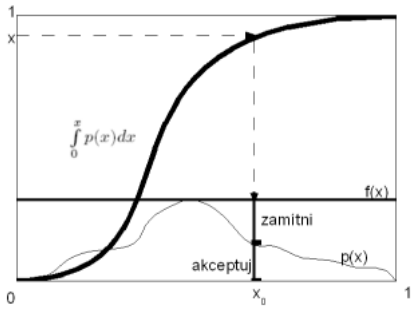
5. Integrace řádků a invertování na formu M^{-1}.

6. Integrace a invertování na formu C^{-1}.

7. Distribuce N primitiv podle M^{-1} a C^{-1}.
   Primitiva jsou distribuovány podle původní PDF. Pro umístění primitiv podle PDF se použije zamítací metoda [11] popsaná dále.

2.5.4 Zamítací metoda

Po invertování na M^{-1} a C^{-1} je potřeba dané body redistribuovat podle původní PDF. K tomu nám dobře poslouží zamítací metoda (v angl. rejection method).



Obrázek 2.8: Zamítací metoda pro generování náhodné proměnné x ze známé PDF $p(x)$, která je menší než nějaká funkce $f(x)$. Nejdříve se najde nová hodnota x_0. Ve druhém kroku pro x_0 vygeneruju náhodnou hodnotu $l : l < f(x_0)$. Pokud $l < p(x_0)$, dané x_0 akceptuji, jinak ho zamítnu.

Mějme novou funkci $f(x)$ takovou, že $p(x) \leq R.f(x)$ pro konstantu R. Pro použití v algoritmu zvolíme $R = max(p(x))$. Pro x najdeme pomocí inverze x_0. Pro redistibuovaný bod x_0 náhodně vybereme číslo $l : l \in [0, R]$. Pokud $p(x_0) < l$ bod můžeme vykreslit, v opačném případě bod zahodíme. Princip metody je lépe vidět na obrázku 2.8.

2.5.5 Kreslící styly

Tento algoritmus může být použitý k rozdílným kreslícím stylům. V základním návrhu potřebuje algoritmus na vstupu jen šedotónní obraz, aby se vygenerovalo PDF.
Pokud se použije místo obrazu na vstupu 3D model, můžeme do obrazu zakódovat dodatečné informace a využít je potom k rozdílným kreslícím stylům.
Následující tři popsané styly se dají aplikovat i na 2D obraz.

