

# Základy počítačové grafiky

## Úvod do předmětu

Michal Španěl

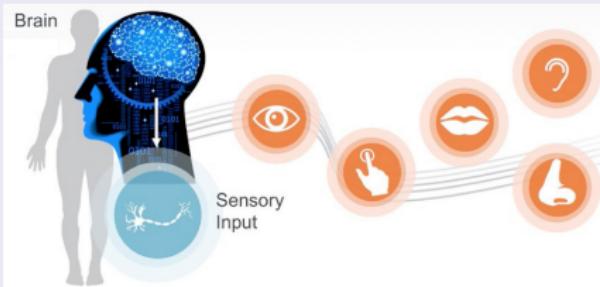
Ústav počítačové grafiky a multimédií  
Fakulta informačních technologií  
Vysoké učení technické v Brně

Brno 2016

# Proč právě grafická informace?

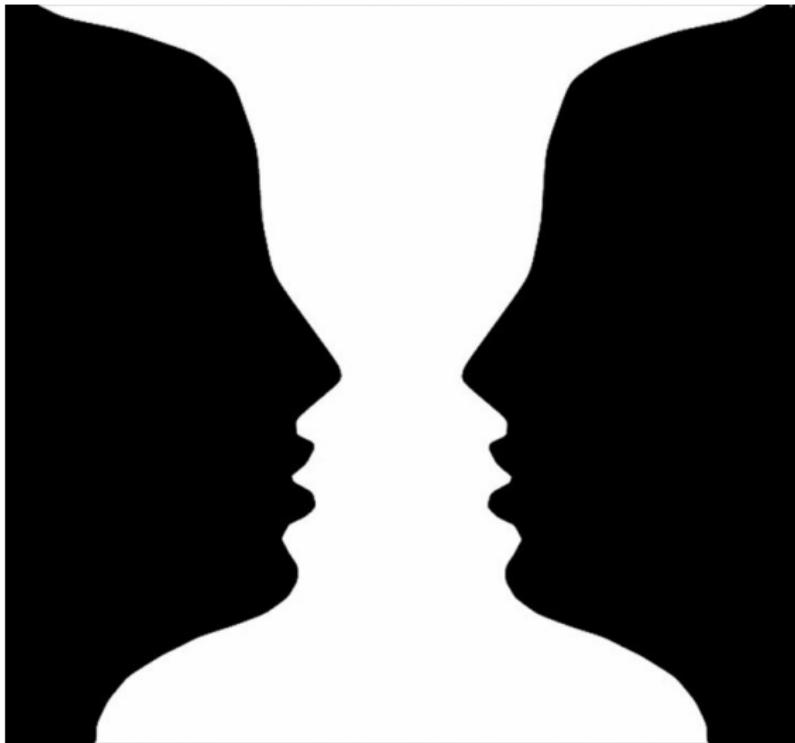
Grafické vyjádření nese velké množství informace

- Rychlé zpracování a dobré interpretační vybavení (oči a mozek)

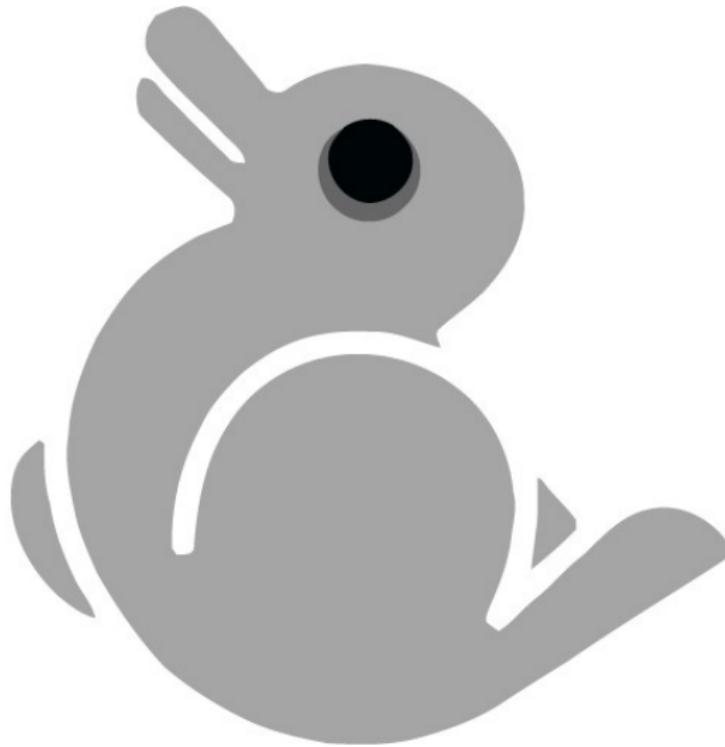


- Jiné formy vyjádření jsou více závislé na interpretaci (větší prostor pro fantazii)
- Špatná interpretace vede ke klamům, desinterpretaci, což se nevyhýbá ani grafice...

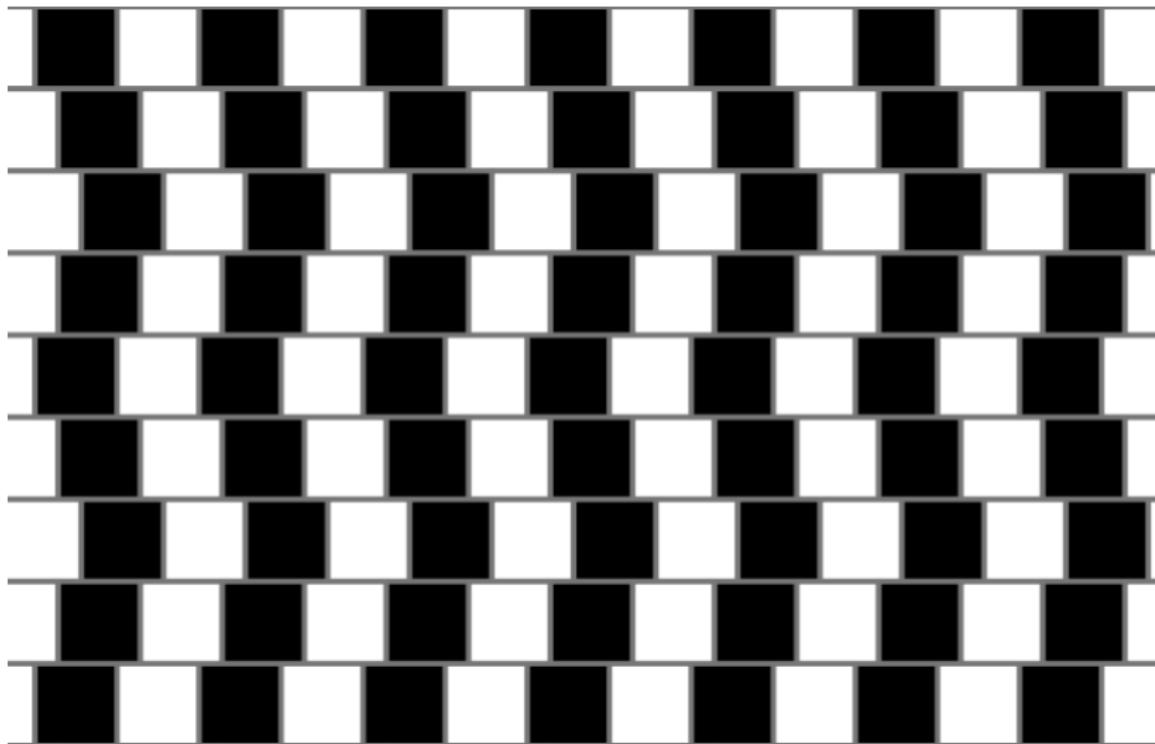
# Grafické klamy – interpretace obsahu



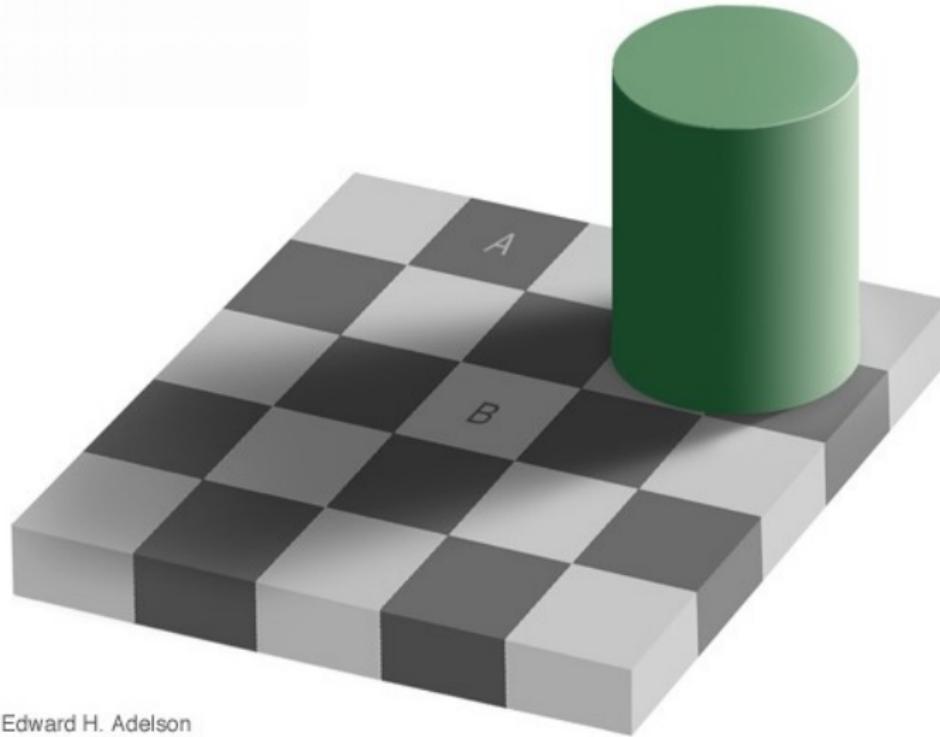
# Grafické klamy – interpretace obsahu



# Grafické klamy – interpretace struktury



# Grafické klamy – interpretace struktury



Edward H. Adelson

# Počítačová grafika

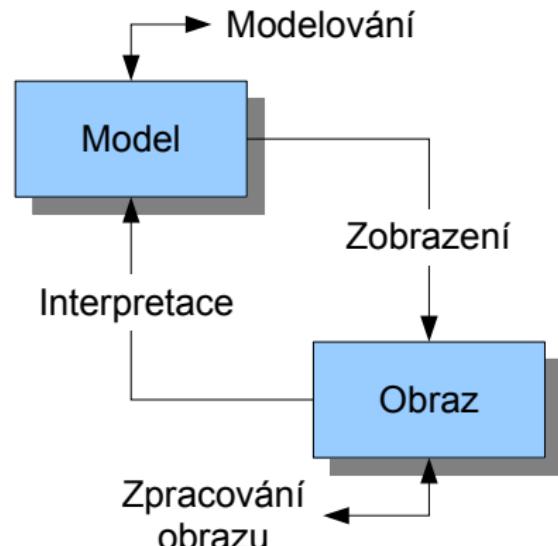
## Definice

Počítačová grafika (Computer Graphics - CG) se zabývá

- analýzou (interpretací)
- tvorbou (syntézou, generováním)

grafické obrazové informace.

- Není jednotná teorie, *syntetický obor*

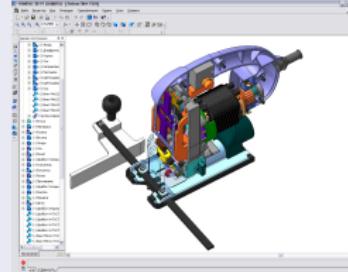
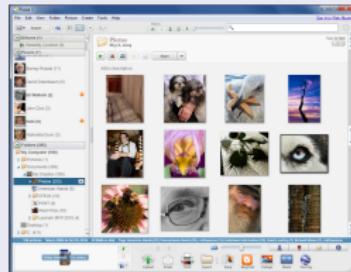


# Obecné dělení Počítačové grafiky

Mnoho různých úhlů pohledů...

## Z pohledu uživatele

- Domácí zpracování fotek nebo videa
- Tvorba grafických aplikací z pohledu programátora
- Prezentační nebo reklamní grafika
- 3D modelování a konstruování ve strojařině, stavařině, atd.



# Obecné dělení Počítačové grafiky

## Podle interactivity práce

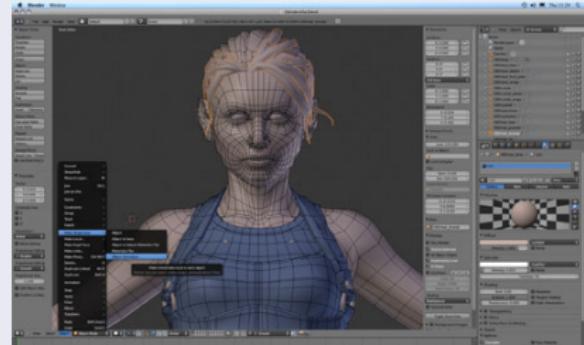
- Interaktivní zpracování dat, malování, modelování atd.
- Offline generování dat - tvorba realistických animací 3D scén ve filmu, architektuře, atd.



# Obecné dělení Počítačové grafiky

## Podle dimenze zpracování dat

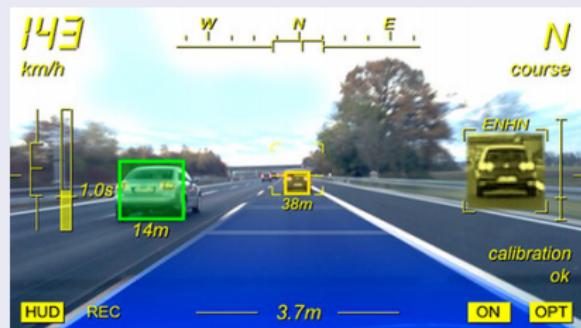
- 2D nebo 3D grafika



# Obecné dělení Počítačové grafiky

## Analýza vs. syntéza obrazových dat

- Rozpoznávání a sledování objektů, třídění, kontrola atd.



# Počítačová grafika je dnes všude... .

- Zpracování fotek a videa
- Reklamní grafika
- Filmy a hry
- Navrhování, konstruování (CAD) a výroba (CAM)
- Grafické uživatelské rozhraní (GUI)
  - Osobní počítače
  - Mobilní telefony, tablety, wearables, atd.
  - Televize, pračky, ledničky atd.
  - Herní konzole
- Vizualizace dat (věda a výzkum, ekonomika, simulace)
- Medicína (diagnostika CT/MR/US, navigace, plánování, ...)
- atd.

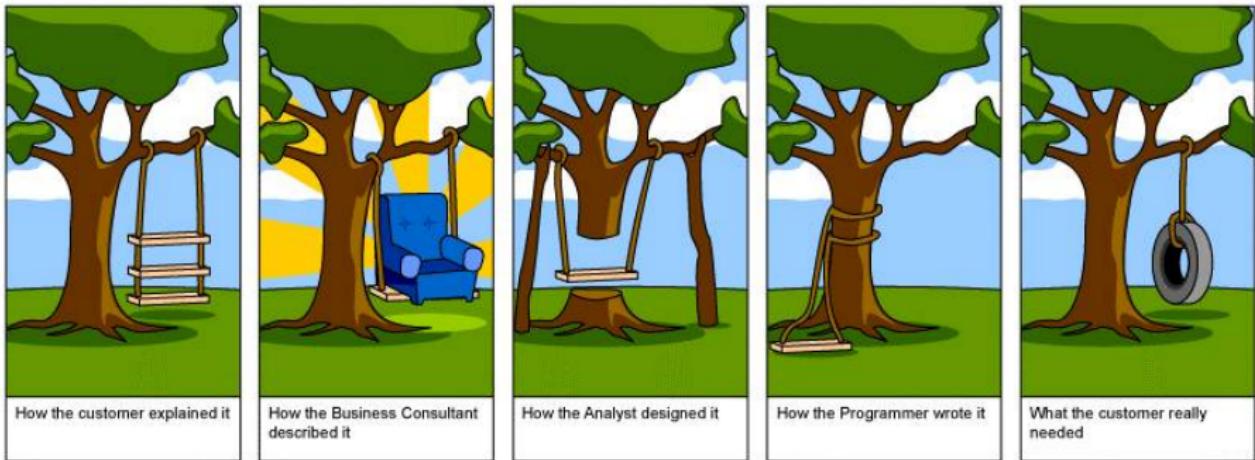
# Cíl předmětu

Co myslíte, že se naučíte?

Co byste se chtěli naučit?

# Cíl předmětu

Získat přehled o základních principech a metodách oboru *Počítačová grafika* z pohledu programátora.



# Koncept předmětu

- Nejde o složitou teorii nebo hlubokou matematiku
  - ...ale s nějakou matikou se potkáme
- IZG je přehledový předmět
  - ...přináší mnoho nových *základních* informací
- Důležité je hledat klíčové poznatky a jejich vzájemné souvislosti (pravidlo 80/20)
  - ...vyžaduje aktivní přístup: poslouchat, přemýšlet a klást si otázky (proč?, jak?, atd.)

"Neboť obávaný klíč všeho opravdového vědění jest otázka:  
Proč?"(Sinuhet, M. Waltari)

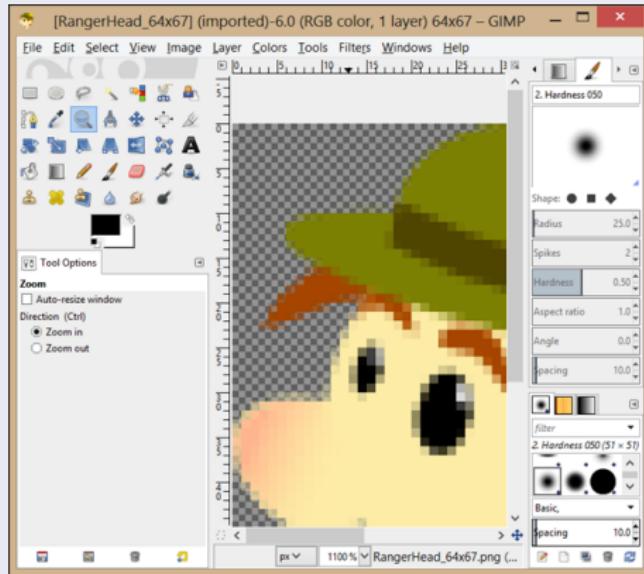
# O čem budou přednášky?

Přednášky se věnují teoretické stránce popisovaných algoritmů a metod (jak to funguje, co je problém, jak to řešit, atd.).

# O čem budou přednášky?

## 2D grafika

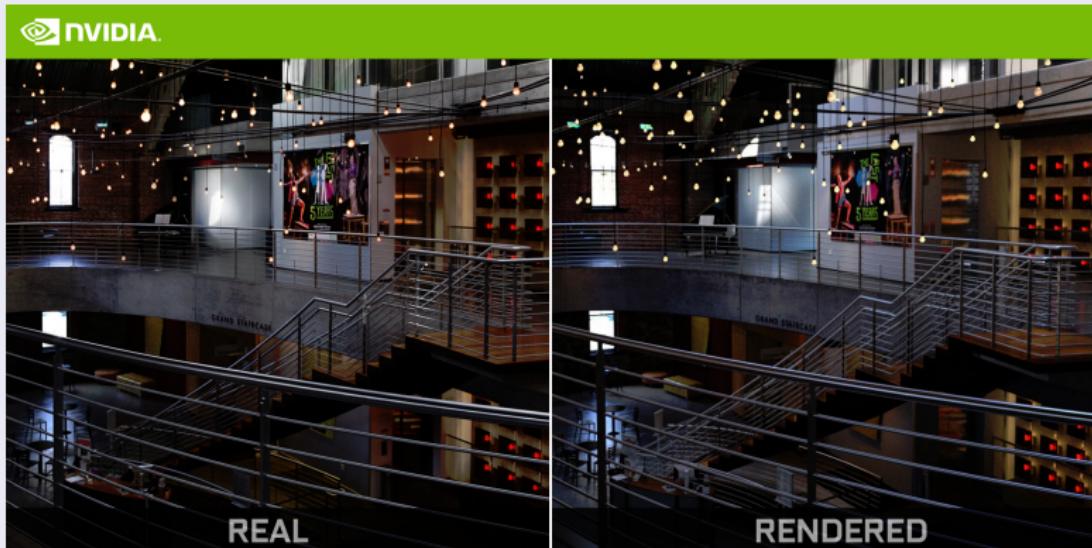
- Principy vykreslování čar, křivek, polygonů do rastrových obrázků
- Transformace a manipulace se 2D objekty



# O čem budou přednášky?

## 3D grafika

- Popis 3D objektů, materiály, texturování, osvětlení, ...
- Principy vykreslování 3D scény



# Organizace přednášek

- 13 přednášek po 3 hodinách
- Dvě přednáškové skupiny
- Přednášející:
  - M. Španěl (většina přednášek)
  - A. Herout, T. Milet (3 přednášky)
- Slide pro přednášky jsou dostupné v PDF na wiki stránce předmětu

# Organizace cvičení

Cvičení se věnují problematice implementace přednášených algoritmů a metod (jak se to dělá, jak se to napíše, atd.).

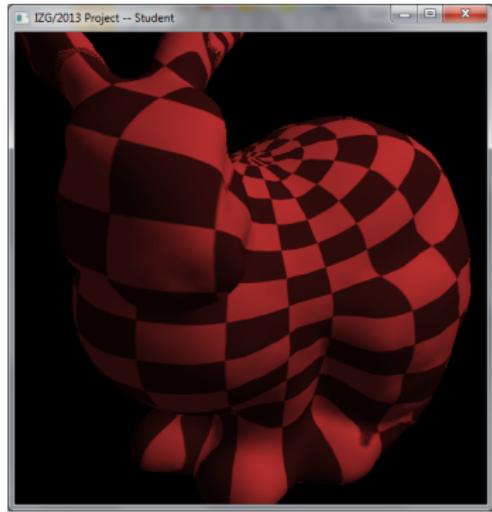
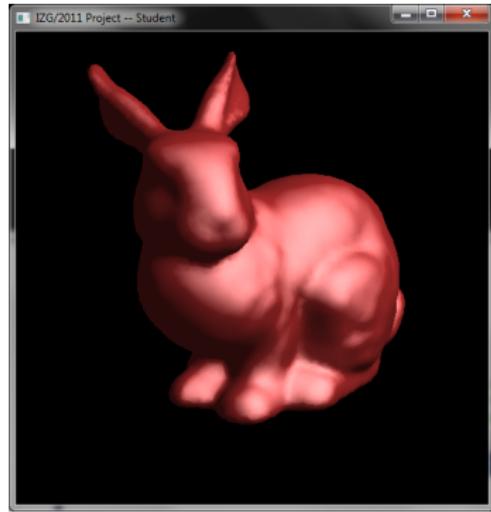
- 6 témat po dvou týdnech = 12 týdnů + zápočtový týden = 13 týdnů
- Každé cvičení má na starosti jiný vyučující
- Reklamace a připomínky ke konkrétnímu cvičení řešte s příslušným vyučujícím
- Studenti navštěvují cvičení v sudých nebo lichých týdnech
- Přihlašování na termíny cvičení prostřednictvím IS FIT

# Organizace cvičení

- Součástí každého cvičení je samostatný bodovaný úkol ohodnocený vždy na konci cvičení 0 - 3 body
- Celkově je možné získat  $6 \times 3 = 18$  bodů
- Materiály pro cvičení (programy, prezentace, atd.) jsou dostupné na wiki stránce předmětu

# Projekt

Projekt je v podstatě rozšířenější cvičení (jak se to dělá, co je efektivní, jak se to napíše, atd.)



- Témata z oblasti 3D zobrazení, stínování, texturování, atd.

# Projekt

- Za projekt je možné získat až 18 bodů
- Základem je funkční program (framework), do kterého doplňujete (implementujete) určené metody (algoritmy)
- Odhadovaná časová náročnost (příprava, psaní kódu, ladění) je 10 hod.
- Materiály k projektu (program, zadání, atd.) budou opět na wiki
- Termín odevzdání projektu je na konci 11-tého týdne semestru!!!

# Zkoušky a písemky

- Doposud bývaly (?) kombinací testových otázek + teoretických slovních otázek
- Závěrečná písemka – maximálně 52 bodů
- Jeden řádný a dva opravné termíny
- Půlsemestrální písemka – maximálně 12 bodů
- Nenahrazuje se!

# Podmínky úspěšného absolvování předmětu

- Celkově získat alespoň 50 bodů a zápočet
- Minimum ze závěrečné písemky je 20 bodů
- Zápočet je potřeba pro připuštění ke zkoušce
- Pro získání zápočtu je třeba během semestru získat alespoň 20 bodů (cvičení, projekt, půlsemestrálka)
- Pokud bude odhalena nedovolená spolupráce na projektu (plagiátorství), znamená to ztrátu zápočtu. Zároveň bude zváženo zahájení disciplinárního řízení...

# Celkový přehled hodnocení předmětu

- Průběh semestru (maximálně 48 bodů):
  - Cvičení – až 18 bodů
  - Projekt – až 18 bodů
  - Půlsemestrální písemka – až 12 bodů
- Zápočet:
  - Podmínka získat alespoň 20 bodů během semestru
  - Zápočet je možné ztratit...
- Semestrální písemka
  - Maximum – 52 bodů
  - Minimum – 20 bodů

# Doporučená literatura (výběr)

- Žára, J., Beneš, B., Felkel, P.: Moderní počítačová grafika, Computer Press, 1998.
- Žára, J., Limpouch, A., Beneš, B., Werner, T.: Počítačová grafika - principy a algoritmy, GRADA, 1992.
- Sochor, J., Žára, J., Beneš, B.: Algoritmy počítačové grafiky, Skriptum ČVUT, Vydavatelství ČVUT, Praha, leden 1998.
- Foley, J., D., A. van Dam, ...: Computer Graphics: Principles and Practice, Addison-Wesley, ISBN: 0201848406, 1995.
- Watt, Allan: 3D Computer Graphics, Addison-Wesley, 2nd edition, ISBN: 0-201-63186-5, 1993.
- Watt, Allan, Watt, Mark: Advanced Animation and Rendering Techniques, Addison-Wesley, ISBN: 0-201-54412-1, 1992.

# Naše dělení Počítačové grafiky

## Zaměření předmětu

Problematika Počítačové grafiky z pohledu programátora, což odpovídá zaměření vašeho studia.

- Z tohoto pohledu je *rozhodující* reprezentace (popis) zpracovávaných dat

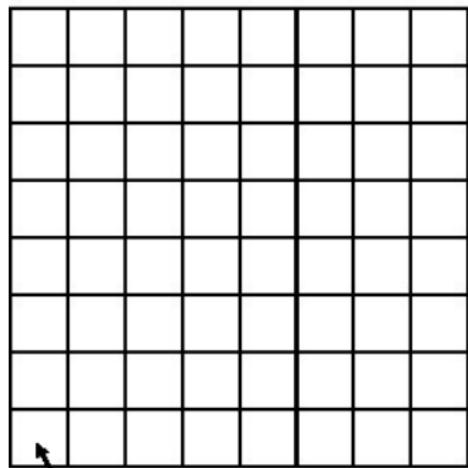
# Rastrová grafika

## Definice

Zpracovávané a zobrazované informace popisujeme a ukládáme diskrétně ve formě *rastrové matice* (2D/3D)

- Jeden prvek obrazové matice nazýváme *pixel*
- Tento popis získáme: manuálně (malování), syntézou (generováním, převodem) nebo snímáním (kamerou)

## Matice



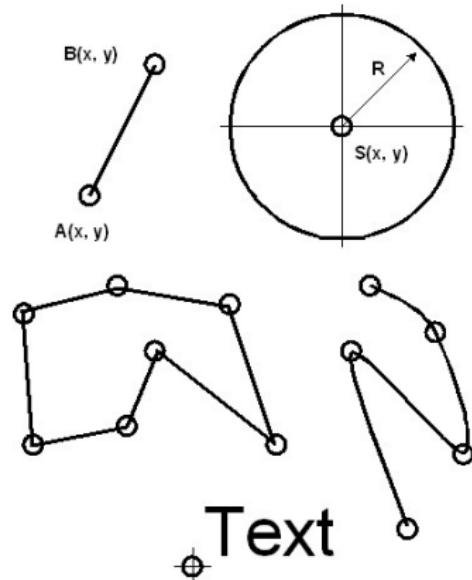
Pixel (hodnota,  
Index, RGB, ...)

# Vektorová grafika

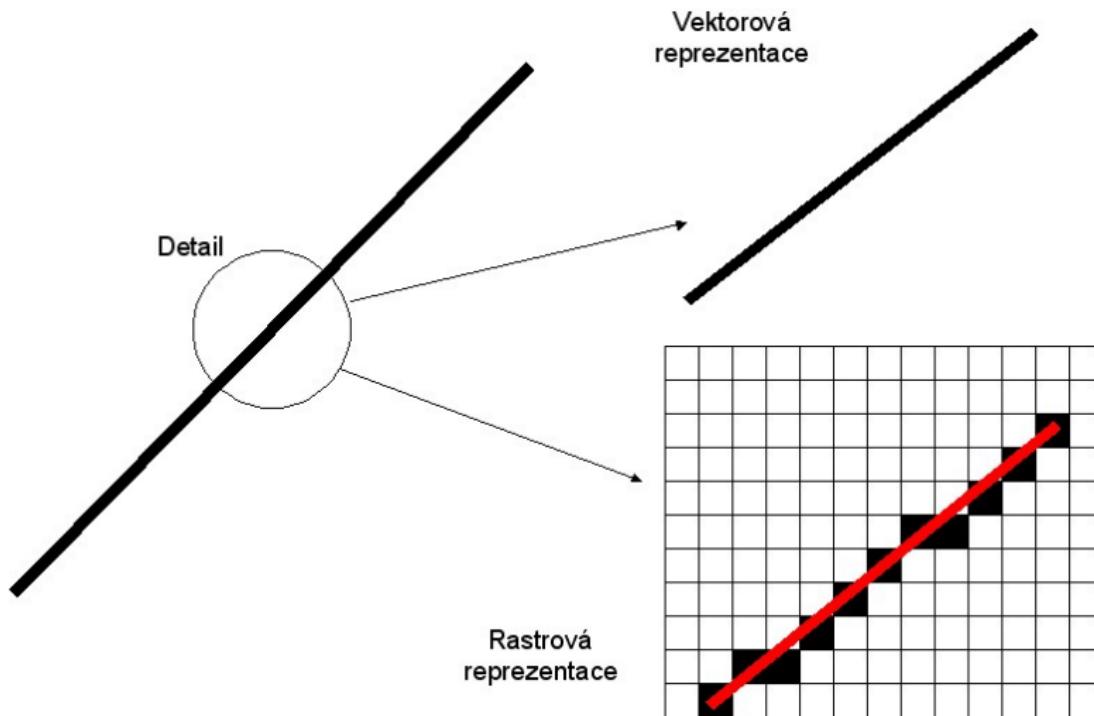
## Definice

Zpracovávané a zobrazované informace popisujeme a ukládáme analyticky ve formě skupiny vektorových entit (úsečky, kružnice, křivky, polygony, atd.).

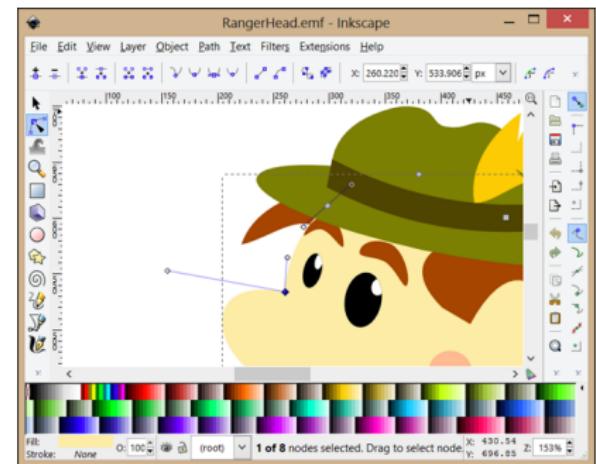
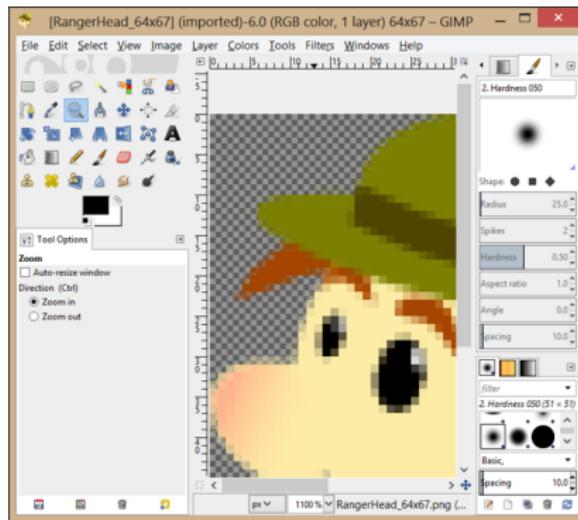
- Tento popis získáme:  
manuálně (kreslení) nebo  
syntézou (generováním)  
nebo převodem z jiného  
popisu



# Vektorová X Rastrová grafika



# Vektorová X Rastrová grafika



# Vektorová X Rastrová grafika

## Vektorová

- Všechny uložené objekty (entity) stále existují
- Vlastnosti (parametry) uložených objektů lze kdykoli měnit
- Přesnost popisu je (teoreticky) neomezená

## Rastrová

- Uchovává se pouze obraz uložených objektů
- Nelze jednoduše měnit vlastnosti (parametry) uložených objektů
- Dané neměnné (snadno) rozlišení (přesnost) popisu

# Vektorová X Rastrová grafika

## Vektorová

- Při zvětšení detailu je zobrazení objektů hladké a spojité



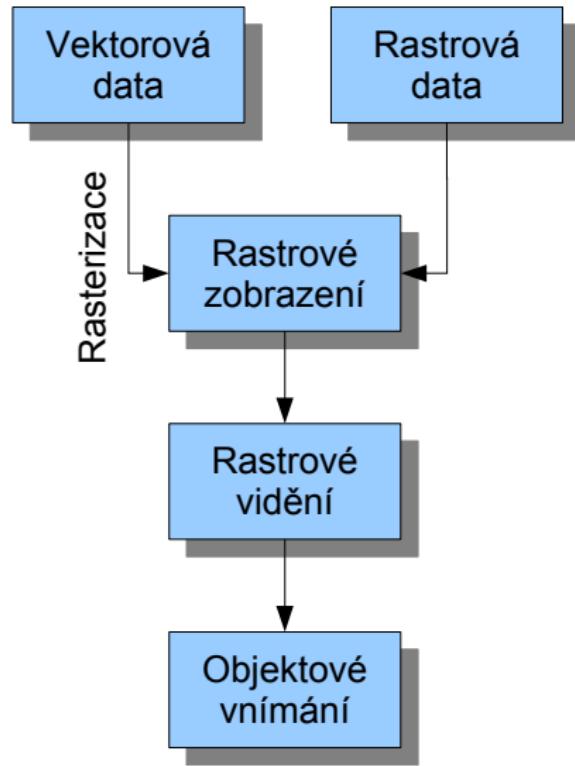
## Rastrová

- Při zvětšení detailu je zobrazení objektů diskrétní (zubaté) a nespojité



# Zobrazení grafické informace

- Popis dat může být vektorový nebo rastrový
- Většina současných zobrazovacích zařízení je rastrová!
- Nezávisle na reprezentaci musíme pro zobrazení zajistit *vhodný rastrový výstup*
- Naše vidění (oko) je rastrové
- Naše vnímání (mozek) je vektorové



# Rasterizace X Vektorizace

## Rasterizace

*Rasterizace* je proces převodu vektorových entit na jejich odpovídající rastrové zobrazení.

