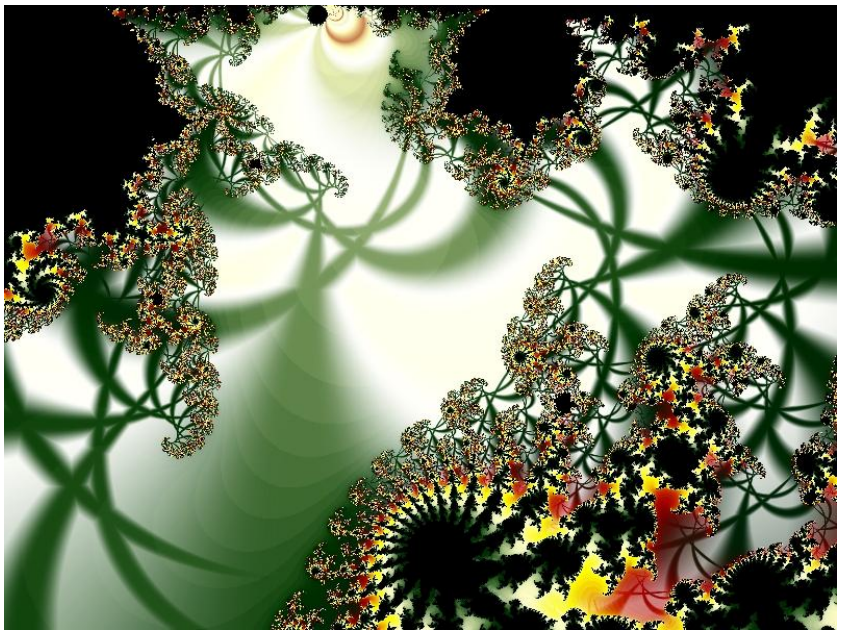
Základy počítačové grafiky



Jana Procházková, Vojtěch Vitásek

Obsah

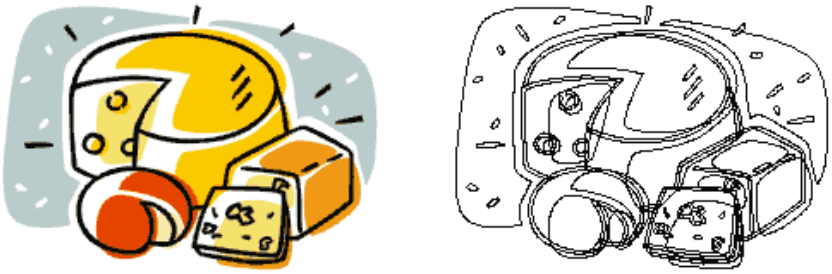
Kapitola 1

Grafická data

Grafická data se obvykle dělí na data vektorová a rastrová. Základní rozdíl je v způsobu popisu daného objektu. V případě vektorové grafiky jsou objekty popsány pomocí geometrických parametrů. U rastrových dat je objekt popsán jako shluk bodů. Úsečka popsaná rastrem je množina bodů o různých souřadnicích, které zobrazené dohromady tvoří obraz úsečky. Úsečka ve vektorovém zápisu bude obsahovat pouze počáteční a koncový bod a informaci o barvě a tloušťce.

1.1 Vektorová data

Vektorový grafický soubor obsahuje informace o objektech složených z křivek a jednoduchých těles, které umožňují jejich geometrickou konstrukci. Je-li takto uložena např. kružnice, soubor neobsahuje informace o všech jednotlivých bodech, které na ní leží. Informuje o tom, že se jedná o kružnici, dále obsahuje souřadnice jejího středu, jednoho bodu, který na ní leží a poslední bod určuje rovinu její konstrukce. Připojeny jsou rovněž informace o barvě objektu a tloušťce čáry, kterou má být sestrojen. Program, pro který jsou tato data určena, musí být schopen tyto informace správně přečíst a musí obsahovat algoritmus, který na základě těchto informací kružnici sestrojí. Vektorová data jsou typická např. pro technické výkresy. Z grafických softwaru je používá např. CorelDraw, Adobe Illustrator.



Obrázek 1.1: Obrázek je popsán pomocí vektorových dat jako soustava křivek

Výhody

- změnou měřítka nedochází ke ztrátě informace,
- výstupní soubory jsou poměrně malé,
- vhodné na tvorbu log, map apod., tedy obrázků, které se používají v různých rozměrech,
- jednoduché kreslení,
- výstupní soubory: .eps, .pdf, .pict (Mac)

Nevýhody

- některé čáry mohou zmizet při velmi velkém změnšení
- při velkém zvětšení jsou vidět chyby v kresbě

1.2 Rastrová data

Rastrová data dohromady tvoří obrázek. Ten se skládá z matice jednotlivých teček (pixelů). Každý má svoji vlastní barvu. Někdy bývají uloženy i informace o způsobu případné komprese a kódování barev. Rastrově jsou ukládány buď informace, které již nebudou dále upravovány systémem, kterým byly vytvořeny (např. žánrový pohled na strojní součást), nebo obrazy, které nebyly pořízeny počítačem (např. fotografie). Na rastrovém principu funguje většina zobrazovacích zařízení (monitory, jehličkové, inkoustové i laserové tiskárny, televize apod.).

Na obr. 1.2 je ukázka rastrového obrázku. V detailu je vidět složení obrázku ze čtverců s danou barvou. Lidské oko nedokáže rozlišit jednotlivé čtverce, proto na nás výsledek působí spojitě.

Počet pixelů záleží na daném obrázku, v jaké velikosti jej chceme tisknout či prezentovat. Aby se dalo porovnávat rozlišení obrázků, byla zavedena jednotka DPI - Dots Per Inch (počet bodů na palec). Rozlišení 800x600 může poskytovat vysoce kvalitní (malý) obraz na čtrnáctipalcovém monitoru a díky omezené rozlišovací schopnosti lidského oka zde již nemá smysl použít rozlišení větší. Totéž rozlišení však bude asi nedostatečné na monitoru jednadvacetipalcovém.

1.2.1 Typy rastrových obrázků

Rastrové obrázky mohou obsahovat libovolný počet barev, ale rozlišujeme čtyři základní kategorie.

1. Line-art. Tyto obrázky obsahují pouze dvě barvy, obvykle černou a bílou. Každý pixel je definová pouze jedním bitem (on=černá, off=bílá).

Obrázek 1.2: Rastrový obrázek

2. Odstíny šedi. Obsahuje paletu stínů šedé, také čistou černou či bílou.
3. Tónované. Tento obrázek obsahuje odstíny dvou či více barev. Nejoblíbenější je použití dvou barev, které se obvykle skládá z černé a druhé barvy.
4. Plně barevné. Barevná informace může být popsána užitím množství barevných prostorů: RGB, CMYK, atp. Více o nich budeme mluvit v kapitole 2.

Obrázek 1.3: 1. Line-art 2. Odstíny šedi 3. Tónované 4. Plně barevné

Výhody

- jednoduché pro výstup, pokud má tiskárna dostatečnou paměť
- výstupní soubory: .eps, .gif, .jpg, .tiff, .pict (Mac).

Nevýhody

- velká paměťová náročnost (A4 formát střední kvality bez komprese má 40 MB),
- nelze libovolně zvětšovat – dochází ke čtvercování,
- při zmenšení ztrácí ostrost.

1.3 Kontrolní otázky

1. Jaký je základní rozdíl vektorovými a rastrovými daty?
2. Jaká je hlavní výhoda užití vektorových dat?
3. Na jaký typ kreseb jsou vhodná vektorová data?
4. Jaké jsou typy rastrových obrázků?
5. Popište způsob uložení rastrových dat.
6. Jaké jsou nevýhody užití rastrových dat?
7. Co znamená zkratka DPI?

1.4 Literatura

www.prepressure.com/image/bitmapvector.htm
http://en.wikipedia.org/wiki/Vector_graphics
http://atlc.sourceforge.net/bmp.html -- popis bmp obrázků
http://www.fileformat.info/mirror/egff/ch01_03.htm

Kapitola 2

Barvy, barvy, barvičky

2.1 Vnímání barev

Světlo, které vnímáme, představuje viditelnou část elektromagnetického spektra. V něm se vyskytují všechny známé druhy záření, např. gama záření či infračervené záření. Lidské oko vnímá pouze oblast vlnových délek 380 až 720 nm. Uvnitř této oblasti vnímáme záření s určitou vlnovou délkou jako barvu.

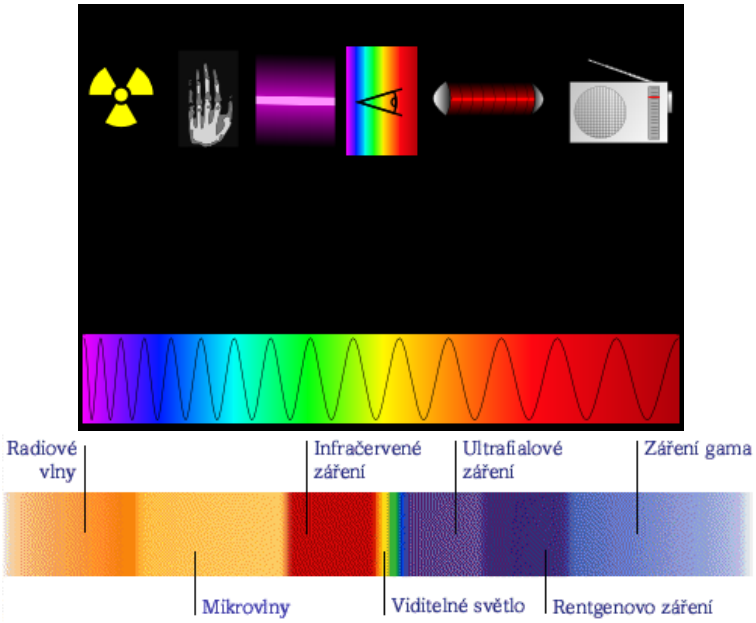
Jak tedy vnímáme barvy? Tak například, když bílé světlo dopadne na červený objekt, tento objekt absorbuje (pohltí) všechny složky bílého světla kromě červené. Objekt pak odráží záření v červené části spektra. Jak již v minulém století zjistil pan Heimholz, lidské oko má buňky (receptory), které jsou citlivé na červenou, zelenou a modrou barvu, tyto barvy jsou též základními složkami bílého světla. Kombinací různých úrovní těchto tří základních barev je lidský mozek schopen vnímat tisíce jemnějších barevných odstínů.

Elektromagnetické záření s kratšími vlnovými délkami nese více energie. V oblasti paprsků x nese záření dostatečnou energii, která umožňuje prostoupit větším objemem hmoty (např. lidské tělo). Toho se využívá v lékařství.

Také gama paprsky mají význam v lékařství. Obrazy získané pomocí gama záření se používají pro odhalení funkcí. Infračervené záření vydávají zahřáté objekty. Toho využívají moderní přístroje pro noční vidění.

Za světelný zdroj nejčastěji bereme Slunce, ale je jím i žárovka. Světelný zdroj vysílá paprsky všech frekvencí, které se skládají ve výsledné bílé světlo. Toto světlo se nazývá achromatické. Paprsky světla dopadají na objekty, některé frekvence se odrazí, jiné jsou pohlceny. Odražené světlo přijímá naše oko a mozek zpracovává a rozpozná výslednou barvu.

Pro popis světla se kromě frekvence používá ještě jas, sytost a světlost. Jas odpovídá intenzitě světla, sytost určuje čistotu barvy a světlost určuje velikost achromatické složky ve světle s určitou dominantní frekvencí (např. světle nebo tmavě modrá).



Obrázek 2.1: Barevné spektrum

2.1.1 Jak funguje lidské oko

Lidské oko je velmi složitý orgán. Příchozí obraz je promítán na sítnici. Tato fotocitlivá vrstva pokrývá dvě třetiny vnitřního povrchu oka. Obsahuje dva druhy receptorů: tyčinky a čípky. Tyčinek je více (120×10^6) a zajišťují noční vidění. Jsou také více citlivé. Čípky (8×10^6) nám zprostředkovávají barevné vidění. Tyčinky a čípky nejsou pravidelně rozmístěny po celé sítnici. Každý čípek je připojen k nervovému zakončení. Oproti tomu je vždy několik tyčinek propojeno pouze k jednomu nervu. Všechny signály z čípků a tyčinek jsou pomocí zrakového nervu předávány do mozku. Zajímavostí je, že po cestě informace do mozku dojde k rekombinaci informací. Z původních tří hodnot červená, zelená a modrá dojde do mozku informace o poměru červená-zelená, žlutá-modrá, zelená-červená.