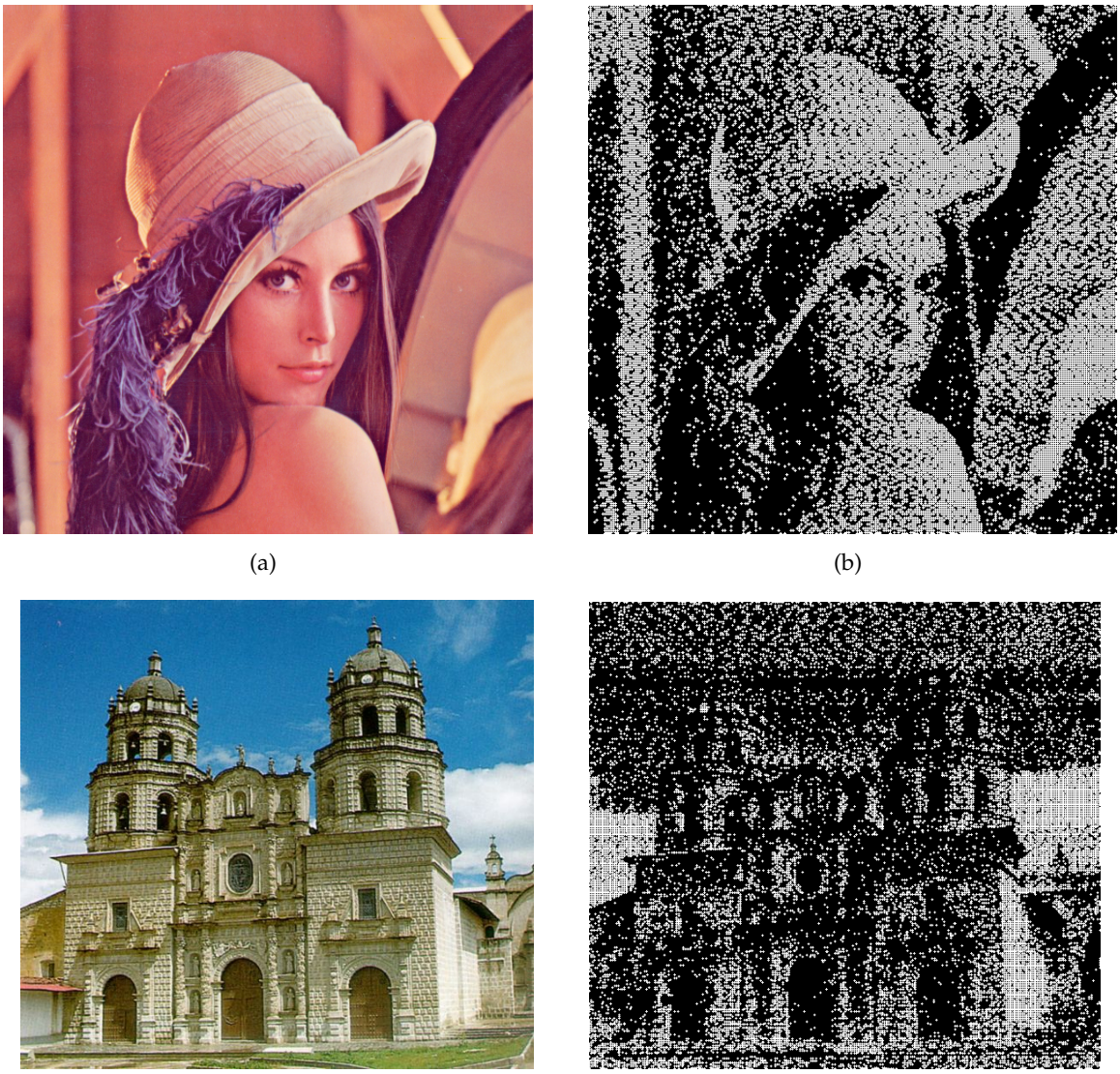
Tečkování - v angl. point stippling

Je to základní styl na renderování ilustrace popsanou metodou, který nepotřebuje z obrazu získávat žádné dodatečné informace.
Ilustraci vykreslenou tímto stylem získáme následujícím způsobem:

1. Načtení vstupního obrazu.
2. Sestavení histogramu vstupního obrazu.
3. Spočtení N. Pro rychlejší výpočet se bere v potaz váha jednotlivých intenzit z histogramu.
   N nejde z rovnice 4 vyjádřit, proto je potřeba použít některou z numerických metod na jeho spočtení, např. metodu půlení intervalu.
4. Spočtení PDF $q(I_j), j = 1..K$, kde K je počet odstínů šedé ve vstupním obraze pomocí rovnice 3. Za N se dosadí hodnota získaná v předcházejícím kroce.
5. Vypočtení marginálních funkcí pro jednotlivé řádky - $\int_0^1 p(x)dx$. Budou potřeba v následujících krocích.
6. Spočítání distribuční funkce $M(y) = \int_0^y m(t)dt$, kde $m(y)$ jsou hodnoty marginálních funkcí z předchozího kroku.
7. Spočítání podmíněné distribuční funkce $C(x|q_{i,y}) = \int_0^x c(s|q_{i,y})ds$. PDF daného bodu se dělí hodnotou marginální funkce pro řádek, na kterém se daný bod nachází.
8. Distribuce primitiv podle původní PDF $q(x)$.
   Pro každý bod v původním obraze nalezneme jeho nové souřadnice pomocí inverze M^{-1} a C^{-1}. Nové souřadnice nalezneme jako $y_0 = M^{-1}(y)$ a $x_0 = C^{-1}(x|y_0)$, kde x a y jsou převedeny na interval $[0, 1]$.
   Nejdříve se zjistí nová souřadnice y_0 a teprve potom můžeme vypočítat i novou souřadnici x_0.
   Jakmile máme vypočteny nové souřadnice, musíme podle původní PDF rozhodnout, jestli se daný bod vykreslí nebo ne. K tomu se použije zamítací metoda popsaná výše.

Šrafování - v angl. hatching

U tohoto stylu se používají čáry rozdílné tloušťky a délky. Orientace čar se volí konstantně.
Postup pro získání obrazu vykresleného tímto stylem je stejný jako u tečkování s jediným rozdílem, místo bodů se používají čáry se středem x_0, y_0.