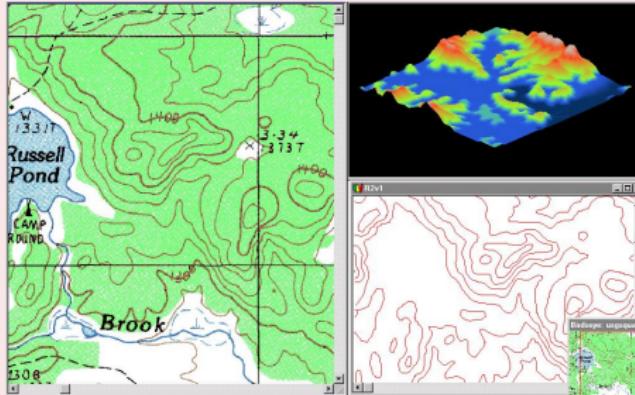
- Tento proces probíhá automaticky se snahou o jeho maximální rychlosť.
- Pro všechny druhy 2D i 3D entit je rasterizace bez problému řešitelná.

# Rasterizace X Vektorizace

## Vektorizace

*Vektorizace* je proces převodu odpovídajícího rastrového zobrazení objektů na jejich vektorový popis.

- Tento proces není triviální a v mnoha případech je nejednoznačný.
- Automatické metody často vyžadují manuální korekci.



# Má smysl (ne)studovat počítačovou grafiku?

## Proč se v současnosti aktivně věnovat počítačové grafice?

- Není to již překonané? Není již vše hotové a vymyšlené?
  - Nejsou již na všechno vytvořeny programy a knihovny?
  - Nestačí jen prostě používat existující nástroje a prostředky?
- 
- Určitě je potřeba používat existující nástroje a prostředky!
  - Zároveň je potřeba dobře znát použité principy a metody, aby
    - ...grafika nevládla nám, ale my jí.
    - ...abychom mohli nástroje efektivně používat a rozvíjet.

# Základy počítačové grafiky

## Barvy a barevné modely

Michal Španěl

Ústav počítačové grafiky a multimédií  
Fakulta informačních technologií  
Vysoké učení technické v Brně

Brno 2016

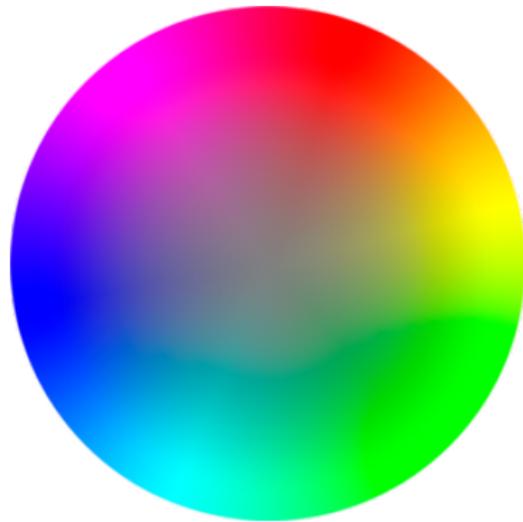
# Cíl přednášky

Seznámit se s problematikou barev v kontextu počítačové grafiky.

Co je to barva?

Jaké barevné modely se používají?

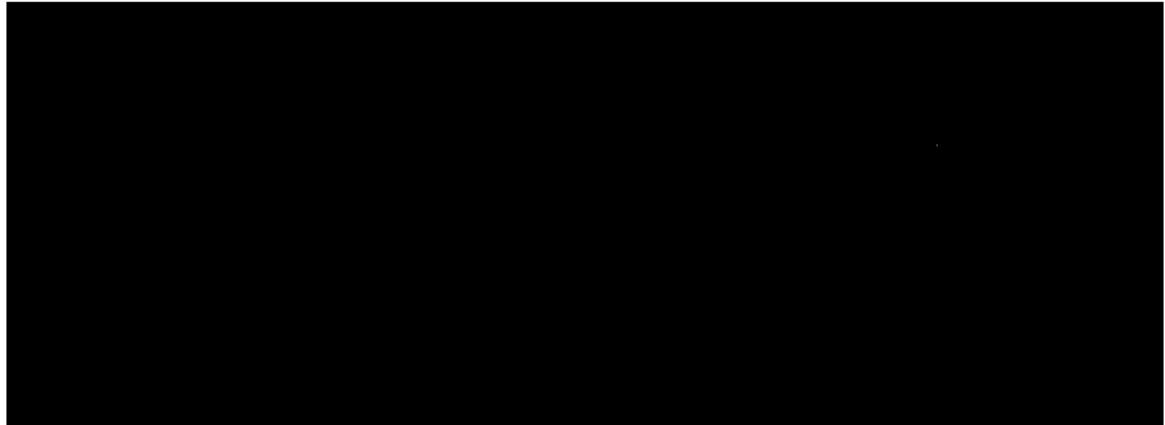
# Barva



Otázka

Co je to BARVA?

# Barva



## Co potřebujeme pro vnímání barvy?

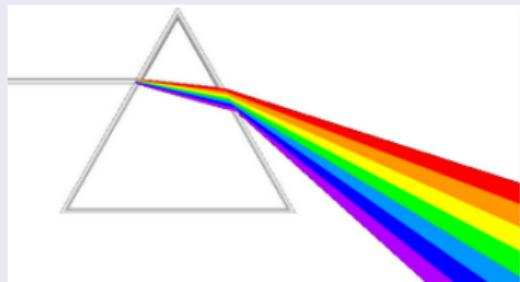
- Světlo (je nositelem informace)
- Objekt (má objektivní fyzikální vlastnosti)
- Pozorovatele (vidí, vnímá světlo / barvu)

# Světlo

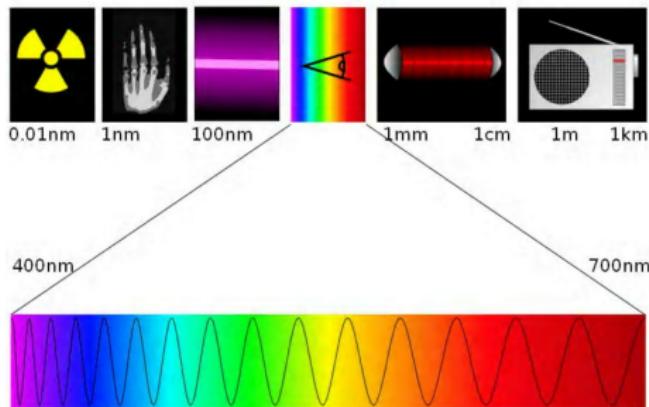
## Co je to světlo?

- Elektromagnetické záření určité vlnové délky
- Je nositelem informace o všem co vidíme
- Achromatické světlo (bílé), rovnoměrné zastoupení vlnových délek
- Chromatické světlo (barevné), nerovnoměrné zastoupení vlnových délek

Rozklad bílého světla hranolem, Isaac Newton, 1666



# Barevné spektrum



## Další charakteristiky světla, kromě barvy

- Jas (svítivost) – intenzita světla, světelného zdroje.
- Sytost – čistota barvy světla, šířka barevného spektra.
- Světllost – velikost achromatické složky světla určité (hlavní) barvy.
- Odstín – dominantní vlnová délka světla (hue).

# Barevný objekt

## Proč vidíme barevné objekty?

- Odrážejí na svém povrchu dopadající světlo
- Podle vlastností povrchu se odrazí jen určité vlnové délky
- **Podle odráženého světla vnímáme barvu objektu**
- Odrazit se mohou jen vlnové délky dopadající na povrch
- **Barevné světlo proto ovlivňuje vnímání barvy objektu, přestože se jeho objektivní vlastnosti nemění**



# Barevné vidění

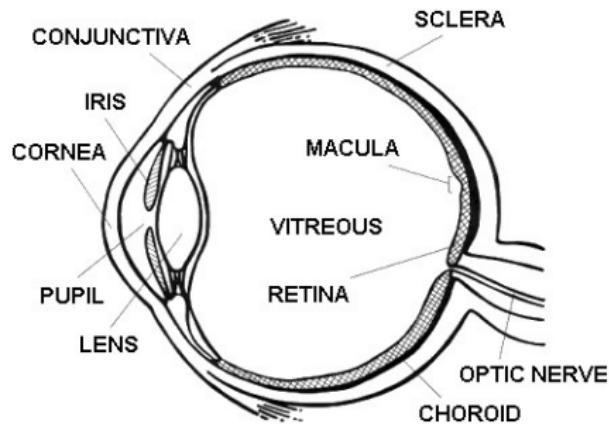
## Vnímání světla

Pro vnímání světla / barvy,  
musíme být vybaveni vhodným  
detektorem = oko

V oku je světlo čočkou  
zaostřeno na sítnici, která  
předá informaci do mozku

Teprve tam vyhodnoceno  
vidění a barva

Barva tedy vzniká až v mozku  
jako vjem !!!

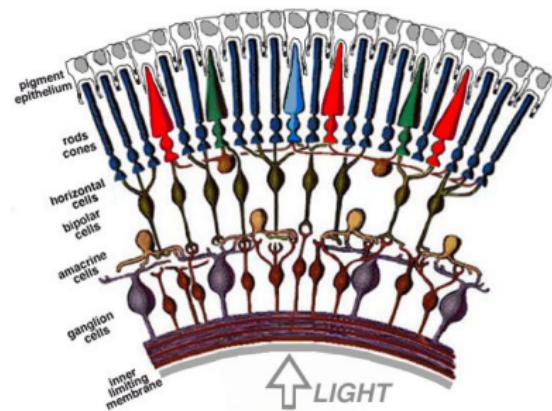


# Barevné vidění

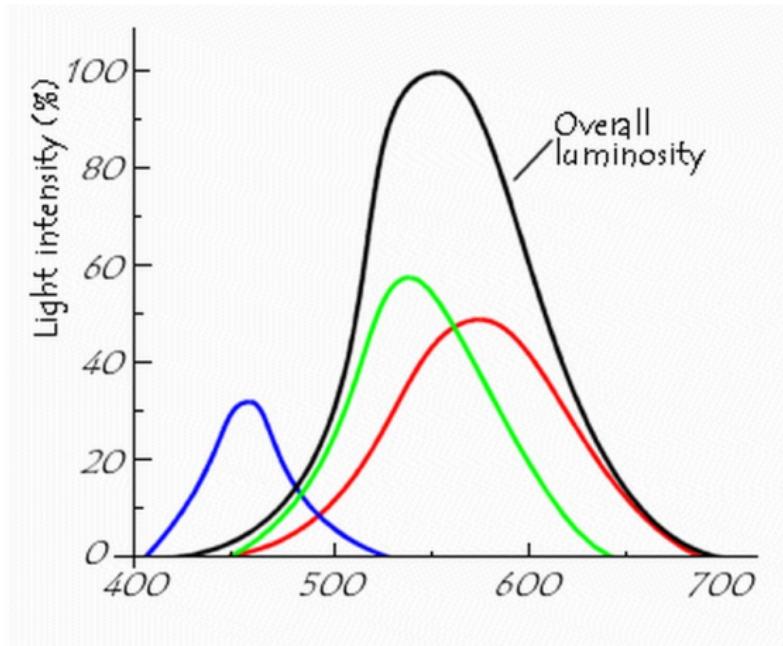
## Sítnice

Na sítnici oka jsou světlocitlivé buňky:

- Tyčinky (rods)
  - Leží více na okraji
  - Reagují na intenzitu světla
  - Noční vidění (šedé)
- Čípky (cones)
  - Leží více ve středu
  - Reagují na různou vlnovou délku světla (RGB)
  - Barevné vidění



# Citlivost oka na vlnové délky světla



# Definice barvy

## Definice

*Barva* je subjektivní vjem vyvolaný působením světla daného spektra šířeného ze zdroje, které se odráží od povrchu objektu (podle jeho objektivních vlastností) a dopadá na sítnici našeho oka.

Oko pak předává získanou informaci do mozku, který ji interpretuje jako barvu odpovídající charakteristickým rozsahům vlnových délek světla odraženého od povrchu objektu

# Míchání barev

## Míchání barev

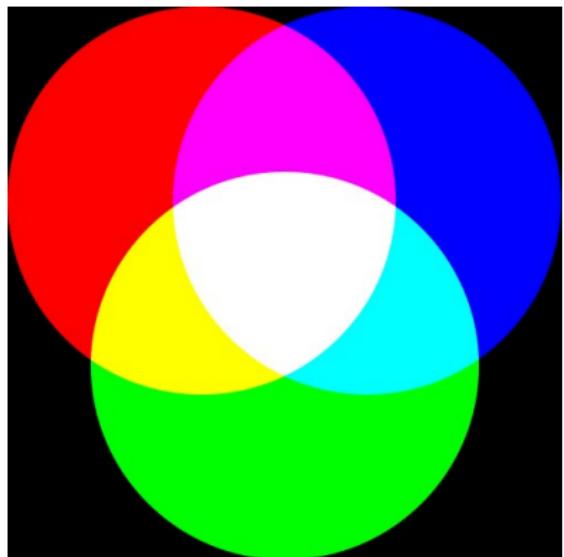
Kombinacemi různých barev (mícháním světla/barev), můžeme získat velké množství různých barevných odstínů:

- Základem jsou vhodně zvolené základní barvy pro míchání dalších odstínů
- Pro všechny existující barvy by bylo potřeba nekonečně mnoho základních barev
- Prakticky však stačí tři základní barvy (Red, Green, Blue)
- Proč právě - Red, Green, Blue?

# Aditivní míchání barev

## Aditivní míchání barev

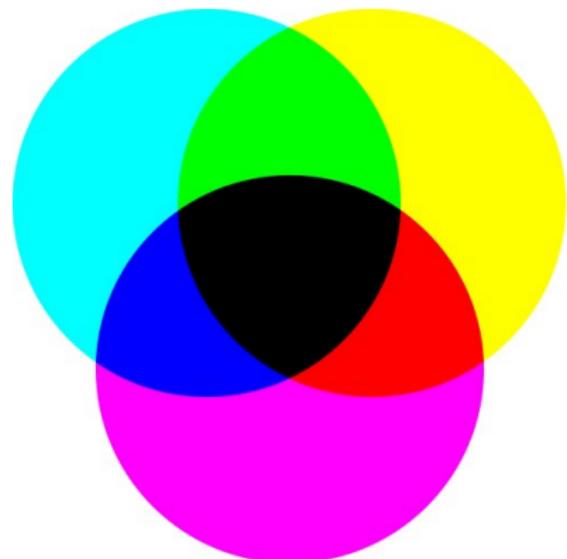
- Práce/míchání se světlem
- Monitory, projektor, kamery atd.
- Základní barvy Red, Green, Blue
- Odpovídá barevnému modelu RGB
- Složením základních barev maximální intenzity vzniká bílá barva



# Subtraktivní míchání barev

## Subtraktivní míchání barev

- Práce/míchání s pigmenty
- Tiskárny, plotry, ofsety atd.
- Základní barvy Cyan, Magenta, Yellow
- Odpovídá barevnému modelu CMY
- Složením základních barev maximální intenzity vzniká černá barva



# Barevné modely

## Barevné modely

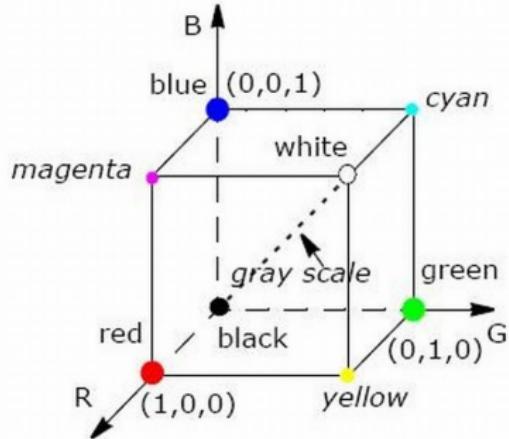
Barevné modely umožňují prakticky pracovat s barvami, uživatelsky vhodně realizují míchání barev, kalibraci a převody barev atd.

Existuje jich velké množství druhů podle specifických požadavků jednotlivých aplikací, např. pro různé druhy uživatelů (programátor, grafik atd.) nebo pro různé oblasti použití (tisk, televizní technika, video, fotografie atd.).

# RGB barevný model

## RGB barevný model

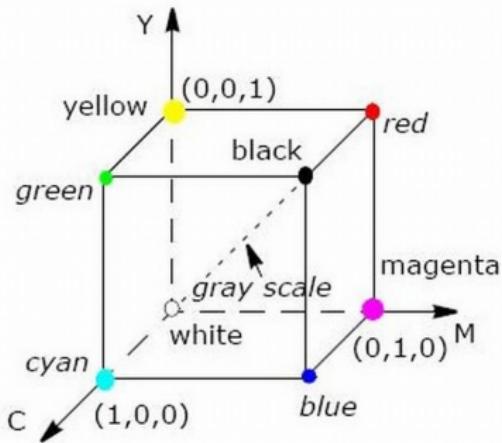
- Odpovídá aditivnímu skládání barev
- Základní barevný model počítačové grafiky a počítačové techniky
- Základní barvy Red, Green, Blue
- Je doplňkový k modelu CMY
- Má tvar jednotkové krychle
- Není příliš intuitivní



# CMY barevný model

## CMY barevný model

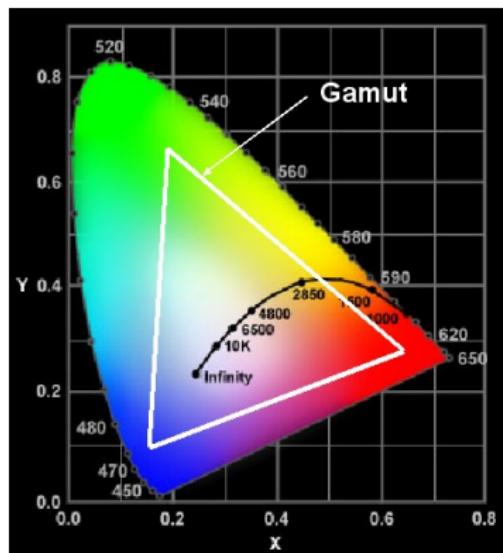
- Odpovídá subtraktivní skládání barev
- Základní barevný model pro tiskárny a tiskový výstup
- Základní barvy Cyan, Magenta, Yellow + black
- Je doplňkový k modelu RGB
- Má tvar jednotkové krychle
- Není příliš intuitivní



# Chromatický diagram

## Chromatický diagram

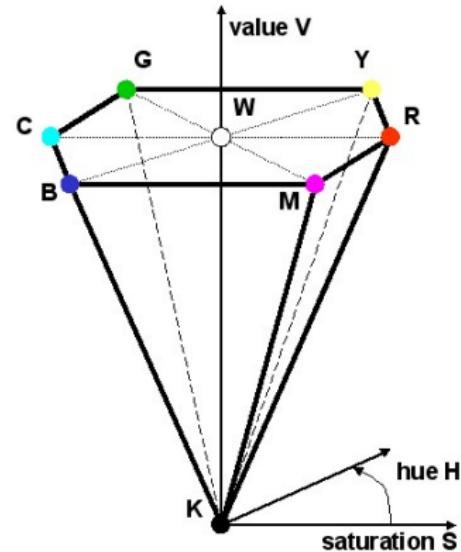
- Mezinárodní standard CIExy
- Barvy slunečního spektra (tzv. locus barev)
- Sestrojen pro dané typy luminoforů
- Kalibr pro porovnání barev
- Ostatní modely jsou jeho podmnožinou (tzv. Gamut)
- Porovnání gamutů výstupních zařízení = kalibrace



# HSV barevný model

## HSV barevný model

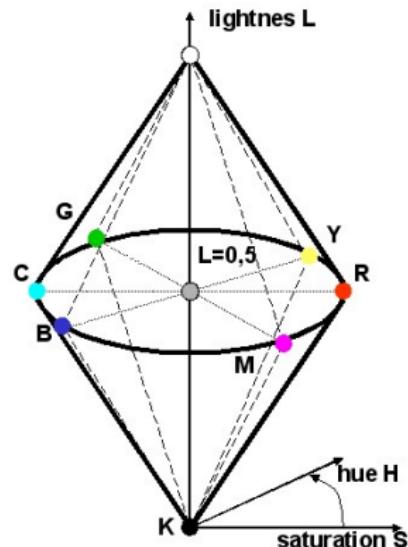
- Uživatelsky orientovaný barevný model
- Nastavuje: sytost (saturation), barevný tón (hue) a světlost (value)
- Intuitivní práce s barvou
- Při úbytku světlosti klesá zároveň také sytost
- Tvar šestibokého jehlanu
- Netriviální převod na RGB/CMY - kontrola mezi



# HLS barevný model

## HLS barevný model

- Uživatelsky orientovaný barevný model
- Nastavuje: sytost (saturation), barevný tón (hue) a jas (lightnes)
- Intuitivní práce s barvou
- Při limitech úbytku jasu klesá zároveň také sytost
- Tvar dvou kuželů
- Netriviální převod na RGB/CMY - kontrola mezí



# Odkazy

## Vyzkoušejte...

- <http://www.etntalk.com/colorpicker/czech/>
- <http://www.netgraphics.sk/conversion-between-rgb-and-hsb-color-models-1>
- [http://www.cs.rit.edu/~ncs/color/a\\_spaces.html](http://www.cs.rit.edu/~ncs/color/a_spaces.html)
- atd.

# Základy počítačové grafiky

## Redukce barevného prostoru

Michal Španěl

Ústav počítačové grafiky a multimédií  
Fakulta informačních technologií  
Vysoké učení technické v Brně

Brno 2016

# Cíl přednášky

Jak smysluplně a efektivně pracovat s barevami?

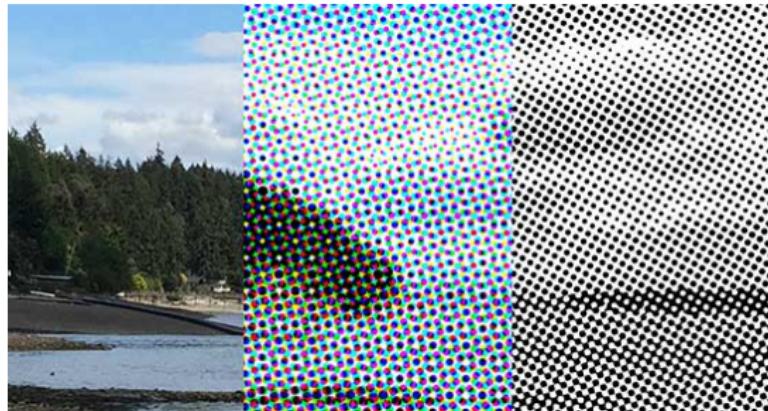
Jak upravit barevná data pro výstup s omezeným počtem barev?



# Motivace

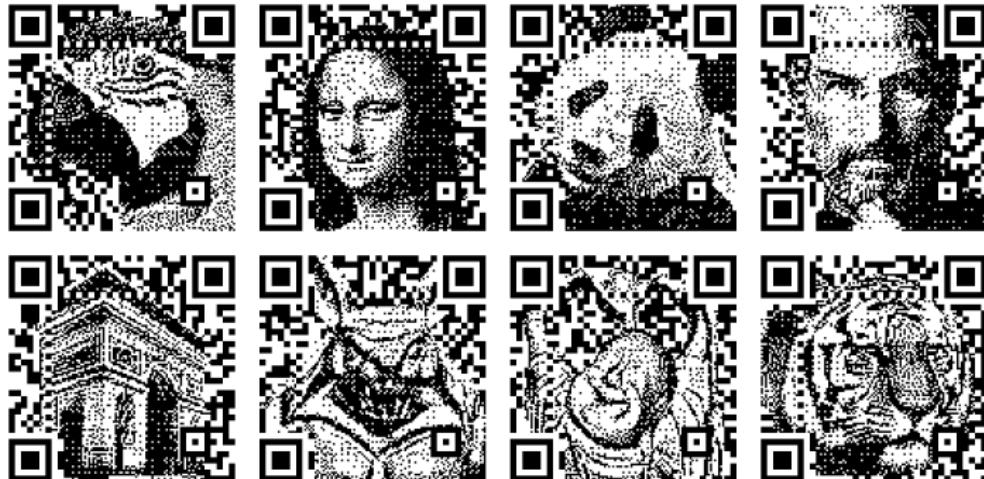
Redukce barevného prostoru pro výstup s omezeným počtem barev

- Tisk na černo-bílé tiskárně (vytvoření šedotónového obrázku).
- Zobrazení na displeji s pouze 256 barvami (mobilní zařízení).
- Komprese obrazových dat omezením počtu barev.



# Trochu jiné použití...

- H.-K. Chu, et al.: Halftone QR Codes (2013)

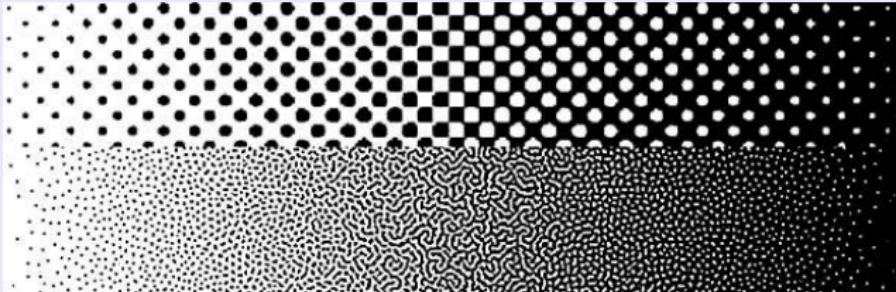


[http://vecg.cs.ucl.ac.uk/Projects/SmartGeometry/  
halftone\\_QR/halftoneQR\\_sigga13.html](http://vecg.cs.ucl.ac.uk/Projects/SmartGeometry/halftone_QR/halftoneQR_sigga13.html)

# Základní principy

Integrační schopnost lidského oka - z několika blízkých barevných bodů si vytvoří barevný odstín

- Kombinace černých a bílých bodů dává stupně šedi.
- Kombinace R, G, B bodů dává barevné odstíny.



# Základní principy

## Komprese obrazu - snížení počtu barev

- Při převodu dochází ke ztrátě dat.



# Barevné obrazy

- Barevné modely *RGB, HSV, ...*
- Typicky tři barevné kanály, každý 8 bitů = *24 bitů na pixel*.



# Achromatické obrazy

## Bílé světlo

Všechny vlnové délky mají stejnou intenzitu.

- > 80% odrazu - bílá barva.
- < 3% odrazu - černá barva.

## Vztah barvy a intenzity

- Empirický vztah:

$$I = 0.299R + 0.587G + 0.114B$$

Převod barevného prostoru na 256 stupňů šedi.



# Achromatické obrazy, pokr.

## Obrazy ve stupních šedi (šedotónové/grayscale obrazy)

- Teoreticky stačí 32-64 stupňů (citlivost oka).
- Prakticky 256 stupňů.



# Achromatické obrazy, pokr.

## Černo-bílé (monochromatické, black-and-white, B/W) obrazy

- Jen dvě úrovně - černá/bílá.



# Metody redukce šedotónového obrazu na černo-bílý

## Dithering (rozptylování)

- Nahrazení původních hodnot intenzity šedé černými a bílými body.
- Snaha o vizuálně maximálně odpovídající podobu.
- Zachovává rozměry obrazu.
- Výstup na obrazovku.

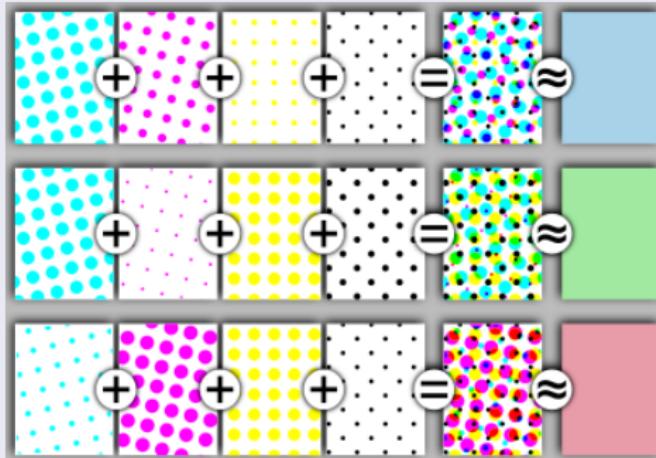
## Halftoning (polotónování)

- Každý pixel nahrazen vzorem černých a bílých bodů dané hodnoty.
- Zvětšuje rozměry obrazu.
- Výstup na tiskárnu.

# Metody redukce šedotónového obrazu na černo-bílý

## Barevné obrazy

- Každý kanál se upravuje zvlášť.
- Další výklad pro převod gray na mono!



# Dithering

## Základní metody

- Prahování
- Náhodné rozptýlení
- Distribuce chyby
- Maticové rozptýlení

# Prahování (angl. thresholding)

- Rozdělení pixelů obrazu podle prahové hodnoty  $T$ .
- Nejprimitivnější metoda.



# Prahování, pokr.

## Algoritmus

- Vstupní obraz  $I(x,y)$ , výstupní binární obraz  $G(x,y)$ .
- Pro každý pixel obrazu:

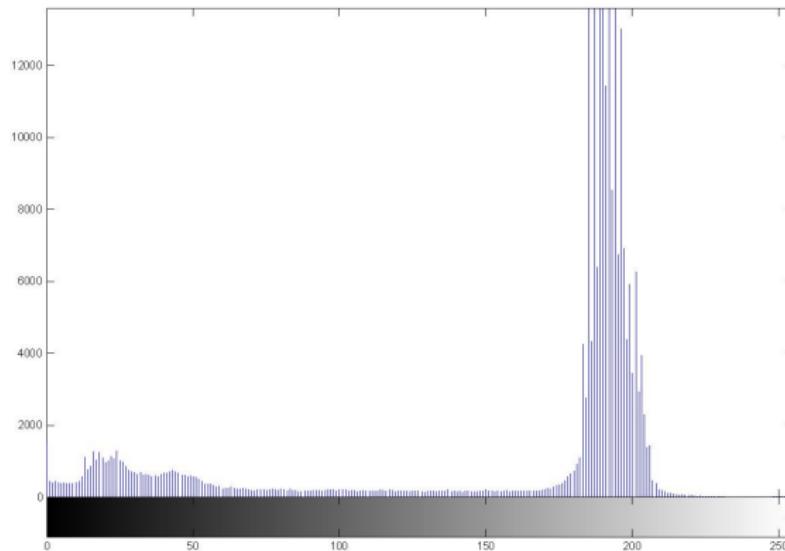
$$G(x, y) = \begin{cases} 1 & \text{pro } I(x, y) \geq T \\ 0 & \text{pro } I(x, y) < T \end{cases}$$

+/-

- Uspokojující pro obrazy s velkým kontrastem.
- Značná degradace obrazu.
- Jak zvolit vhodný práh?

# Optimální výběr prahu

- Práh zvolený uživatelem
- Střední hodnota, medián, apod.
- Analýza histogramu



# Náhodné rozptýlení

- Hodnota prahu generována náhodně pro každý pixel obrazu.
- Efekt "hrubého zrna" simulující staré fotografie.