2.2 Barevné modely

Při tisku nebo zobrazení obrázku jsou vytvářeny jeho barvy většinou pomocí tři složek - červená (red), zelená (green), modrá (blue), které nabývají většinou hodnot od 0 do 255. (V některých systémech může být rozmezí jiné, např. 0,1) To odpovídá kódování každé ze složek RGB v jednom bytu. Hodnota nula znamená, že daná barva není ve výsledné vůbec zastoupena, hodnota 255 znamená, že daná složka má největší intenzitu.

Máme tedy tři barvy (R,G,B) a jejich hodnotami můžeme určit $256^3 = 16777216$ barev. Lidské oko dokáže rozpoznat asi deset milionů barev, proto je tento počet dostatečný. Tyto barvy označujeme jako `true color`. V minulosti byla termínem

true color označována i zařízení o 15 resp 16 bitech na pixel (32768 resp. 65536 barev).

Pro reprodukci, tj. zobrazení barev se používají dvě metody: subtraktivní a aditivní míchání barev.

1. **Aditivní metoda**

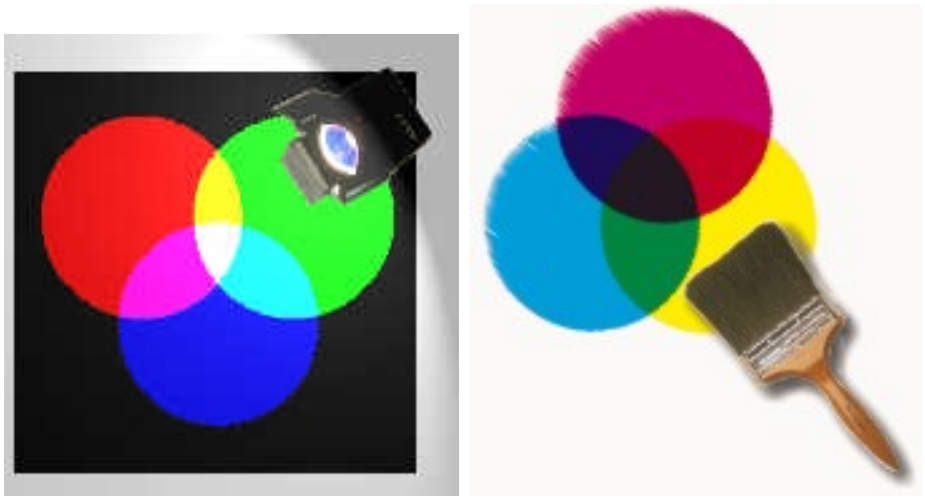
Když použijeme červené, zelené a modré světlo (RGB) používáme aditivní míchání barev. Termín aditivní – doplňkové znamená, že začneme s černým pozadím a přidáváme jednotlivé barevné RGB složky pro vytvoření konkrétního barevného odstínu. Kombinací všech tří RGB barev vznikne bílá barva. Barevná televize, počítačový monitor, scanner a jevištní osvětlovací technika pracuje tímto způsobem. Složením (kombinací) tří základních barev vzniknou další barvy.

2.**Subtraktivní metoda**

Subtraktivní míchání barev je opačný k aditivnímu. Začínáme s bílým pozadím a přidáváme barvy Cyan, Magenta a Yellow (zkratka CMY), abychom jejich překrytím vytvořili černou barvu. Subtraktivní metoda se používá v tiskovém průmyslu, kde se obvykle tiskne na bílý papír. CMY pigmenty absorbují (odečítají) konkrétní vlnové délky dopadajícího světla a naopak dovolují ostatním, aby se od potištěného papíru odrazily. Tak tedy Cyan absorbuje (pohltí - odečte) červenou část dopadajícího světla. Magenta absorbuje zelené světlo a Yellow absorbuje modré světlo. Když sloučíme subtraktivní barvy získáme opět základní aditivní barvy.

Černá barva, která vzniká smísením všech složek CMY však není čistě černá, jedná se spíše o tmavě hnědou. Z tohoto důvodu se k složkám CMY přidává samostatná černá barva K (blacK). Vzniká tak prostor CMYK.

Obrázek 2.2: Citlivost lidského oka na barvy



Obrázek 2.3: Barevný systém RGB

Uživatelsky však zadávání barev pomocí prostorů RGB nebo CMY není snadné. Odhadnout, které barvy a v jakém poměru smíchat k výslednému odstínu není vůbec intuitivní, proto se zavedly další barevné modely pro snažší práci s barvami (využívají se v grafických softwarech).

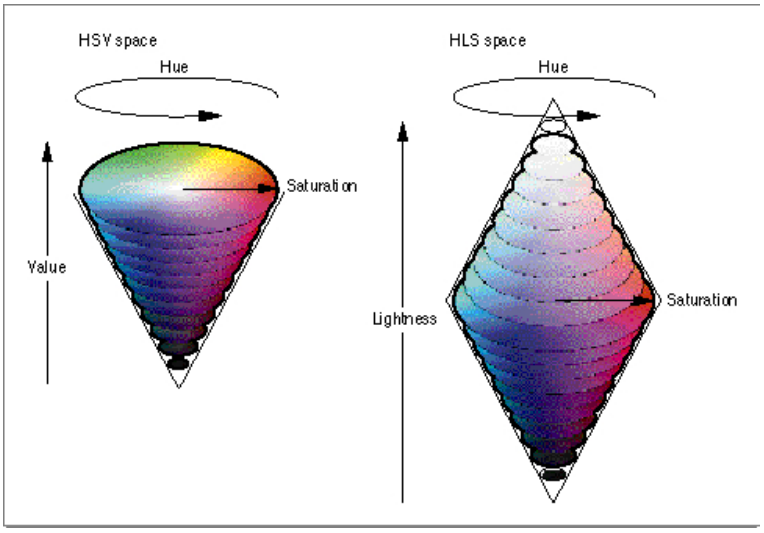
Modely HSV, HLS

Barva je definována opět trojicí složek. Nejsou to ale barvy jako u RGB či CMYK. Hlavní využití je v grafických programech pro snadnější míchání barev. Jsou to ty lišty při návrhu barvy, kde lze jednoduše změnit světlost či sytost barvy. **HSV** - Hue (barevný tón)+Saturation(sytost)+Value(jas)
HLS - Hue (barevný tón)+Lightness(světlost)+Saturation(sytost)

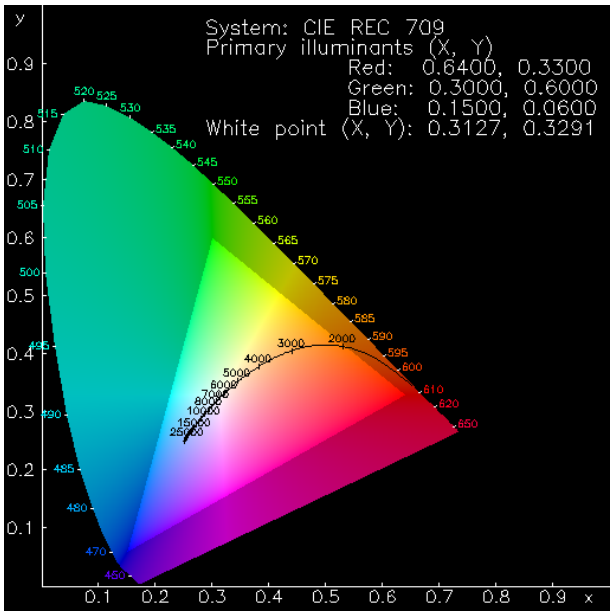
Modely CIE, YUV, YIQ

Každou barvu vnímá každý člověk jinak. Záleží na únavě očí, světle okolí. Přesné rozlišování barev trápilo v době před nástupem počítačů zejména výrobce barev pro tiskárny a textilky, a tak se v roce 1931 sešli odborníci na barvy v organizaci CIE (Commission Internationale de l'Eclairage) a vytvořili hypotetického standardního pozorovatele a jemu odpovídající barevný prostor, nezávislý na použité technologii. Tento prostor, dnes nazývaný CIE-XYZ, umožňuje popsat jakoukoliv barvu (dokonce i takovou, kterou nelze namíchat ze třech základních), či přesněji vjem z této barvy, pouze dvěma hodnotami (velmi zjednodušeně je lze nazvat poměrná červenost a poměrná modrost). Třetí hodnotou je pak světlost barvy.

Po nástupu počítačů vznikla potřeba přizpůsobit prostor CIE-XYZ tak, aby bylo možné co nejomezenějším počtem bitů, které budou každé složce přiděleny, pokrýt co nejlépe možnosti barevného rozlišení oka. A tak vznikly barevné prostory CIE



Obrázek 2.4: Barevné modely HSV, HLS



Obrázek 2.5: Barevný model CIE

L*u*v* a CIE L*a*b*, oba značně výpočetně náročné. To sice nevadí při přípravě pro tisk, ale při zobrazování v reálném čase ano. S dalšími podobnými prostory přišli tvůrci norem pro barevné televizní vysílání (YUV, YIQ).

2.3 Kontrolní otázky

1. Jakou funkci mají tyčinky a čípky?

2. Popište aditivní metodu mísení barev.
3. Popište subtraktivní metodu mísení barev.
4. Co znamená RGB?
5. Jaká barva vznikne smísením červené a zelené?
6. Proč se zavedly barevné modely HSV a HLS?
7. Co to je CIE?

2.4 Literatura

Žára, Beneš, Sochor, Felkel - Moderní počítačová grafika
http://www.dtpstudio.cz/obchod/Gretag/rychlokurz.html
http://www.root.cz/clanky/nez-zacneme-skenovat/
http://developer.apple.com/documentation/mac/ACI/ACI-48.html
http://viz.aset.psu.edu/gho/sem_notes/color_2d/index.html
http://dx.sheridan.com/advisor/cmyk_color.html

Kapitola 3

Úpravy obrazu

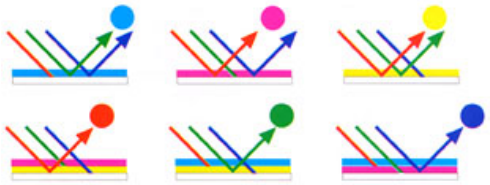
V následující kapitole se seznámíme se základními typy úpravy obrazu. První z nich je transformace barev pro výstupní zařízení, dále práce s barvami a expozicí pomocí histogramu a úpravy obrázků s vysokým dynanickým rozlišením (obrázky obsahující zdroj světla). Z metod transformace obrázku uvedeme warping a morfing, které se využívají v grafickém designu.

3.1 Transformace barev

Úprava barvy, například zesvětlení či ztmavení, přidání konstrastu atd., se velice často používají při úpravě fotografií. Práce s barvami je ale potřebná i při tisku. Obraz je upraven s ohledem na technologii barevného tisku a sníží se celkový počet použitých barev.

Pokud omezíme počet barev, dochází automaticky ke ztrátě informace. Současné algoritmy se snaží, aby oko člověka tyto změny nedokázalo poznat. Metody, které se používají, se nazývají polotónování (halftoning) a rozptylování (dithering). Obě metody vychází z toho, že naše oko dokáže mít vjem jednoho odstínu barvy jako kombinace několika barevných bodů blízko u sebe.

Podle obr. 3.1 vidíme. Na předmět, který obsahuje fialovou a žlutou dopadá světlo. Vyruší se a výsledná barva, kterou oko přijímá a mozek interpretuje, je červená. Takže, když se díváte na knihu s červeným přebalem, tak pod mikroskopem byste viděli síť fialových a žlutých bodů. Analogicky je to na obrázku pro ostatní barvy.



Obrázek 3.1: Odraz a vstřebávání paprsků světla

3.2 Warping a morfing

Hlavní využití obou transformací je v oblasti počítačové animace. Filmový průmysl pracuje s měnícími se objekty a scény nejsou reálné, pouze se generují na výkonných počítačích. Objekty se mohou různě pohybovat, vzájemně ovlivňovat a v některých případech i měnit svůj tvar. Pokud chceme tento jev zachytit i v počítačové animaci, používáme animační techniku známou jako morphing. Morphing samozřejmě nemá uplatnění pouze v animaci přírodních jevů (růstové simulace), ale používá se zejména pro tvorbu nejrůznějších speciálních efektů. Warping se spíše užívá v oblasti designu, kdy nepotřebujeme vidět průběh transformace, ale pouze výsledek.