(a) (b)

(c) (d)

Obrázek 2.9: Ukázka po aplikování algoritmu.

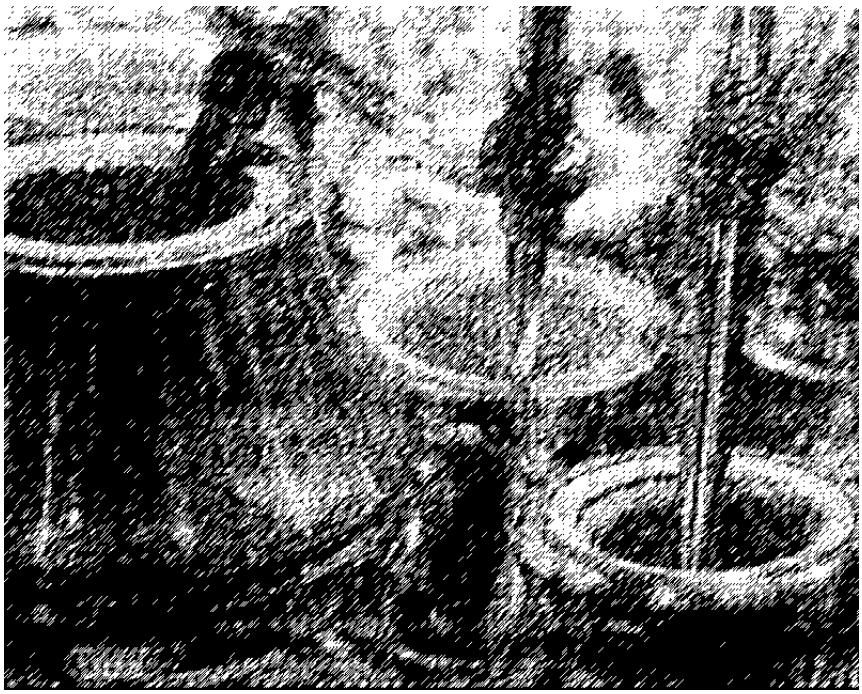
Šrafování s různou orientací čar - v angl. cross hatching

Tento styl je velice podobný předchozímu. Pouze u čar není pevně dána orientace, ta se spočítá pro každou vykreslovanou čáru zvlášť z původního obrazu.
Pro zjištění orientace můžeme použít sobelův operátor [10]. Spočítáme gradient ve směru x a ve směru y.

$$G_x = \begin{vmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{vmatrix} * A \text{ a } G_y = \begin{vmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{vmatrix} * A.$$

Úhel α spočítám jako $\alpha = \frac{\pi}{2} - \arctan \frac{G_y}{G_x}$. Jednotlivé čáry se potom v obraze vykreslí natočené podle tohoto úhlu.

(a)



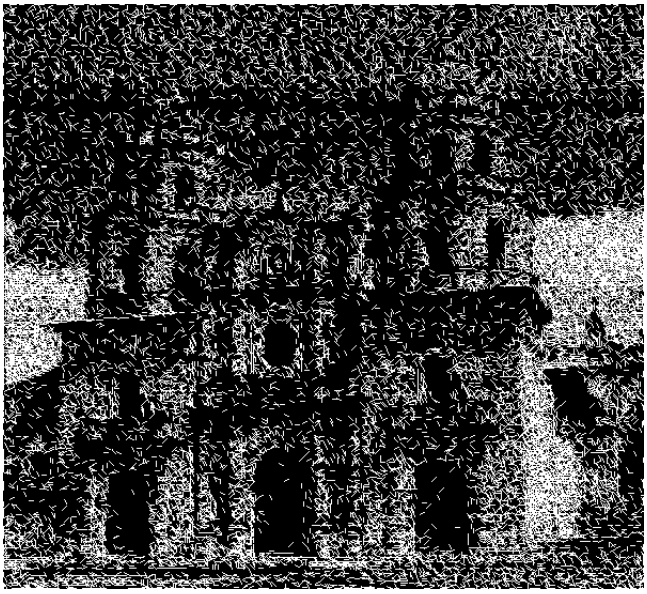
(b)

Obrázek 2.10: Ukázka po aplikování algoritmu - šrafování.

2.5.6 Korekce tónů

Při používání primitiv větších než pixel může docházet k překrývání jednotlivých bodů v generovaném obraze. Například při použití bodů o poloměru 3 pixelů ztrácíme velkou

(a)



(b)

Obrázek 2.11: Ukázka po aplikování algoritmu - šrafování s rozdílnou orientací čar.