# Náhodné rozptýlení, pokr.

## Algoritmus

- Pro každý pixel obrazu:

$$G(x, y) = \begin{cases} 1 & \text{pro } I(x, y) \geq \text{random}(I_{max}) \\ 0 & \text{jinak} \end{cases}$$

+/-

- Jednoduchá a rychlá metoda.
- Zachovává jasové poměry v obrazu.
- Rovnoměrná intenzita u velkých ploch.
- Vhodná pro velké obrazy s konstantními plochami .
- Lze modifikovat pro obecný počet úrovní. **Jak?**

# Náhodné rozptýlení, pokr.

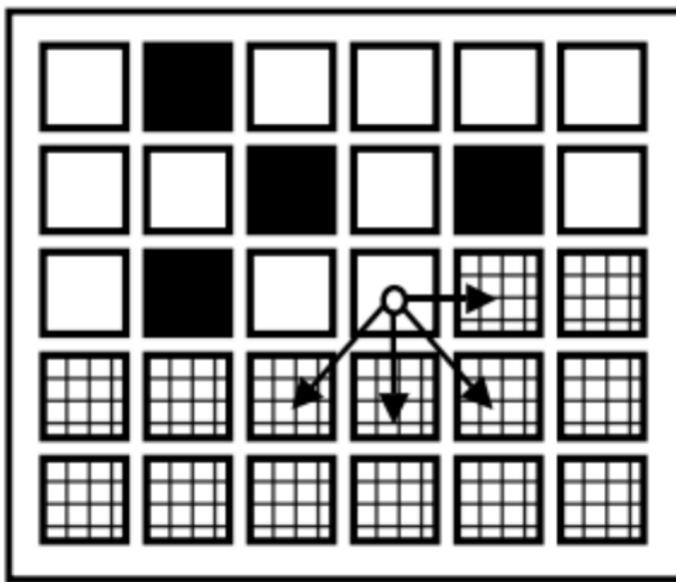
## Algoritmus 2

Pro každý pixel obrazu:

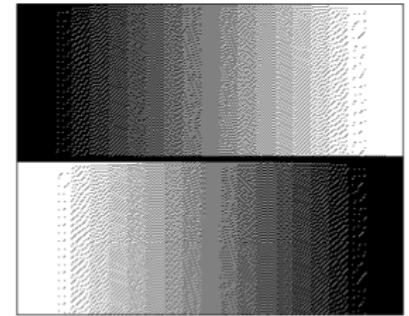
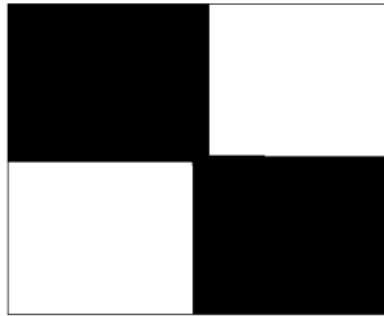
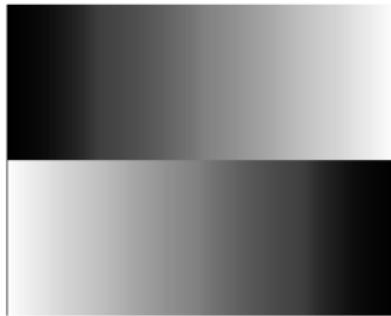
- Inicializuj  $G(x, y) = 0$
- Generuj náhodné prahy  $T_1, \dots, T_n$
- Je-li  $I(x, y) > T_i$ , pak  $G(x, y) += 1$

# Distribuce chyby

- Distribuce vznikající chyby okolním pixelům.
- Maximální využití vstupní informace.



# Prahování vs. prahování s distribucí chyby



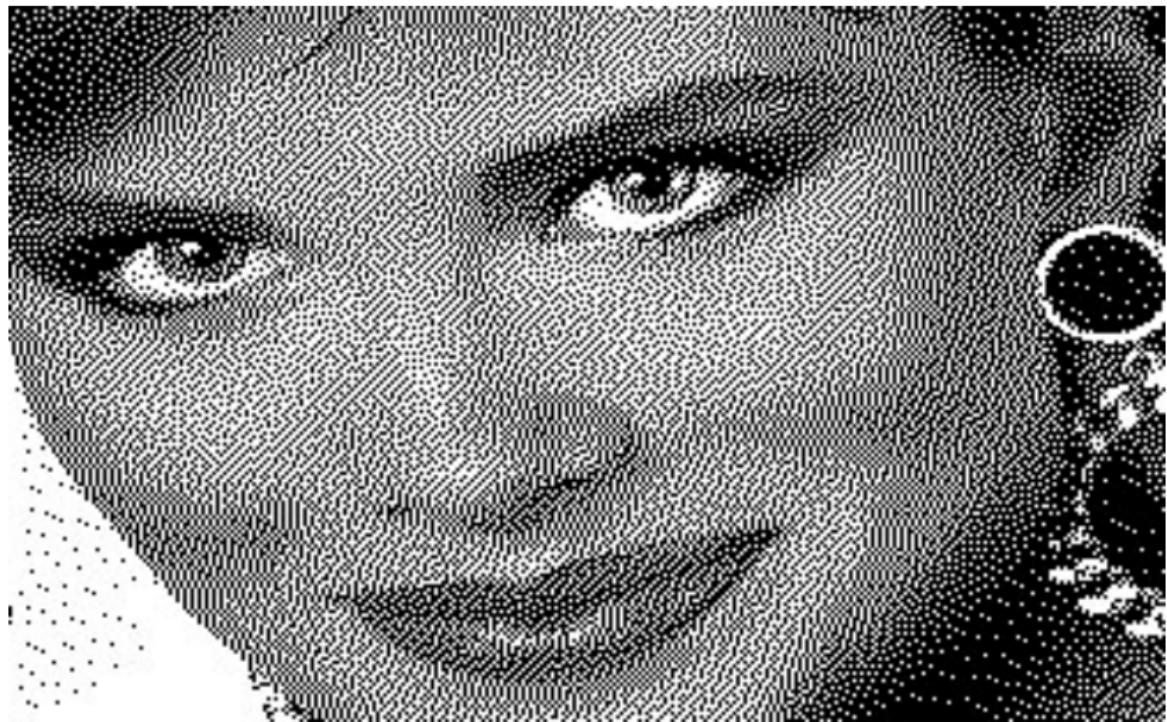
# Distribuce chyby

## Algoritmus

Pro každý pixel obrazu:

- Urči hodnotu  $G(x, y)$  podle dané metody rozptýlení.
- Vypočti chybu  $E$   
Je-li  $G(x, y) = 1$ , pak  $E = I(x, y) - I_{max}$   
jinak  $G(x, y) = 0$  a  $E = I(x, y) - 0$
- Distribuce chyby sousedům (modifikace hodnot pixelů).

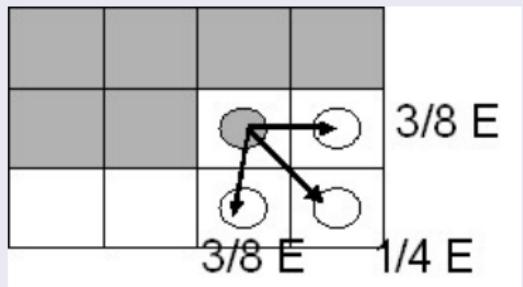
## Příklad rozptýlení s distribucí chyby



# Distribuce chyby, pokr.

- Různé metody rozptýlení hodnot (nejčastěji prahování).
- Různé metody distribuce chyby (Floyd, Bayer, Burkes, Stucky).

## Floyd-Steinberg



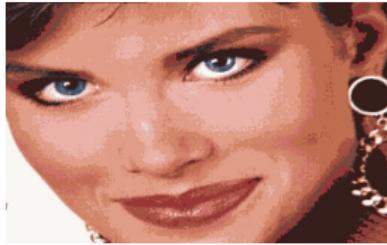
## Pozn.

- Pozor na přetečení rozsahu hodnot pixelu!
- **Jak by se dalo řešit?**
- Omezení (saturace) hodnot
- Pomocný řádkový buffer o velikosti  $N + 1$

# Příklady



Originální barevný RGB obraz



Barevný obraz s paletou 16 barev.



Barevný obraz s paletou 16 barev.  
Použita distribuce chyby.



256 stupňů šedi



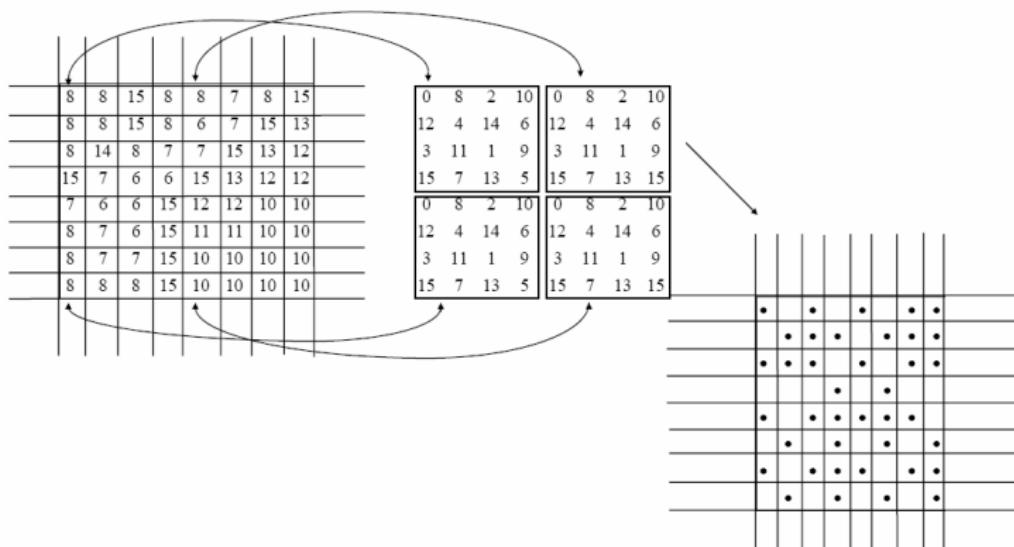
16 stupňů šedi.



16 stupňů šedi, distribuce chyby.

# Maticové rozptylení

- Porovnání pixelů obrazu s odpovídajícími hodnotami distribuční (rozptylovací) matice a prahování.
- Dithering - plochu obrazu pokryjeme maticemi.
- Halftoning - každý pixel nahradíme maticí.



# Maticové rozptýlení

## Algoritmus

Pro každý pixel obrazu:

- Inicializuj  $G(x, y) = 0$
- Je-li  $I(x, y) > M_{x_m, y_m}$ , pak  $G(x, y) = 1$   
 $x_m = x \bmod n$   
 $y_m = y \bmod n$   
n ... řád matice

# Rozptylové matice $n \times n$ různých řádů



1



2



3

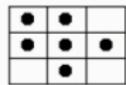
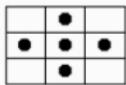
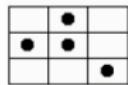
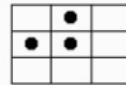
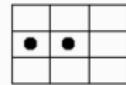
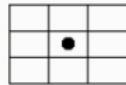
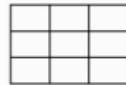


4

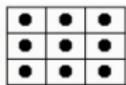
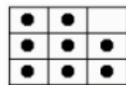


5

$$\approx^{(2)} T = \begin{bmatrix} 3 & 1 \\ 2 & 4 \end{bmatrix}$$



$$\approx^{(3)} T = \begin{bmatrix} 7 & 9 & 5 \\ 2 & 1 & 4 \\ 6 & 3 & 8 \end{bmatrix}$$



$$\approx^{(4)} T = \begin{bmatrix} 1 & 9 & 3 & 15 \\ 13 & 5 & 14 & 7 \\ 4 & 10 & 2 & 12 \\ 16 & 8 & 11 & 6 \end{bmatrix}$$

# Rozptylové matice, pokr.

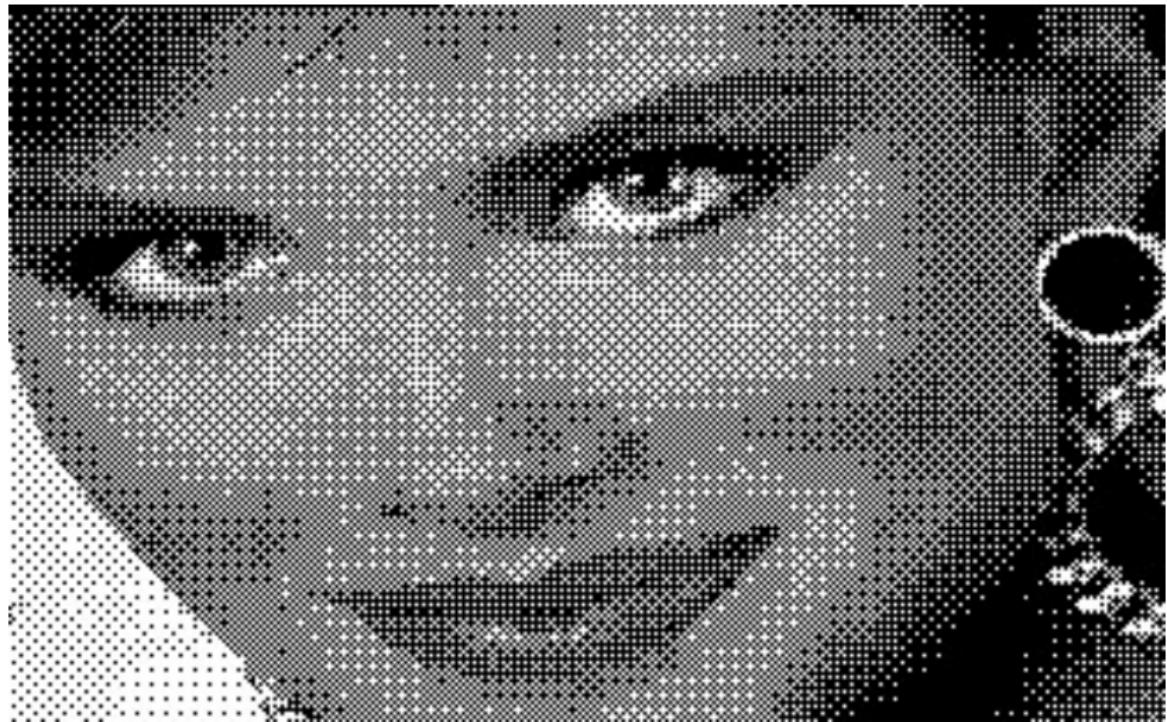
## Příklady používaných matic

$$M_d = \begin{pmatrix} 0 & 12 & 3 & 15 \\ 4 & 8 & 11 & 7 \\ 2 & 14 & 1 & 13 \\ 10 & 6 & 9 & 5 \end{pmatrix}, \quad M_p = \begin{pmatrix} 1 & 5 & 9 & 2 \\ 8 & 12 & 13 & 6 \\ 4 & 15 & 14 & 10 \\ 0 & 11 & 7 & 3 \end{pmatrix}$$

## Pozn.

- Matice vyšších řádů lze algoritmicky vytvářet z menších.
- Pozor na permutace uvnitř a na okrajích matice → artefakty (pruhы, vzory) v obrazu.

# Dithering pomocí matice $M_d$ .



# Halftoning pomocí matice $M_p$ - upravený rozměr.



# Barevná paleta 332

- Redukce RGB obrazu (16 mil. barev) na 256 barevný obraz.
- R,G kanály - 3 bity.
- B kanál - 2 bity.

## Algoritmus

Pro každý pixel obrazu:

- Nalezni nejbližší barvu z palety 332 → index  $i$ .
- Nastav hodnotu pixelu  $G(x,y) = i$ .
- Případně určení chyby hodnot R, G, B a distribuce chyby.

# Příklad redukce barevného prostoru RGB zmenšením počtu barev paletou 332.



+/-

- Nevyužití celého rozsahu palety → další degradace obrazu.

# Generování barevné palety 332

## Výpočet barev

$$R = ((i >> 5) * 255) / 7$$

$$G = (((i >> 2) \& 7) * 255) / 7$$

$$B = ((i \& 3) * 255) / 3$$

## Index barvy v paletě

$$I = (R_3 << 5) + (G_3 << 2) + B_2$$



# Převod šedotónového obrazu do ASCII

- Části obrazu o velikosti nahradíme jedním znakem (bloky např.  $7 \times 12$ ).
- Vhodný znak vybereme na základě průměrné intenzity v bloku.
- Černý blok ( $\sim 0$ )  $\rightarrow$  ''
- Bílý blok ( $\sim 255$ )  $\rightarrow$  '#'

## Příklady používaných znakových sad

"# "
"10"
"@%#*+=-:. "
"#WMNRXVYIti+=;:,.. "



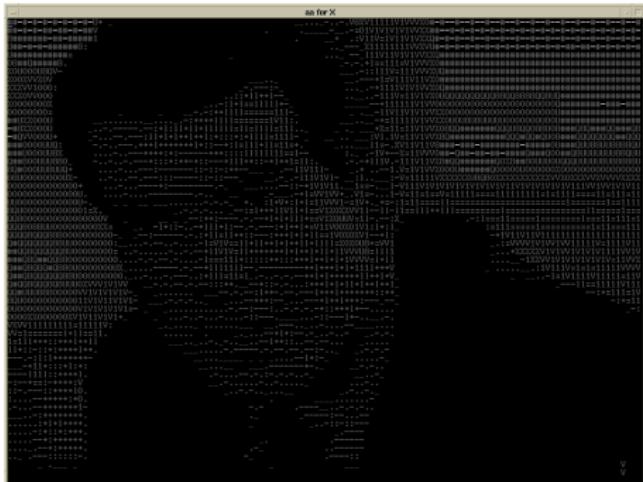
# Převod obrazu do ASCII



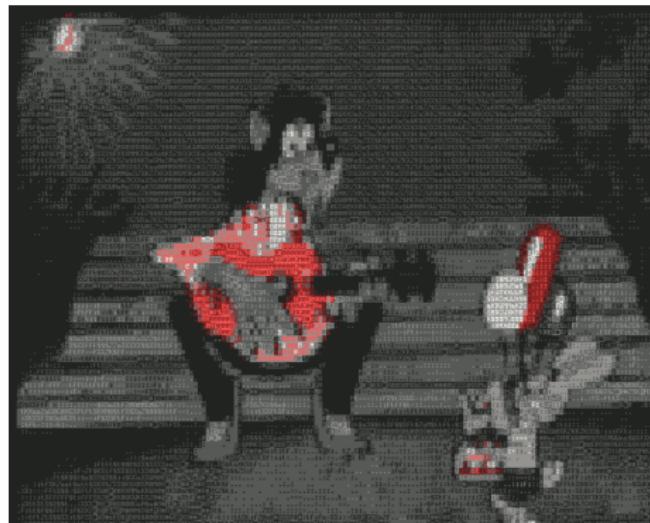


# Video v ASCII

- Vyzkoušejte MPlayer ...



<http://www.mplayerhq.hu/>



<http://www.root.cz/clanky/mplayer-a-mencoder-hrajeme/>

# Základy počítačové grafiky

## Rasterizace objektů ve 2D

Michal Španěl

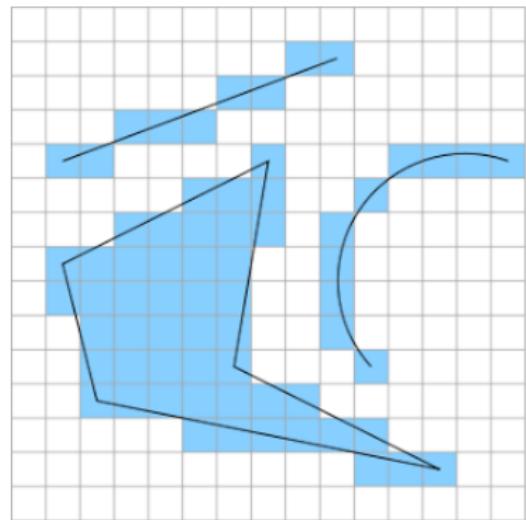
Ústav počítačové grafiky a multimédií  
Fakulta informačních technologií  
Vysoké učení technické v Brně

Brno 2016

# Cíl přednášky

Seznámit se s hlavními algoritmy pro převod základních vektorových entit na rastrové zobrazení.

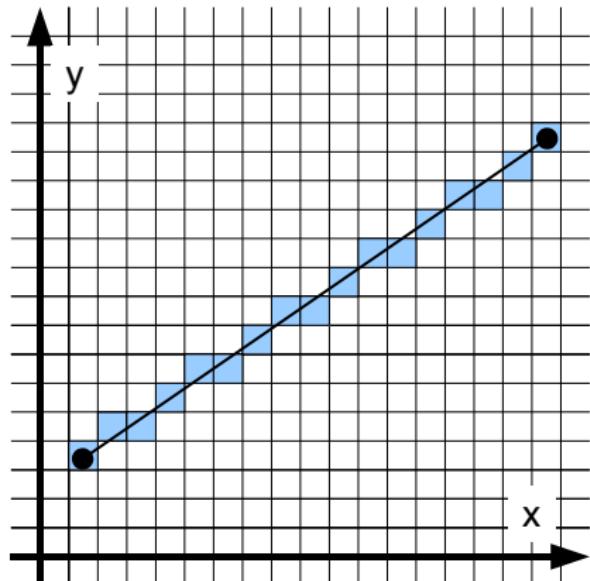
Prozatím se budeme zabývat úsečkou, kružnicí a elipsou.



# Rasterizace

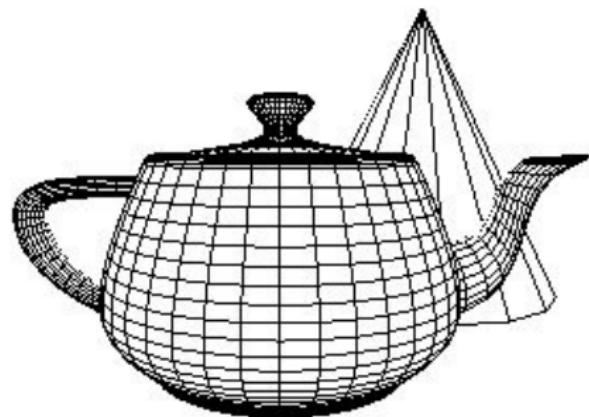
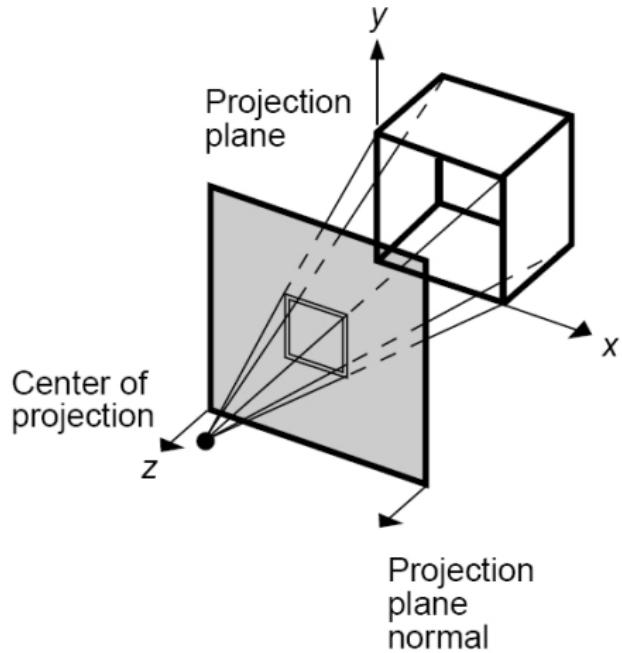
## Definice

Proces převodu vektorové reprezentace dat na jejich rastrovou formu s cílem dosáhnout maximální možnou kvalitu a zároveň rychlost výsledného zobrazení.



# Rasterizace, pokr.

- Při zobrazování je rasterizace velmi často opakovaná operace!
- Realizována v HW grafické karty.

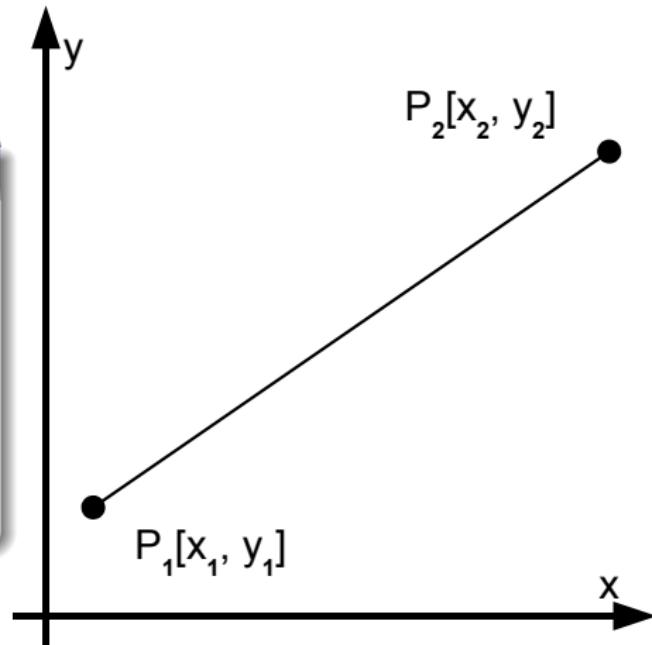


# Úsečka

## Definice

Úsečka je základní geometrická vektorová entita definovaná:

- souřadnicemi dvou koncových bodů
- rovnicí přímky popisující geometrii



# Úsečka

## Obecná rovnice úsečky

$$Ax + By + C = 0, \quad A = (y_1 - y_2), \quad B = (x_2 - x_1)$$

## Parametrické vyjádření

$$x = x_1 + t(x_2 - x_1), \quad y = y_1 + t(y_2 - y_1), \quad t \in <0, 1>$$

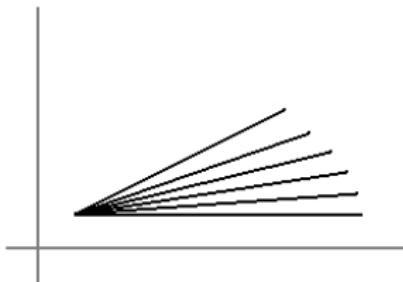
## Směrnicový tvar

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

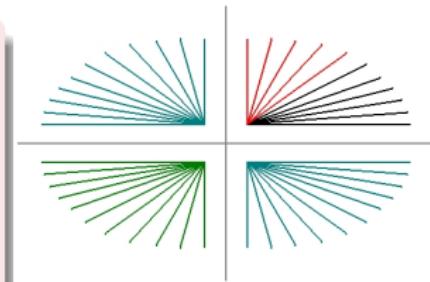
# Platnost algoritmů

Následující algoritmy pro vykreslení úsečky jsou odvozeny pro případ, kdy

- úsečka leží v prvním kvadrantu,
- je rostoucí od počátečního bodu  $P_1$  ke koncovému  $P_2$
- a nejrychleji roste ve směru osy X.



Ostatní polohy úsečky  
je potřeba převést na  
tento případ  
(prohození souřadnic,  
os, apod.)



# DDA algoritmus

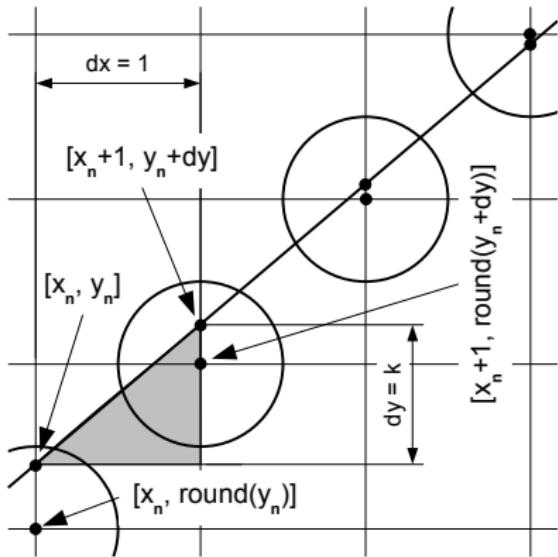
## Popis

- DDA - Digital Differential Analyser
- Jeden z prvních algoritmů rasterizace úsečky.
- Používá "floating-point aritmetiku".
- Nízká efektivita, náročná implementace do HW.

## Princip

- Vykreslujeme po pixelu od bodu  $P_1$  k bodu  $P_2$ .
- V ose X postupujeme s přírůstkem  $dx = 1$ .
- V ose Y je přírůstek dán velikostí směrnice úsečky.
- Souřadnice Y se zaokrouhuje na nejbližší celé číslo.

# DDA algoritmus



```
LineDDA(int x1, int y1, int x2, int y2)
{
    double k = (y2-y1) / (x2-x1);
    double y = y1;

    for (int x = x1; x <= x2; x++)
    {
        draw_pixel( x, round(y));
        y += k;
    }
}
```

$$x_{n+1} = x_n + dx, \quad dx = 1$$

$$y_{n+1} = y_n + dy, \quad dy = k = \frac{(y_2 - y_1)}{(x_2 - x_1)}$$

# Bresenhamův (Midpoint) algoritmus

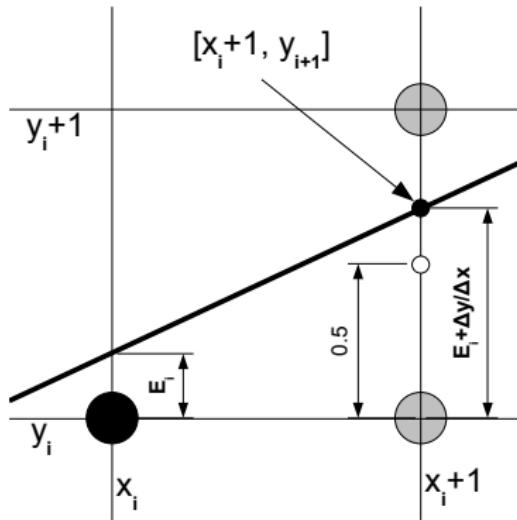
## Popis

- Nejčastěji používaný algoritmus rasterizace úsečky.
- Používá celočíselnou aritmetiku, sčítání, porovnání.
- Velmi efektivní, snadná implementace do HW.

## Princip

- Vykreslujeme po pixelu od bodu  $P_1$  k bodu  $P_2$ .
- V ose X postupujeme s přírůstkem  $dx = 1$ .
- Opusnutí v ose Y rozhodujeme podle **znaménka tzv. prediktoru**.