Warping = metoda, která z jednoho obrázků vykreslí zmodifikovaný jiný (zkřivený, deformavaný). Změnu určí uživatel sám na původním obrázku.

Morfing = posloupnost, která zobrazuje přechod od jednoho k druhému obrázku. Jako vstup jsou potřeba dva obrázky – původní a ten, ke kterému chceme přejít.

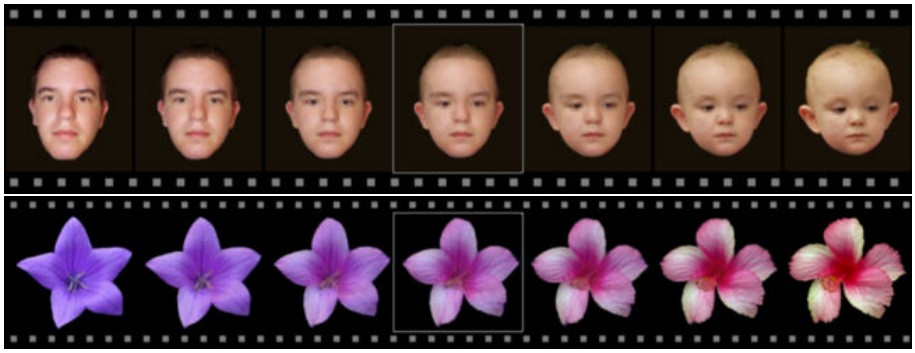
3.2.1 Warping

Warping lze podle způsobu provedení rozdělit do dvou tříd. První třída pouze předepsanou transformací upraví vložený obrázek, například udělá vlnu či spirálu. Druhá třída obsahuje obecné metody, které umožňují libovolnou transformaci. Lze použít síťový či úsečkový warping. Síťový pokryje obrázek sítí křivek (většinou bezierových kubik či spline křivek) a těmi uživatel pohybuje. Ty ovlivní pixely obrázku a podle algoritmu jej přepočítají. Úsečkový warping se používá při lokálních změnách obrazu. Na obr. 3.2 jsou v prvním okně vyznačeny části, které se budou měnit. Ve druhém okně je potom výsledná proměna.

Obrázek 3.2: Příklad úsečkového warpingu

3.2.2 Morfing

Úlohu morphingu lze formalizovat následovně. Máme dány dva zdrojové objekty – výchozí a cílový. Úkolem je nalézt transformaci mezi výchozím a cílovým objektem. Takových transformací existuje jistě velké množství, ale pro nás je důležitá ta třída transformací, která působí reálným dojmem.



Obrázek 3.3: Morfing

3.3 Obrazy s vysokým dynamickým rozsahem (High Dynamic Range – HDR)

Ve svém okolí můžeme vnímat různé rozsahy intenzity světla. V tmavém lese vidíme stíny s jasem v řádu 10^{-5} cd m^{-2}, ale písek na pláži v poledne může mít jas až 10^5. Dynamický rozsah obrázku se vyjadřuje jako poměr nejvyššího k nejnižšímu jasu. Obyčejná scéna z výletu má dynamický rozsah kolem 15:1, ale stůl s rozsvícenou lampou může mít i rozsah 50 000:1.

Pokud sami děláte fotografie, není možné na jedno nastavení expozice zachytit nejtmavější a nejsvětlejší části scény. Obrázky s HDR (High Dynamic Range) se získávají např. kombinací různých snímků s různou expozicí nebo jsou k dispozici speciální snímače.

Monitory mají dynamický rozsah asi 100:1, tiskárny kolem 30:1. Obrázek se tedy musí transformovat (mapovat) na daný rozsah. Tomuto úkolu se říká mapování tónů (tone mapping). Tato metoda má za úkol ze získaného HDR snímku udělat obrázek na výstupní zařízení, který bude odpovídat scéně, jak ji viděl člověk, který snímek vytvořil. Existuje několik metod k mapování tónů – globální či lokální, které dokáží s různou přesností zrekonstruovat a zobrazit původní scénu. Zajímavé a v praxi užitečné (například ve filmu) jsou časově závislé metody, které berou v úvahu časovou asiaci oka. Ve tmě se oko přizpůsobí a po nějaké době slabě vnímá okolní předměty.

3.4 Histogram

Histogram je graf, který vám dá přehled o tom, kolik pixelů je na vašem snímku obsaženo ve škále od nejtemnější do největšího jasu; bývá ve výbavě lepších editorů a velmi dobrých fotoaparátů.

Jeho hlavní využití je při kontrole expozice dané fotografie a její další úpravě v počítači. Na malém display digitálního fotoaparátu je těžko rozeznatelná správná či špatná expozice, proto se v lepších fotoaparátech zobrazuje histogram. Pokud tuto funkci ve fotoaparátu nemáte, nabízí se možnost zpětné úpravy histogramu ve vhodném počítačovém programu (např. Photoshop, Gimp – free software).

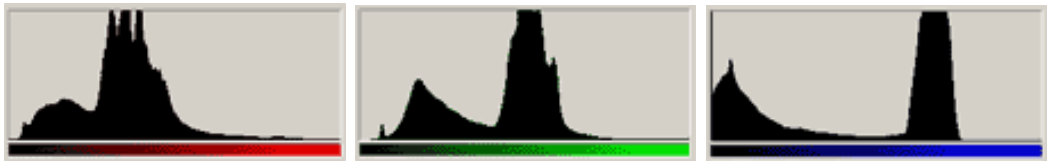
Jak tedy rozpoznat správně exponovanou fotografii? Musí obsahovat všechny úrovně jasu. Je však nutné si dát pozor na snímky, které jsou tmavší a obsahují pouze malé množství světlých pixelů (viz obr. 3.4).

Obrázek 3.4: Tmavý obrázek pokoje a jeho histogram

Špatně naexponované snímky mají silně zastoupeny pouze některé z krajních hodnot (bílá vpravo nebo černá vlevo v grafu) jak je vidět na obr. 3.4. Pro barevné fotografie jsou většinou ukázány tři histogramy pro každou z barevných složek RGB. Výsledný jasový diagram celé fotografie se vypočítá podle vztahu

```
Jas  =   0.3 R + 0.59 G + 0.11 B
```

U barevných fotografií si je nutné dávat pozor na přeexponování pouze jedné barvy, která ve výsledném jasovém diagramu nemusí být poznat, neboť ve výpočtu jasu má jen 30-ti procentní vliv.

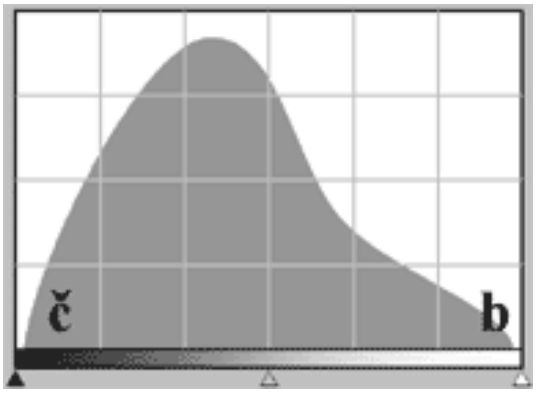


Obrázek 3.5: RGB histogram

Obrázek 3.6: Histogram špatně naexponovaného obrázku – podexponovaný a přeexponovaný obrázek

Shrnutí

Histogram je graf, který zobrazuje počet jednotlivých jasových stupňů v obrázku. Pokud máte fotoaparát se zobrazením histogramu, snažte se, aby pro vaše fotky vypadal podobně jako na obr. Když tuto funkci nemáte, zkontrolujte si histogram fotografií v programu v počítači a upravte jej na ideální tvar.



Obrázek 3.7: Histogram ideální expozice

3.5 Kontrolní otázky

1. Co je to warping?
2. Popište princip morfingu.
3. Co znamená zkratka HDR?
4. Kdy vznikají obrázky s HDR?
5. Co je to histogram?
6. Jak se dá histogram využít při úpravě fotografií?

3.6 Literatura

Úprava barev
http://www.root.cz/clanky/programujeme-jpeg-transformace-a-podvzorkovani-barev/

Warping a morphing
http://herakles.zcu.cz/~jparus/diploma-cz.php (česky)
http://www.cs.cmu.edu/~german/research/ImageWarping/imagewarping.html
http://www.blackbeltsystems.com/mdemos.html (demo ukázky)
http://www.ababasoft.com/mindconcentration/morfing.html (morfing fraktálů)

Práce s obrázky s HDR
http://www.cambridgeincolour.com/tutorials/high-dynamic-range.htm
http://www.cis.rit.edu/mcsl/icam/hdr/

Histogram
http://www.fotografovani.cz/art/fotech_df/histogram.html
http://www.digineff.cz/cojeto/ruzne/histogram.html
http://www.digineff.cz/cojeto/histogram/histogram1.html
http://fotoroman.cz/techniques2/exposure_histo.htm#Jas%20(Brightness)

Kapitola 4

Rasterizace objektů

Rasterizace je proces při kterém se vektorově definovaná grafika konvertuje na rastrově definované obrazy. Při zobrazení reálného modelu ve světových souřadnicích na výstupní zařízení potřebujeme zajistit co nejvěrnější podobnost reálného a zobrazovaného modelu. Omezíme se v tomto textu na rastrové výstupní zařízení, jehož nejjednodušším grafickým prvkem je bod. Protože složitější objekty jsou jen skládankou jednodušších objektů, potřebujeme mít k dispozici algoritmy na výpočet polohy bodů jednoduchých objektů jako úsečky, kružnice, elipsy, oblouků, atd... Ukážeme si některé algoritmy pro výpočet bodů úsečky a kružnice.

4.1 Úsečka a lomenná čára

Úsečka je nejjednodušším grafickým prvkem. Jejich skládáním se dají vytvářet lomené čáry a další křivočaré útvary. Vykreslení úsečky by mělo být co nejefektivnější a nejrychlejší.

Úsečka lze zapsat dvěma způsoby:

1. souřadnicemi počátečního a koncového bodu $[x_1, y_1], [x_2, y_2]$,
2. pomocí počátečního bodu $[x_1, y_1]$ a směrového vektoru

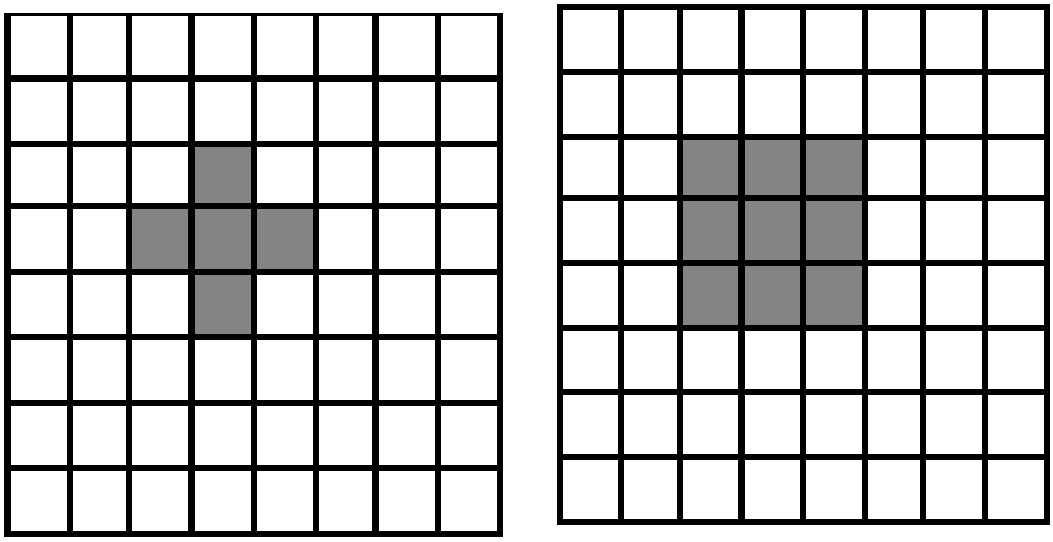
$$m = \frac{\Delta x}{\Delta y} = \frac{y_2 - y_1}{x_2 - x_1}$$

Podle velikost směrnice m se určí, vzhledem ke které ose se bude vzorkovat. Pro $|m| < 1$ je úsečka blíž k ose x a tedy bude vzorkována vzhledem k ní. Pro $|m| > 1$ je úsečka vzorkována na ose y a pro $|m| = 1$ může být řídící osa libovolná.