část obrazové informace.

Aby se uchoval správný tón výstupního obrazu, musíme již vzít v úvahu vztahy mezi pixely. Převedeme původní postup do 2D. Místo $p(x_i)$ budeme používat $p(x_{i,j})$.
Po této úpravě můžeme přepsat pravděpodobnost na

$$p(x_{i,j} = 1) = 1 - \left(1 - \sum_{k=1}^{n_x} \sum_{l=1}^{n_y} q(x_{k,l}) s(x_{k-i,l-j})\right)^N \tag{5}$$

kde $i = 1...n_x, j = 1...n_y$ a $s(x_{k-i,l-j})$ je binární obraz získaný původním algoritmem.

Obrázek 2.12: Tečkování při nastavení poloměru 2

Hodnota $s(x)$ je tedy rovna jedné nebo nule. Dále je potřeba upravit rovnici 2, získáme

$$\sum_{k=1}^{n_x}\sum_{l=1}^{n_y} q(x_{k,l}) = 1. \tag{6}$$

Stejně jako v původním případě, musí být tyto dvě rovnice řešeny souběžně, abychom získali $q(x_{i,j})$ a N. Tento systém rovnic ale nemá řešení, pokud se ve vstupním obraze vyskytují místa, která jsou užší než použitá primitiva.
Přidáním podmínky $q(x_{k,l}) \geq 0$ lze docílit minimálních rozdílu mezi jednotlivými stranami rovnice 5. Rovnici 5 můžeme ještě aproximovat a získáme tak

$$p(x_{i,j} = 1) = 1 - \left(1 - \sum_{k=1}^{n_x}\sum_{l=1}^{n_y} q(x_{k,l}) s(x_{k-i,l-j})\right)^N \tag{7}$$

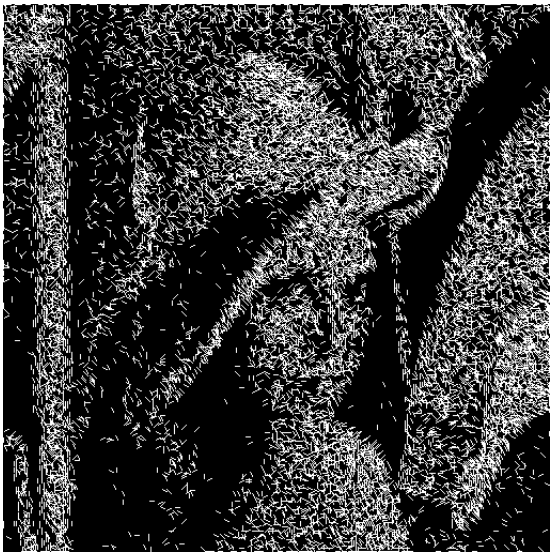
$$\approx 1 - \left(1 - q(x_{k,l}) \sum_{k=1}^{n_x}\sum_{l=1}^{n_y} s(x_{k-i,l-j})\right)^N \tag{8}$$

$$= 1 - (1 - q(x_{k,l}) s_{i,j})^N \tag{9}$$

kde $s_{i,j} \equiv \sum_{k=1}^{n_x}\sum_{l=1}^{n_y} s(x_{k-i,l-j})$. $s_{i,j}$ představuje počet pixelů (x_k, y_l), které mohou způsobit, že se díky nim vykreslí i pixel $(x_{i,j})$.[8]

Pro aplikování korekce tónů je zapotřebí nejprve provést původní algoritmus. Místo vykreslování obrazu budeme ale počítat $s_{i,j}$. Jakmile známe $s_{i,j}$, můžeme jej použít k vypočtení

(a)



(b)

(c)

Obrázek 2.13: Ukázka při použití korekce tónů.

nového N - počet primitiv k vykreslení.

$$\sum_{k=1}^{n_x}\sum_{l=1}^{n_y}\frac{1-\sqrt[N]{1-I_{i,j}}}{s_{i,j}}=1 \tag{10}$$

N stejně jako v původním algoritmu nejde vypočítat přímo, ale je opět nutné použít numerickou metodu, např. metodu půlení intervalu. Poté dosadíme nově spočtené N do rovnice 9 a spočítáme z ní $q_{k,l}$. Další postup se shoduje s původním algoritmem. Spočítání marginální funkce, distribuční funkce... s jediným rozdílem, místo $q(I_j)$ se použije $q(x_{k,l})$.