# Bresenhamův algoritmus



## Rozhodování a výpočet chyby vykreslování $E$

$$E_i + \frac{\Delta y}{\Delta x} \begin{cases} < 0.5 & \text{krok } (x_i + 1, y_i) \\ \geq 0.5 & \text{krok } (x_i + 1, y_i + 1) \end{cases} \quad \begin{aligned} E_{i+1} &= E_i + \frac{\Delta y}{\Delta x} & E_{i+1} &= E_i + \frac{\Delta y}{\Delta x} - 1 \end{aligned}$$

# Bresenhamův algoritmus, pokr.

## Převod porovnání s 0.5 na test znaménka

- Nerovnice násobíme  $2\Delta x$

$$2\Delta x E_i + 2\Delta y - \Delta x \begin{cases} < 0 & E_{i+1} = E_i + 2\Delta y \\ \geq 0 & E_{i+1} = E_i + 2\Delta y - 2\Delta x \end{cases}$$

- Rozhodovací člen nazveme **prediktorem**  $P_i$

$$P_i = 2\Delta x E_i + 2\Delta y - \Delta x \begin{cases} < 0 & P_{i+1} = P_i + 2\Delta y \\ \geq 0 & P_{i+1} = P_i + 2\Delta y - 2\Delta x \end{cases}$$

## Počáteční hodnota predikce ( $E_0 = 0$ )

$$P_0 = 2\Delta y - \Delta x$$

# Zjednodušená implementace

```
LineBres(int x1, int y1, int x2, int y2)
{
    int dx = x2-x1, dy = y2-y1;
    int P = 2*dy - dx;
    int P1 = 2*dy, P2 = P1 - 2*dx;
    int y = y1;

    for (int x = x1; x <= x2; x++)
    {
        draw_pixel( x, y);
        if (P >= 0)
            { P += P2; y++; }
        else
            P += P1;
    }
}
```

# DDA s fixed-point aritmetikou

## Floating-point aritmetika

- S ... znaménko
- M ... mantisa
- E ... exponent



$$X = S \cdot M \cdot 2^E$$

abcdefgijklmnopq

=0,abcdefgijklmnopq

= $a*1/2 + b*1/4 + c*1/8 + d*1/16 + \dots$

## Fixed-point aritmetika

qponmlkjihgfedcba

=qponmlkjihgfedcba,0

= $a*1 + b*2 + c*4 + d*8 + \dots$

dcbaefghijklmnopq

=dcba,efghijklmnopq

= $a*1 + b*2 + c*3 + d*4 + e*1/2 + f*1/4 + g*1/8 + h*1/16 + \dots$

# DDA s fixed-point aritmetikou

```
#define FRAC_BITS 8

LineDDAFixed(int x1, int y1, int x2, int y2)
{
    int y = y1 << FRAC_BITS;
    int k = (y2-y1) << FRAC_BITS / (x2-x1);

    for (int x = x1; x <= x2; x++)
    {
        draw_pixel( x, y >> FRAC_BITS);
        y += k;
    }
}
```

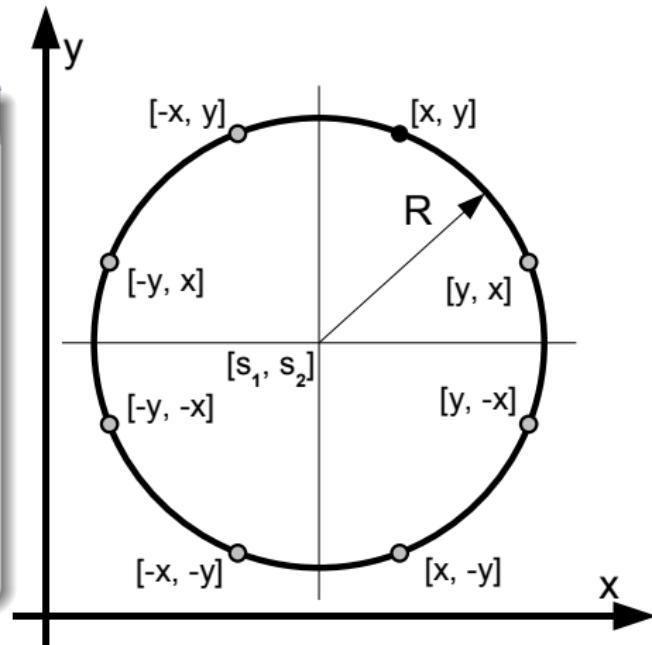
# Kružnice

## Definice

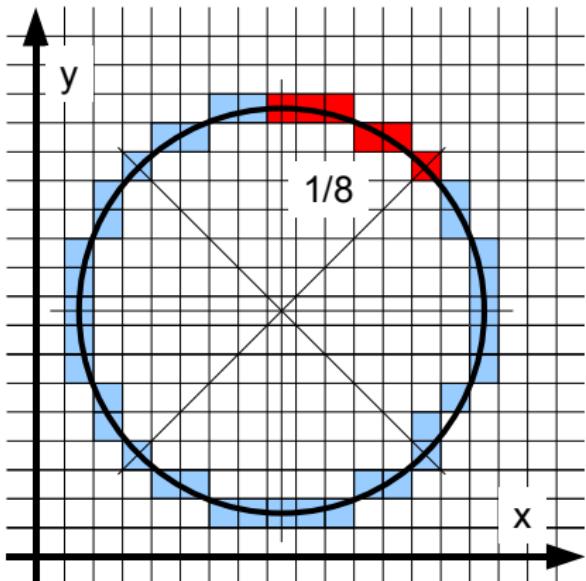
Kružnice je základní geometrická vektorová entita definovaná:

- souřadnicemi středu
- hodnotou poloměru
- rovnicí kružnice popisující geometrii

$$(x - s_1)^2 + (y - s_2)^2 - R^2 = 0$$



# Kružnice



## Vlastnosti

- Je 8x symetrická
- Provádíme výpočet pro 1/8 bodů v 1/2 prvního kvadrantu
- Zbylé body získáme záměnou souřadnic
- **Algoritmy jsou odvozeny pro kružnici se středem v počátku [0, 0]**

# Vykreslení kružnice po bodech

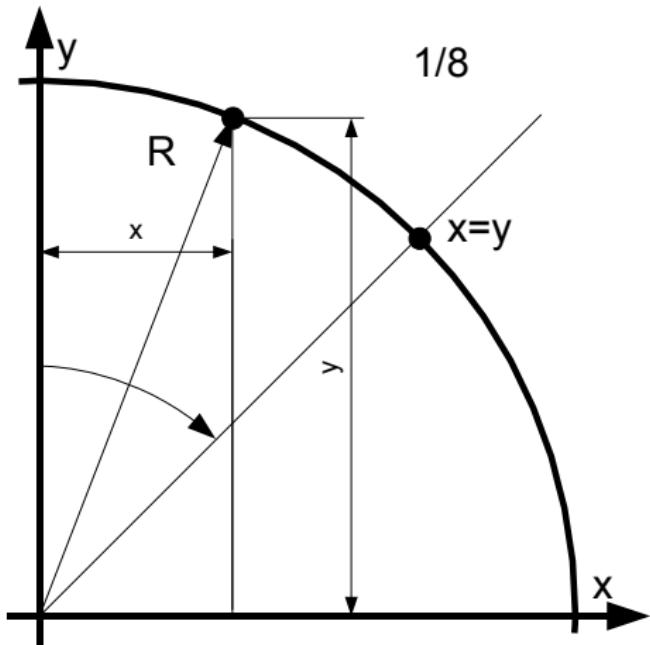
## Popis

- "Naivní" algoritmus rasterizace kružnice.
- Používá "floating-point aritmetiku".
- Nízká efektivita, náročná implementace v HW.

## Princip

- Vykreslujeme ve směru hodinových ručiček.
- jdeme po pixelu od bodu  $[0, R]$ , dokud není  $x = y$ .
- V ose X postupujeme s přírůstkem  $dx = 1$ .
- Pozici v ose Y vypočteme podle vztahu  $y = \sqrt{R^2 - x^2}$ .
- Souřadnice Y se zaokrouhuje na nejbližší celé číslo.

# Vykreslení kružnice po bodech



```
CircleByPoints(int s1, int s2, int R)
{
    int x = 0, y = R;
    while (x <= y)
    {
        draw_pixel_circle(x, y);
        x++;
        y = sqrt(R*R - x*x);
    }
}
```

# Vykreslení kružnice jako N-úhelník

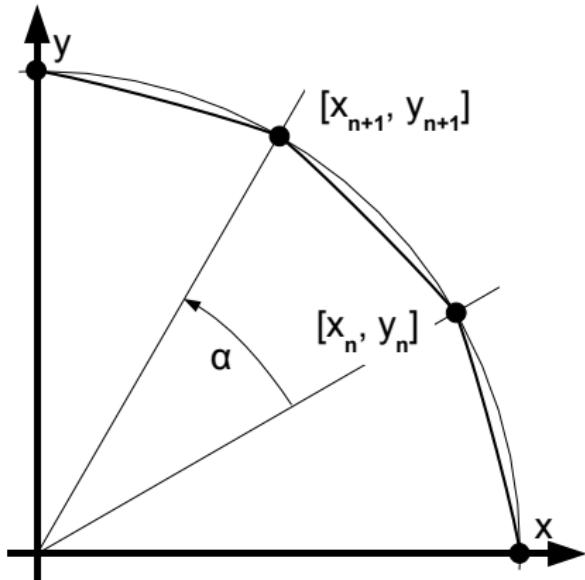
## Popis

- Varianta algoritmu DDA pro kružnici.
- Aplikace rotační transformace bodu.
- Používá "floating-point aritmetiku".
- Nízká efektivita, náročná implementace do HW.

## Princip

- Rekurentně se posouváme o konstantní přírůstek úhlu.
- Funkce  $\sin$  a  $\cos$  jsou vypočítány pouze jednou!
- Souřadnice X a Y se zaokrouhlují na nejbližší celé číslo.
- Vypočtené souřadnice spojujeme úsečkami.

# Vykreslení kružnice jako N-úhelník



```

CircleDDA(int R, int N)
{
    double cosa = cos(2*PI/N);
    double sina = sin(2*PI/N);
    int x1 = R, y1 = 0, x2, y2;

    for (int i = 0; i < N; i++)
    {
        x2 = x1*cosa - y1*sina;
        y2 = x1*sina + y1*cosa;
        draw_line(x1, y1, x2, y2);
        x1 = x2;
        y1 = y2;
    }
}

```

$$x_{n+1} = x_n \cos \alpha - y_n \sin \alpha$$

$$y_{n+1} = x_n \sin \alpha + y_n \cos \alpha$$

# Midpoint algoritmus pro kružnici

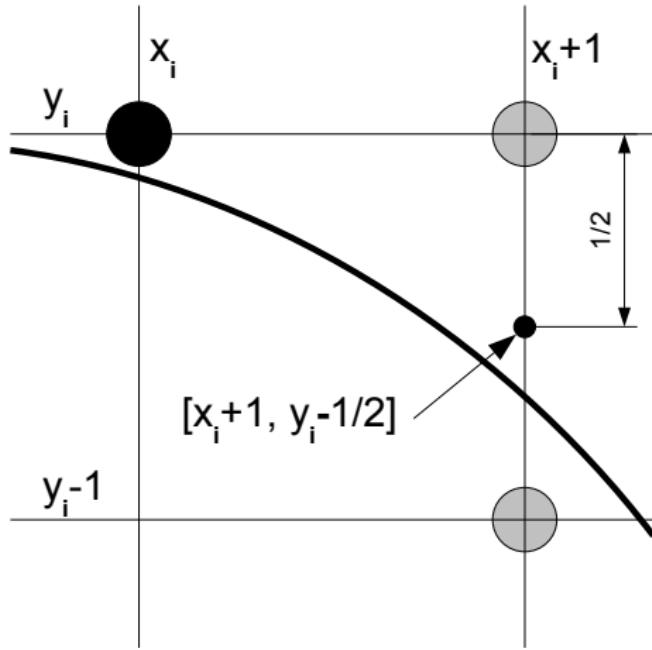
## Popis

- Variace na Bresenhamův algoritmus, stejný přístup.
- Určování polohy "Midpointu" vůči kružnici (in, out).
- Používá celočíselnou aritmetiku, sčítání, porovnání.
- Velmi efektivní, snadná implementace v HW.

## Princip

- Vykreslujeme po pixelu od bodu  $[0, R]$ , dokud není  $x = y$ .
- V ose X postupujeme s přírůstkem  $dx = 1$ .
- Opusnutu v ose Y rozhodujeme podle **znaménka prediktoru**.

# Midpoint algoritmus pro kružnici



```

CircleMid(int s1, int s2, int R)
{
    int x = 0, y = R;
    int P = 1-R, X2 = 3, Y2 = 2*R-2;

    while (x < y)
    {
        draw_pixel_circle(x, y);

        if (P >= 0)
            { P += -Y2; Y2 -= 2; y--; }

        P += X2;
        X2 += 2;
        x++;
    }
}

```

# Midpoint algoritmus pro kružnici

## Odvození prediktoru

$$F(x, y) = x^2 + y^2 - R^2 = 0$$

$$p_i = F(x_i + 1, y_i - \frac{1}{2})$$

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

$$p_i < 0 \implies y_{i+1} = y_i$$

$$p_i \geq 0 \implies y_{i+1} = y_i - 1$$

# Midpoint algoritmus pro kružnici

## Odvození rekurentního prediktoru

$$p_{i+1} = (x_{i+1} + 1)^2 + \left(y_{i+1} - \frac{1}{2}\right)^2 - R^2$$

$$p_{i+1} = p_i + 2x_i + 3 \iff p_i < 0$$

$$p_{i+1} = p_i + 2x_i - 2y_i + 5 \iff p_i \geq 0$$

Startovací hodnota prediktoru je  $p_i = 1 - R$  ???

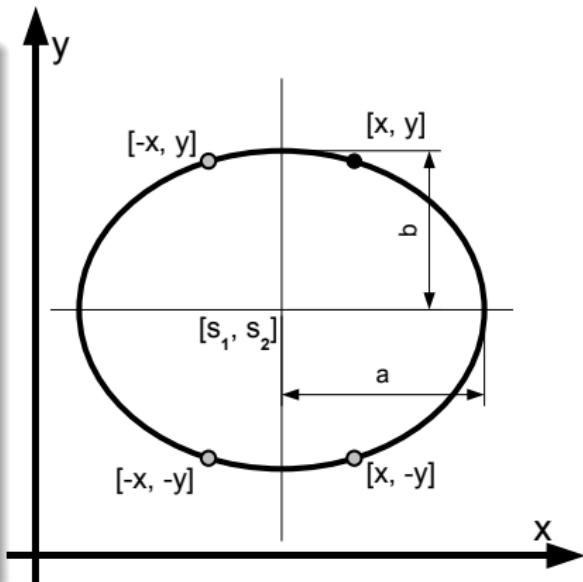
# Elipsa

## Definice

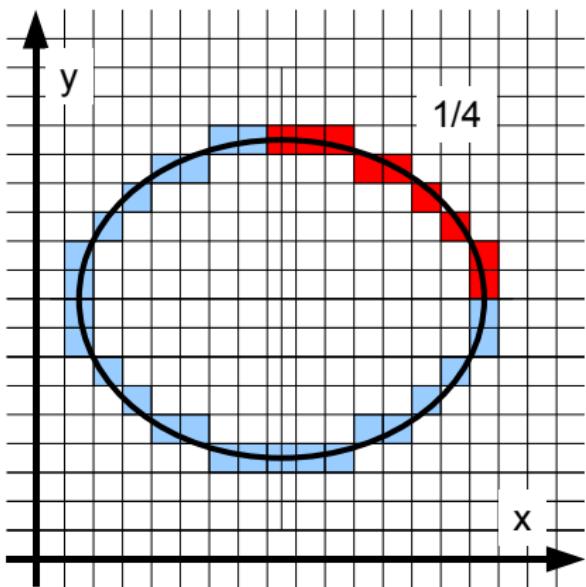
Elipsa je základní geometrická vektorová entita definovaná:

- souřadnicemi středu
- hodnotami hlavní a vedlejší poloosy
- úhlem natočení hlavní poloosy
- rovnicí elipsy popisující geometrii

$$F(x, y) : b^2x^2 + a^2y^2 - a^2b^2 = 0$$



# Elipsa



## Vlastnosti

- Je 4x symetrická
- Provádíme výpočet pro 1/4 bodů
- Zbylé body získáme záměnou souřadnic
- Dvě oblasti výpočtů - jak je poznáme?
- Algoritmy jsou odvozeny pro elipsu se středem v  $[0, 0]$  a nulovým otočením

# Midpoint algoritmus pro elipsu

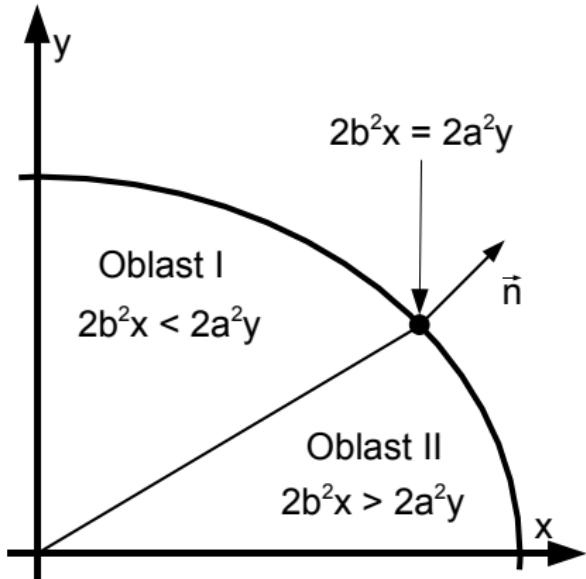
## Popis

- Ekvivalent Midpoint algoritmu pro kružnici.
- Určování polohy "Midpointu" vůči elipse (in, out).
- Používá celočíselnou aritmetiku, sčítání, porovnání.
- Velmi efektivní, snadná implementace v HW.

## Princip

- Pro oblast I jdeme po pixelu od bodu  $[0, b]$ , dokud nejsou parciální derivace podle  $x$  a  $y$  rovny  $(2b^2x = 2a^2y)$ .
- Pak pro oblast II až do bodu  $[a, 0]$ .
- V ose X/Y postupujeme s přírůstkem  $dx/dy = 1$  (oblast I/II)
- Pusun v ose Y/X určuje **znaménko prediktoru**

# Midpoint algoritmus pro elipsu



```

ElipseMid(int A, int B)
{ int x = 0, y = B, AA = A*A, BB = B*B;
  int P = BB - AA*B + AA/4;

  while (AA*y > BB*x)
  { draw_pixel_elipse(x, y);
    if (P < 0)
      { P += BB*(2*x+3); x++; }
    else
      { P += BB*(2*x+3) + AA*(2-2*y); x++; y--; }

    P = BB*(x+0,5)*(x+0,5)+AA*(y-1)*(y-1)-AA*BB;
    while (y >= 0)
    { draw_pixel_elipse(x, y);
      if (P < 0)
        { P += BB*(2*x+2) + AA*(3-2*y); x++; y--; }
      else
        { P += AA*(3-2*y); y--; }
    }
  }
}

```

$$\frac{\partial F(x, y)}{\partial x} = 2b^2x \quad , \quad \frac{\partial F(x, y)}{\partial y} = 2a^2y$$

# Midpoint algoritmus pro elipsu

## Odvození prediktoru, pro oblast I

$$F(x, y) = b^2x^2 + a^2y^2 - a^2b^2 = 0$$

$$p_i = F(x_i + 1, y_i - \frac{1}{2})$$

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

$$p_i < 0 \implies y_{i+1} = y_i$$

$$p_i \geq 0 \implies y_{i+1} = y_i - 1$$

# Midpoint algoritmus pro elipsu

## Odvození rekurentního prediktoru, pro oblast I

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

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

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

Startovací hodnota prediktoru je  $p_i = b^2 - a^2b + \frac{1}{4}a^2$

# Midpoint algoritmus pro elipsu

## Odvození prediktoru, pro oblast II

$$F(x, y) = b^2x^2 + a^2y^2 - a^2b^2 = 0$$

$$p_i = F\left(x_i + \frac{1}{2}, y_i - 1\right)$$

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

$$p_i \geq 0 \implies x_{i+1} = x_i$$

$$p_i < 0 \implies x_{i+1} = x_i + 1$$

# Midpoint algoritmus pro elipsu

## Odvození rekurentního prediktoru, pro oblast II

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

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

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

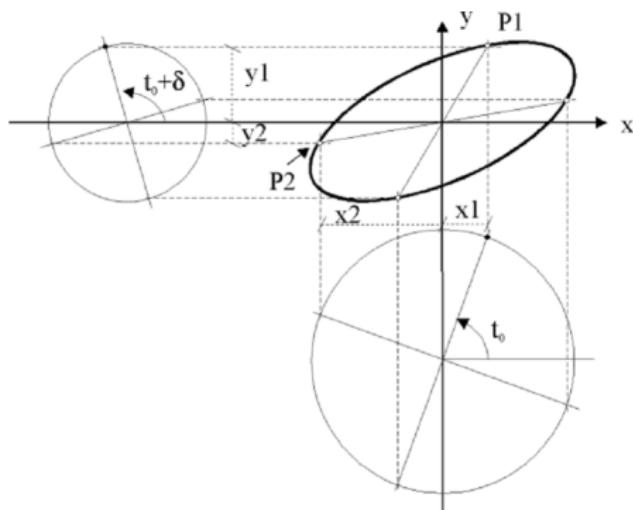
REstartovací hodnota prediktoru je

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

# Jak vykreslit elipsu v obecné poloze?

## Elipsa pomocí dvou kružnic

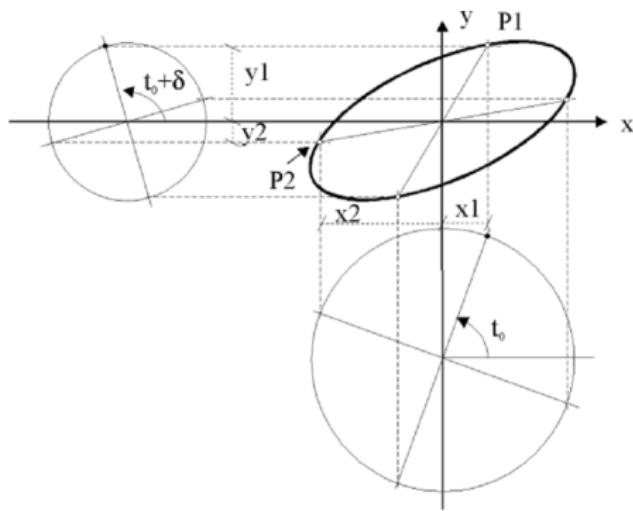
- Složení dvou rotačních pohybů se stejnou úhlovou rychlostí
- Dvě kružnice s různým poloměrem
- $x \dots$  dána polohou bodu na kružnici s poloměrem A
- $y \dots$  dána polohou bodu na kružnici s poloměrem B



# Elipsa pomocí dvou kružnic

Parametrické vyjádření elipsy se středem v počátku

- $f(t) = f(x(t), y(t))$
- $f(t) = (A \cdot \sin(t), B \cdot \sin(t + \delta))$



# Základy počítačové grafiky

## Antialiasing

Michal Španěl

Ústav počítačové grafiky a multimédií  
Fakulta informačních technologií  
Vysoké učení technické v Brně

Brno 2016

# Cíl přednášky

Cílem této přednášky je seznámit se s příčinami vzniku artefaktů v rastrovém obrazu a metodami jeho odstraňování.



# Co je to Aliasing?

Nechtěný (rušivý) artefakt při snímání (generování) signálu či obrazu

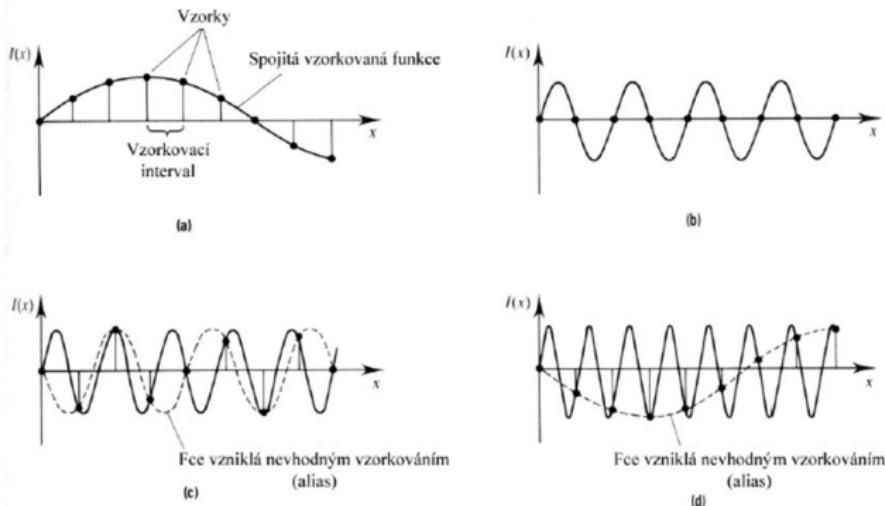
- Vzniká při syntéze (rasterizaci, vzorkování) spojitého signálu, obrazu
- Nejčastější projevy: "zubaté" hrany, poruchy textur, atd.



# Aliasing signálu

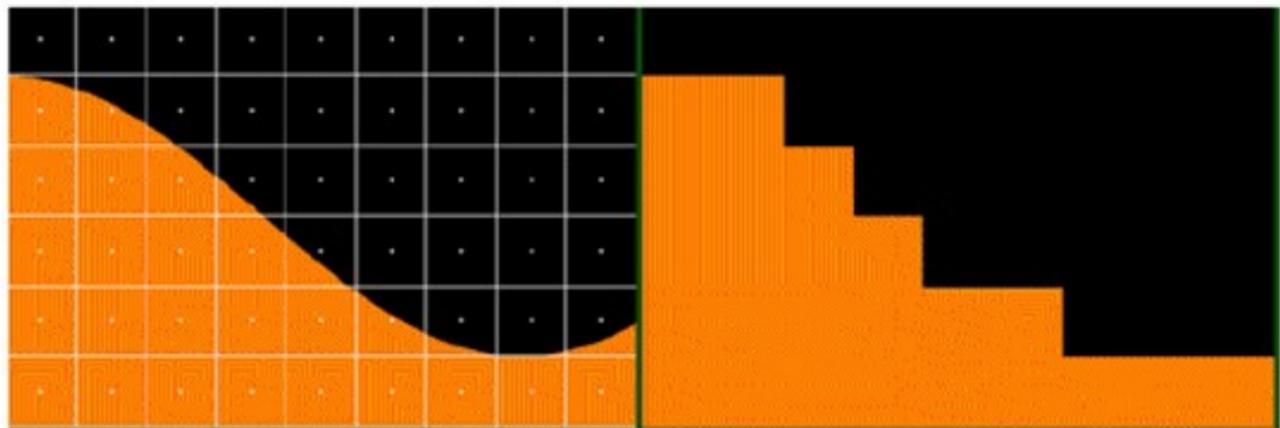
## Definice

Alias je jev, který vzniká nízkofrekvenčním vzorkováním signálu, který má relativně vysokou frekvenci.



# Aliasing spojité oblasti – zubaté hrany

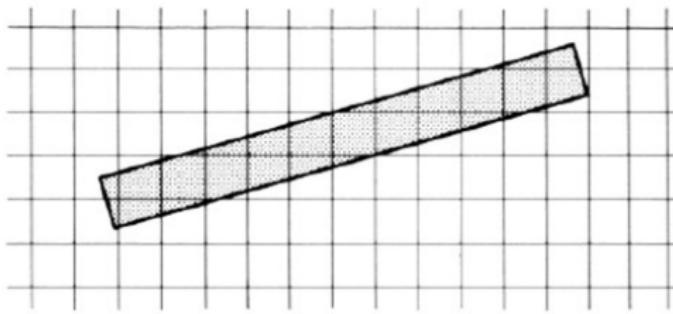
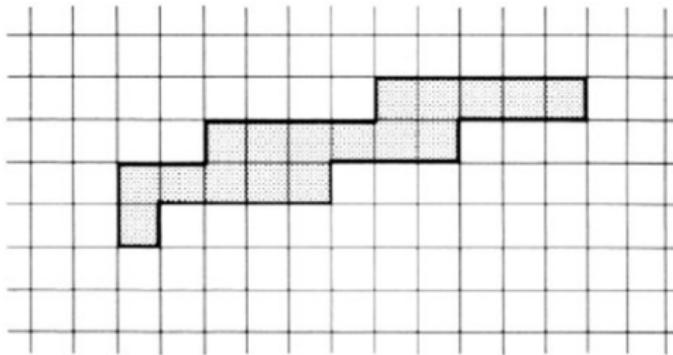
Problém příliš přesného a pravidelného vzorkování!



# Aliasing v herních enginech



# Aliasing vektorové entity – zubaté hrany

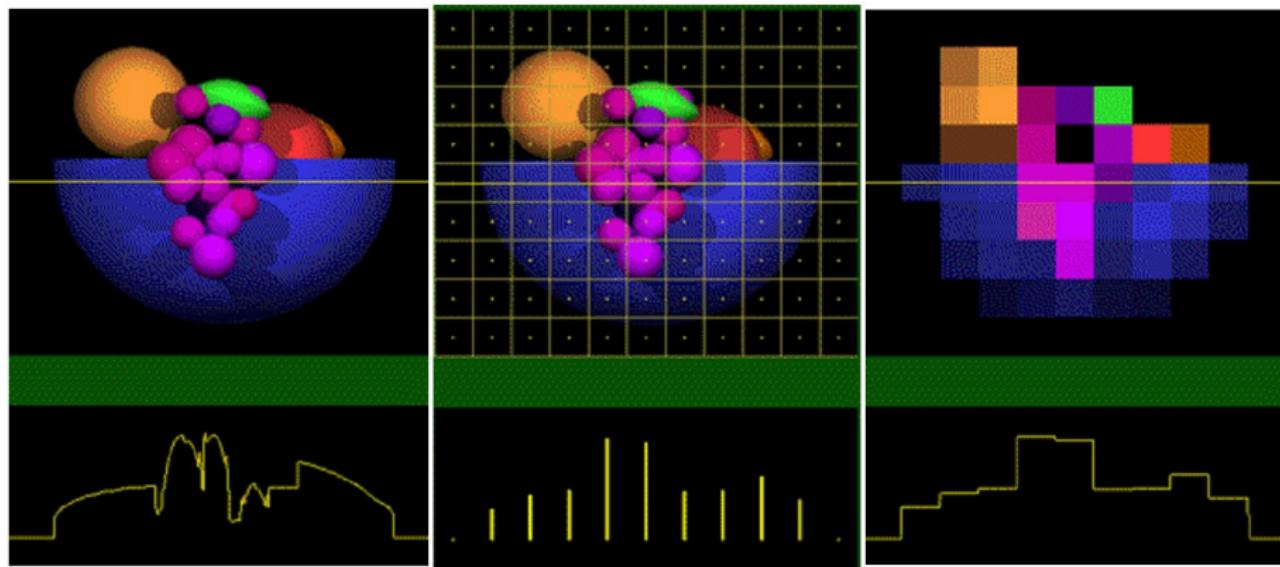


# Aliasing textury

Vzorkujeme rastrový (pravidelný) vzor (rastrovou texturu)!

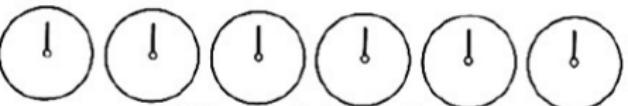
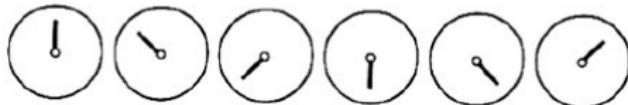
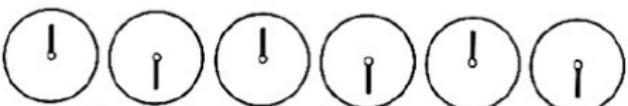
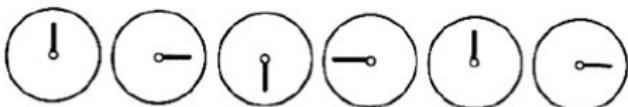
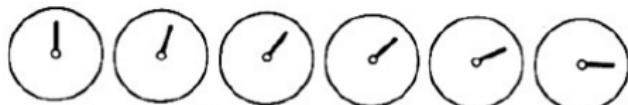


# Aliasing 3D scény – degradace obrazu



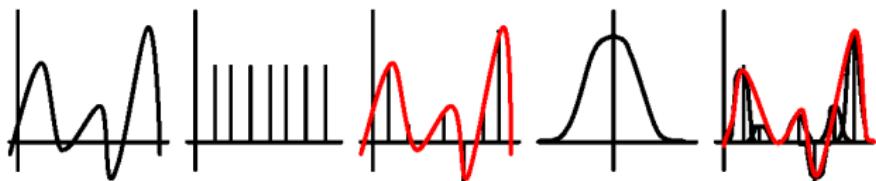
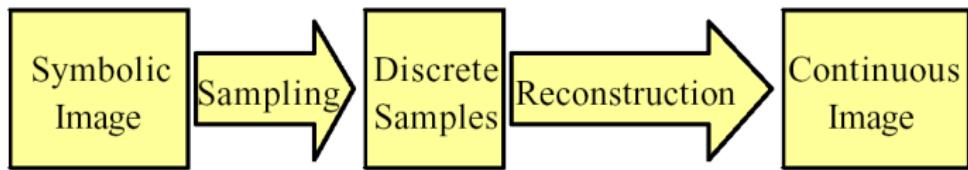
# Aliasing času

- Vzorkujeme periodický děj (stroboskopie)!
- Tzv. wagon-wheel effect (<https://youtu.be/VNftf5qLpiA>)



# Podstata problému aliasingu

## Proč vzniká aliasing?



# Podstata problému aliasingu

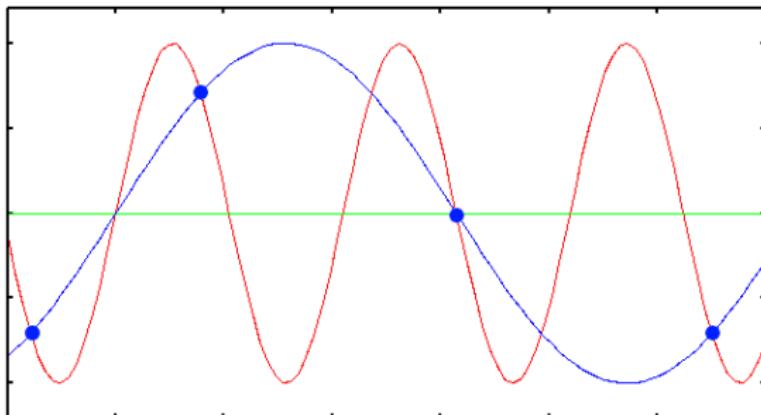
## Proč vzniká aliasing?

- Vznik aliasingu souvisí se způsobem vzorkování vstupní informace (model) pro získání výstupních dat (obraz)
- Vzorkujeme s příliš malou vzorkovací frekvencí
- Vzorkujeme příliš pravidelně
- Vzorkujeme příliš přesně (ostře, deterministicky)

# Shannonův vzorkovací teorém

## Definice

Přesná rekonstrukce spojitého, frekvenčně omezeného, signálu z jeho vzorků je možná tehdy, pokud byl vzorkován frekvencí alespoň dvakrát vyšší než je maximální frekvence rekonstruovaného signálu.