Rozlišujeme dva typy posloupnosti pixelů (obr. 4.1):

1. osmispojitá (8-connected) - každý pixel má osm sousedů čtyři ve vodorovném a svislém směru a čtyři úhlopříčně

2. čtyřspojitá (4-connected) - používá se zřídka, každý pixel má jen čtyři sousedy ve vodorovném s svislém směru.



Obrázek 4.1: 4-spojitý a 8-spojitý pixel

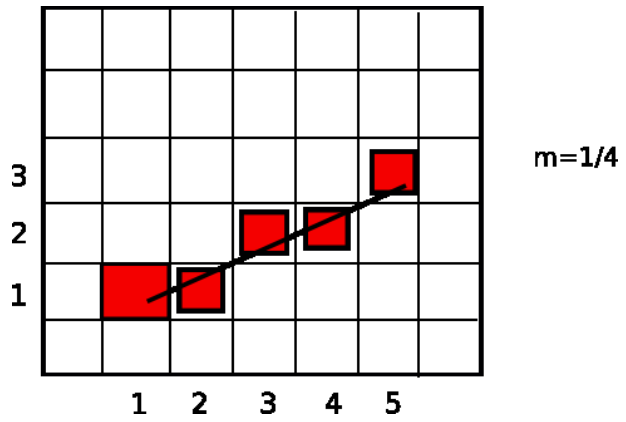
Vzhledem ke zvolenému typu pixelů se objekty vykreslují vybarvováním pixelů, které co nejvíce vystihují úsečku. Pro určení, které pixely budou vykresleny existuje několik algoritmů. My si ukážeme algoritmus DDA a Bresenhamův algoritmus pro kresbu úsečky.

4.1.1 Algoritmus DDA

Jedná se o přírůstkový algoritmus pro výpočet bodů úsečky. Postup spočívá v tom, že postupně zvedáme o jeden pixel hodnoty na x-ové souřadné ose a dopočítáváme odpovídající bod y. Jelikož jsou souřadnice pixelů vždy celá čísla, provede s zaokrouhlení hodnoty a pixel se vykreslí. Algoritmus je jednoduchý, ale relativně pomalý, protože pracuje v oboru reálných čísel.

Alogitmus pro rasterizaci ve směru osy x. Pro $|m| > 1$ je prováděna rasterizace ve směru osy y a v algoritmu se zamění operace s x a y.

1. Z koncových bodů $[x_1, y_1], [x_2, y_2]$ urči směrnici m podle vzorce výše.
2. Inicializuj bod $[x, y]$ hodnotou $[x_1, y_1]$.
3. Dokud je $x \leq x_2$ opakuj:
   (a) Vykresli bod [x, zaokrouhlené (y)]
   (b) $x = x + 1$
   (c) $y = y + m$

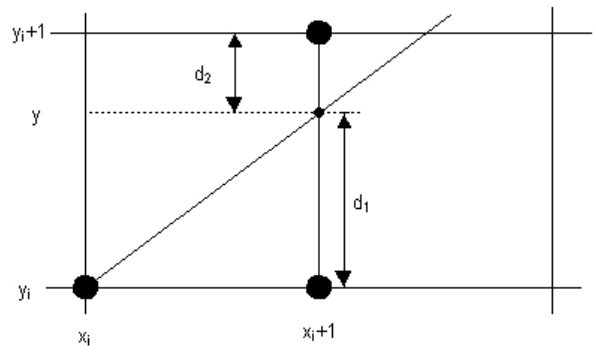


Obrázek 4.2: Vykreslení úsečky algoritmem DDA (rasterizace ve směru osy x)

4.1.2 Bresenhamův algoritmus pro kresbu úsečky

Tento algoritmus byl objeven v šedesátých letech minulého století panem Bresenhamem. Jeho výhoda je v tom, že pracuje pouze s celými čísly.

Víme, že od daného pixelu $[x, y]$ můžeme umístit další pixel (při rasterizace po ose x) pouze na dvou pozicích – $[x+1, y]$ nebo $[x+1, y+1]$. Rozdíly mezi souřadnicí y středů uvedených pixelů a skutečnou souřadnicí y označíme d_1, d_2 a můžeme je celočíselně vypočítat z parametrické rovnice kreslené úsečky. Jejich rozdíl potom vyhodnotíme. Pokud je kladný, je $d_1 > d_2$ a bližším pixelem je ten ve vzdálenosti d_2. Postup je názorně předveden na obr. 4.3.

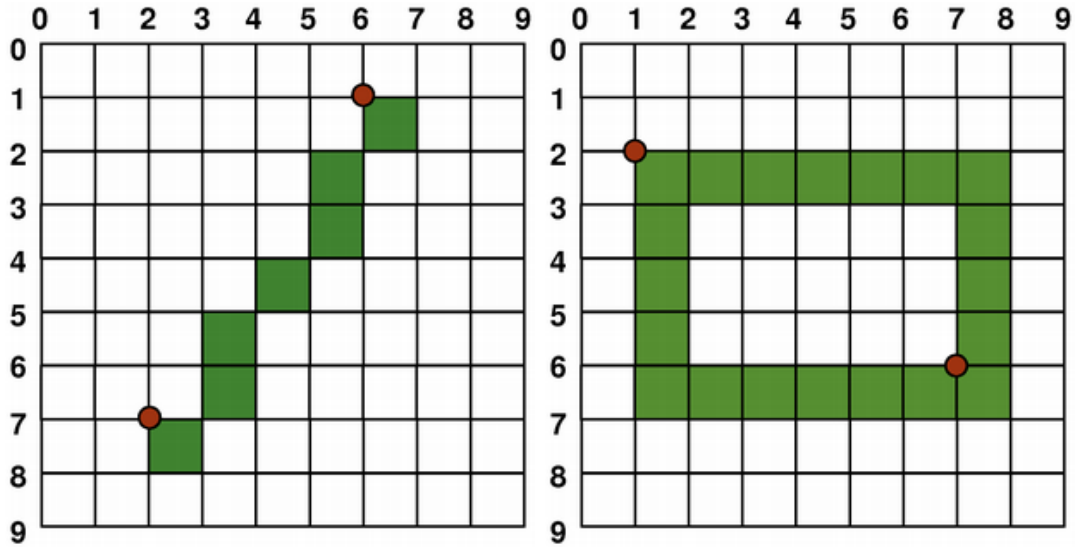


Obrázek 4.3: Princip Bresenhamova algoritmu

Na obr. 4.4 je ukázka rasterizace obecné úsečky a obdélníka.

4.1.3 Kresba silné a přerušované čáry

Silnou čáru můžeme kreslit pomocí dvou typů algoritmů. Jednoduchý postup je rychlý, ale dochází v něm k nepřesnostem. Jedná se o Bresenhamův algoritmus, kdy se v každém kroku vykresluje několik pixelů nad sebou či vedle sebe. Má však



Obrázek 4.4: dd

nevýhody. První z nich je, že se tloušťka čáry liší podle jejího sklonu (vzhledem k metrice rastru) a druhou nevýhodou je zakončení čar, které je rovnoběžné s některou souřadnou osou.

Přesný postup je však početně náročnější. Pracuje na principu vyplňování plochy, která určuje silnou čáru. Jako nejjednodušší tvar se používá obdélník, u něj je problém při napojování lomené čáry. Vhodnější je kresba pomocí obdélníků se zakulacenými rohy.

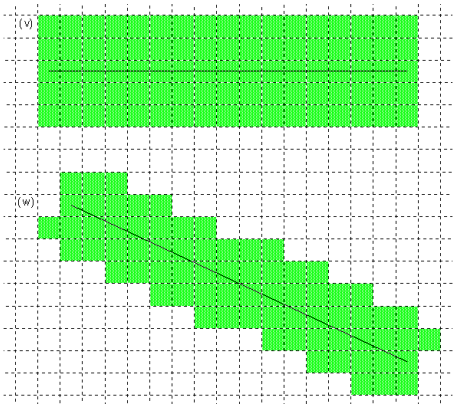
Pro kresbu přerušované čáry opět uvedeme dva algoritmy. První je založen na známém Bresenhamově algoritmu, kdy přidáme informaci o tom, zda jsme v úseku, který má být zobrazen či ne. Nevýhodou je opět metrika na rastrovém poli. Při různých sklonech úsečky budou dělící dílky nestejně dlouhé, což opticky působí nepřirozeně.

V praxi se většinou používá přesnější postup, založený na výpočtu délek dělících úseček. Potom se teprve použije známý algoritmus pro každý dílek samostatně. V závislosti na kvalitě programu je možné pozorovat různé zobrazení koncových dílku. Některé algritmy umí rozdělit úsečku tak, aby poslední dílek byl vždy viditelný.

4.2 Kružnice a elipsa

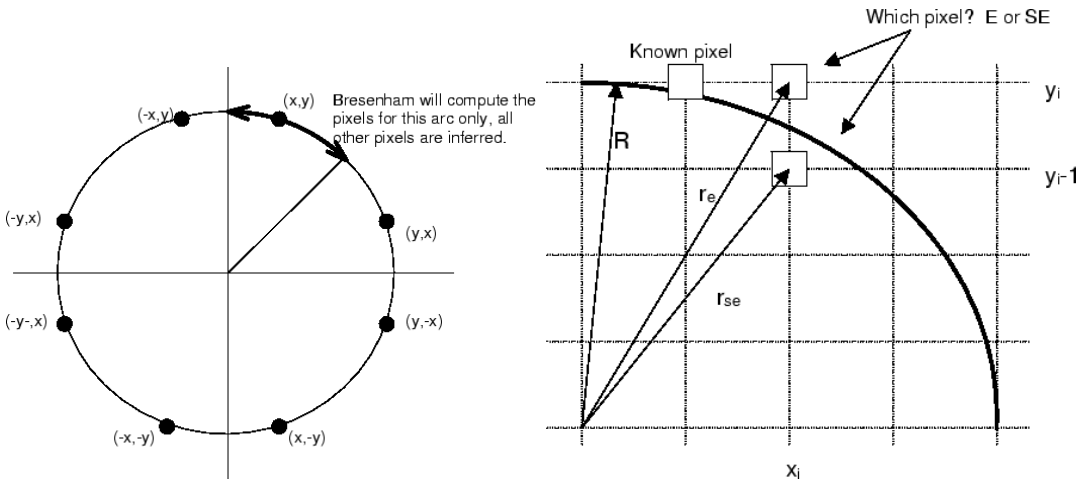
Kružnice se klasicky definuje středem a poloměrem. Elipsa je zase dána středem a velikostí hlavní a vedlejší poloosy. Otočením ji lze umístit do libovolné polohy. Při kresbě eliptických či kružnicových oblouků je navíc nutné definovat počáteční a koncový bod oblouku.

Při výpočtu bodů na kružnici stačí nalézt souřadnice oblouku 45 stupňů (obr. 4.6 vlevo), neboť všechny další body dostaneme záměnou souřadnic a změnou znamének.



Obrázek 4.5: Kresba silné čáry v rastru

Princip je opět na bázi Bresenhamova algoritmu pro úsečky. Jsme ve výchozím pixelu a ptáme se, kam teď můžu jít a testujeme, která pozice je blíž vykreslované kružnici (obr. 4.6 vpravo). Výhodu tohoto postupu je opět rychlost, neboť vždy pracujeme pouze s celými čísly.



Obrázek 4.6: Rasterizace kružnice

Elipsa se na rozdíl od kružnice musí vygenerovat pro celý kvadrant, tj. devadesát stupňů. Opět se použije Bresenhamův algortimus pouze s celočíselnou aritmetikou. Jediný rozdíl je ve tvaru testovací funkce, který z možných kandidátů na další bod leží blíž elipse.

4.3 Kontrolní otázky

1. Popište Bresenhamův algoritmus kreslení úsečky.
2. Co je algoritmus DDA?

3. Jaký je rozdíl mezi 8-spojitým a 4-spojitým pixelem.
4. Jaký je princip kreslení přerušované čáry?
5. Jaký je princip kreslení tlusté čáry?
6. Popište myšlenku kreslení kružnice.