2.5.7 Srovnání metod

V následující tabulce je shrnutí a porovnání předchozích metod.

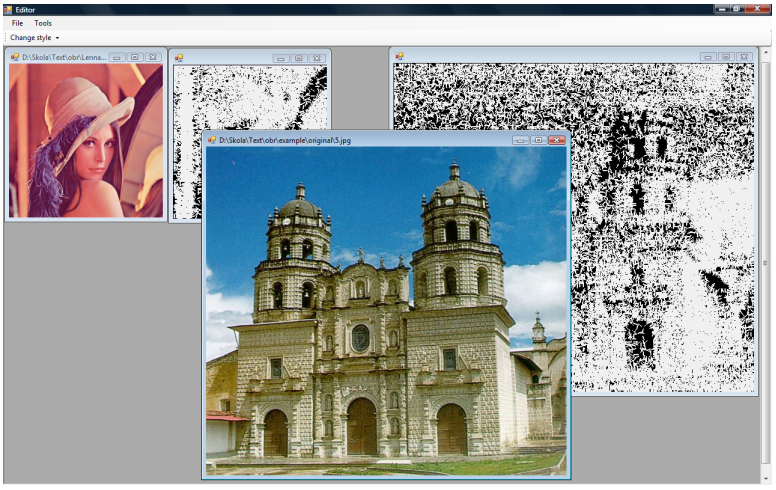
Metoda	**Druh vstupu**	**Zásah uživatele**	**Druh výstupu**
Image-space ilustrace se vzhledem perokresby pro textury	2D, 3D	ano	bin. obraz
Ilustrace stromů se vzhledem perokresby	3D model stromu	ne	bin. obraz
Perokresba pro parametrické povrchy	3D	ne	bin. obraz
Automatické NPR odstraněním měkkých stínů	2D	ne	RGB obraz
Ilustrace pomocí rychlé distribuce kreslících primitiv	2D, 3D	ne	bin. obraz

Z tabulky je vidět, že pro vstup se mohou používat 2D obrazy i 3D modely. Většina metod pak funguje bez zásahu uživatele a generuje pouze binární obrazy.