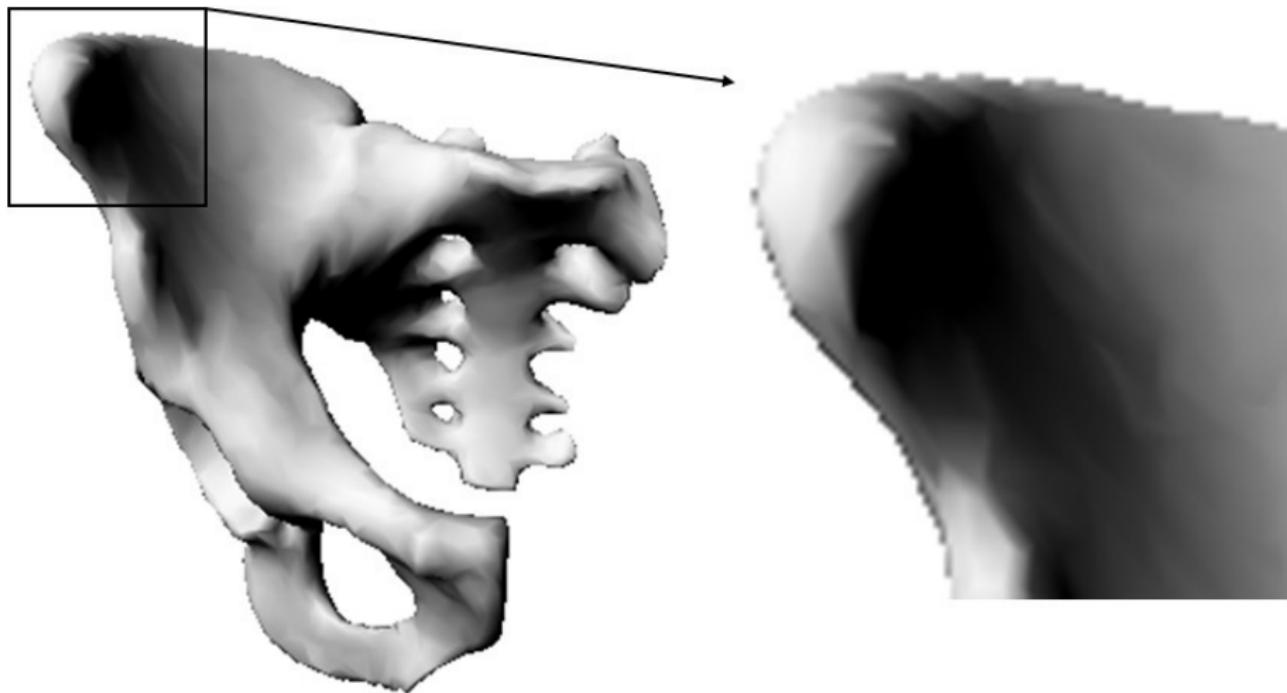
# Možnosti řešení problému aliasingu

## Zvýšení vzorkovací frekvence

- Zvětšení rozlišení výstupních dat – obrazu
- Přímo to odpovídá závěrům Shannonova vzorkovacího teorému
- Dostáváme velmi dobré výsledky
- Cenou je příliš velká datová náročnost výstupních dat – obrazu

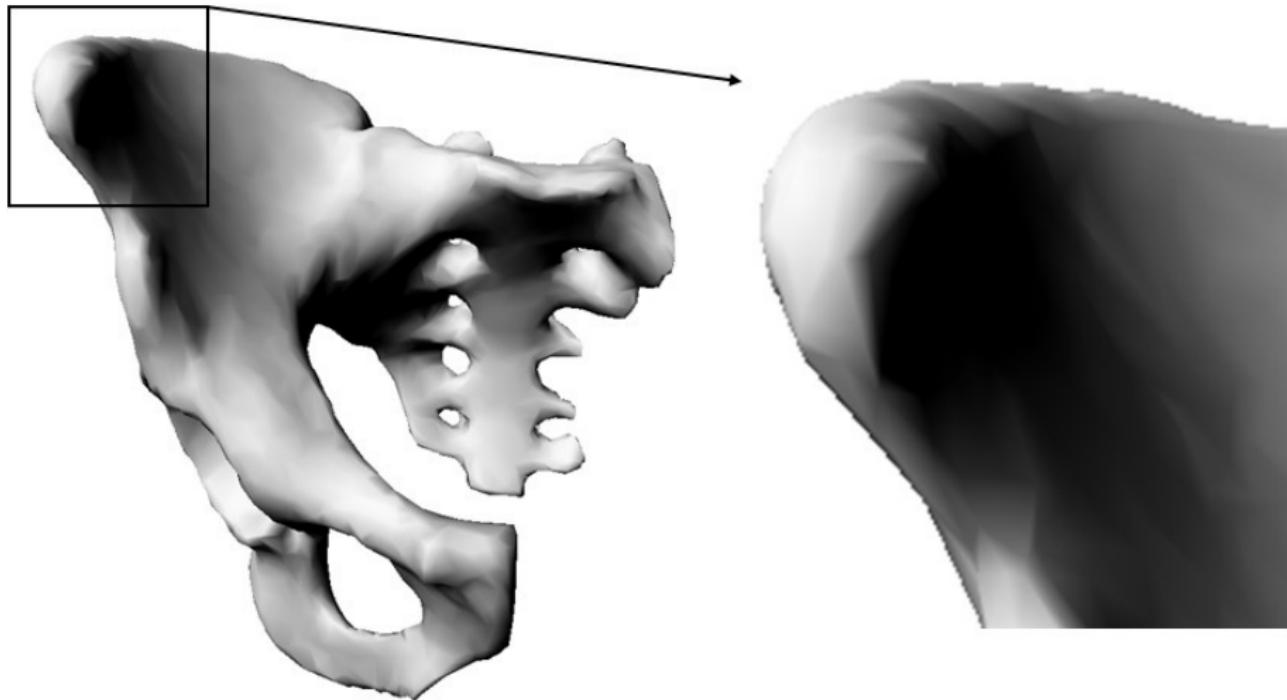
# Zvýšení vzorkovací frekvence

3D model, rozlišení 280 x 290



# Zvýšení vzorkovací frekvence

3D model, rozlišení 650 x 665



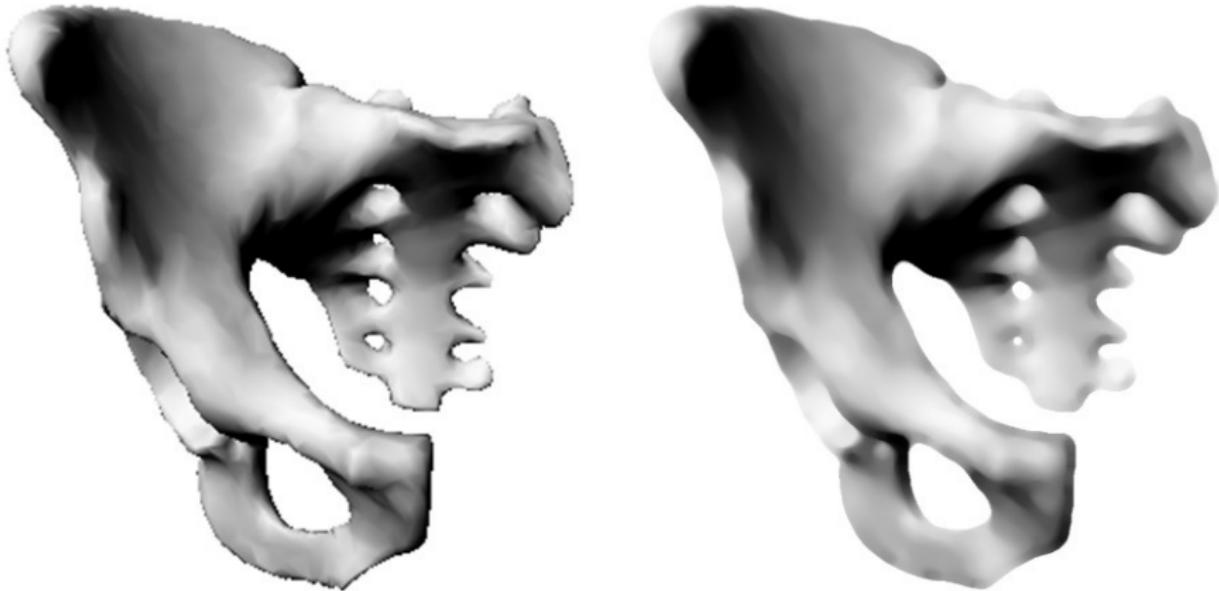
# Možnosti řešení problému aliasingu

## Přefiltrování výstupních dat

- Vyhlazení, rozmazání výstupních dat – obrazu
- Provádí se až na konec celého procesu – postprocessing
- Cenou je ztráta detailů výstupních dat – obrazu

# Přefiltrování výstupních dat

3D model, rozlišení 280 x 290 + vyhlazený výstupní obraz

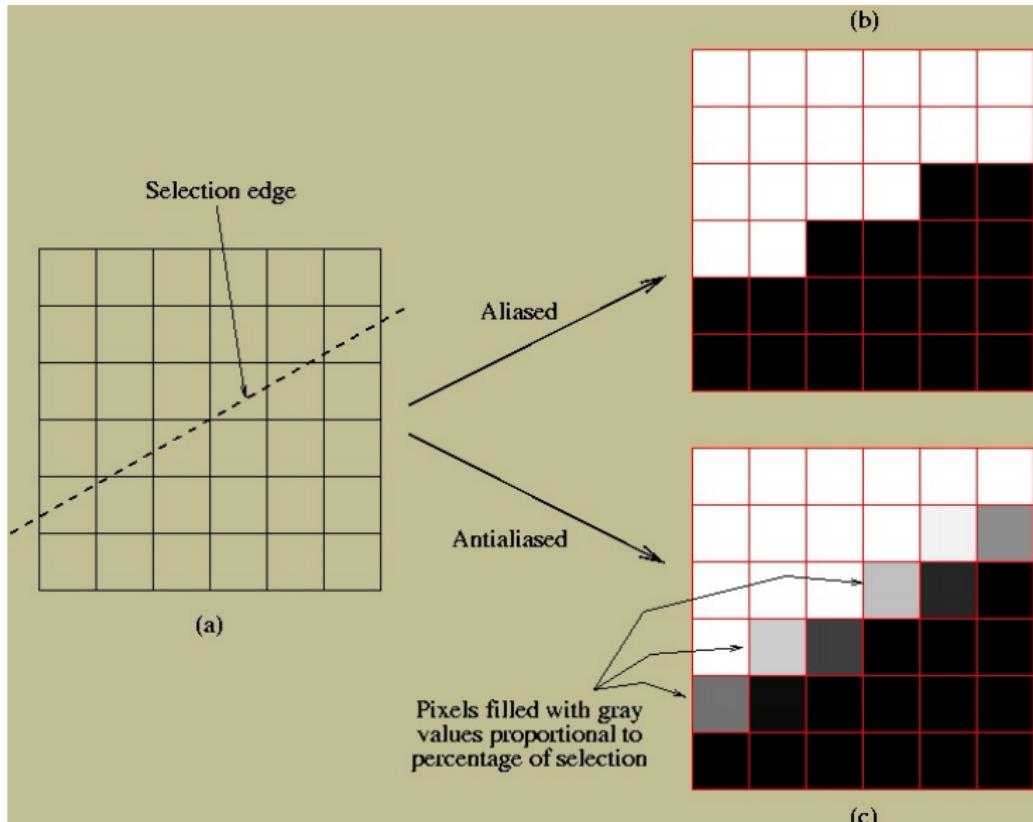


# Možnosti řešení problému aliasingu

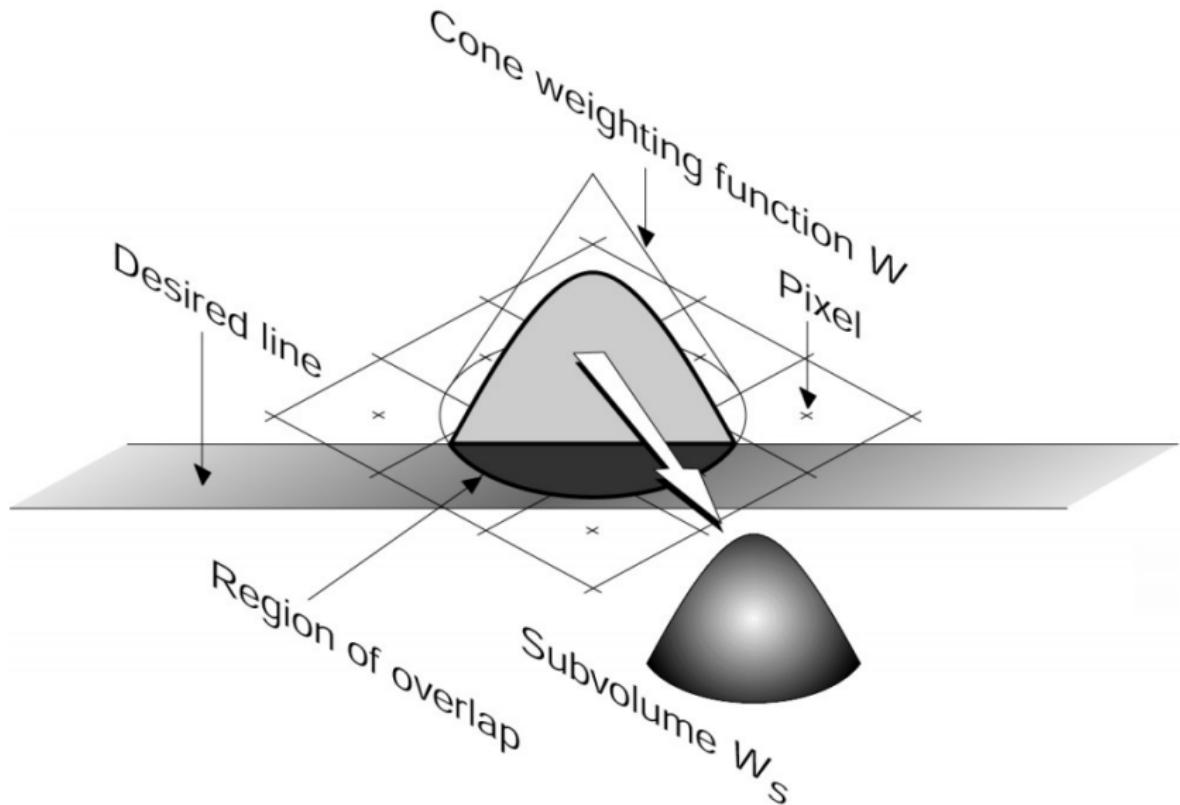
## Před-filtrování (vyhlazení) vstupních dat

- Zachovává rozlišení výstupních dat – obrazu
- Zvyšujeme však vzorkovací frekvenci – více vstupních vzorků na jeden výstupní
- Probíhá během vlastního zpracování vstupních dat
- Cenou je ztráta výkonu (rychlosti) zpracování dat

# Výhlazení vstupních dat



# Výhlazení vstupních dat

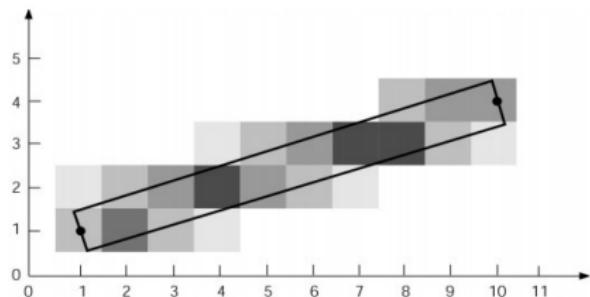
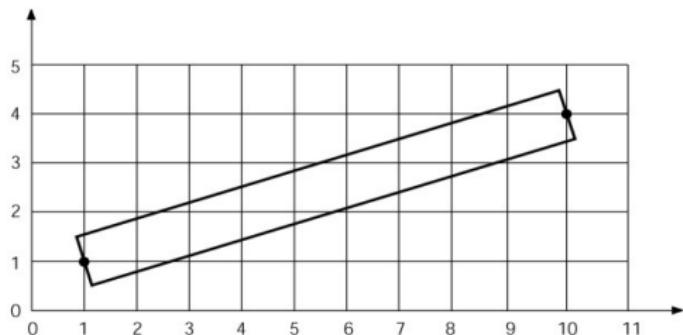


# Výhlazení vstupních dat

$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{16}$
$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{8}$
$\frac{1}{16}$	$\frac{1}{8}$	$\frac{1}{16}$

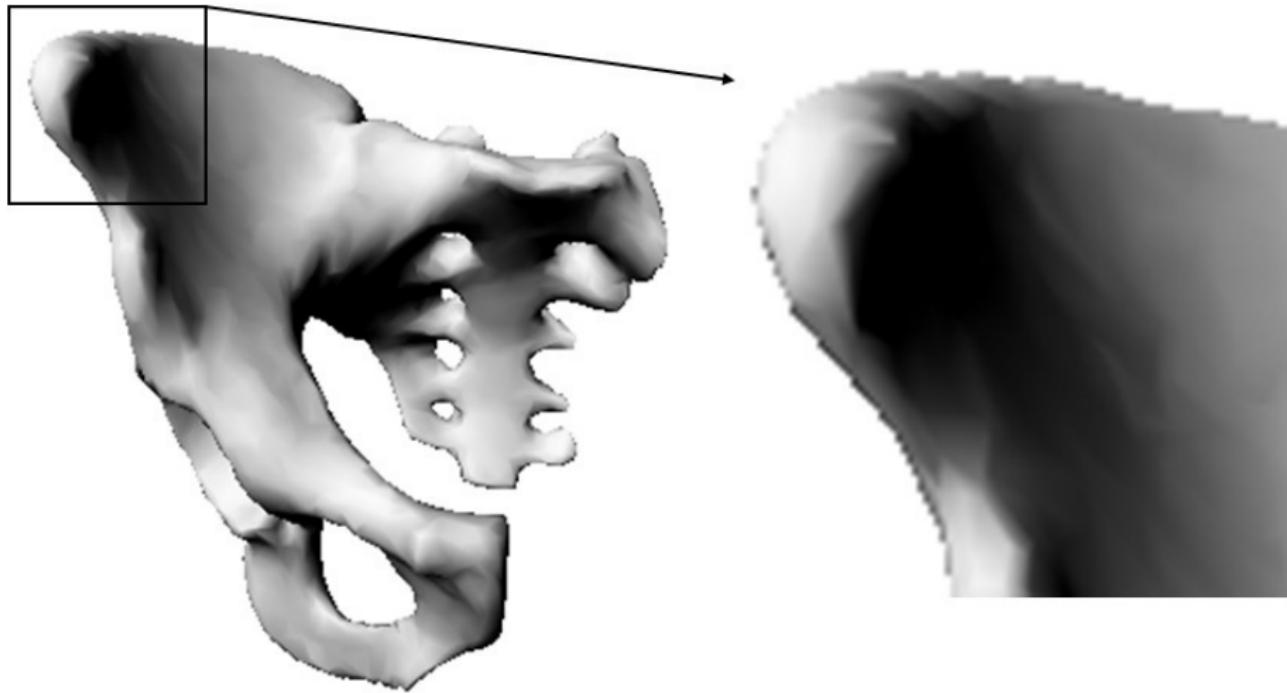


# Výhlazení vstupních dat



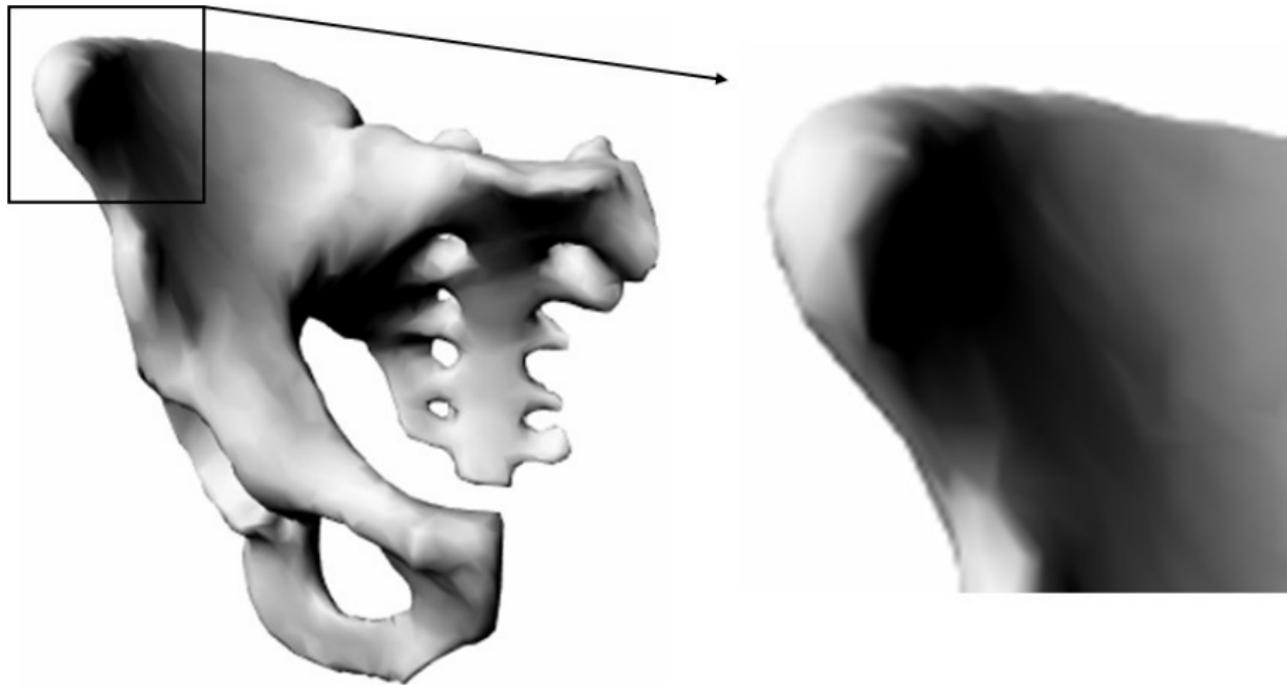
# Výhlazení vstupních dat

3D model, rozlišení 280 x 290



# Výhlazení vstupních dat

3D model, rozlišení 280 x 290 + antialiasing



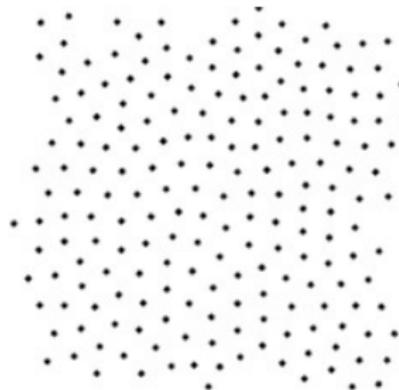
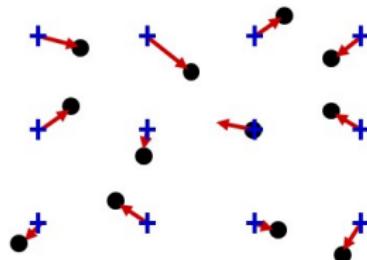
# Možnosti řešení problému aliasingu

## Nepravidelné vzorkování vstupních dat

- Zachovává rozlišení výstupních dat – obrazu
- Zachováváme vzorkovací frekvenci – jeden vstupní vzorek na jeden výstupní
- Náhodně drobně posouváme pozice vstupních vzorků (na úrovni rozlišení vstupních dat)
- Probíhá během vlastního zpracování vstupních dat
- Používá se v kombinaci s ostatními metodami
- Jittering, Poisson disk

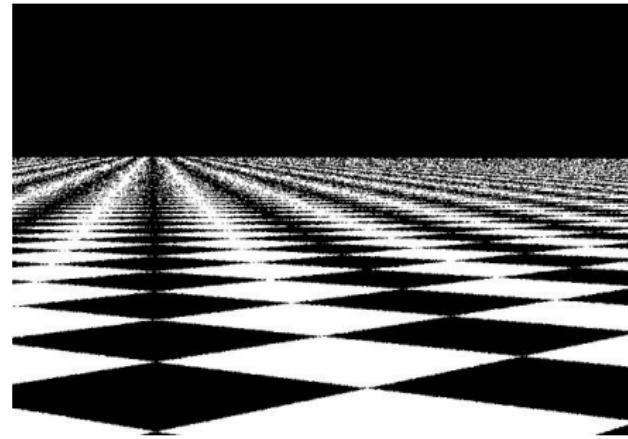
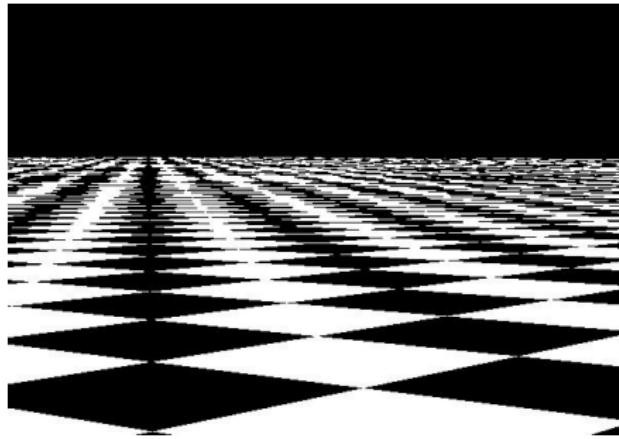
# Nepravidelné vzorkování vstupních dat

Jittering (vlevo) a rozmístění vzorků na obrazu (vpravo)



# Nepravidelné vzorkování vstupních dat

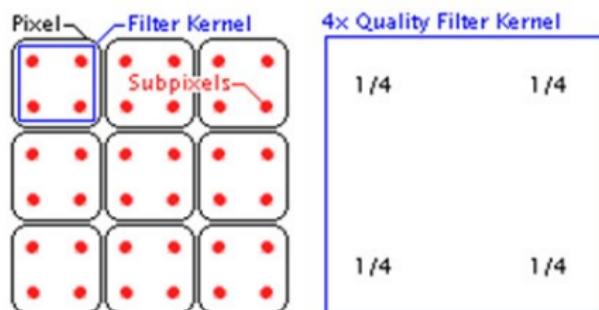
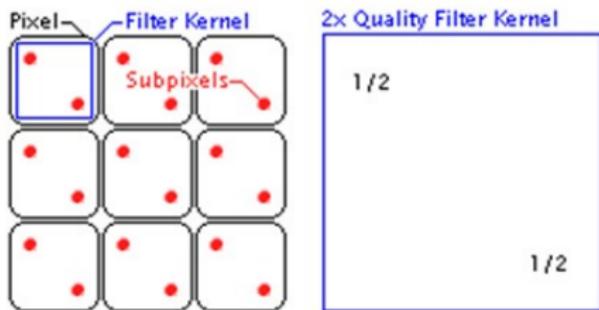
Uniformní vzorkování (vlevo) a jittering (vpravo)



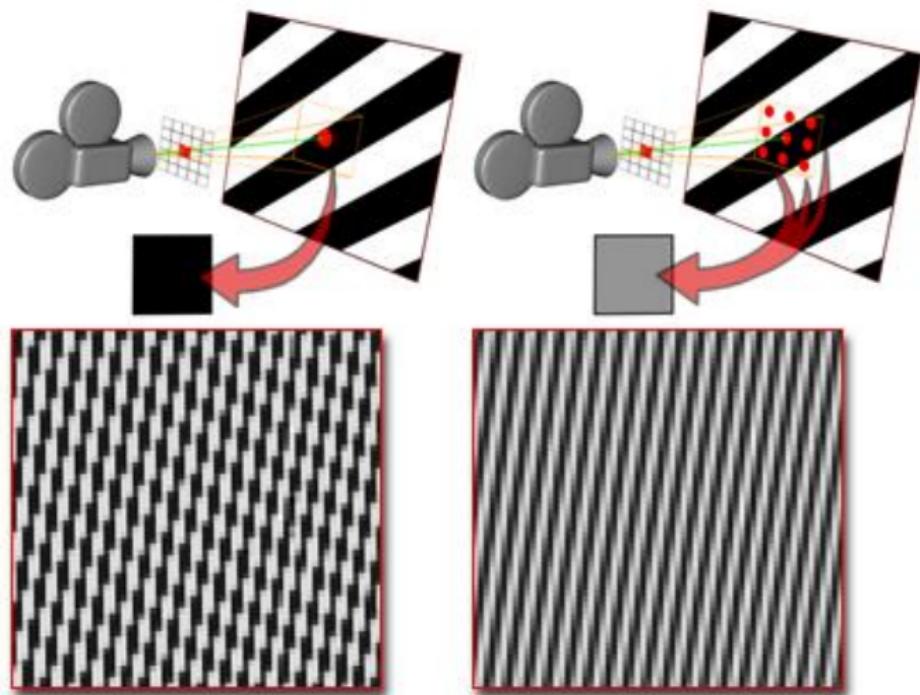
# Supersampling

## Charakteristika

- Odpovídá metodě řešení vyhlazením vstupních dat
- Každý pixel je rozdělen na několik vzorků
- Výsledná hodnota pixelů je složena z hodnot vzorků
- Složení vzorků zajistí konvoluční filtr
- Vyhlažuje celý výsledný obraz, hrany i textury
- Výrazný pokles výkonu zobrazení



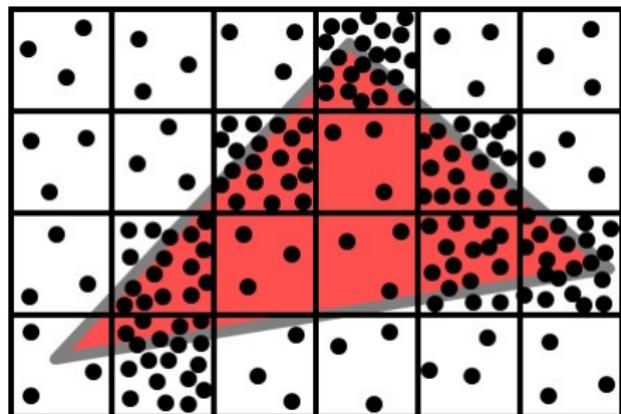
# Supersampling



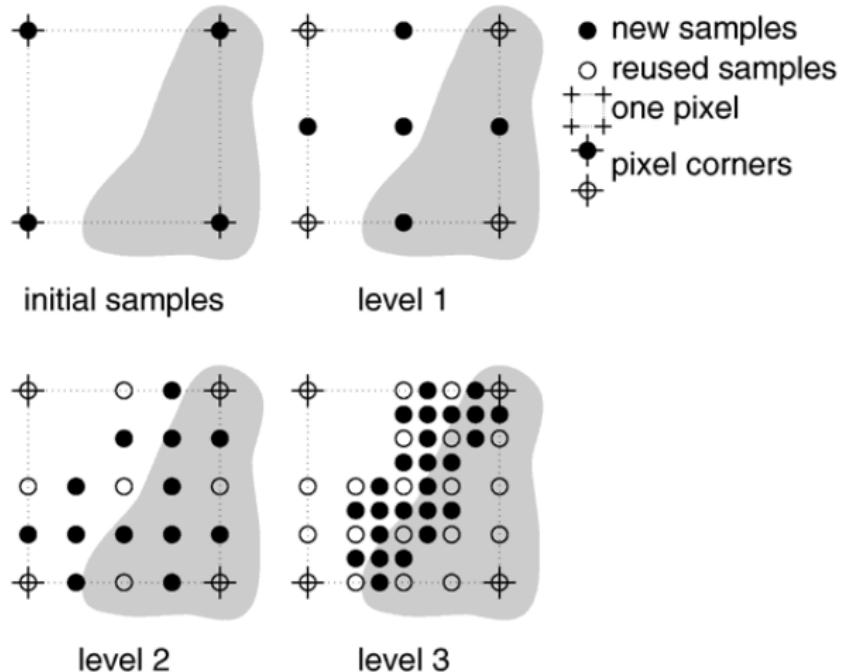
# Multisampling

## Charakteristika

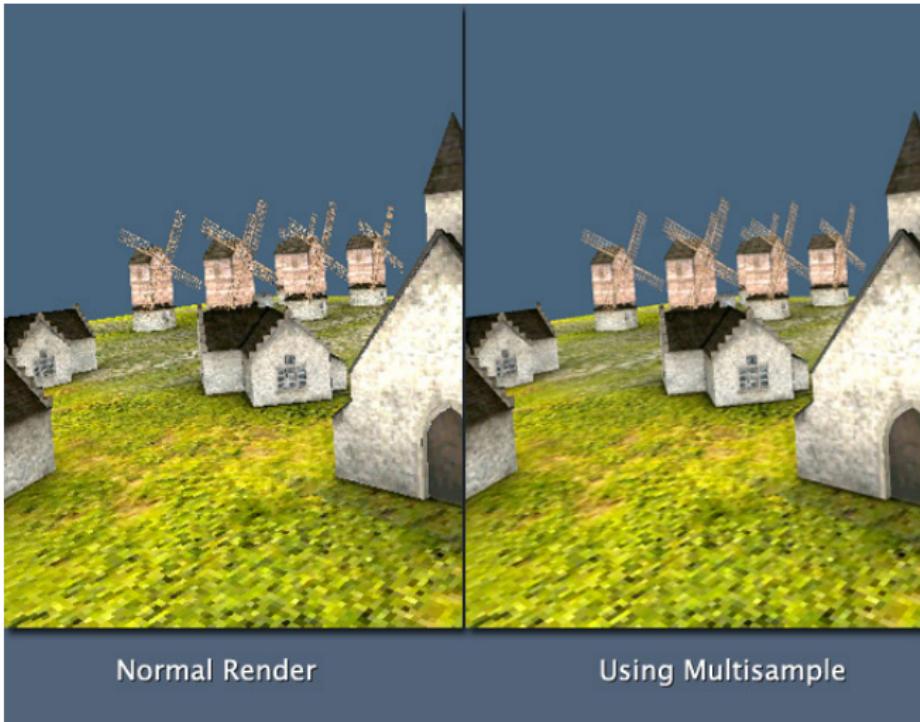
- Adaptivní Supersampling
- Hustější vzorkování pouze v oblasti hran objektů
- Uvnitř a mimo objekty řidší vzorkování
- Při gradientech vzorků se zahustí vzorkování
- Zpracování dat jako u Supersamplingu
- Výrazně vyhlazuje hrany objektů, méně textury
- Menší pokles výkonu



# Multisampling



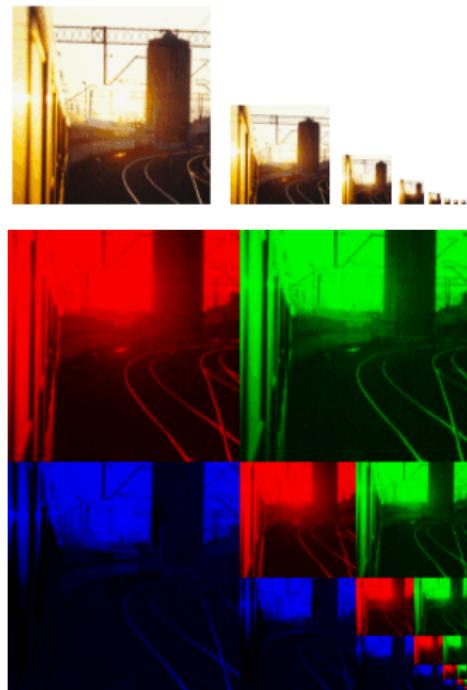
# Multisampling



# MIP ("multum in parvo") Mapping textur

Uložení hierarchie rozlišení textury do jedné matice

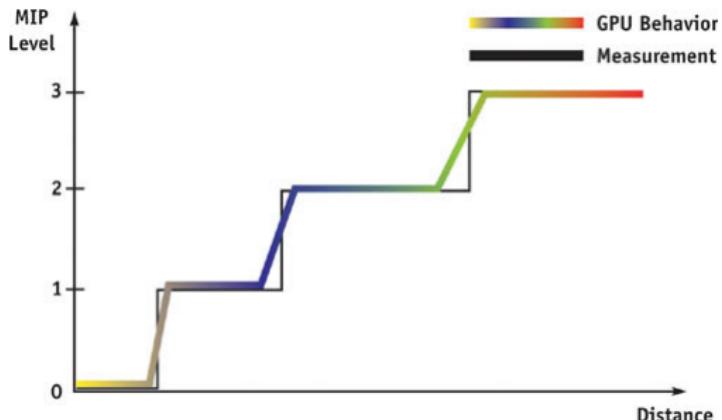
- Použití vhodného rozlišení podle jejího aktuálního vzorkování při zobrazení
- Předzpracování textur
- Vzorkování pro různá rozlišení s pokročilou interpolací hodnot



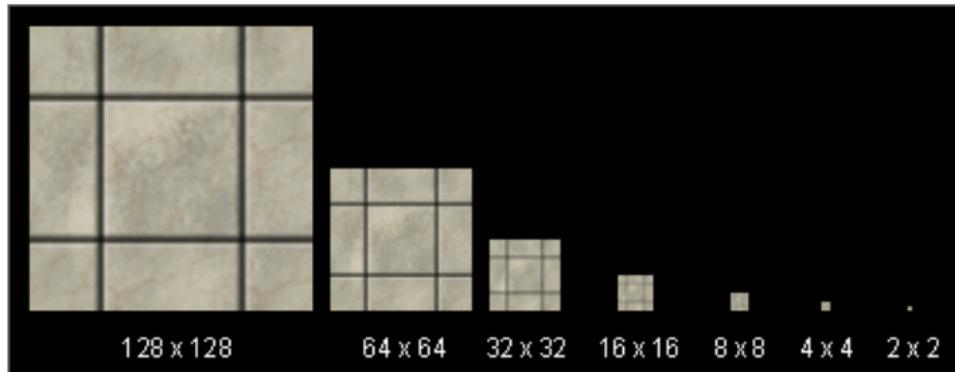
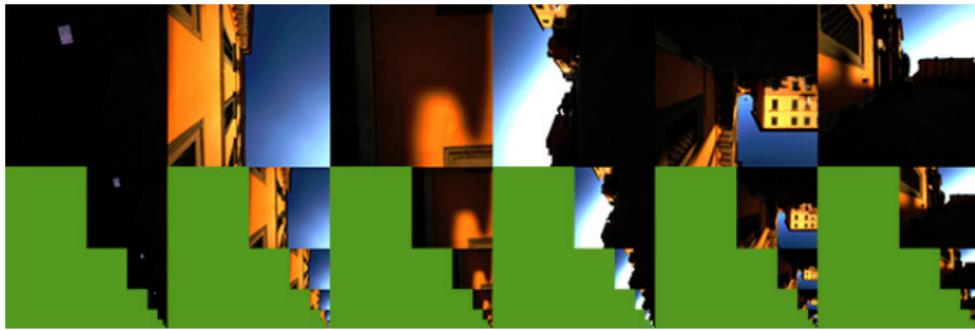
# MIP Mapping textur

## Kdy se přepíná úroveň/rozlišení?

- V zásadě podle vzdálenosti textury od kamery (velikosti obrazu textury)
- *Viz. přednáška o texturování*



# Příklad MIP textur



# Základy počítačové grafiky

## Vyplňování 2D oblastí

Michal Španěl  
Rostislav Hulík

Ústav počítačové grafiky a multimédií  
Fakulta informačních technologií  
Vysoké učení technické v Brně

Brno 2016

# Cíl přednášky

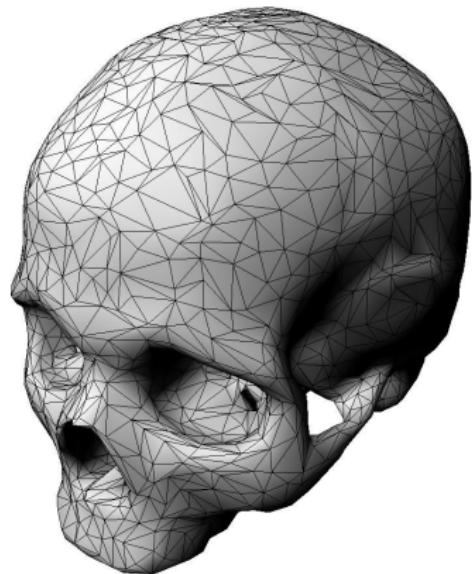
Seznámit se s algoritmy pro vyplňování uzavřených rovinných oblastí s hranicí definovanou rastrově nebo vektorově.