4.4 Literatura

Bresenham algoritmus

`http://slady.cz/java/Bresenham/ (česky s Javou)`
`http://graphics.idav.ucdavis.edu/education/`
`GraphicsNotes/Bresenhams-Algorithm/Bresenhams-Algorithm.html`
`http://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html`
`http://en.wikipedia.org/wiki/Bresenham's_line_algorithm`

Kresba tlusté čáry a kružnice

`http://www.acm.uiuc.edu/bug/Be%20Book/The%20Interface%20Kit/`
`3_CoordinateSpace.html`
`http://rich12345.tripod.com/circles/ // algoritmus pro kružnici`

Kapitola 5

Řešení viditelnosti

Řešit viditelnost ve scéně umí většina grafických programů. Cílem je určit ty objekty, resp. jejich části, které jsou viditelné z určitého místa. Tyto algoritmy jsou vždy svázány s určitou reprezentací objektů ve scéně. Podle toho, v jakém tvaru jsou výstupní data, rozdělujeme algoritmy do dvou skupin:

1. Vektorové algoritmy – jejich výstupem je soubor geometrických prvků, např. úseček, které představují viditelné části zobrazovaných objektů. Používá se především v technických aplikacích. Bez změny viditelnosti lze plynuje úsečky zvětšovat. Nevýhodou je, že při numerické chybě je špatně celá úsečka, což výrazně mění celý pohled.

2. Rastrové algoritmy – pracují pouze v rastru. Výsledek je obraz, jehož jednotlivé pixely obsahují barvu odpovídajících viditelných ploch.Nevýhodou je pevný rozměr obrázku. Většina metod patří do této skupiny. Mezi rastrové algoritmy patří z-buffer, malířův algoritmus či algoritmus plovoucího horizontu.

5.1 Paměť hloubky – z-buffer

Tato metoda patří k nejznámějším a nejefektivnějším. Základem je použití paměti hloubky – z-bufferu. Ta tvoří dvojrozměrné pole, jehož rozměry odpovídají velikosti obrazu. Každá položka paměti hloubky obsahuje souřadnici z toho bodu, který leží nejblíže pozorovateli a jehož průmět leží v odpovídajícím pixelu v rastru. Na obr. vidíme scénu a odpovídající z-buffer. Čím blíže je předmět k pozorovateli, tím světlejší má odstín.

Výhody metody:

- není třeba třídit,
- umí správně vykreslit nestandardní situace (např. průseky),
- rychlost, jednoduchost výpočtu.

Obrázek 5.1: Z-buffer

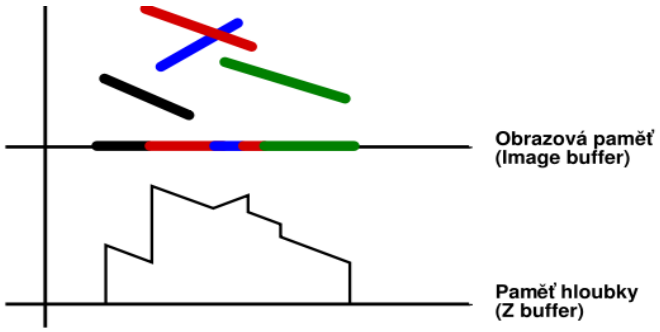
Nevýhody metody:

- větší paměťová náročnost,
- některé pixely se vícekrát překreslují.

Algoritmus

Pro každý pixel ukládáme jeho barvu a vzdálenost od pozorovatele (z-buffer). V inicializační části nastavíme barvu pro všechny pixely na barvu pozadí, paměť hloubky vypníme hodnotou minus nekonečno.

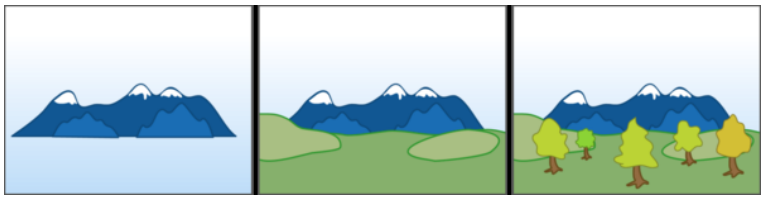
1. Každou plochu rozlož na pixely a pro každý její pixel $[x_i, y_i]$ stanov hloubku z_i
2. Má-li z_i větší hodnotu než položka $[x_i, y_i]$ v z-bufferu pak:
   a. obarvi pixel$[x_i, y_i]$ v obrazové paměti barvou této plochy
   b. položku $[x_i, y_i]$ v z-bufferu aktualizuj hodnotou z_i



Obrázek 5.2: Princip fungování algoritmu z-buffer

5.2 Malířův algoritmus – Painter's algorithm

Algoritmus je založen na myšlence vykreslování všech ploch postupně odzadu dopředu - přes objekty v pozadí se kreslí objekty v popředí. Inspirováno představou práce malíře který na podkladovou barevnou vrstvu nanáší další vrstvy (viz obrázek 5.3).

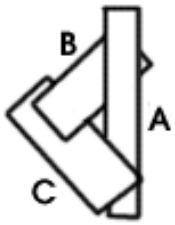


Obrázek 5.3: Princip malířova algoritmu

Algoritmus

Nejprve nalezneme pro každou plochu její nejmenší z-ovou souřadnici. a podle ní plochy uspořádáme. První plocha v seznamu je označena jako aktivní a je podrobena několika testům překrývání s ostatními plochami a pokud lze po nich rozhodnout, že leží za všemi ostatními je vykreslena a vyřazena ze seznamu. V opačném případě dojde v seznamu k výměně aktivní plochy s plochou u které dopadly všechny testy negativně.

Malířův algoritmus není vhodný pro všechny případy. Některé pozice (protínání) zobrazovaných objektů mohou vést ke špatnému zobrazení. Pokud se objekty navzájem překrývají může to vést až k zacyklení algoritmu. Takový případ nastane například při situaci na obrázku 5.4.



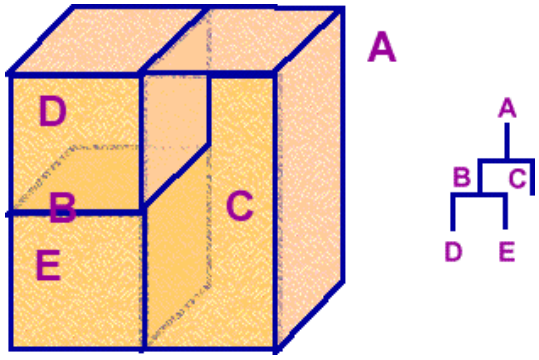
Obrázek 5.4: Problémový případ pro malířův algoritmus

```
Setřiď objekty podle hloubky.
Vstup: plošky, které se neprotínají
for všechny objekty v určeném pořadí do {
  for každý pixel[x][y] pokrytý objektem do{
    Obarvi pixel[x][y] barvou objektu.
  }
}
```

5.3 Malířův algoritmus se stromem BSP

BSP (Binary Space Partitioning trees) jsou datové struktury, které popisují danou scénu s jejími vazbami. Pomáhají urychlit řadu algoritmů, ale jsou vhodné pouze pro statické scény. Popisují složení objektů a jejich vzájemné vazby.

Při zobrazování procházíme strom BSP od kořene a v každém uzlu určujeme, který poloprostor je vůči poyorovateli vzdálenější a který bližší. Nejprve vykreslíme vzdálenější část prostoru, poté obsah aktuálního uzlu a nakonec oblast bližší pozorovateli.



Obrázek 5.5: BSP strom

5.4 Metoda vržení paprsku

Tato metoda spočívá ve vysílání paprsku od pozorovatele a následné hledání průsečíků s objekty ve scéně a určení toho, který je pozorovateli nejblíže. Pokud paprsek protíná pouze jeden objekt, je situace jednoduchá. Jestliže protíná průnik více těles, je nutné užít množinové operace.

```
for každý pixel[x][y] obrazu do {
/* Nejbližší objekt zasažený promítacím paprskem */
   for každý objekt ve scéně do
       if existuje průsečík paprsku s objektem
          { Porovnej hloubku z a vyber bližší objekt }.
   if nalezen objekt
         { Obarvi pixel[x][y] barvou objektu }
   else
         { Obarvi pixel[x][y] barvou pozadí }
}
```

5.5 Kontrolní otázky

1. Na jaké druhy rozdělujeme algoritmy viditelnosti podle typu výstupu?
2. Jaké jsou výhody a nevýhody z-bufferu?
3. Popište princip z-buffer určování viditelnosti.
4. Popište princip malířova algoritmu.
5. Jaké jsou nevýhody malířova algoritmu?
6. Co je to BSP strom?
7. Popište základní myšlenku metody vržení paprsku.

5.6 Literatura

Z-buffer
http://cs.wikibooks.org/wiki/Viditelnost
http://cgg.ms.mff.cuni.cz/~pepca/lectures/pgr003.html
http://www.pctuning.cz/index.php?option=com_content&task
=view&id=8721&Itemid=44&limit=1&limitstart=3

Malířův algoritmus
http://pocitacova-grafika.kvalitne.cz/maliruv-algoritmus
http://www.fi.muni.cz/~sochor/M4730/Slajdy/Viditelnost.pdf
http://en.wikipedia.org/wiki/Painter's_algorithm
http://medialab.di.unipi.it/web/IUM/Waterloo/node67.html
http://www.ibiblio.org/e-notes/3Dapp/Hidden.htm
Žára, Beneš, Sochor, Felkel - Moderní počítačová grafika

BSP stromy
http://cgg.ms.mff.cuni.cz/~pepca/prg022/vondrak.html

Kapitola 6

Objektově orientované programování

Objektově orientované programování (zkracováno na OOP, z anglického Object oriented programming) je metodika vývoje softwaru, založená na těchto myšlenkách, koncepci:

6.1 Koncepce

Objekty – jednotlivé prvky modelované reality (jak data, tak související funkčnost) jsou v programu seskupeny do entit, nazývaných objekty. Objekty si pamatují svůj stav a navenek poskytují operace (přístupné jako metody pro volání).

Abstrakce – programátor, potažmo program, který vytváří, může abstrahovat od některých detailů práce jednotlivých objektů. Každý objekt pracuje jako černá skříňka, která dokáže provádět určené činnosti a komunikovat s okolím, aniž by vyžadovala znalost způsobu, kterým vnitřně pracuje.

Zapouzdření – zaručuje, že objekt nemůže přímo přistupovat k vnitřním parametrům jiných objektů, což by mohlo vést k nekonzistenci. Každý objekt navenek zpřístupňuje rozhraní, pomocí kterého (a nijak jinak) se s objektem pracuje.

Skládání – objekt může využívat služeb jiných objektů tak, že je požádá o provedení operace.

Dědičnost – objekty jsou organizovány stromovým způsobem, kdy objekty nějakého druhu mohou dědit z jiného druhu objektů, čímž přebírají jejich schopnosti, ke kterým pouze přidávají svoje vlastní rozšíření. Tato myšlenka se obvykle implementuje pomocí rozdělení objektů do tříd, přičemž každý objekt je instancí nějaké třídy. Každá třída pak může dědit od jiné třídy (v některých programovacích jazycích i z několika jiných tříd).

Polymorfismus – odkazovaný objekt se chová podle toho, jaký je jeho skutečný typ. Pokud několik objektů poskytuje stejné rozhraní, pracuje se s nimi stejným způsobem, ale jejich konkrétní chování se liší. V praxi se tato vlastnost projevuje např. tak, že na místo, kde je očekávána instance nějaké třídy, můžeme dosadit i instanci libovolné její podtřídy (třídy, která přímo či nepřímo z této třídy dědí), která se může chovat jinak, než by se chovala instance rodičovské třídy, ovšem v rámci mantinelů, daných popisem rozhraní.

Pokud předchozí koncepci již rozumíme, bude pro nás pochopení objektově orientovaného programování jednoduché. Pokud však nerozumíme, což se předpokládá, nevadí. K dalšímu vysvětlení nám vřele pomůže ukázkový příklad.

6.2 Ukázkový příklad

Nejprve si představme, že celý svět se skládá z objektů. Objektem můžeme rozumět města, ulice, domy, stromy, auta, silnice, atd. Aby se nám tyto objekty nepletli, rozdělíme si je na různé druhy, resp. různé **třídy**. V programování bývá nejčastějším ukázkovým příkladem tříd třída aut.

Představme si auto (objekt). Každé auto, pokud neuvažujeme některé speciální případy, je přesně určeno státní poznávací značkou, respektive písmeny a číslicemi na SPZ. Můžeme tedy říci, že SPZ je charakteristická vlastnost auta, neboli vlastnost objektu. Pochopitelně má každé auto mnohem více vlastností, jako je například značka, barva, typ, objem motoru, atd.