Kapitola 3

Realizovaný program

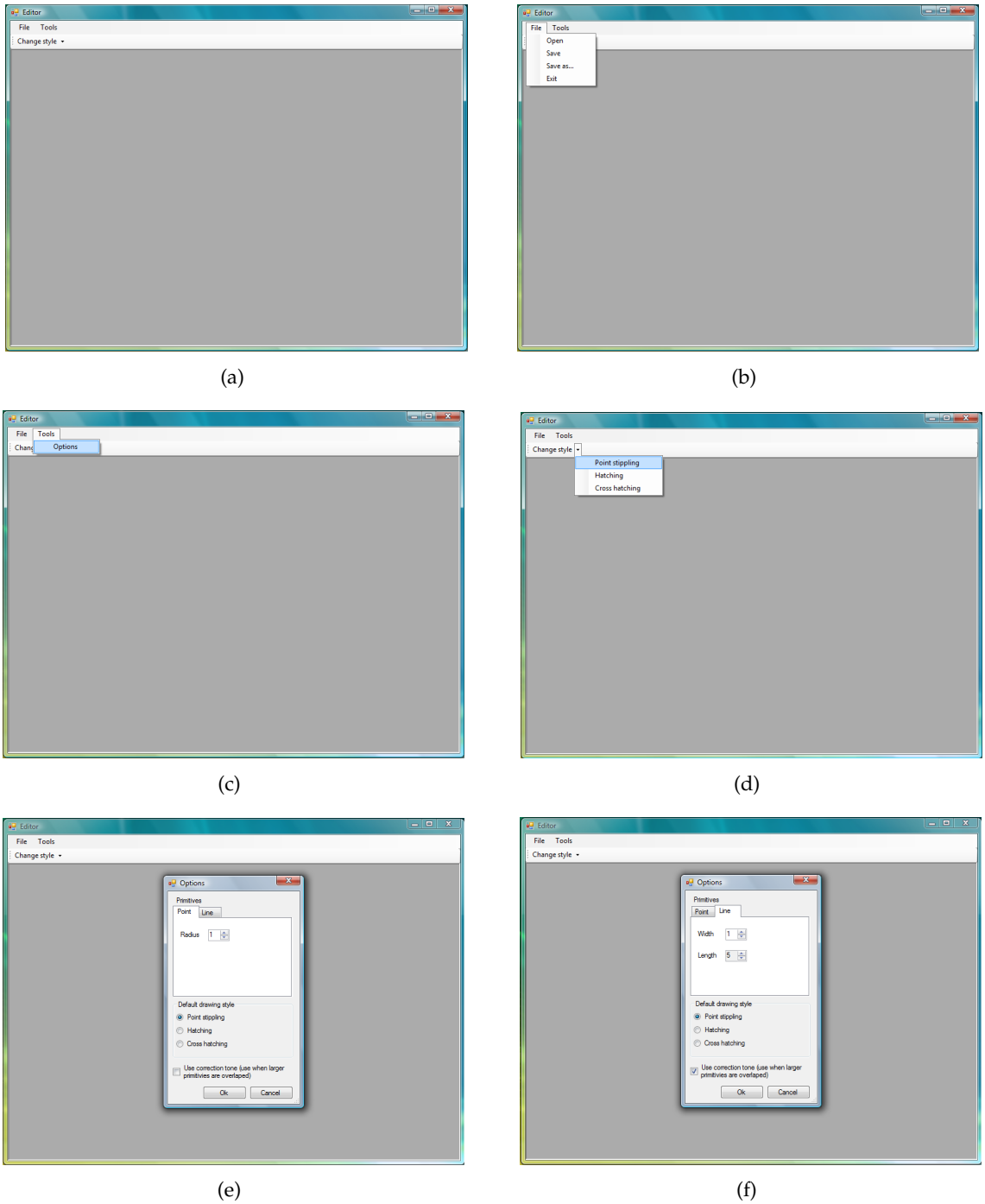
Důležitou částí aplikace je okno pro nastavení stylů 3.2(e) a 3.2(f). Horní polovina slouží pro nastavení jednotlivých primitiv, pro bod se nastavuje poloměr. Velikost 0 zde znamená, že se jedná o pouhý pixel.
U čáry(úsečky) se nastavuje její šířka a délka.
V dolní části okna nastavení je možnost si vybrat jeden ze tří stylů vykreslování(tečkování, šrafování a šrafování s různou orientací úseček). Pod přehledem stylů je možnost zašktrnout použití korekce tónů, která slouží k přepočítání výsledné ilustrace, pokud se primitiva v obraze překrývají a v ilustraci nelze řádně rozeznat původní obrázek. Tato volba ale zpracování vstupu zpomaluje.
Pokud chceme pro danou ilustraci změnit styl kreslení a nechceme přenastavovat celou aplikaci přes Tools/Options, použijeme Change style z Toolbaru. Aplikace znovu vykreslí ilustraci podle daného stylu pro aktivní okno.



Obrázek 3.1: Aplikace s hotovou ilustrací.

Aplikace pro vstup používá pouze obrázky typu png, jpeg, gif a bmp. Na výstup jsou použity formáty png a gif. Formát jpeg jsem nepoužil kvůli ztrátové kompresi.

Pro kvalitnější výstupní obraz doporučuji používat obrázky větších rozměrů(s menší velikostí obrázku klesá kvalita výstupu) a velikost primitiv nevolit příliš velikou.



(a) (b)

(c) (d)

(e) (f)

Obrázek 3.2: Ukázka aplikace.

Kapitola 4

Závěr

Při zpracovávání této práce jsem zíslal základní znalosti o jedné z mnoha metod pro nefotorealistické zobrazování, na jejichž základě jsem byl schopen vytvořit jednoduchou aplikaci pro tvorbu ilustrací.
Při zpracování jsem se naučil i další metody z počítačové grafiky, např. vykreslování kružnice pomocí Bresenhamova algoritmu, vzplňování ohraničeného prostoru...

Dále jsem měl v rámci práce příležitost porovnat tuto metodu s několika dalšími. Zadaná metoda se mi na rozdíl od ostatních metod zmiňovaných v této práci, zdála asi jako nejjednodušší. Některé z NRP metod bych mohl složitostí přirovnat i k fotorealistickým metodám zobrazování.

Ukázky výstupu z aplikace jsou na přiloženém CD ve složce samples.