# Vyplňování oblastí

## Definice

Proces nalezení a označení (obarvení) všech vnitřních bodů dané oblasti.

- Vstupní informací je popis hranice oblasti.
- Výstupem je rastrový popis vyplněné oblasti.
- V podstatě jde o rasterizaci dané oblasti.



# Popis a typy oblastí

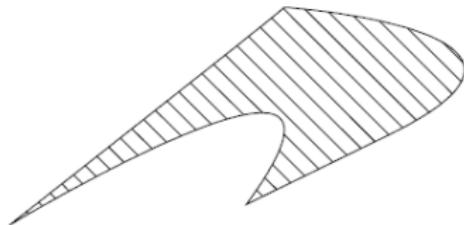
## Rastrové oblasti

Hranice popsána v rastrové matici hodnotou pixelů na hranici nebo barvou vyplňované oblasti.



## Vektorové oblasti

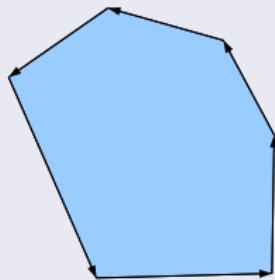
Hranice popsána seznamem vektorových entit (úseček, kruhových oblouků, křivek, atd.).



# Dělení oblastí podle geometrie

## Konvexní (vypouklé, vyduté)

Pro libovolné dva body oblasti platí, že jejich spojnice je součástí oblasti, neprotíná její hranice.



?

Jak určíme, kde je uvnitř a kde vně oblasti?

## Konkávní (nekonvexní, prohnuté, duté)

Oblast, která není konvexní.

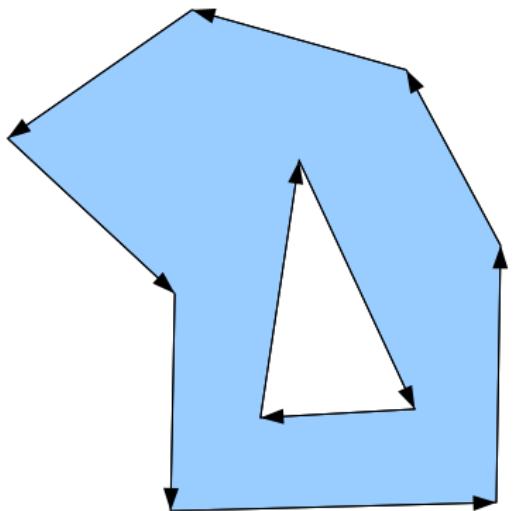


## S vnitřními otvory

Hranice je tvořena více nezávislými smyčkami, obrysem a otvory.



# Vektorový popis oblastí



## Korektní definice oblasti

- Seznam hraničních entit vyplňované oblasti musí být *orientovaný* a *spojitý*.
- Vnitřní díry mají vždy opačnou orientaci.

## Jak otestovat orientaci?

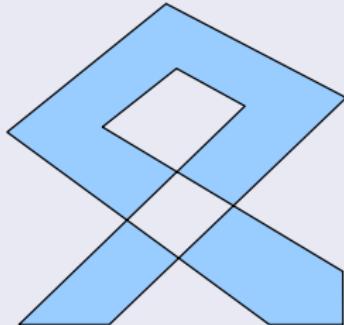
- Vektorový součin dvou sousedních hran (pravidlo pravé ruky).
- V případě nekonvexních polygonů
  - suma přes celý polygon.

# Pravidla vyplňování složitých oblastí

- Řeší nejednoznačnost při vyplňování složitých oblastí (hranice protínají sami sebe).

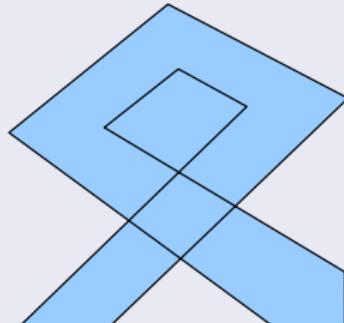
## Paritní vyplňování

Hranice odděluje vyplněný a nevyplněný prostor, nejběžnější způsob.



## Vnitřní vyplňování

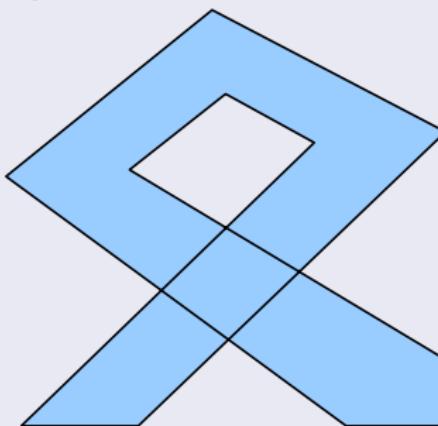
Jsou vyplněny všechny body, které nejsou vně oblasti.



# Pravidla vyplňování složitých oblastí, pokr.

## Nenulové vyplňování

Nenulové objetí bodu uzavřenou smyčkou při vyřešení sebeprotínání.



# Druhy výplní

Konstantní barvou (*solid*)

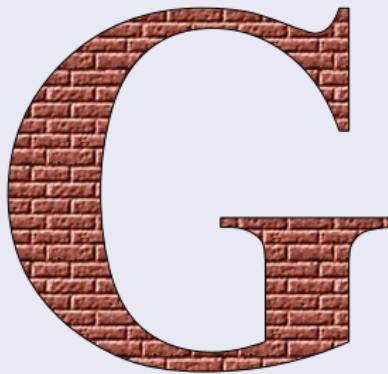


Jednoduchým čárovým  
vzorem (šrafovou, angl. *hatch*)



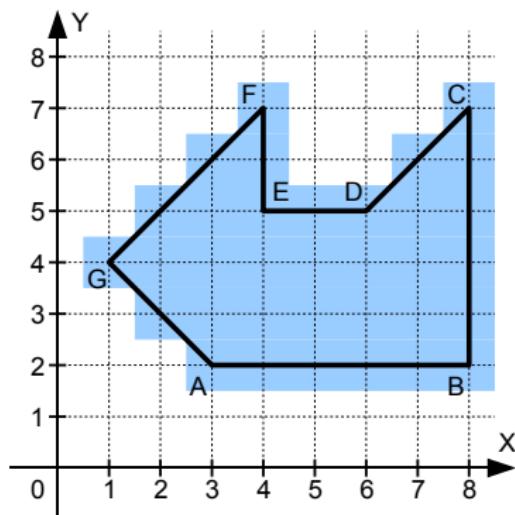
# Druhy výplní, pokr.

Složitějším obecným vzorem nebo  
texturou (*pattern*)



# Řádkové vyplňování (angl. Scanline Fill)

- Základní algoritmus vyplňování obecných mnohoúhelníků.
- Seznam hraničních entit (hran) oblasti  $\Omega$ .
- Orientovaný seznam vrcholů úseček.



## Základní myšlenka

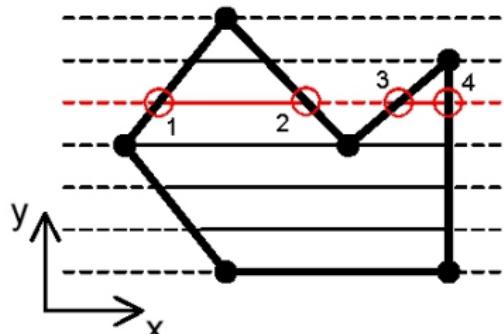
- Procházení oblasti  $\Omega$  po jednotlivých řádcích výsledného rastru.
- Úseky ležící uvnitř oblasti jsou vyplněny barvou nebo vzorem.

# Řádkové vyplňování, pokr.

## Algoritmus (pro paritní vyplňování)

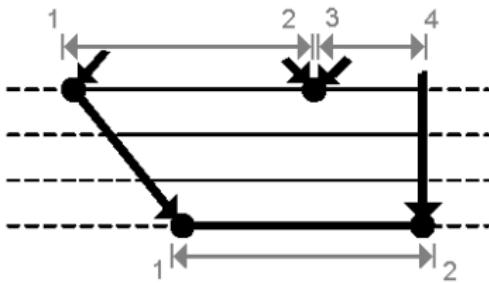
Pro každý řádek  $Y_j \in \langle Y_{min}, Y_{max} \rangle$  oblasti  $\Omega$ :

- Vytvoř seznam souřadnic  $x_i$  průsečíků řádku  $Y_j$  se všemi hraničními usečkami. *Vodorovné hrany se vyneschávají!*
- Setřídění seznamu průsečíků podle souřadnic  $x_i$ .
- Vykreslení vodorovných úseků řádku  $Y_j$  mezi lichými a sudými průsečíky seznamu (dvojice tvoří úsek uvnitř oblasti).



# Problém krajních bodů úseček (lichý počet průsečíků hran)

- Algoritmus předpokládá sudý počet průsečíků.
- Nemusí platit, pokud řádek protíná některý vrchol hranice!



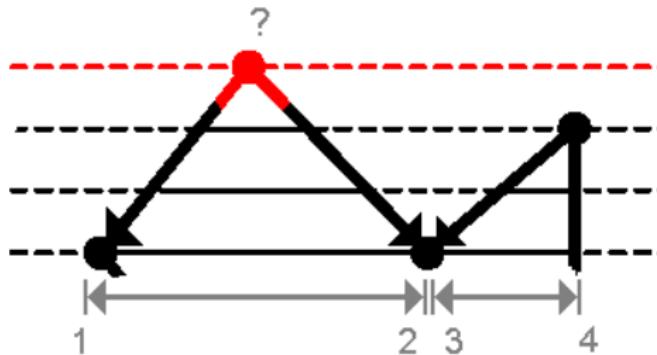
## Obecné řešení

- Ve vrcholech, které jsou lokálním extrémem v ose Y generovat průsečíky obou hran.
- V ostatních vrcholech generovat průsečík pouze jednou.

# Problém krajních bodů úseček, pokr.

## Analýza typu vrcholů

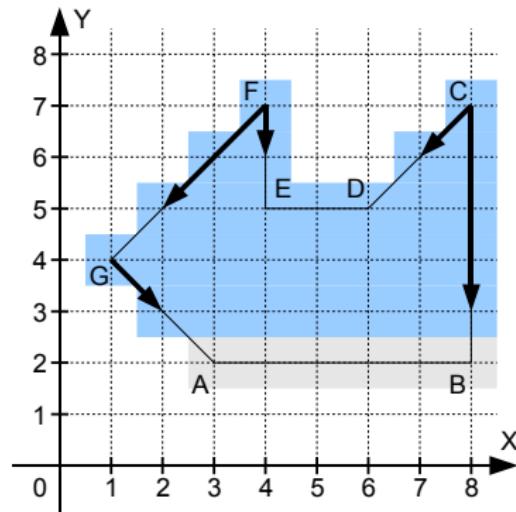
- Generování správného počtu průsečíků testováním extrému v ose Y.
- Implementace algoritmu se komplikuje a zpomaluje!



# Problém krajních bodů úseček, pokr.

Zkrácení dolního okraje všech hran

- Zkrátit hranu ve směru osy Y o 1.
- Vodorovné hrany se vypouštějí.
- Je nutné překreslit obrys oblasti!



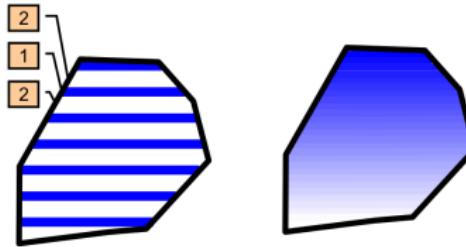
# Šrafování, gradient

## Vyplnění vodorovnou šrafovou

- Jednoduchá úprava vyplňovacích algoritmů.
- Přeskakování řádků pomocí parametru.

## Vyplnění gradientem

- Podobné šrafování.
- Parametr inkrementující se každý řádek.



# Šrafování, gradient pod libovolným úhlem

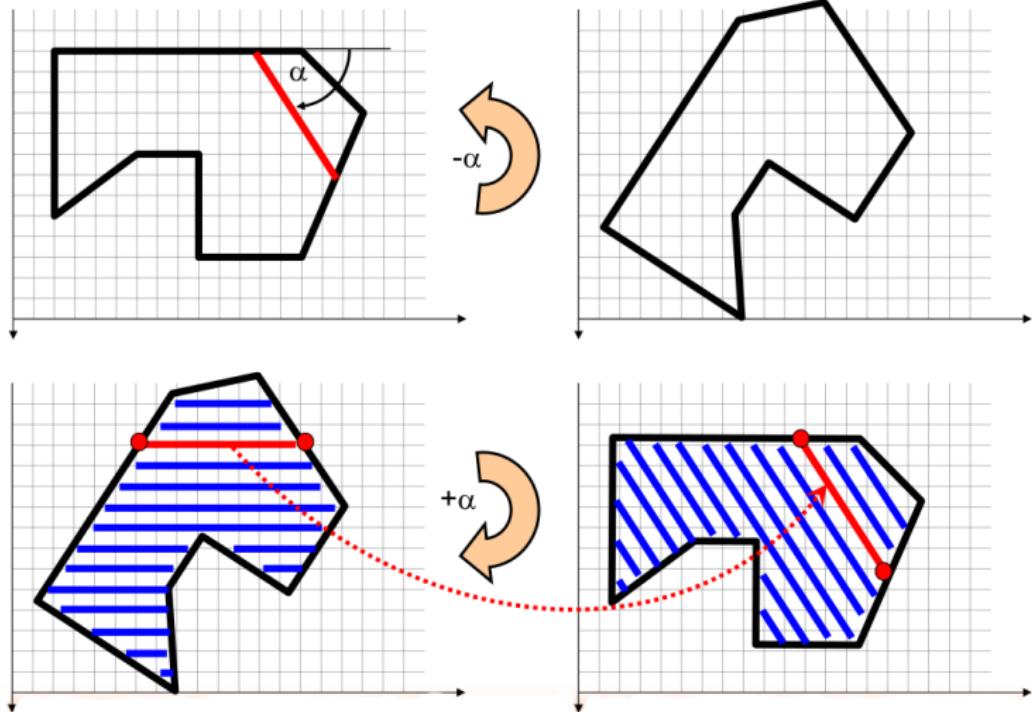
## Algorimus pro šrafování pod libovolným úhlem

### Modifikace vodorovného šrafování

- Pro šrafu/gradient pod úhlem  $\alpha$  nutné nejprve otočit oblast o úhel  $-\alpha$
- Šrafovat vodorovně
- Otočit oblast zpět o úhel  $\alpha$

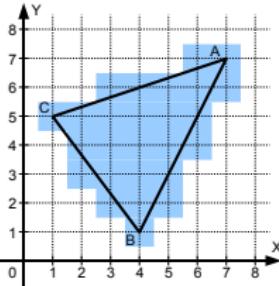
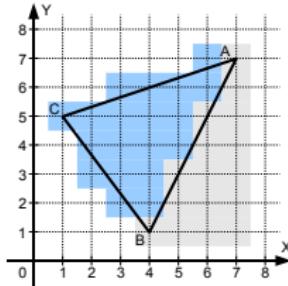
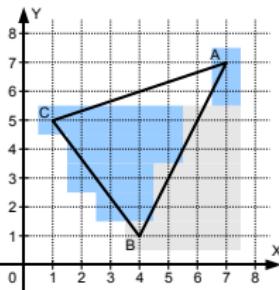
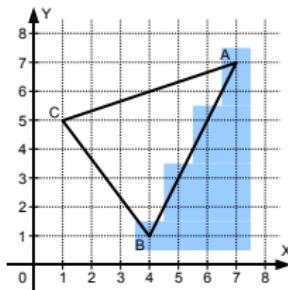
Otočení se řeší transformační maticí

# Šrafování, gradient pod libovolným úhlem



# Inverzní řádkové vyplňování

- Odstraňuje nutnost třídit průsečíky pro každý řádek.
- Mnohoúhelníky, seznam hran oblasti nebo orientovaný seznam vrcholů.



## Základní myšlenka

- Procházení oblasti  $\Omega$  po jednotlivých hranách  $e_i \in \Omega$ .
- Od každé hrany je napravo po řádcích provedeno vyplňení s inverzí hodnot pixelů.

# Inverzní řádkové vyplňování, pokr.

## Algoritmus (pro paritní vyplňování)

Nalezení maximální souřadnice  $X_{max}$  v ose  $X$ .

Každou hranu  $e_i \in \Omega$  oblasti zpracuj následovně:

- Získání  $Y_{min}$  a  $Y_{max}$  pro danou hranu  $e_i$ .
- Pro každý řádek  $Y_j \in \langle Y_{min}, Y_{max} \rangle$ , kromě vodorovných hran, proved'：
  - Najdi průsečík  $P_{ij} = (x_{ij}, y_{ij})$  řádku  $Y_j$  s aktuální hranou  $e_i$ .
  - Invertuj hodnoty pixelů v řádku  $Y_j$  od průsečíku  $P_{ij}$  po maximální souřadnici oblasti  $X_{max}$ .

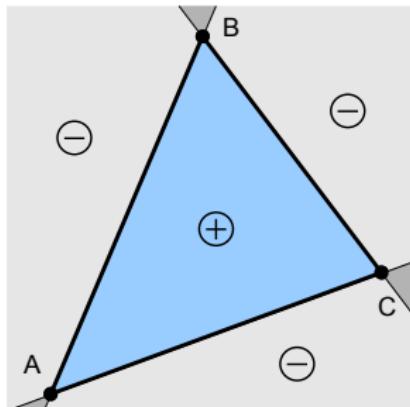
# Inverzní řádkové vyplňování, pokr.

+/-

- Lineární časová složitost v závislosti na počtu hran.
- Po vyplnění je nutné překreslit obrys oblasti.
- Binární operace inverze pixelů → binární obraz.
- Ovlivňuje okolí oblasti → generování šablony v pomocném bufferu.

# Pinedův algoritmus (J. Pineda, 1988)

- Pracuje pouze s *konvexními mnohoúhelníky*.
- Vyplňovaná oblast  $\Omega$  je opět popsána seznamem hraničních entit (hran)  $\Omega = \{e_1, e_2, \dots, e_n\}$ .
- Nejčastěji používán pro trojúhelníky (vždy konvexní).
- Snadná realizace v HW.



## Základní myšlenka

- Rozdělení roviny oblasti  $\Omega$  na poloroviny hran  $e_i \in \Omega$ .
- Body roviny, které leží na kladné straně všech polorovin hran  $e_i$  jsou uvnitř oblasti  $\Omega$ .

# Test polohy bodu $P(x, y)$ k přímce

## Hranová funkce $E_i(x, y)$

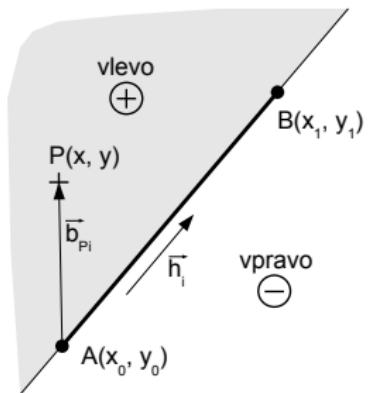
Vekt. součin vektoru  $\vec{h}_i$  hrany a vektoru  $\vec{b}_{Pi}$  z počátku hrany k testovanému bodu  $P$ .

$$\vec{h}_i = (x_{i1} - x_{i0}, y_{i1} - y_{i0}) = (\Delta x_i, \Delta y_i)$$

$$\vec{b}_{Pi} = (x - x_{i0}, y - y_{i0})$$

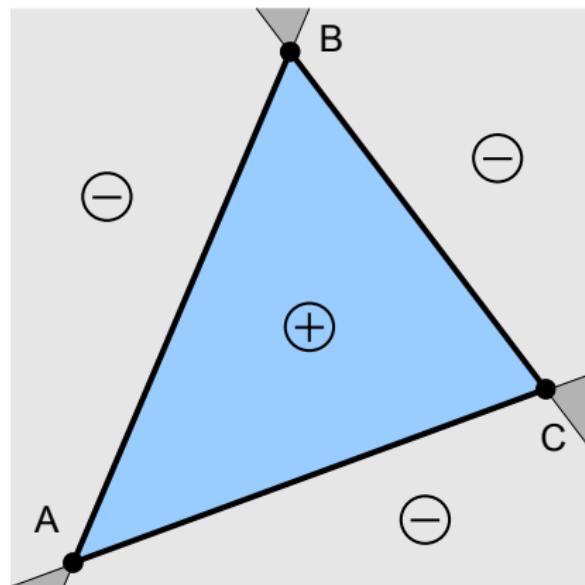
$$E_i(x, y) = \vec{h}_i \times \vec{b}_{Pi}$$

$$E_i(x, y) = (x - x_{i0})\Delta y_i - (y - y_{i0})\Delta x_i$$



# Trojúhelníková oblast a její poloroviny

- Je-li hodnota všech hranových funkcí  $E_i(x, y) \geq 0$ , pak bod  $P(x, y)$  leží uvnitř nebo na hranici oblasti.



# Výpočet hranové funkce

- Není nutné vyhodnocovat hranové fce  $E_i(x, y)$  pro každý bod!
- Hodnotu lze určit na základě sousedního bodu  $P(x \pm 1, y)$  nebo  $P(x, y \pm 1)$ .

## Odvození

$$E_i(x, y) = (x - x_{i0})\Delta y_i - (y - y_{i0})\Delta x_i$$

$$E_i(x + 1, y) = (x + 1 - x_{i0})\Delta y_i - (y - y_{i0})\Delta x_i$$

$$E_i(x + 1, y) = E_i(x, y) + \Delta y_i$$

$$E_i(x \pm 1, y) = E_i(x, y) \pm \Delta y_i$$

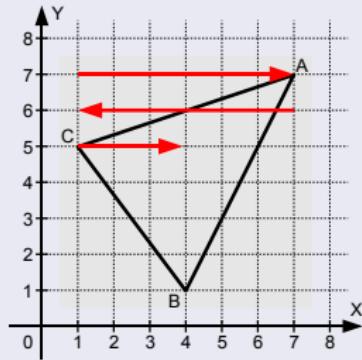
$$E_i(x, y \pm 1) = E_i(x, y) \pm \Delta x_i$$

# Základní algoritmus

- Nalezení  $x_{min}$ ,  $x_{max}$ ,  $y_{min}$  a  $y_{max}$ .
- Pro každou hranu oblasti inicializuj  $E_i(x_{min}, y_{min})$ .
- Cyklus přes všechny body  $(x, y)$  v obdélníku  $(x_{min}, y_{min}), (x_{max}, y_{max})$ :
  - Je-li  $E_i(x, y) \geq 0$  pro všechny hrany, pak nastav hodnotu pixelu.
  - Aktualizace  $E_i(x, y)$ .

Procházení opsaného obdélníka po řádcích

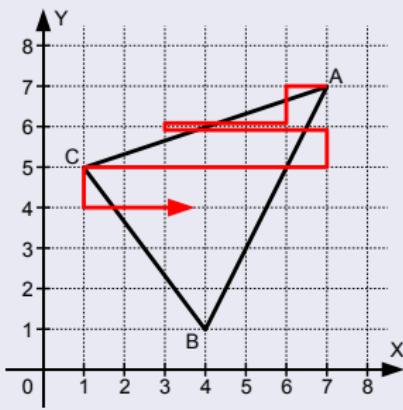
Zbytečný průchod prázdnou  $\sim 1/2$  obdélníka.



# Základní algoritmus

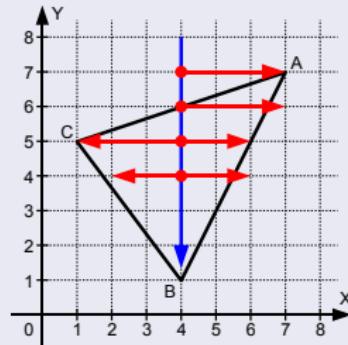
Procházení od maximálního vrcholu s obratem a přechodem na další řádek

Algoritmicky složitější.



Procházení po řádcích od svislého plotu

Plot ve středu opsaného obdélníka nebo skrz prostřední vrchol. Možnost paralelizace (strany nezávisle).

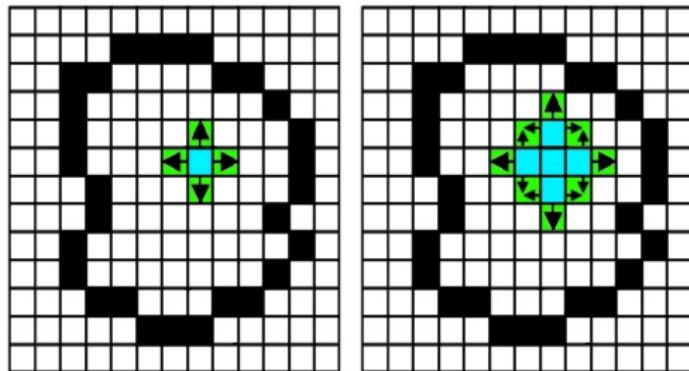


# Semínkové vyplňování (angl. Flood-fill)

- Vyplňování rastrových oblastí.
- Startovací bod (semínko) uvnitř oblasti.

## Základní myšlenka

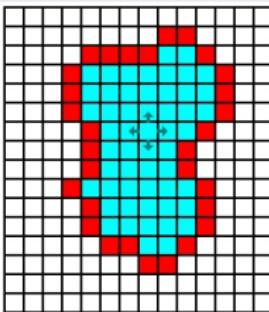
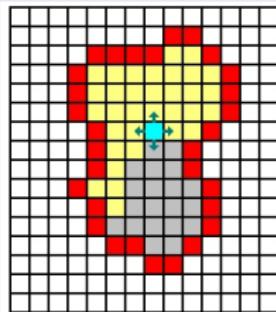
- Semínko uvnitř oblasti šíříme na sousedy (obarvování sousedních pixelů).
- Obarvené pixely se rekursivně stávají semínky.



# Definice hranice oblasti

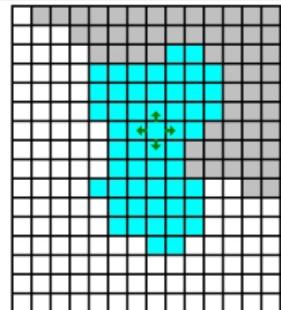
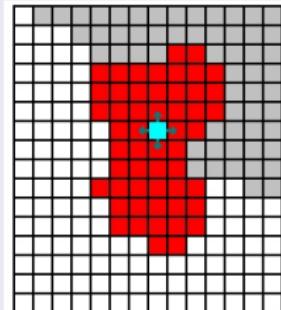
## Hraniční

Oblast definována spojitou hranicí z pixelů dané barvy.



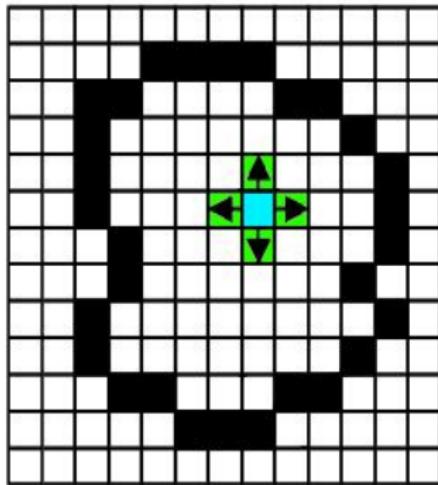
## Záplavová

Oblast definována spojitou množinou vnitřních pixelů dané barvy.



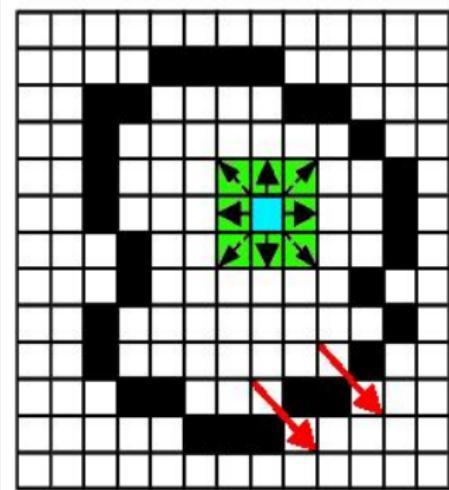
# Způsob výběru sousedů

## 4-okolí



## 8-okolí

Vyžaduje "silnější" hranice.



# Příklady implementace

## Rekurzivní implementace

```
FloodFill(int x, int y)
{
    if (get_pixel(x,y) == background_color)
    {
        set_pixel(x, y, fill_color);
        FloodFill(x, y - 1);
        FloodFill(x, y + 1);
        FloodFill(x - 1, y);
        FloodFill(x + 1, y);
    }
}
```

- Hrozí nebezpečí přetečení zásobníku!