6.3 Definice třídy

Nadefinujme si tedy třídu aut, tak aby obsahovala tyto vlastnosti: `spz`, `typ`, `objem Motoru`. Třídu aut, jak je v Delphi zvykem, nazveme `TAuto`. V této třídě budeme uvažovat `spz` a `typ` jako řetězce a `objemMotoru` jako celočíselnou hodnotu. Syntaxe této třídy je následující:

```
type
  TAuto = class
     spz, typ: String;
     objemMotoru: Integer;
  end;
```

Tuto definici nové třídy `TAuto` umístíme v Delphi do oblasti, kde definujeme nové typy, tedy za příkaz `type`. Každá nová třída se zadává stejným způsobem: `jmenotridy = class`, kde `class` nám udává, že se jedná o třídu. Po výčtu vlastností třídy, ukončíme definici příkazem `end;`, jak je vidět výše.

6.4 Vytvoření objektu

Nyní máme nadefinovanou třídu `TAuto`, ale zatím nemáme žádný objekt. Představme si, že vlastníme dva automobily (objekty): služební a soukromý. Aby jsme mohli tyto objekty vytvořit, musíme si je nejdříve deklarovat. Objekty budeme deklarovat stejně jako proměnné v části programu určené k deklaraci (tedy za klíčovým slovem `var`) a tyto objekty budou typu `TAuto`.

```
var
  sluzebniAuto, soukromeAuto: TAuto;
```

Abychom mohli s objekty pracovat, je zapotřebí je nejdříve vytvořit. Vytvoření se provede pomocí konstruktoru (konstruktor bude vysvětlen později) tímto způsobem:

```
sluzebniAuto := TAuto.Create;
soukromeAuto := TAuto.Create;
```

Takto byli vytvořeny dva objekty `sluzebniAuto` a `soukromeAuto`. Přesněji řečeno, jsme si uvolnily v paměti počítače místo pro tyto objekty a nyní s nimi můžeme pracovat. V objektově orientovaném programování je zapotřebí stále pamatovat na to, že objekty musíme nejdříve vytvořit a až pak s nimi můžeme pracovat. Jestliže už objekt k práci nepotřebujeme, je zapotřebí objekt zrušit, přesněji uvolnit zabrané místo v paměti počítače. To se provede příkazem `Free` následovně:

```
sluzebniAuto.Free;
soukromeAuto.Free;
```

Po tomto příkazu již naše objekty neexistují, nemůžeme tedy s nimi pracovat.

6.5 Vlastnosti objektu

Vraťme se nyní trochu zpět a předpokládejme, že máme vytvořeny objekty `sluzebni Auto` a `soukromeAuto`. Tyto objekty jsou typu `TAuto` a tudíž obsahují vlastnosti `spz`, `typ` a `objemMotoru`. Pokud chceme přistupovat k vlastnostem objektu, budeme k nim přistupovat pomocí tečky. Pokud tedy chceme zadat vlastnosti našich aut, uděláme to takto:

```
sluzebniAuto.spz := '1B0 25-69';
sluzebniAuto.typ := 'Škoda';
sluzebniAuto.objemMotoru := 1600;

soukromeAuto.spz := '2B3 85-31';
soukromeAuto.typ := 'Porshe';
soukromeAuto.objemMotoru := 3200;
```

Naopak, pokud budeme chtít vypsat do dialogového okna typ našeho služebního auta, napíšeme tento příkaz:

```
MessageDlg('Typ mého soukromého auta je:
' + soukromeAuto.typ, mtInformation, [mbOK], 0);
```

6.6 Metody objektu

Dosud jsme si vysvětlili, že každý objekt může obsahovat různé vlastnosti a my k nim pak jednoduše pomocí tečky přistupujeme. Nabízí se otázka: "K čemu je to dobré?", když stejně tak funguje v Delphi záznam, neboli `record`. Vysvětlení je jednoduché. Objekty nám nabízejí mnohem více možností a jednou z nich jsou metody, neboli funkce či procedury uvnitř objektu.

Vraťme se zpět k našim autům a chtějme sledovat stav benzínu v nádrži pomocí vlastnosti `palivo`. Dále budeme předpokládat, že množství paliva ubývá, pokud ujedeme nějakou vzdálenost v závislosti na spotřebě (`spotreba`) a přibývá, pokud tankujeme na čerpací stanici. Tyto dva poznatky nám zaručí procedury `Ujed` a `Tankuj`. Deklarace třídy `TAuto` bude vypadat následovně:

```
type
  TAuto = class
     spz, typ: String;
     objemMotoru: Integer;
     palivo, spotreba: Double;
     procedure Ujed(vzdalenost: Double);
     procedure Tankuj(mnozstviPaliva: Double);
  end;
```

Dále je zapotřebí vytvořit těla našich nových dvou procedur. To provedeme kdekoliv v části zdrojového textu, tedy za příkazem `implementation`, jak jsme zvyklí. Procedury budou vypadat následovně:

```
procedure TAuto.Ujed(vzdalenost: Double);
begin
  palivo := palivo - vzdalenost * (spotreba / 100);
end;

procedure TAuto.Tankuj(mnozstviPaliva: Double);
begin
  palivo := palivo + mnozstviPaliva;
end;
```

Nyní si všimněme, že při definici nových procedur jsme jako název uvedli `TAuto.Ujed`, resp. `TAuto.Tankuj` místo pouhého `Ujed`, resp. `Tankuj`. Důvod je velmi jednoduchý. Pokud bychom v programu měli více tříd, které by obsahovaly proceduru `Ujed` a přitom v každé třídě by procedura měla jinou definici, nastal by při překladu zdrojového textu zmatek.

Další důležitý poznatek je v proměnné `palivo`. Všimněme si, že neuvádíme např. `soukromeAuto.palivo`, ale pouze `palivo`. Jak tedy program pozná jaké `palivo` má změnit? Jednoduše. Změní `palivo` toho objektu, na kterém byla daná procedura volána. Tedy pokud voláme nad objektem soukromého auta (`soukromeAuto.Ujed(50)`), odečte požadovanou hodnotu z paliva tohoto objektu, tedy z `soukromeAuto.palivo`.

6.7 Konstruktor a destruktor objektu

Už dříve jsme se zmínili o konstruktoru. Používá se k tomu, aby nám vytvořil objekt, resp. aby alokoval místo v paměti pro tento náš objekt. Často je však velmi vhodné, když už při vytváření objektu specifikujeme některé vlastnosti, v našem případě `spz`, `typ`, `objemMotoru`, `palivo` a `spotreba`. Docílíme toho tak, že předefinujeme klasický konstruktor Delphi. Do definice třídy `TAuto` přidáme zmínku o tom, že budeme konstruktor vytvářet vlastní. Definice třídy pak bude vypadat:

```
type
  TAuto = class
     spz, typ: String;
     objemMotoru: Integer;
     palivo, spotreba: Double;
     constructor Create(sspz, ttyp: String;
     oobjemMotoru: Integer;
     sspotreba: Double);
     procedure Ujed(vzdalenost: Double);
     procedure Tankuj(mnozstviPaliva: Double);
  end;
```

Konstruktor je uveden klíčovým slovem `constructor` a název může být jakýkoliv, avšak je vhodné název ponechat tak, jak je zvykem, tedy `Create`.

Definice samotného konstruktoru se opět provádí v části implementace a může vypadat takto:

```
constructor TAuto.Create(sspz, ttyp: String;
 oobjemMotoru: Integer; sspotreba: Double);
begin
  inherited Create;
```

```
  spz := sspz;
  typ := ttyp;
  objemMotoru := oobjemMotoru;
  palivo := 0;
  spotreba := sspotreba;
end;
```

Příkaz `inheroted Create` nám slouží k vlastnímu vytvoření objektu, další příkazy jsou snad zřejmé a jednoduché. Pokud tedy budeme chtít vytvořit opět služební a soukromý vůz, budeme postupovat takto:

```
sluzebniAuto := TAuto.Create('1B0 25-69', 'Škoda', 1600, 7.8);
soukromeAuto := TAuto.Create('2B3 85-31', 'Porshe', 3200, 11.4);
```

Někdy je zapotřebí něco provést, než se objekt zruší. Třeba uložit důležitá data do souboru. K tomu nám hravě poslouží předefinování destruktoru. Je to stejné jako v případě konstruktoru, pouze klíčové slovo je `destructor` a je běžné používání názvu `Destroy`. Destruktor pak může vypadat takto:

```
destructor TAuto.Destroy;
begin
  UlozeniDat;
  inherited Destroy;
end;
```

Pozorný čtenář si všimne, že výše jsme k zrušení objektu použili příkazu `Free`, nikoliv `Destroy`. Bylo to z důvodu, že procedura `Free` nejdříve zjistí zda daný objekt existuje a pokud existuje, tak pak teprve volá destruktor třídy `Destroy`.

6.8 Dědičnost

Dědičnost je další důležitou vlastností při objektově orientovaném programování. Představme si, že máme již vytvořenou třídu `TAuto`, které splňuje požadavky pro veškerá auta. Nyní však budeme chtít vytvořit třídy pro auta osobní a nákladní, které budou mít stejné vlastnosti jako třída `TAuto`. Navíc u třídy `TOsobniAuto` budeme požadovat vlastnost `pocetMist` a u třídy `TNakladniAuto` budeme požadovat vlastnost `nosnost`.

Díky dědičnosti v OOP je definice těchto tříd velmi jednoduchá:

```
type
  TOsobniAuto = class(TAuto)
    pocetMist: Integer;
  end;
  TNakladniAuto = class(TAuto)
    nosnost: Integer;
  end;
```

Jak je vidět, syntaxe nových tříd je velmi jednoduchá, stačí pouze do závorky za příkaz `class` napsat jméno třídy, po které bude nová třída dědit vlastnosti a metody. Znamená to tedy, že např. nová třída `TOsobniAuto` obsahuje vlastnosti `spz`, `typ`, `objemMotoru`, `palivo`, `spotreba` a `pocetMist` a metody `Ujed` a `Tankuj`.

Pokud chceme aby vlastnost `pocetMist` byla zadávána již při vytváření objektu osobního auta, musíme změnit konstruktor třídy `TOsobniAuto`. To uděláme obdobně, jako jsme to dělali ve třídě `TAuto`.

6.9 Kontrolní otázky

6.10 Literatura

Kapitola 7

Křivky a plochy technické praxe

V technické praxi se setkáváme s tím, že potřebujeme křivky a plochy, které se dají libovolně upravovat a zároveň je jejich matematické vyjádření jednoduché. V šedesátých letech minulého století se objevily Fergusonovy, později Bézierovy a Coonsovy křivky a plochy. Na ně navázaly splajny a v současné době se v moderních CAD systémech používají NURBS křivky a plochy. V roce 2003 se ve studiu Maya objevily plochy T-spline.

7.1 Základní princip

Základem všech výše uvedených ploch a křivek jsou řídicí body. Křivka či plocha jimi obecně neprochází, jedná se tedy o aproximační křivky a plochy. K těmto bodů jsou podle typu vybrány řídicí – **bázové** funkce - bersteinovy polynomy pro bézierovy křivky a plochy, B-spline funkce pro splajny a NURBS.

Výsledný bod na křivce či ploše je dán jako lineární kombinace řídicího bodu s funkci k němu patřící. Vysvětlíme si to na Bézierově křivce.

Máme dány čtyři řídicí body P_0, P_1, P_2, P_3 a bázové bersteinovy polynomy:

$$\begin{aligned} B_0(t) &= (1-t)^3 \\ B_1(t) &= 3t\,(1-t)^2 \\ B_2(t) &= 3t^2\,(1-t) \\ B_3(t) &= t^3 \end{aligned} \tag{7.1}$$

Parametr t probíhá v intervalu od nuly do jedné a pro každou hodnotu určí jeden bod na výsledné křivce. Vezmeme pevně parametr t roven $0,25$ a vypočítáme výsledný bod jako kombinaci

$$C(0,25) = P_0B_0(0,25) + P_1B_1(0,25) + P_2B_2(0,25) + P_3B_3(0,25)$$

Vypočítáme hodnoty bersteinových polynomů a pak postupně vezmeme x, y, z-ovou souřadnici jednotlivých bodů a dostaneme tři čísla, která udávají výsledný bod na křivce. Matematický zápise je:

$$Q(t) = \sum_{i=0}^{3} P_i B_i(t),\ t \in \langle 0, 1 \rangle \tag{7.2}$$

Pro plochy je výpočet obtížnější. Jako vstup máme síť řídicích bodů a tedy každý bod má dva indexy – řádkový a sloupcový. Při výpočtu přiřadíme každému bodu dvě bázové funkce, které odpovídají indexům daného bodu, tzn. bod P_{31} bude násoben bersteinovými polynomy B_3 a B_1. Jak víme plocha je dána dvěma parametry, proto první funkce bude vyčíslena s prvním parametrem a druhá s druhým parametrem. Matematicky můžeme všechny tyto kombinace zapsat jako:

$$S(r, s) = \sum_{i=0}^{3} \sum_{j=0}^{3} P_{ij} B_i(r) B_j(s), \tag{7.3}$$

Tento princip funguje i pro další typy křivek a ploch jen s různými tvary bázových polynomů.