Literatura

[1] *Salisbury M. P., Wong M. T., Hughes J. F., Salesin D. H.*, Orientable Textures for Image-Based Pen-and-Ink Illustration
University of Washington, 1997
Dokument dostupný na http://www.cs.brown.edu/courses/cs224/papers/orient.pdf (leden 2008)

[2] *Deussen O., Strothotte T.*, Computer-Generated Pen-and-Ink Illustration of Trees
Faculty of Computer Science, University of Magdeburg, 2000

[3] *Greenworks GbR.*, Domovská stránka systému xfrog http://www.greenworks.de

[4] http://cs.wikipedia.org/wiki/Prahování (leden 2008)

[5] *Žára J., Beneš B., Sochor J., Felkel P.*, Moderní počítačová grafika (2. vydání)
Computer Press, Brno, 2004

[6] *Winkenbach G., Salesin D. H.*, Rendering Parametric Surfaces in Pen and Ink, Department of Computer Science & Engineering
University of Washington, Seattle, 1996

[7] *Olmos A., Kingdom F. A. A.*, Automatic non-photorealistic rendering through soft-shading removal: a colour-vision approach
2nd International Conference on Vision, Video and Graphics (VVG '05), pp. 203–208, Edinburgh, 2005
Dokument dostupný na http://www.eg.org/EG/DL/PE/VVG/VVG05/203-207.pdf.abstract.pdf (leden 2008)

[8] *Secord A., Heidrich W., Streit L.*, Fast Primitive Distribution for Illustration
Department of Computer Science, The University of British Columbia, Vancouver, 2002
Dokument dostupný na http://www.cs.ubc.ca/ heidrich/Papers/RW.02.pdf (leden 2008)

[9] *Secord A. J.*, Random Marks on Paper Non-Photorealistic Rendering with Small Primitives
B.Math., University of Waterloo, 2000

[10] http://en.wikipedia.org/wiki/Sobel_operator (leden 2008)

[11] *Press W. H., Teukolsky S. A., Vetterling W. T., Flannery B. P.*, Numerical Recipes in C. The Art of Scientific Computing, Second Edition
Press Syndicate of the University of Cambridge, 1993

Příloha A

Obsah přiloženého CD

- Zdrojový kód bakalářské práce ve formátu pro LaTeXa samotná práce ve formátu pdf
- Zdrojové kódy aplikace pro Visual C++ 2005
- Ukázkový výstup aplikace
- Zkompilovaná verze aplikace