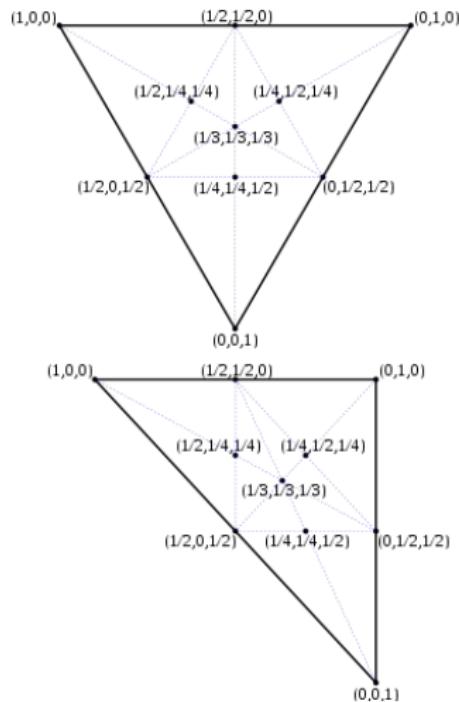
## Implementace s využitím fronty/zásobníku

- Vlož počáteční semínko  $(x_0, y_0)$  do fronty.
- Dokud fronta není prázdná:
  - Vyber semínko  $(x_i, y_i)$  ze začátku fronty.
  - Je-li uvnitř oblasti, pak nastav hodnotu pixelu a vlož sousedy  $(x_i + 1, y_i), \dots$  do fronty.

# Barycentrické souřadnice

## Barycentrické souřadnice

- Představeny A. F. Möbiem (1827).
- Souřadnice vztažené relativně k  $n$  bodům.
- Pro  $n$ -úhelník lze bod vyjádřit  $n$  souřadnicemi.
- Představují vliv "hmotnosti" jednotlivých bodů  $n$ -úhelníku (barycenter = těžiště) v konkrétním bodě.



# Barycentrické souřadnice pro trojúhelník

## Vlastnosti

- Mějme libovolný bod  $r = (\lambda_1, \lambda_2, \lambda_3)$  v barycentrických souřadnicích
- Mějme  $p_1, p_2, p_3$  body trojúhelníku
- Pak platí
  - $r = \lambda_1 p_1 + \lambda_2 p_2 + \lambda_3 p_3$
  - $\lambda_1 + \lambda_2 + \lambda_3 = 1$  pokud bod  $r$  náleží trojúhelníku  $(p_1, p_2, p_3)$

## Vhodné pro určení polohy bodu vůči trojúhelníku

- $\lambda_1 + \lambda_2 + \lambda_3 = 1$  pokud bod  $r$  náleží trojúhelníku  $(p_1, p_2, p_3)$
- $\lambda_1 + \lambda_2 + \lambda_3 \neq 1$  pokud bod  $r$  leží mimo trojúhelník  $(p_1, p_2, p_3)$
- $\lambda_1 = 0$  a  $\lambda_2 + \lambda_3 = 1$  pokud bod  $r$  leží na hrani protilehlé bodu  $p_1$

# Výpočet barycentrických souřadnic pro trojúhelník

## Konverze z kartézských souřadnic

- $\lambda_1 = A(r, p_2, p_3) / A(p_1, p_2, p_3)$
- $\lambda_2 = A(r, p_1, p_3) / A(p_2, p_1, p_3)$
- $\lambda_3 = A(r, p_1, p_2) / A(p_3, p_1, p_2)$

# Barycentrické souřadnice pro N-úhelník

## Zobecnění souřadnic pro N-úhelník

- Pro N-úhelník je bod vyjádřen n souřadnicemi.
- Platí

$$\sum_{i=1..n} \lambda_i = 1$$

pro bod náležící n-úhelníku.

- Platí

$$r = \sum_{i=1..n} \lambda_i r_i$$

pro libovolný bod.

# Příklady implementace

## Optimalizace vyplňování

- Řádkové vyplňování se seznamem aktivních hran.
- Řádkové semínkové vyplňování (angl. scanline seed fill).

## Druhy výplní

- Použití šablon pro šrafování.
- Vyplnění vzorkem.

## Literatura

- Žára, J., Beneš, B., Felkel, P., *Moderní počítačová grafika*, ComputerPress, 1999.
- ... [www.google.com](http://www.google.com) ...

# Základy počítačové grafiky

## Geometrické transformace ve 2D a 3D

Michal Španěl  
Rostislav Hulík  
Jiří Havel

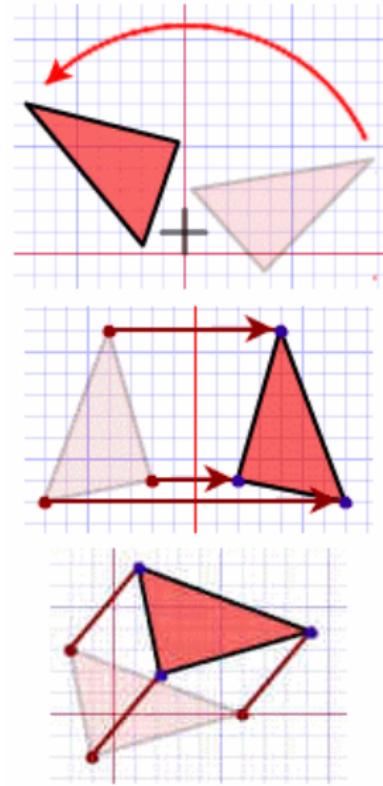
Ústav počítačové grafiky a multimédií  
Fakulta informačních technologií  
Vysoké učení technické v Brně

Brno 2016

# Cíl přednášky

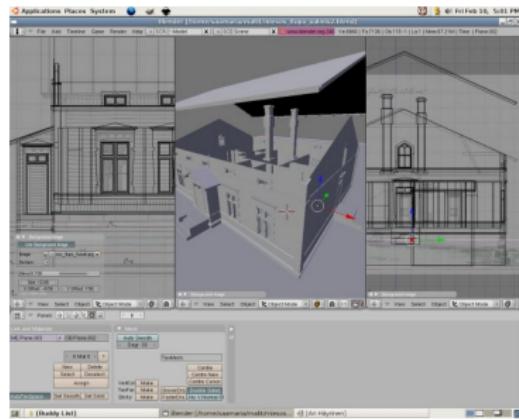
Bez geometrických transformací není moderní vektorové počítačové grafiky a její hardwarové akcelerace!

Seznámit se s principy transformací vektorových objektů ve 2D a 3D prostoru.



# Geometrické transformace vektorových objektů

- Popis objektů založen na *uzlových bodech, vrcholech*.
- Vytváření a zobrazování objektů → operace posunutí, otočení, zmenšení/zvětšení, zkosení vrcholů.
- **Nejčastější operace v současné grafice!**
- Prvním krokem pro akceleraci bývá *HW implementace transformací*.



# První vektorová 3D grafika ...

- Slabý výpočetní výkon, nulová podpora grafiky.
- Jednoduché objekty a geometrické transformace – *základ veškerého zobrazování*.



# Počítačová 3D grafika včera ...

- Osvětlení, stínování, texturování, částicové systémy, ...
- *HW akcelerace.*
- Stále "stejné" geometrické transformace!



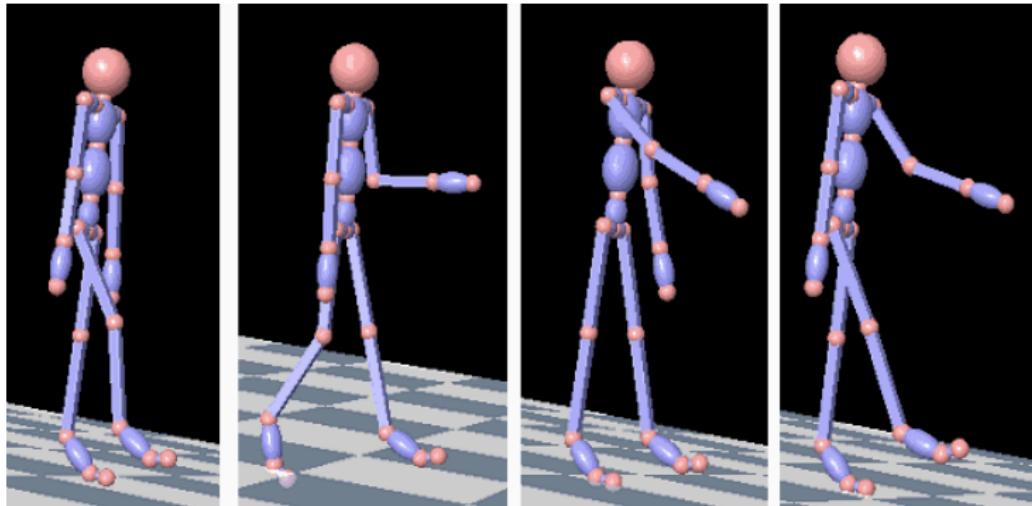
# Počítačová 3D grafika dnes ...

- Generování geometrie, šíření světla, ...
- Masivní *HW akcelerace*.
- Pořád "stejné" geometrické transformace!



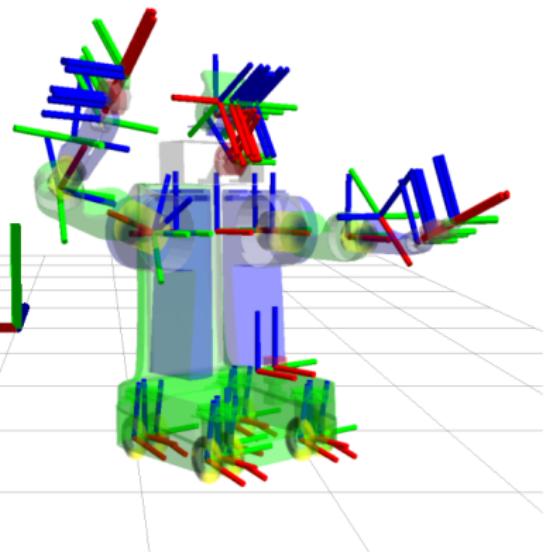
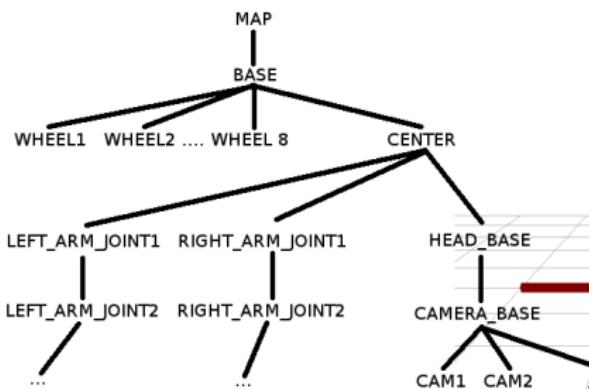
# Animace kloubových soustav

- Postava složena z dílčích modelů.
- Animace realizována pomocí transformací aplikovaných na jednotlivé části.



# Vztahy mezi komponenty

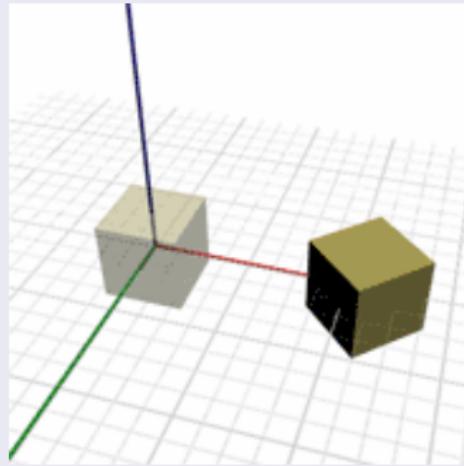
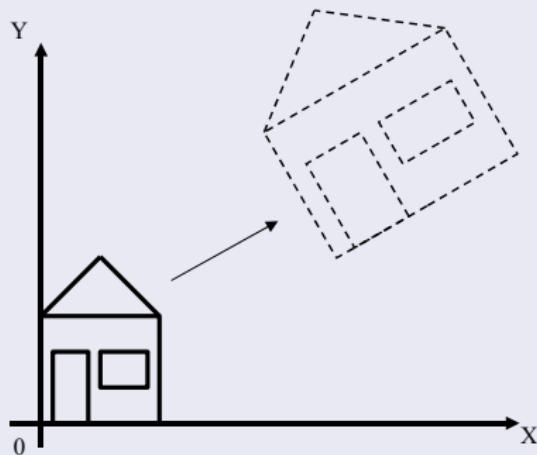
- Vztahy mezi komponenty
- Pozice v prostoru
- Strom transformací (hledání cesty)



# Způsob aplikace transformace

Změna polohy vrcholů objektu v souřadném systému

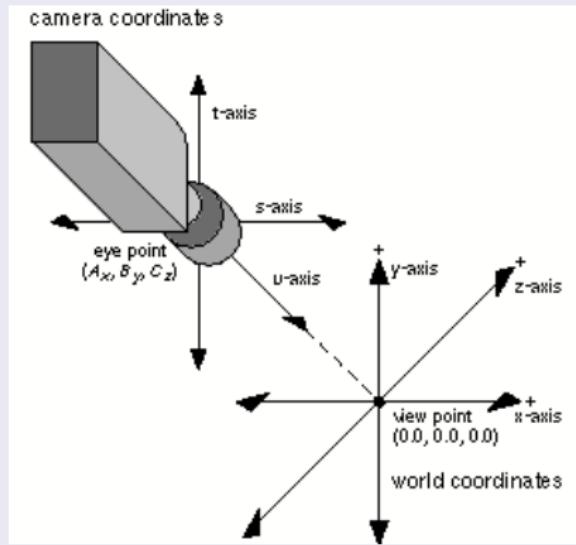
- Operace s objekty (posunutí, rotace, atd.).



# Způsob aplikace transformace, pokr.

## Změna souřadného systému do vhodnější pozice

- Např. pro zjednodušení výpočtu perspektivní/paralelní projekce.

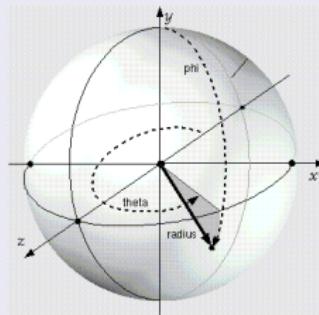
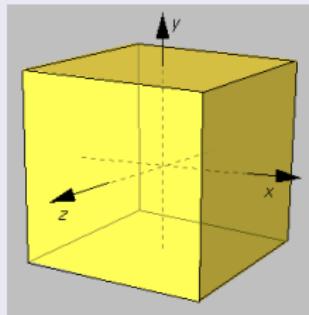


# Zobrazování 3D scény

- 3D modely jednotlivých objektů.
- Model scény (rozmístění objektů, poloha kamery, apod.).

## 3D model objektu

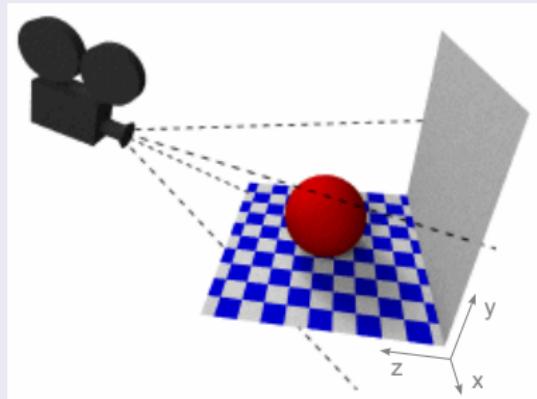
- Nejčastěji obecný model (koule, židle, atd.)
- Vhodně **zvolený souřadný systém**.



# Zobrazování 3D scény, pokr.

## Vytvoření 3D scény

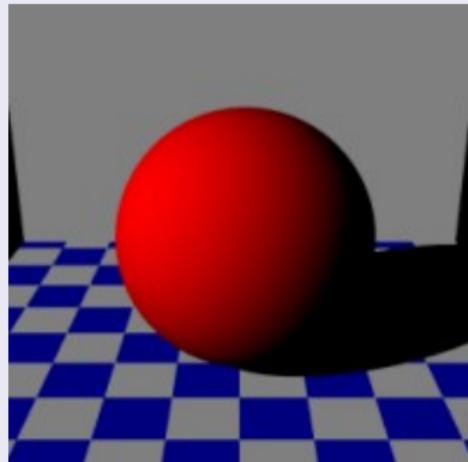
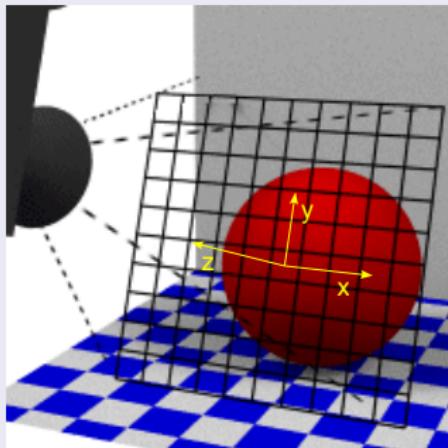
- Souřadný systém scény (angl. *world coordinates*).
- Umístění objektu do scény → transformace do prostoru scény (posunutí, rotace, apod.).
- Definice kamery (poloha, směr, úhel).



# Zobrazování 3D scény, pokr.

## Zobrazení scény

- Transformace do souřadného systému kamery.
- (Perspektivní) projekce.



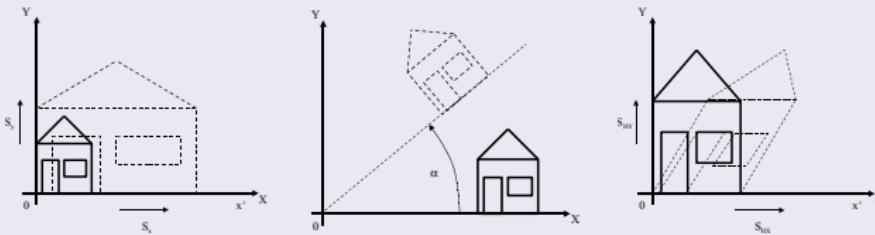
# Lineární transformace

## Definice

Lineární transformace je zobrazení  $f$  z jednoho vektorového prostoru do druhého  $f : V \rightarrow W$ , které zachovává lineární kombinace. Pro libovolné dva vektory  $\vec{x}_1, \vec{x}_2$  a skalár  $\alpha$  platí:

$$\begin{aligned} f(\vec{x}_1 + \vec{x}_2) &= f(\vec{x}_1) + f(\vec{x}_2), \\ f(\alpha \vec{x}_1) &= \alpha f(\vec{x}_1). \end{aligned}$$

## Měřítko, rotace a zkosení jsou lineární transformace



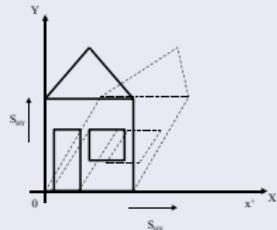
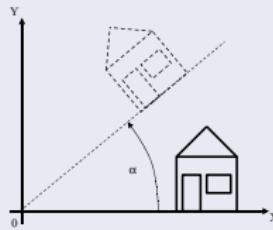
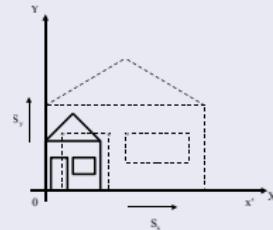
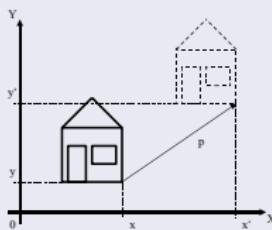
# Afinní transformace

## Definice

*Afinní transformace* je zobrazení  $f$  z jednoho vektorového prostoru do druhého  $f : V \rightarrow W$ , které zachovává kolinearitu (tzn. body ležící na přímce budou ležet na přímce i po zobrazení) a dělící poměr.

**Lze vyjádřit jako lineární transformaci následovanou posunem.**

Všechny základní geometrické transformace jsou affinní



# Homogenní souřadnice ve 2D

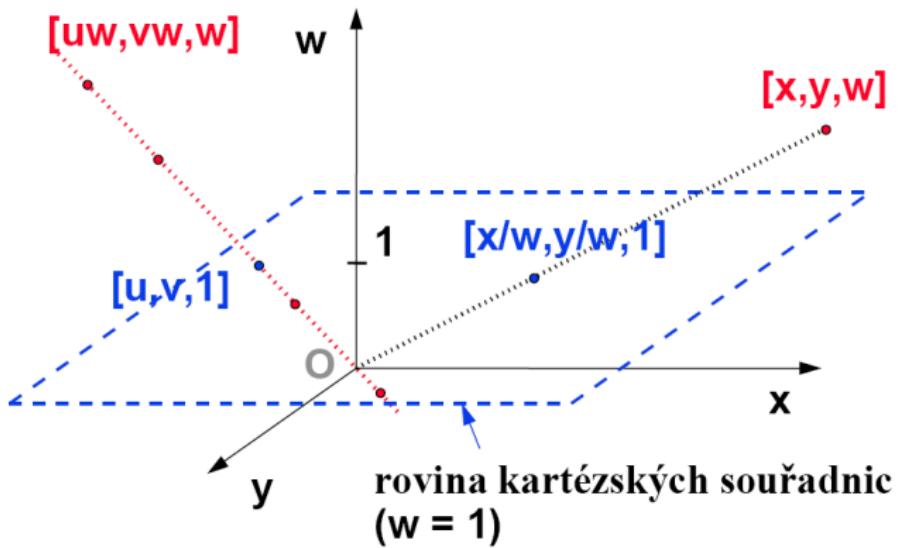
- Umožňují pracovat se všemi druhy základních transformací jednotně, pomocí *maticového zápisu* (viz. dále).
- Skládání transformací.

## Definice

*Homogenní souřadnice* bodu ve 2D s kartézskými souřadnicemi  $[x, y]$  je uspořádaná trojice  $[X, Y, w]$  pro kterou platí  $x = X/w$  a  $y = Y/w$ . Souřadnici  $w$  nazýváme **váhou bodu**.

- V případě affinních transformací je  $w = 1$ .
- Vektory  $\vec{v} = (x, y)$  reprezentujeme trojicí  $\vec{v} = (x, y, 0)$ , kde  $w = 0$ .

# Geometrická představa homogenních souřadnic ve 2D



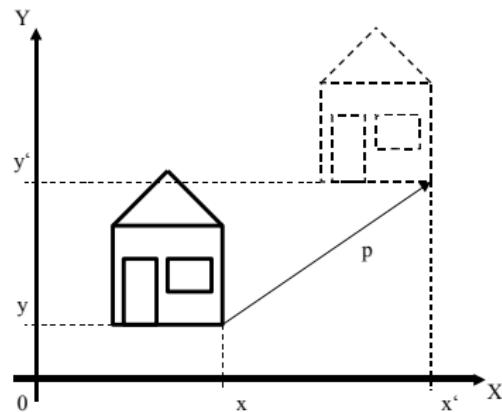
# Posunutí ve 2D (angl. translation)

- Posunutí bodu v rovině s homogenními souřadnicemi  $P(x, y, 1)$ .
- Vektor posunutí  $\vec{T}(d_x, d_y)$ .
- $x' = x + d_x, \quad y' = y + d_y$

## Maticový zápis transformace

$$[x', y', 1] = [x, y, 1] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ d_x & d_y & 1 \end{bmatrix}$$

$$P' = P \cdot T$$



# Inverzní transformace (posunutí opačným směrem)

## Maticový zápis inverzní transformace

$$[x', y', 1] = [x, y, 1] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -d_x & -d_y & 1 \end{bmatrix}$$

$$P' = P \cdot T^{-1}$$

## Transformační matice pro posunutí ve 2D

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ d_x & d_y & 1 \end{bmatrix} \quad T^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -d_x & -d_y & 1 \end{bmatrix}$$

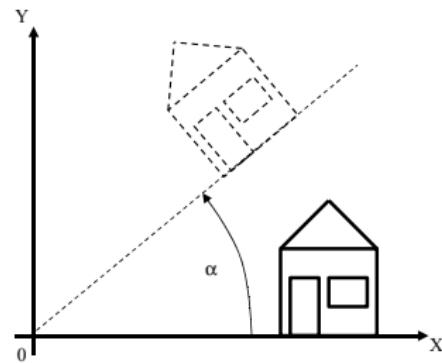
# Otočení ve 2D (rotation)

- Otočení bodu v rovině s homogenními souřadnicemi  $P(x, y, 1)$  o úhel  $\alpha$ .
- Střed otáčení v počátku souřadného systému.
- $x' = x \cdot \cos \alpha - y \cdot \sin \alpha, \quad y' = x \cdot \sin \alpha + y \cdot \cos \alpha$

## Maticový zápis transformace

$$[x', y', 1] = [x, y, 1] \cdot \begin{bmatrix} \cos \alpha & \sin \alpha & 0 \\ -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

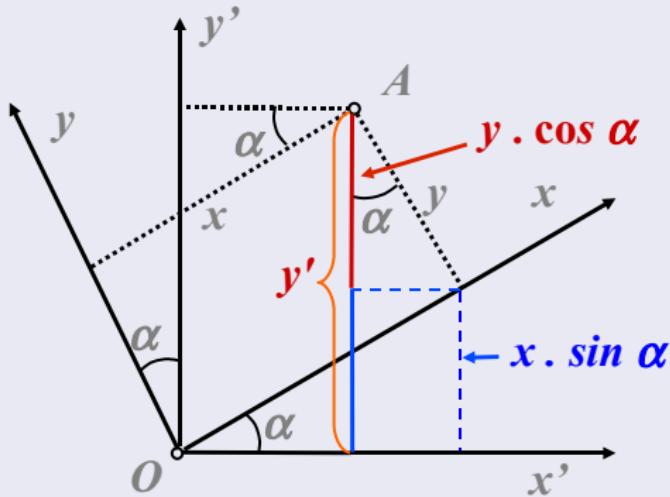
$$P' = P \cdot R$$



# Otočení ve 2D, pokr.

## Odvození transformace

- $y' = x \cdot \sin \alpha + y \cdot \cos \alpha$



# Inverzní transformace

## Transformační matice pro otočení ve 2D

$$R = \begin{bmatrix} \cos \alpha & \sin \alpha & 0 \\ -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad R^{-1} = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

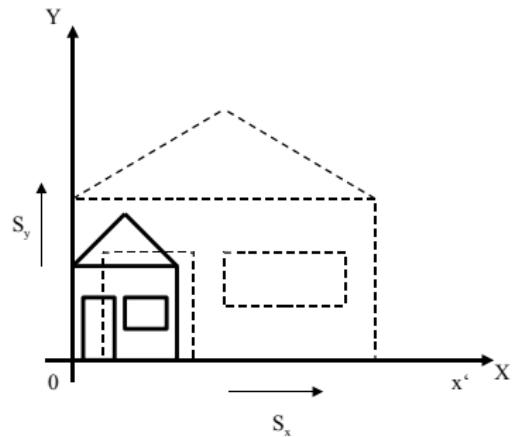
# Změna měřítka ve 2D (scale)

- Změna měřítka s faktory  $S_x$  a  $S_y$  ve směru jednotlivých os.
- $S_{x,y} > 1 \rightarrow$  zvětšení.
- $0 < S_{x,y} < 1 \rightarrow$  zmenšení.
- $S_{x,y} < 0 \rightarrow$  dochází k převrácení (zrcadlení).
- $x' = x \cdot S_x, \quad y' = y \cdot S_y$

## Transformační matice

$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$S^{-1} = \begin{bmatrix} 1/S_x & 0 & 0 \\ 0 & 1/S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



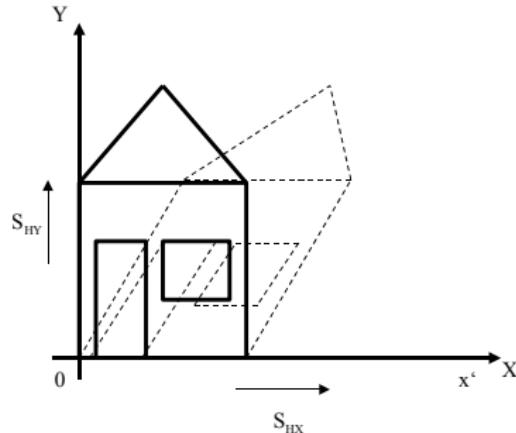
# Zkosení ve 2D (shear)

- Zkosení bodu v rovině s homogenními souřadnicemi  $P(x, y, 1)$  a faktory zkosení  $S_{hx}$  a  $S_{hy}$ .
- $x' = x + S_{hx} \cdot y, \quad y' = y + S_{hy} \cdot x$

## Transformační matice

$$S_H = \begin{bmatrix} 1 & S_{hy} & 0 \\ S_{hx} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$S_H^{-1} = \begin{bmatrix} 1 & -S_{hy} & 0 \\ -S_{hx} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



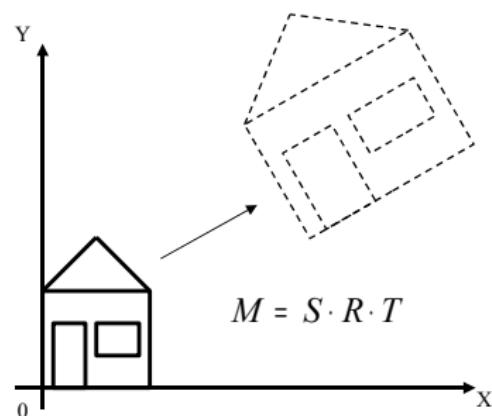
# Příklad č. 1 - Transformace

- Mějme čtverec ABCD, kde A=(0, 0), B=(2, 0), C=(2, 2), D=(0, 2)
- Aplikujte zkosení  $S_hx = 1,5$  a  $S_hy = 0$
- Transformační matice:

$$S_H = \begin{bmatrix} 1 & S_{hy} & 0 \\ S_{hx} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

# Skládání transformací

- Každá komplexní lineární transformace se dá rozložit na základní transformace.
- Složená transformace se dá vyjádřit jedinou maticí.
- *Skládání se provádí násobením matic.*
- **Záleží na pořadí transformací!**
- Matice násobíme zprava ve stejném pořadí!



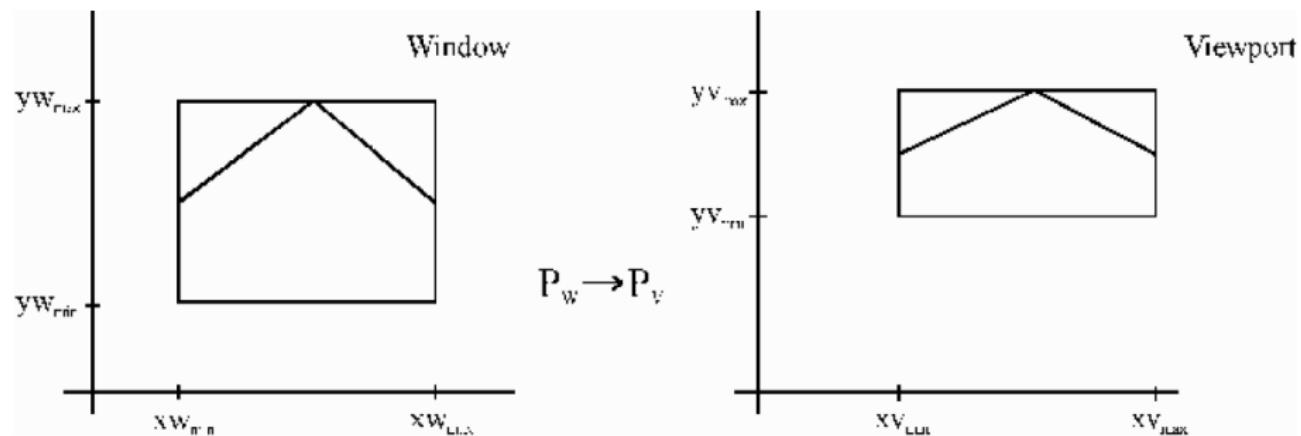
## Příklad č. 2 - skládání Transformací

- Mějme čtverec ABCD, kde A=(2, 2), B=(4, 2), C=(4, 4), D=(2, 4)
- Otočte tento čtverec okolo bodu P=(3, 3) o úhel  $\alpha = \pi/4$
- Transformační matice:

$$R = \begin{bmatrix} \cos \alpha & \sin \alpha & 0 \\ -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ d_x & d_y & 1 \end{bmatrix}$$

# Transformace okna pohledu



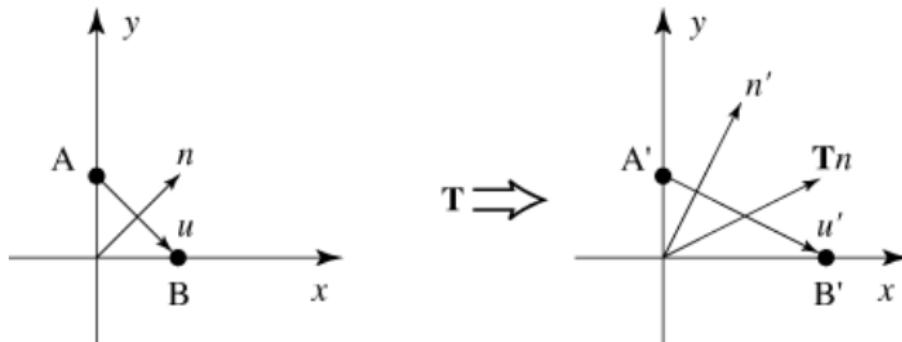
$$x' = s_x(x - x_{w_{min}}) + x_{v_{max}} \quad s_x = \frac{x_{v_{max}} - x_{v_{min}}}{x_{w_{max}} - x_{w_{min}}}$$

$$y' = s_y(y - y_{w_{min}}) + y_{v_{max}} \quad s_y = \frac{y_{v_{max}} - y_{v_{min}}}{y_{w_{max}} - y_{w_{min}}}$$

# Transformace normály

- Směrové vektory v homogenních souřadnicích  $\vec{v} = (x, y, 0)$  transformujeme stejně jako body.
- Neplatí pro normálové vektory**
- Transformace normál provádíme *násobením inverzní transpozicí matici  $M$ :*

$$\vec{n}' = \vec{n} \cdot M^{-1T}$$



## Příklad č. 3 - Transformace normálového vektoru

- Mějme vektor  $\vec{v} = (1, 1, 0)$  a jeho normálu  $\vec{n} = (-1, 1, 0)$
- Vypočítejte  $\vec{v}'$  a  $\vec{n}'$ , aplikujte měřítko  $S_x = 2$  a  $S_y = 1$
- Transformační matice:

$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

# Transformace ve 3D

- Zobecnění 2D transformací.
- Body popsány homogenními 3D souřadnicemi  $P(x, y, z, w)$ , kde  $w = 1$  pro bod a  $w = 0$  pro vektor.
- Skládání transformací – násobením dílčích matic.