7.2 Základní typy křivek a ploch technické praxe

7.2.1 Fergusonovy křivky a plochy

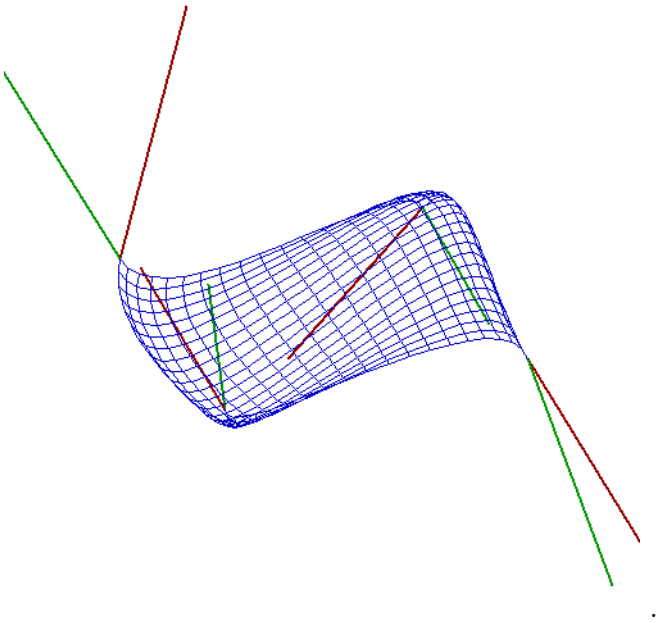
Základní oblouk je určen čtyřmi body P_0, P_1, P_2, P_3. Křivka má krajní body P_0, P_3. Tečné vektory v těchto bodech jsou $\overrightarrow{P_0P_1}, \overrightarrow{P_3P_2}$. Velikost těchto vektorů ovlivňuje tvar křivky – čím je vektor větší, tím více se křivka k vektoru přibližuje.

Křivka je definována parametrickými rovnicemi:

$$Q(t) = \sum_{i=0}^{3} P_i F_i(t),\ t \in \langle 0, 1 \rangle \tag{7.4}$$

$$\begin{aligned} F_0(t) &= t^3 - t^2 - t + 1 \\ F_1(t) &= t^3 - 2t^2 + t \\ F_2(t) &= -3t^3 + 4t^2 \\ F_3(t) &= t^3 - t^2 \end{aligned} \tag{7.5}$$

Dvojrozměrným zobecněním Fergusonových křivek dostaneme Fergusonovy plochy. Jsou dány sítí reálných bodů P_{ij} a jejich rovnice je:



Obrázek 7.1: Fergusonova plocha

$$Q(r,s) = \sum_{i=0}^{3}\sum_{j=0}^{3} P_{ij}F_i(r)F_j(s), \tag{7.6}$$

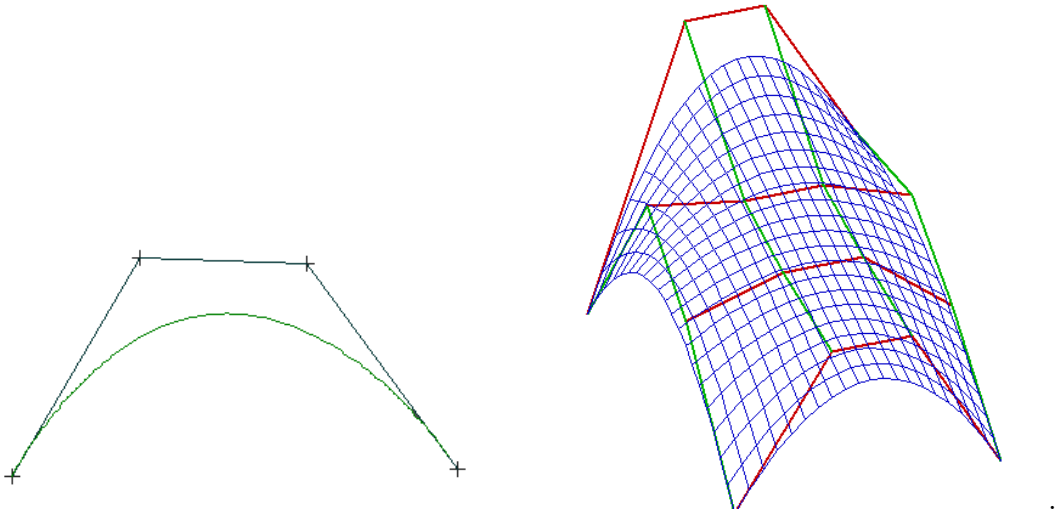
kde F_i jsou polynomy definované v rovnici (7.5) a $(r,s) \in \langle 0,1\rangle \times \langle 0,1\rangle$.

7.2.2 Bézierovy křivky a plochy

Bézierovy křivky a plochy jsou velmi často používány v technické praxi. Za jejich zakladatele je považován P. E. Bézier a nezávisle na něm také P. de Casteljau. Křivka je určena čtyřmi body a prochází prvním a posledním z nich. Vektory $\overrightarrow{P_0P_1}, \overrightarrow{P_3P_2}$ určují tečny v krajních bodech. Jejich směrnice jsou rovny třetině délky těchto úseček.
Křivka je definována parametrickými rovnicemi:

$$Q(t) = \sum_{i=0}^{3} P_iB_i(t),\ t \in \langle 0,1\rangle \tag{7.7}$$

$$\begin{aligned} B_0(t) &= (1-t)^3 \\ B_1(t) &= 3t\,(1-t)^2 \\ B_2(t) &= 3t^2\,(1-t) \\ B_3(t) &= t^3 \end{aligned} \tag{7.8}$$



Obrázek 7.2: Bézierova křivka a plocha

Obecná Bézierova křivka vznikne zobecněním předchozího případu. Bázové polynomy tvoří binomický rozvoj.

Mějme dáno $n+1$ řídicích bodů, potom je Bézierova křivka tvaru:

$$Q(t) = \sum_{i=0}^{n} P_i B_i^n(t),\ t \in \langle 0,1 \rangle\,, \tag{7.9}$$

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}.$$

Dvojrozměrným zobecněním Bézierových křivek dostaneme Bézierovy plochy. Jsou dány sítí reálných bodů P_{ij} a rovnice plochy je:

$$Q(r,s) = \sum_{i=0}^{3} \sum_{j=0}^{3} P_{ij} B_i(r) B_j(s), \tag{7.10}$$

kde B_i jsou polynomy definované v rovnici (7.8) a $(r,s) \in \langle 0,1 \rangle \times \langle 0,1 \rangle$.

7.2.3 Coonsovy křivky a plochy

Podobné Fergusonovým křivkám jsou Coonsovy křivky, pojmenované po S. A. Coonsovi. Jediným rozdílem je tvar bázových polynomů. Základní oblouk je opět určen čtyřmi body P_0, P_1, P_2, P_3. Křivka však nezačíná ani nekončí v žádném z těchto bodů. Počáteční a koncový bod leží v antitěžištích trojúhelníků $P_0P_1P_2$ a $P_1P_2P_3$.

Křivka je definována parametrickými rovnicemi:

$$Q(t) = \frac{1}{6}\sum_{i=0}^{3} P_i C_i(t),\ t \in \langle 0,1\rangle \tag{7.11}$$

$$\begin{aligned} C_0(t) &= (1-t)^3 \\ C_1(t) &= 3t^3 - 6t^2 + 4 \\ C_2(t) &= 3t^3 + 3t^2 + 3t + 1 \\ C_3(t) &= t^3 \end{aligned} \tag{7.12}$$

Jednotlivé Coonsovy kubiky je možné hladce napojit. Pokud má navazující oblouk první tři body shodné s posledními třemi předchozího oblouku, dostáváme po částech polynomiální křivku, která se nazývá Coonsův kubický B-spline.

Dvojrozměrným zobecněním Coonsových křivek dostaneme Coonsovy plochy. Jsou dány sítí reálných bodů P_{ij} a rovnice plochy je:

$$Q(r,s) = \sum_{i=0}^{3}\sum_{j=0}^{3} P_{ij} C_i(r) C_j(s), \tag{7.13}$$

kde C_i jsou polynomy definované v rovnici (7.12) a $(r,s) \in \langle 0,1\rangle \times \langle 0,1\rangle$

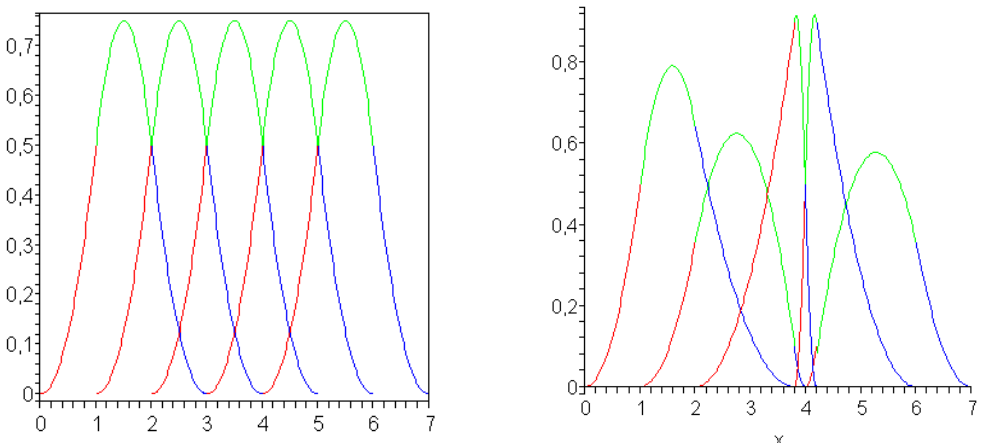
7.2.4 B-spline křivky a plochy

Ve všech předchozích případech probíhal parametr t interval od nuly do jedné a záleželo pouze na tvaru bázových funkcí. Přidejme nyní k bázových funkcím posloupnost reálných čísel, podle které se budou počítat jednotlivé hodnoty funkcí. Tyto bázové funkce se nazývají **B-spline funkce** a posloupnosti se říká **uzlový vektor**. Na uzlový vektor jsou kladena další omezení – je neklesající, uzly se mohou opakovat jen tolikrát, kolik je stupeň křivky a většinou bývá v intervalu od nuly do jedné, ale obecně nemusí. B-spline funkce je definována rekurentně následujícím předpisem:
Nechť $\mathbf{t} = (t_0, t_1, \ldots t_n)$ je uzlový vektor. **B-spline funkce** je definována jako:

$$N_i^0(t) = \begin{cases} 1 & \text{pro} \quad t \in \langle t_i, t_{i+1}) \\ 0 & \text{jinak} \end{cases}$$

$$N_i^k(t) = \frac{t - t_i}{t_{i+k} - t_i} N_i^{k-1}(t) + \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} N_{i+1}^{k-1}(t), \tag{7.14}$$

$$\text{kde } 0 \le i \le n-k-1, 1 \le k \le n-1, \tfrac{0}{0} := 0.$$



Obrázek 7.3: Ukázka B-spline funkcí - ekvidistantní a neekvidistantní

B-spline křivka je dána klasicky jako kombinace bodů s hodnotou příslušné bázové funkce. Výpočet je však náročnější a používá se k němu deBoorův algoritmus, který je založen na určitých vlastnostech B-spline funkcí.

Mějme dáno $m+1$ řídicích (kontrolních) bodů $P_0, P_1, \ldots, P_m$, kde $P_i \in \mathbb{R}^d$, uzlový vektor $\mathbf{t} = (t_0, t_1, \ldots t_{m+n+1})$. B-spline křivka stupně n pro řídicí body P_i a uzlový vektor $\mathbf{t}$ je definována jako:

$$C(t) = \sum_{i=0}^{m} P_i N_i^n(t), \tag{7.15}$$

kde N_i^n jsou bázové B-spline funkce.