A.1 Zdrojový kód

```
//Zakladni funkce volana pro redistribuci primitiv.
Bitmap^ redistribute(System::Drawing::Bitmap^ bmp)
{
  Thread^ oThread = gcnew Thread( gcnew ParameterizedThreadStart( &
    Thread_wait_form::ThreadProc ) );
      oThread->Start();

  this->setBitmap(bmp);
  this->setHistogram();
  this->setN(false);
  this->setQij();
  this->setMy(false);
  this->setCxy(false);

  oThread->Abort();

  return this->draw_into_array();
}

/**
 * spocita pocet primitiv na redistribuci
 * pocet omezen intervalem 20%-100% vsech pixelu
 */
void setN(bool correction)
{
  int n;
  int min = 0;
  int max;

  if(correction)
  {
    //pri korekci tonu
    max = this->N; // max. pocet primitiv na zobrazeni, maximalne jako puvodni N
    n = 1;
  }
  else
  {
    n = this->bitmap->Width * this->bitmap->Height - 1; // pocet pixelu - 1
    max = n; // max. pocet primitiv na zobrazeni
    int min = (int)Math::Round(n*0.2f); // min. pocet primitiv na zobrazeni
    int max = min;
  }

  int mid;
```

```
while(min < max)
{
  double sum = 0;
  mid = (min+max)/2;

  if (correction)
  {
    // spocitam nove N - pro korekci tonu
    for(int x = 0; x < this->bitmap->Width; x++)
    {
      for(int y = 0; y < this->bitmap->Height; y++)
      {
        if (this->Sij[x, y] > 0)
        {
          double intensity = (double)(this->getIntensity(x, y))/(K-1);
          double root = Math::Pow(1.0 - intensity, 1.0 / (double)mid);
          sum += (1.0 - root)/this->Sij[x, y];
        }
      }
    }
  }
  else
  {
    for(int i = 0; i < K; i++)
    {
      sum += this->histogram[i] * Math::Pow(1-(double)i/(K-1), 1.0/(double)mid);
    }
  }

  if (correction)
  {
    if (sum < n)
    {
      max = mid - 1;
    }
    else
    {
      min = mid + 1;
    }
  }
  else
  {
    if (sum < n)
    {
      min = mid + 1;
    }
```

```
        else
        {
          max = mid - 1;
        }
      }
    }
    this->N = mid;
}


/**
 * spocita PDF
 */
void setQij ()
{
    double base;

    this->Qij = gcnew array<double>(K);

    for(int i = 0; i < K; i++)
    {
      base = 1.0 - ((double)i / (K-1));
      this->Qij[i] = 1 - Math::Pow(base,1.0/(double)this->N);
    }
}


/**
 * spocita CDF pro redistribuci primitiv na ose Y
 * soucasne spocita i marginal f. -> usetreni jednoho pruchodu pres vysku obrazku
 */
void setMy(bool correction)
{
    this->marginal = gcnew array<double>(this->bitmap->Height); // init pole
    this->marginal_max = 0.0; // init max. hodnoty

    this->My = gcnew array<double>(this->bitmap->Height); // init pole

    //init pro prvni radek
    this->marginal[0] = 0;
    for(int x = 0; x < this->bitmap->Width; x++)
    {
      if(correction)
      {
        this->marginal[0] += this->CQij[x, 0];
      }
      else
```

```
    {
      this->marginal[0] += this->Qij[this->getIntensity(x, 0)];
    }

    this->marginal_max = this->marginal[0];
  }

  this->My[0] = this->marginal[0]; // konec prvni radek

  for(int y = 1; y < this->bitmap->Height; y++)
  {
    this->marginal[y] = 0;
    for(int x = 0; x < this->bitmap->Width; x++)
    {
      if(correction)
      {
        this->marginal[y] += this->CQij[x, y];
      }
      else
      {
        this->marginal[y] += this->Qij[this->getIntensity(x, y)];
      }
    }

    // najde max. hodnotu
    if(this->marginal[y] > this->marginal_max)
    {
      this->marginal_max = this->marginal[y];
    }

    //spocitam CDF
    this->My[y] = this->My[y-1]+this->marginal[y];
  }
}


/**
 * spocita  cumulative density function
 */
void setCxy(bool correction)
{
  this->Cxy = gcnew array<double, 2>(this->bitmap->Width, this->bitmap->
    Height); // init pole
  this->max_Cxy = gcnew array<double>(this->bitmap->Height);

  // init  prvniho sloupce
  for(int y = 0; y < this->bitmap->Height; y++)
```

```
{
  if(this->marginal[y]>0)
  {
    if(correction)
    {
      this->Cxy[0, y] = this->CQij[0, y]/this->marginal[y];
    }
    else
    {
      this->Cxy[0, y] = this->Qij[this->getIntensity(0, y)]/this->marginal[y];
    }
  }
  else
  {
    this->Cxy[0, y] = 0;
  }

  // init pro zjisteni max. hodnot cond. pdf na radcich
  this->max_Cxy[y] = 0;
}

for(int y = 0; y < this->bitmap->Height; y++)
{
  for(int x = 1; x < this->bitmap->Width; x++)
  {
    this->Sij[x, y] = 0; // vunuluju puvodni obraz
    if(this->marginal[y]>0)
    {
      double cond_px = 0.0;

      if(correction)
      {
        cond_px = this->CQij[x, y]/this->marginal[y];
      }
      else
      {
        cond_px = this->Qij[this->getIntensity(x, y)]/this->marginal[y];
      }

      if(cond_px > this->max_Cxy[y])
      {
        this->max_Cxy[y] = cond_px;
      }

      this->Cxy[x, y] = this->Cxy[x - 1, y] + cond_px;
    }
    else
```

```
        {
          this->Cxy[x, y] = this->Cxy[x - 1, y];
        }
      }
    }
}


// zakladni vykresleni se provede vzdy
for(int y = 0; y < this->bitmap->Height; y++)
{
  for(int x = 0; x < this->bitmap->Width; x++)
  {
    new_y = this->searchRedistribution((double)y/(this->bitmap->Height - 1));
    if (! this->rejection_method(new_y))
    {
      continue;
    }

    new_x = this->searchRedistribution((double)x/(this->bitmap->Width - 1), new_y)
    ;
    if (this->rejection_method(new_x, new_y))
    {
      this->draw(new_x, new_y);
    }
  }
}
```