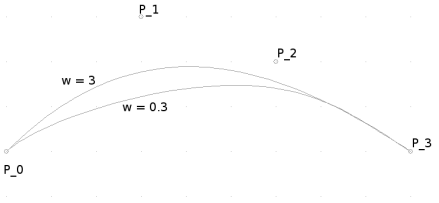
Plocha je definována nad sítí bodů se dvěma stupni, dvěma uzlovými vektory pro řádky a sloupce.

$$S(r,s) = \sum_{i=0}^{k} \sum_{j=0}^{l} P_{ij} N_i(r) N_j(s), \tag{7.16}$$

7.2.5 NURBS křivky a plochy

Zkratka NURBS bývá vtipně vysvětlována jako Nobody Understand Rational B-Spline. Ve skutečnosti však znamená neuniformní racionální B-spline. Kromě slova racionální jsme se již se všemi ostatními pojmy setkali. B-spline funkce a tedy i křivky a plochy byly probrány v předchozí části. Jsou to aproximační křivky a plochy dané řídicími body a spjaté s B-spline funkcemi. Tyto funkce jsou zadány rekurentně a jejich tvar závisí na uzlovém vektoru.

Neuniformní znamená, že rozdíl sousedních čísel v uzlovém vektoru není konstantní. Tj. průběh křivkou není plynulý, což dává větší možnosti při úpravě tvaru.



Obrázek 7.4: Vliv váhy na tvar křivky

Pojem racionální vznikl přidáním vah k bodům. Každému bodu je přiřazeno obvykle kladné číslo, které udává, jakou silou se bude křivka k danému bodu přitahovat. Pokud je v intervalu ¡0,1¿, znamená to, že vliv je malý a křivka jde od bodu dál. Pokud je větší než jedna, zvyšuje se síla tohoto bodu a ten si křivku přitáhne. Geometricky je výpočet plochy s váhami řešen rozšířením prostoru o dimenzi výše. Za čtvrtou souřadnici bodu se bere váha a počítá se s ní stejně jako s ostatními souřadnicemi. Na konci výpočtu se vahou první tři vypočtené hodnoty vydělí. Matematicky to lze zapsat jako:

$$C(t) = \frac{\sum_{i=0}^{m} w_i P_i N_i^n(t)}{\sum_{i=0}^{m} w_i N_i^n(t)}, \tag{7.17}$$

NURBS křivky a plochy se používají v CAD systémech pro jejich velmi dobré modelační schopnosti. Změnou polohy bodu, velikosti váhy či tvarem uzlového vektoru se může měnit lokálně tvar křivky či plochy.

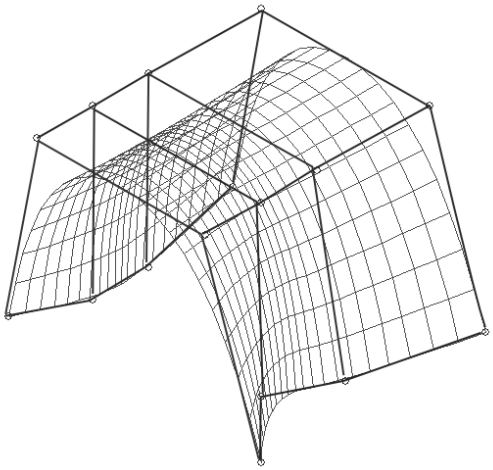
Oproti předchozím typům křivek dovedou NURBS jako jediné přesně vyjádřit kuželosečky a tedy s nimi jdou konstruovat základní tělesa (kužel, válec, anuloid, atd.). Obecná NURBS plocha vypadá matematicky již poměrně složitě:

$$C(u,v) = \frac{\sum_{i=0}^{q} \sum_{j=0}^{r} w_{ij} P_{ij} N_i^m(u) N_j^n(v)}{\sum_{i=0}^{q} \sum_{j=0}^{r} N_i^m(u) N_j^n(v)}, \tag{7.18}$$

7.2.6 T-spline

Následovníkem NURBS se staly T-spline plochy. Jsou ve všem podobné NURBS plochám – řídicí body, bázové B-spline funkce, váhy ALE NEPOTŘEBUJÍ PRAVIDELNOU SÍŤ. Jejich uspořádání je libovolné a jsou propojeny pouze pomocí lokálních uzlových vektorů, které popisují jejich vztah k ostatním bodům v okolí. Nepravidelné navazování je nazýváno T-junctions, což je místo, kde v síti bodů leží jeden krajní bod hrany uprostřed hrany jiné.

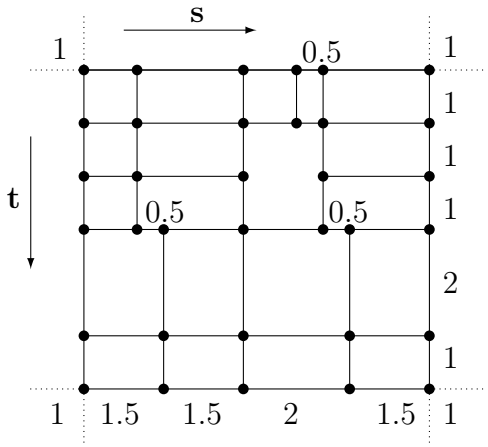
Výhodou používání T-spline je omezení počtu kontrolních bodů. Každý NURBS plocha je zároveň i T-spline. Pomocí algoritmu nazvaného T-spline simplification lze omezit počet řídicích bodů původní NURBS plochy na polovinu až třetinu. Dále se zbavíme tzv. superflous(nadbytečných) bodů , které jsou v NURBS síti pouze pro doplnění její pravidelnosti.



Obrázek 7.5: NURBS plocha a její řídicí síť

Další předností T-spline je velmi dobré napojování libovolně velkých ploch. Pro grafiky je také velmi výhodná funkce přidávání nových bodů pomocí algoritmu T-spline refinement bez změny tvaru výsledné plochy a bez přidávání celého sloupce a řádku do sítě.

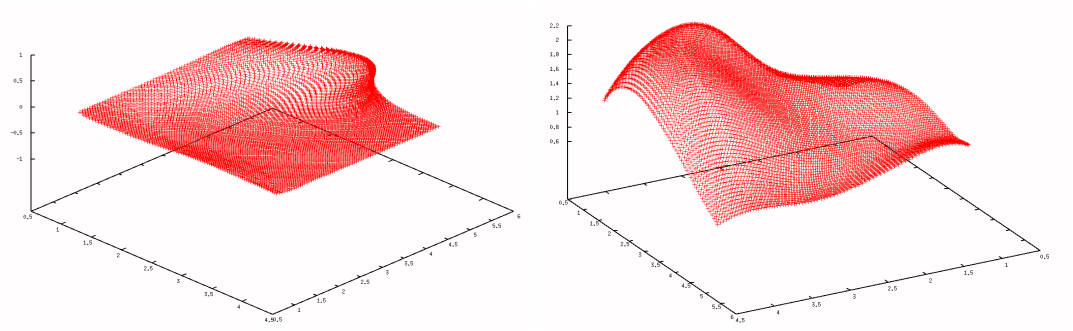
Pravidla pro tvorbu T-mesh sítí a T-spline ploch jsou náročnější, ale zájemci je naleznou v odkazech.



Obrázek 7.6: Obecná T-mesh

7.3 Kontrolní otázky

1. Co jsou to matematické křivky a plochy?



Obrázek 7.7: Obecná T-spline plocha ($z = 0, z$ obecné)

2. Jaké znáte matematické křivky?
3. Vysvětlete základní princip tvorby mat. křivek.
4. Prochází bézierova křivka krajními body a proč?
5. Jaký je rozdíl mezi bézierovými a B-spline křivkami a plochami?
6. Co znamená zkratka NURBS?
7. Jaký je hlavní rozdíl mezi NURBS a T-spline?
8. Co dělají algoritmy T-spline simplification a T-spline refinement?

7.4 Literatura

Procházková, J. - Disertační práce
Martišek, D. - Matematické principy grafických systémů

Kapitola 8

Transformace v rovině

8.1 Transformace v rovině

Mějme dány body $A\,[1,2]\,, B\,[3,5]$ a rovnice

$$\begin{aligned} x_1' &= x_1 + 6 \\ x_2' &= x_1 + x_2 + 3 \end{aligned} \tag{8.1}$$

Pomocí těchto bodů získáme body A', B'

$$\begin{aligned} a_1' &= a_1 + 6 = 1 + 6 = 7 \\ a_2' &= a_1 + a_2 + 3 = 1 + 2 + 3 = 6 \end{aligned} \tag{8.2}$$

Bod A' má souřadnice $[7,6]$ a bod $B' = [9,11]$
Provedli jsme transformaci úsečky AB na úsečku AB pomocí rovnic 8.1. Rovnice 8.1 budeme nazývat **transformační rovnice** a lze jej maticově zapsat jako

$$\mathbf{x}' = T * \mathbf{x} + \mathbf{r} \tag{8.3}$$

Matice **T** se nazývá **transformační matice**.V našem případě vypadá takto

$$T = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

Příklad:
Mějme dány body $A\,[1,-5]\,, B\,[7,6]$ a rovnice

$$\begin{aligned} x_1' &= x_1 + x_2 + 6 \\ x_2' &= x_1 + 3 * x_2 + 3 \end{aligned} \tag{8.4}$$

Pomocí těchto bodů vypočtěte body A', B' a napište transformační matici **T**

Homogenizace

Při práci s transformace se používá homogenizace ke zjednodušení výpočtu. Vezme se transformační matice **T** a vektor **r** z rovnice transformace $\mathbf{x}' = T * \mathbf{x} + \mathbf{r}$ a vytvoří se z nich jediná matice 3 krát 3.

$$H = \begin{pmatrix} t_{11} & t_{12} & r_1 \\ t_{21} & t_{22} & r_2 \\ 0 & 0 & 1 \end{pmatrix}$$

Pro použití v programu je nutné tuto matici transponovat

$$H = \begin{pmatrix} t_{11} & t_{21} & 0 \\ t_{12} & t_{22} & 0 \\ r_1 & r_2 & 1 \end{pmatrix}$$

Typy zobrazení

1. Osová souměrnost

   z rovnic

   $$\begin{aligned} x_1' &= x_1 \\ x_2' &= -x_2 \end{aligned} \tag{8.5}$$

   vyplývá matice

   $$H_o = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

2. Středová souměrnost

   z rovnic

   $$\begin{aligned} x_1' &= -x_1 \\ x_2' &= -x_2 \end{aligned} \tag{8.6}$$

   vyplývá matice

   $$H_s = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

3. Posunutí

   z rovnic

   $$\begin{aligned} x_1' &= x_1 + u_1 \\ x_2' &= x_2 + u_2 \end{aligned} \tag{8.7}$$

vyplývá matice

$$H_o = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ u_1 & u_2 & 1 \end{pmatrix}$$

4. Stejnolehlost – homotetie
z rovnic

$$\begin{aligned} x_1' &= \lambda x_1 \\ x_2' &= \lambda x_2 \end{aligned} \tag{8.8}$$

vyplývá matice

$$H_o = \begin{pmatrix} \lambda & 0 & 0 \\ 0 & \lambda & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

5. Osová afinita
z rovnic

$$\begin{aligned} x_1' &= p_1 x_1 \\ x_2' &= p_2 x_1 + x_2 \end{aligned} \tag{8.9}$$

vyplývá matice

$$H_o = \begin{pmatrix} p_1 & p_2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

8.2 Transformace v prostoru

Rozdíl mezi transformací v rovině a v prostoru je pouze v tom, že transformační matice je o jeden rozměr větší, tedy 4*times*4. Můžeme ji zapsat jako:

$$H = \begin{pmatrix} t_{11} & t_{21} & t_{31} & 0 \\ t_{12} & t_{22} & t_{32} & 0 \\ t_{13} & t_{23} & t_{33} & 0 \\ r_1 & r_2 & r_3 & 1 \end{pmatrix}$$

Na ukázku uvedeme několik typů transformačních matic pro transformace v prostoru. Může to být například otáčení kolem některé ze souřadných os, středová souměrnost podle počátku či posunutí. Obecné transformace dostáváme složením těchto základních. Prakticky je to v programu prováděno postupným součinem transformačních matic.

Příklady transforamčních matic

Souměrnost podle počátku

$$H_s = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Posunutí

$$H_o = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ u_1 & u_2 & u_2 & 1 \end{pmatrix}$$

Osová souměrnost podle roviny xy.

$$H_{xy} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Osová souměrnost podle osy y

$$H_y = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

8.3 Kontrolní otázky

1. Jaké znáte transformace?
2. Jaký je matematický rozdíl mezi transformací v rovině a prostoru?
3. Napište transformační matici středové souměrnosti v rovině.
4. Napište transformační matici osové souměrnosti v rovině.
5. Jak probíhá skládání transformací?

8.4 Literatura

Martišek, D. - Matematické principy grafických systémů

Index