## Maticový zápis

$$[x', y', z', 1] = [x, y, z, 1] \cdot \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & 1 \end{bmatrix}$$

$$P' = P \cdot M$$

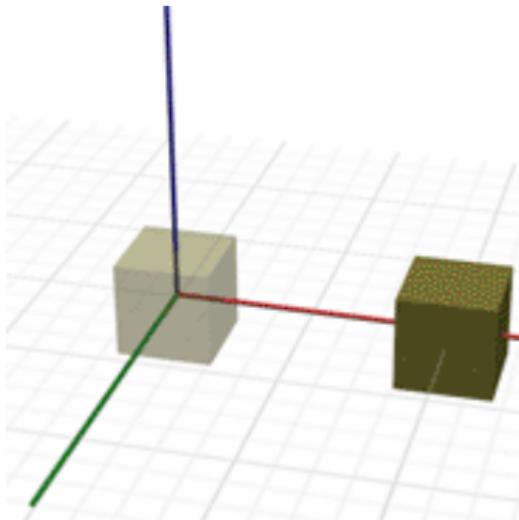
# Posunutí ve 3D

- Pouhé rozšíření dimenze 2D matice.

## Transformační matice

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ d_x & d_y & d_z & 1 \end{bmatrix}$$

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -d_x & -d_y & -d_z & 1 \end{bmatrix}$$



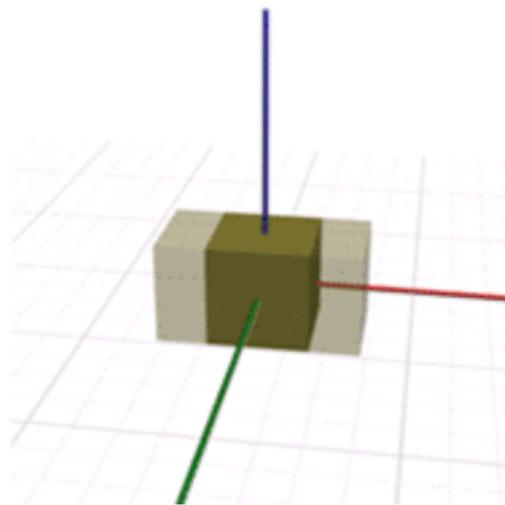
# Změna měřítka ve 3D

- Opět pouhé rozšíření dimenze 2D matice.

## Transformační matice

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$S^{-1} = \begin{bmatrix} 1/S_x & 0 & 0 & 0 \\ 0 & 1/S_y & 0 & 0 \\ 0 & 0 & 1/S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



# Zkosení ve 3D

- Transformace je opět rozdělena na tři různé operace, podle směrů ve kterých zkosení probíhá.
- Tři transformační matice  $S_{HYZ}$ ,  $S_{HXZ}$ ,  $S_{HXY}$  pro zkosení ve směrech  $YZ$ ,  $XZ$  a  $XY$ .

## Transformační matice

$$S_{HYZ} = \begin{bmatrix} 1 & S_{hy} & S_{hz} & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad S_{HXZ} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ S_{hx} & 1 & S_{hz} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

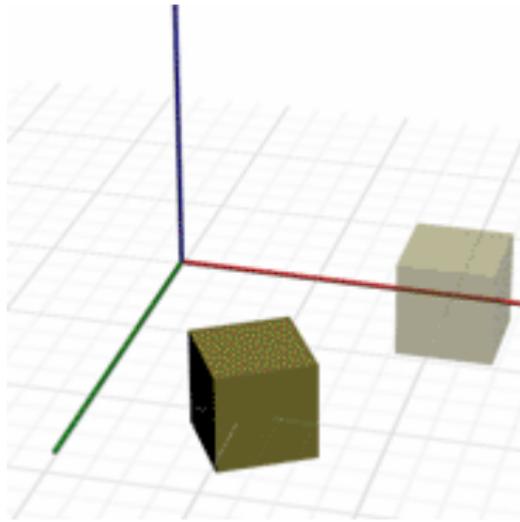
$$S_{HXY} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ S_{hx} & S_{hy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Rotace ve 3D

- Rotace kolem počátku souřadného systému.
- Různé transformační matice  $R_x$ ,  $R_y$ ,  $R_z$  pro rotaci okolo souřadných os  $X$ ,  $Y$  a  $Z$ .

## Příklad transformační matice

$$R_y = \begin{bmatrix} \cos \alpha & 0 & \sin \alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \alpha & 0 & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



# Rotace ve 3D, pokr.

## Transformační matice

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & \sin \alpha & 0 \\ 0 & -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_y = \begin{bmatrix} \cos \alpha & 0 & \sin \alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \alpha & 0 & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

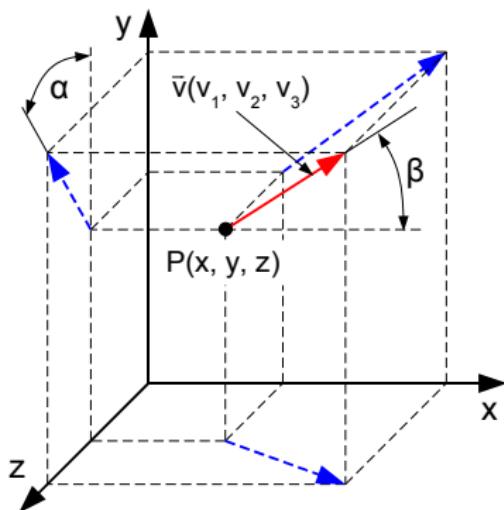
$$R_z = \begin{bmatrix} \cos \alpha & \sin \alpha & 0 & 0 \\ -\sin \alpha & \cos \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Rotace ve 3D kolem obecné osy

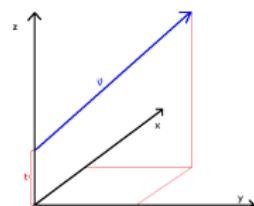
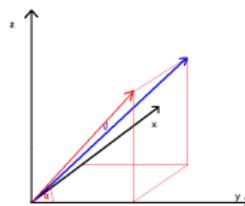
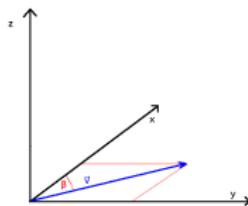
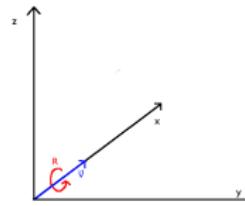
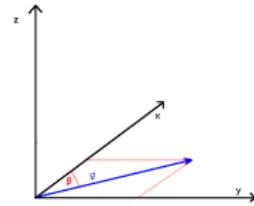
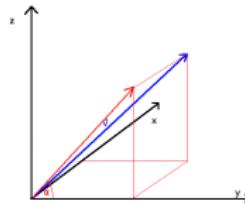
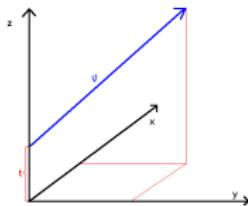
- Osa rotace je dána směrovým vektorem  $\bar{v}$  a bodem umístění  $P$ .
- Nelze přímo použít některou z variant základní 3D rotace.
- → je třeba rozložit na posloupnost několika transformací.

## Postup obecné rotace

- Posunutí osy do počátku souřadného systému.
- Otočení posunuté osy do jedné ze souřadných rovin (např. XY).
- Otočení sklopené osy do jedné ze souřadných os (X).
- Provedení požadované rotace o úhel  $\omega$  kolem příslušné osy (X).
- Vrácení osy do původní polohy.



# Rotace ve 3D kolem obecné osy, pokr.



# Rotace ve 3D kolem obecné osy, pokr.

## Maticový zápis

$$M = T \cdot R_X \cdot R_Z \cdot R_{X(\omega)} \cdot R_Z^{-1} \cdot R_X^{-1} \cdot T^{-1}$$

- $\alpha$  ... směrový kosinus průmětu vektoru osy do kolmé souřadné roviny (YZ).
- $\beta$  ... směrový kosinus mezi vektorem osy a osy do které sklápíme (XZ).

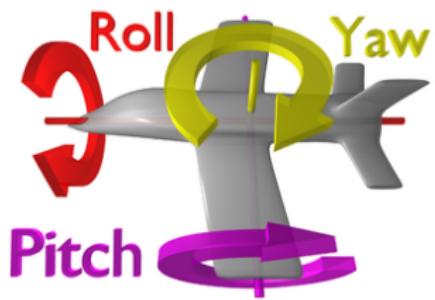
## Pozn.

- Rotaci kolem obecné osy procházející počátkem lze rozložit na dílčí rotace kolem os X, Y a Z - tzv. **Eulerovy úhly**

# Trable s Eulerovými úhly

## Eulerovy úhly

- Rotace podle tří základních os - yaw, pitch, roll
- ZXZ, XYX, ... (12 kombinací)

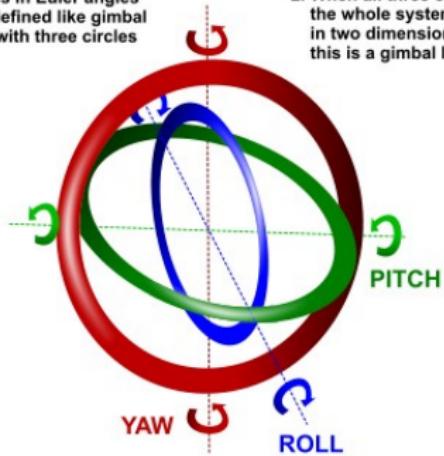


## Pozor!

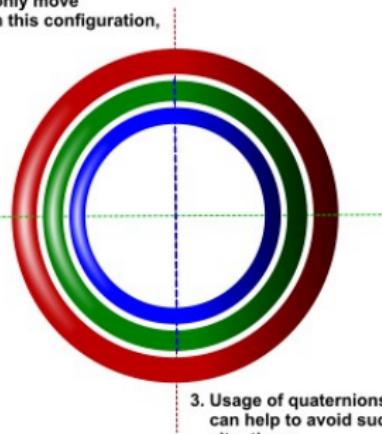
- Záleží na pořadí dílčích rotací.
- Spojitá rotace může způsobit skokovou změnu některých úhlů.
- Gimbal lock...

# Trable s Eulerovými úhly

1. Rotations in Euler angles can be defined like gimbal system with three circles



2. When all three circles are lined up, the whole system can only move in two dimensions from this configuration, this is a gimbal lock



3. Usage of quaternions can help to avoid such situations

## Gimbal lock

- Ztráta stupňů volnosti při nevhodném natočení "kruhů" (gimbals)

## Kvaterniony

- Robustnější způsob práce s rotačními transformacemi

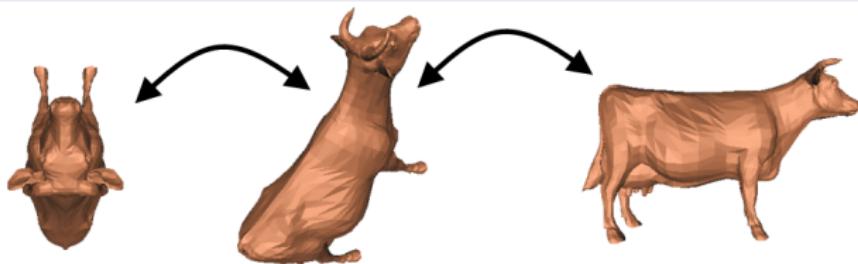
# Rotace pomocí transformačních matic

Nejsou vždy nejfektivnější

- Např. stačily by 3 úhly natočení...

Jsou obtížně interpolovatelné

- Jak realizovat postupný přechod z jedné polohy do druhé?



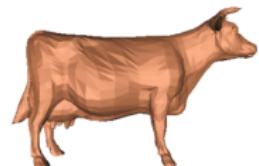
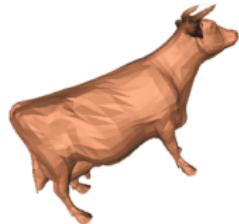
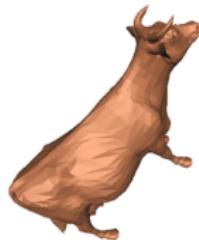
# Kvaterniony

- W. R. Hamilton v 19.století
- Efektivní způsob reprezentace rotace ve 3D.
- Kvantová mechanika, počítačová animace, atd.

## Reprezentace kvaternionu

$$q = [q_1, q_2, q_3, q_4]$$

# Příklad - Interpolace pomocí kvaternionů



# Kvaternion jako rozšíření komplexních čísel

- Kvaternion je lineární kombinací prvků  $1, i, j, k$ .
- $q = q_0 + iq_1 + jq_2 + kj_3$
- Jedna reálná, tři imaginární složky.

## Platí vztahy

$$\begin{aligned}i^2 &= j^2 = k^2 = ijk = -1 \\i &= jk = -kj \\j &= ki = -ik \\k &= ij = -ji\end{aligned}$$

# Komplexní čísla a rotace ve 2D

## Maticový zápis rotace ve 2D

$$[x', y'] = [x, y] \cdot \begin{bmatrix} \cos \alpha & \sin \alpha \\ -\sin \alpha & \cos \alpha \end{bmatrix}$$

## Zápis pomocí komplexních čísel

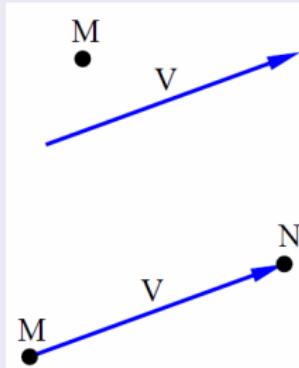
$$\begin{aligned}(x + yi)(\cos \alpha + \sin \alpha i) &= \\ &= (x \cos \alpha - y \sin \alpha) + (x \sin \alpha + y \cos \alpha)i\end{aligned}$$

- Násobením komplexních čísel lze realizovat rotaci.
- 2D rotaci odpovídá jednotkové komplexní číslo  $e^{i\alpha}$ !

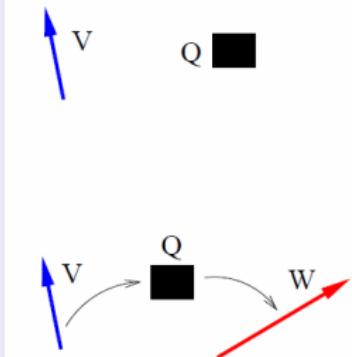
# Kvaternion jako skalár a vektor

- $q = \langle s, v \rangle$
- kde  $s = q_0$ ,  $v = [q_1, q_2, q_3]$

Vektor reprezentuje vztah  
mezi dvěma body



Kvaternion reprezentuje  
vztah mezi dvěma vektory



# Násobení kvaternionů $q_0 q_1$

$$q_0 = w_0 + x_0 i + y_0 j + z_0 k, \quad q_1 = w_1 + x_1 i + y_1 j + z_1 k$$

$$\begin{aligned} q_0 q_1 = & (w_0 w_1 - x_0 x_1 - y_0 y_1 - z_0 z_1) + \\ & (x_0 w_1 + w_0 x_1 + y_0 z_1 - z_0 y_1) i + \\ & (y_0 w_1 + w_0 y_1 + z_0 x_1 - x_0 z_1) j + \\ & (z_0 w_1 + w_0 z_1 + x_0 y_1 - y_0 x_1) k \end{aligned}$$

$$q_0 = \langle s_0, v_0 \rangle, \quad q_1 = \langle s_1, v_1 \rangle$$

$$q_0 q_1 = (s_0 s_1 - v_0 v_1, s_0 v_1 + s_1 v_0 + v_0 \times v_1)$$

## Knihovny

- Boost.Quaternions
- GLM
- A spousta dalších.

# Další vlastnosti kvaternionů

Kvaternion sdružený,  $q = w + xi + yj + zk$

$$q^* = w - xi - yj - zk$$

$$q^*q = qq^* = 1$$

Velikost kvaternionu

$$|q| = \sqrt{w^2 + x^2 + y^2 + z^2}$$

# 3D rotace pomocí kvaternionů

- Libovolnou rotaci lze popsat úhlem  $\alpha$  a jednotkovým vektorem  $a = (a_0, a_1, a_2)$ , který reprezentuje osu otáčení.

Takové rotaci odpovídá kvaternion

$$q = \cos(\alpha/2) + a_0 \sin(\alpha/2)i + a_1 \sin(\alpha/2)j + a_2 \sin(\alpha/2)k$$

Zkráceně

$$q = \cos(\alpha/2) + a \sin(\alpha/2)$$

# 3D rotace pomocí kvaternionů, pokr.

Otočení vektoru  $v = (v_0, v_1, v_2)$  jednotkovým kvaternionem  $q$  odpovídá

$$v' = qvq^*$$

Pozn.:

- Reálná část kvaternionu, který reprezentuje vektor  $v$ , je vždy nulová!
- $v = v_0i + v_1j + v_2k$

# Skládání rotací

- Složení rotací odpovídá násobení kvaternionů!

Rotace kvaternionem  $q$  a následně  $r$

$$v'' = rv'r^* = rqvq^*r^*$$

# Porovnání s transformačními maticemi

## Skládání transformací - rotační matice

- Operace násobení 27x
- Sčítání/odečítání 18x

## Operace rotace - rotační matice

- Násobení 9x
- Sčítání/odečítání 6x

## Skládání transformací - kvaterniony

- Násobení 16x
- Sčítání/odečítání 12x

## Operace rotace - kvaterniony

- Násobení 21x
- Sčítání/odečítání 18x

# Převod mezi kvaterniony a rotačními maticemi

- Nutné zejména z pohledu grafického HW, grafické karty pracují s maticemi!

Kvaternion  $q = w + xi + yj + zk \rightarrow$  rotační matice

$$R(q) = \begin{bmatrix} 1 - 2y^2 - 2z^2 & 2xy + 2wz & 2xz - 2wy & 0 \\ 2xy - 2wz & 1 - 2x^2 - 2z^2 & 2yz + 2wz & 0 \\ 2xz + 2wy & 2yz - 2wx & 1 - 2x^2 - 2y^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Další info

- Google, Wiki, apod.
- SLERP – Sférická lineární interpolace...
  - <https://nccastaff.bournemouth.ac.uk/jmacey/WebGL/QuatSlerp/>
- Arcball interface...
  - [http://www.math.tamu.edu/~romwell/arcball\\_js/index.html](http://www.math.tamu.edu/~romwell/arcball_js/index.html)

# Základy počítačové grafiky

## Křivky v počítačové grafice

Michal Španěl

Ústav počítačové grafiky a multimédií  
Fakulta informačních technologií  
Vysoké učení technické v Brně

Brno 2016

# Proč potřebujeme umět popsat křivé tvary?

Rovnost přímek a kulatost kružnic je ideál, který se ve skutečnosti příliš nevyskytuje!



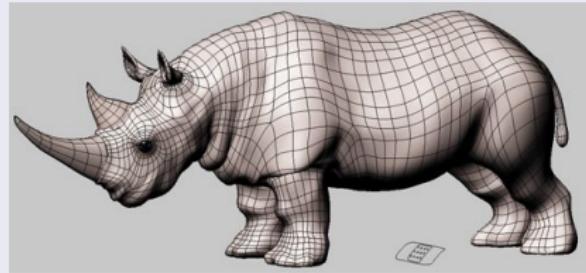
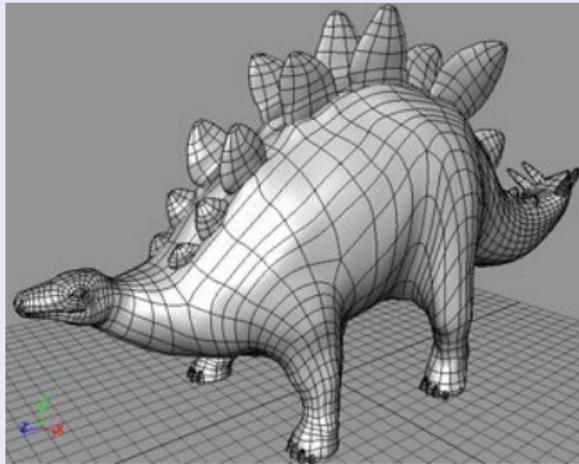
# Cíl přednášky

Seznámit se s jednotlivými typy křivek ve 2D, metodami jejich reprezentace a základními principy vykreslování.



# Použití křivek v počítačové grafice

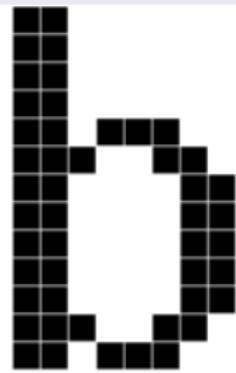
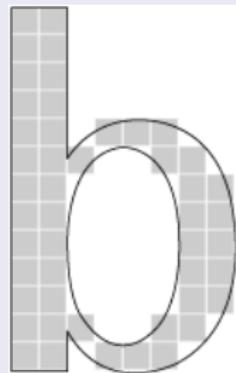
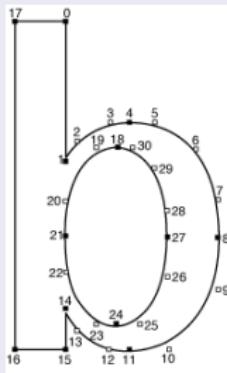
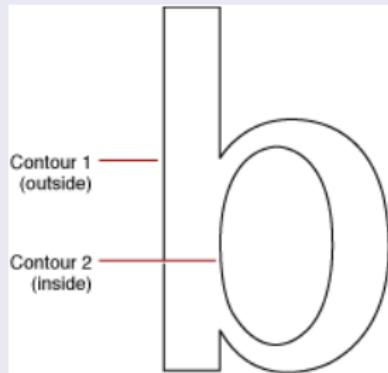
## Definice objektů (modelování)



[[www.3drender.com](http://www.3drender.com)]

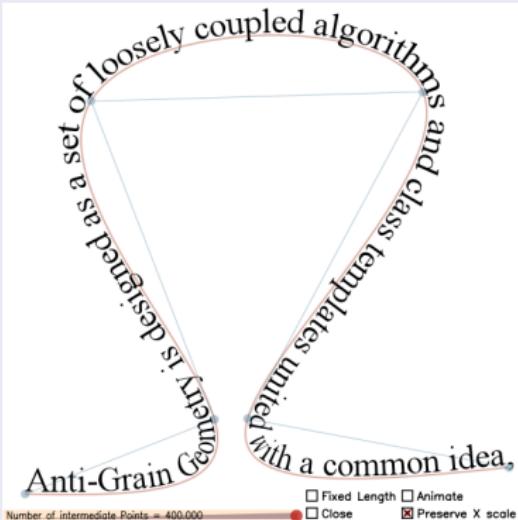
# Použití křivek v počítačové grafice, pokr.

## Definice fontů



# Použití křivek v počítačové grafice, pokr.

## Kreativní grafika

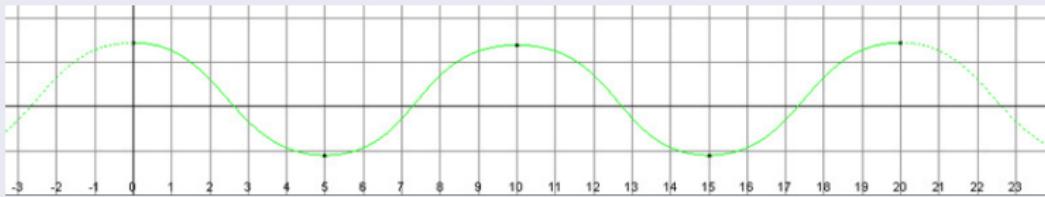
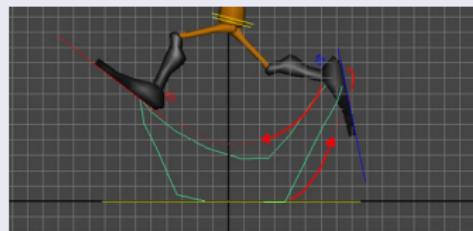
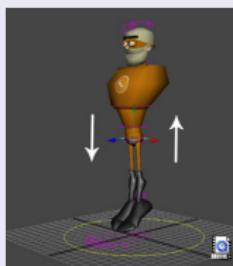


[[www.antigrain.com](http://www.antigrain.com)]

# Použití křivek v počítačové grafice, pokr.

## Určování dráhy objektů při animaci

- Run Cycle - *Rodri Torres* [<http://rodri.aniguild.com/>]



# Použití křivek v počítačové grafice, pokr.

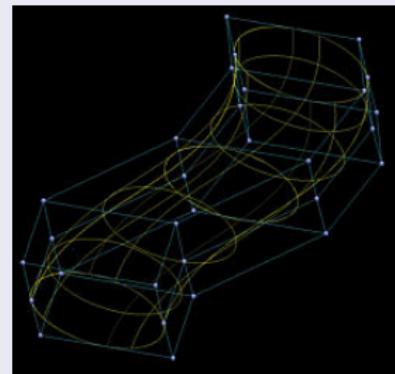
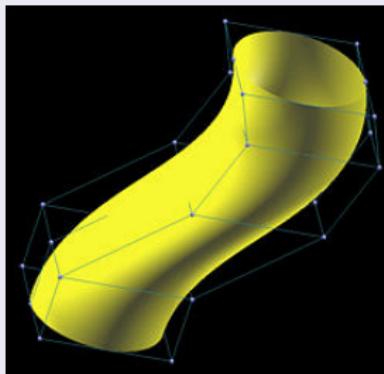
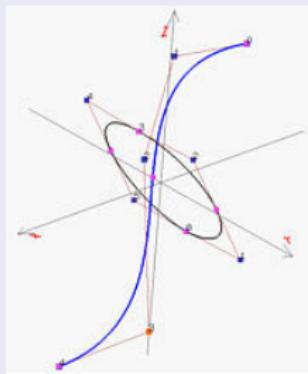
## Run Cycle - *Rodri Torres*



## Video ukázka

# Použití křivek v počítačové grafice, pokr.

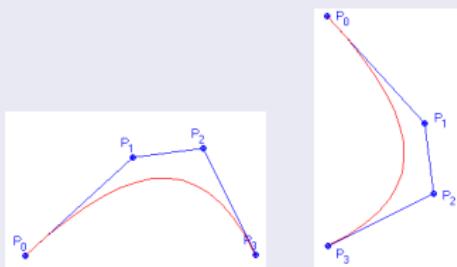
## Šablonování



# Požadováné vlastnosti křivek

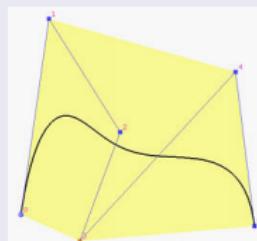
## Invariance k affiním transformacím a projekci

- Rotace řídících bodů nemá vliv na tvar křivky.



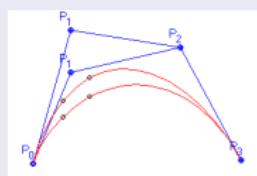
## Interpolace krajních bodů

- Křivka prochází krajními body.

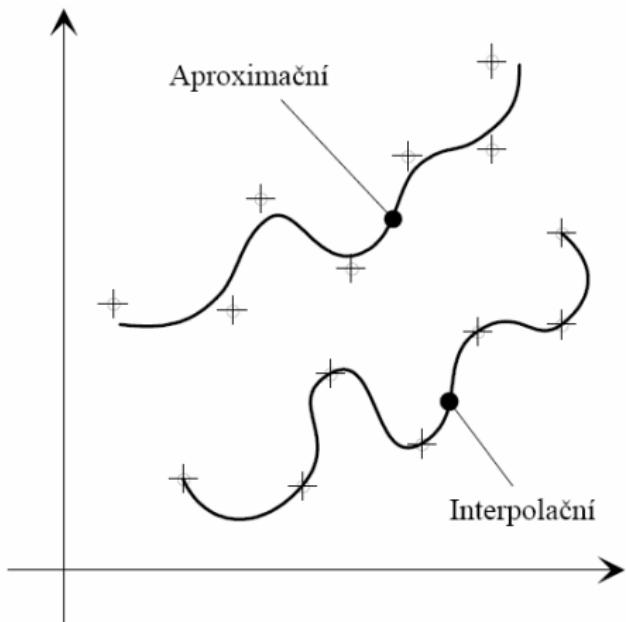


## Konvexní obálka

- Křivka leží v konvexní obálce svých řídících bodů.



# Základní druhy křivek



## Interpolační křivka

- Křivka přímo prochází body ("proložení" bodů křivkou).

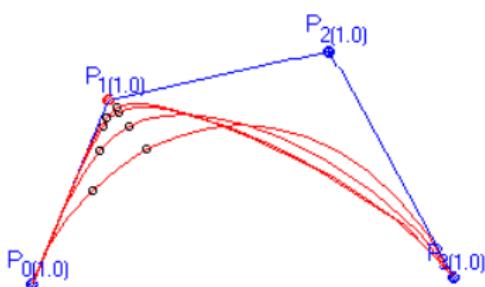
## Aproximační křivka

- Neprochází body.
- Tzv. řídící body).

# Základní druhy křivek, pokr.

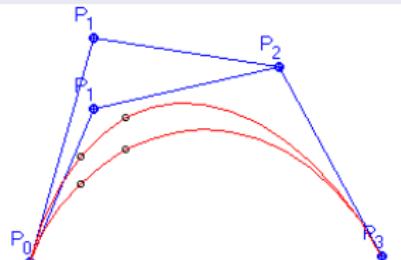
## Racionální křivka

- Váhové koeficienty  $w_i$  řídících bodů.
- Invariantní vůči perspektivní projekci.



## Neracionální křivka

- Tvar křivky ovlivňujeme pouze změnou polohy řídících bodů.
- Speciální případ racionální křivky, kdy  $w_i = 1$ .
- Nejsou invariantní vůči perspektivní projekci!



# Reprezentace křivek v počítačové grafice

Jak můžeme reprezentovat křivku?

# Parametrické vyjádření křivky

- Klasické matematické vyjádření,  $y = F(x)$ , není pro počítačovou grafiku vhodné!

## Parametrizovaná 2D křivka

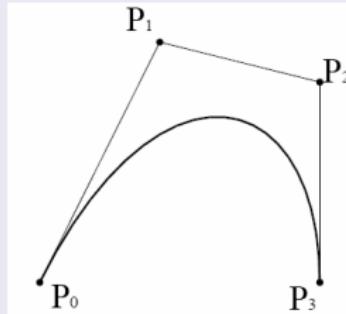
$$Q(t) = [x(t), y(t)]; \quad t \in \langle 0, 1 \rangle$$

## Polynomiální křivka

- Kubické polynomy

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$

$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$$



# Maticový zápis

- Základní tvar

$$\begin{aligned} x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x \\ y(t) &= a_y t^3 + b_y t^2 + c_y t + d_y \end{aligned} \longrightarrow Q(t) = T \cdot C = [t^3 \ t^2 \ t \ 1] \cdot \begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix}$$

- Konstantní matici C můžeme rozepsat do součinu  $C = MP$

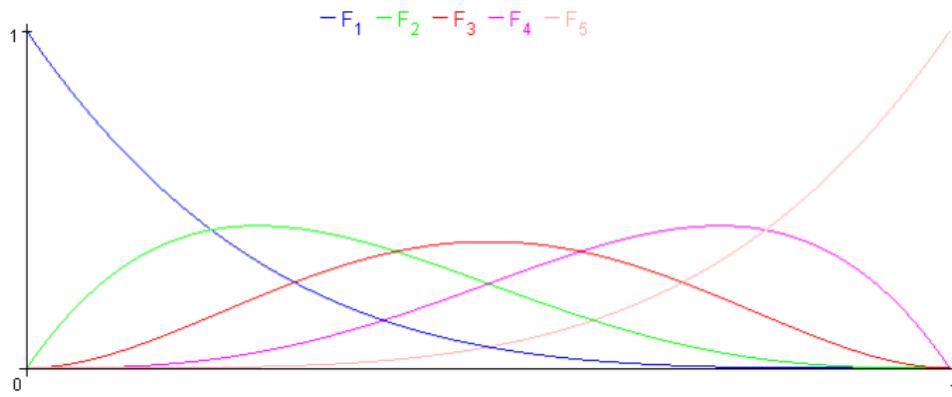
$$Q(t) = T \cdot MP = [t^3 \ t^2 \ t \ 1] \cdot \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \cdot \begin{bmatrix} x_0 & y_0 \\ x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \end{bmatrix}$$

- $M$  ... bázová matice (dána použitou metodou).
- $P$  ... geometrický vektor (řídící body, řídící vektory),  $P_i = [x_i, y_i]$ .

# Jiný zápis

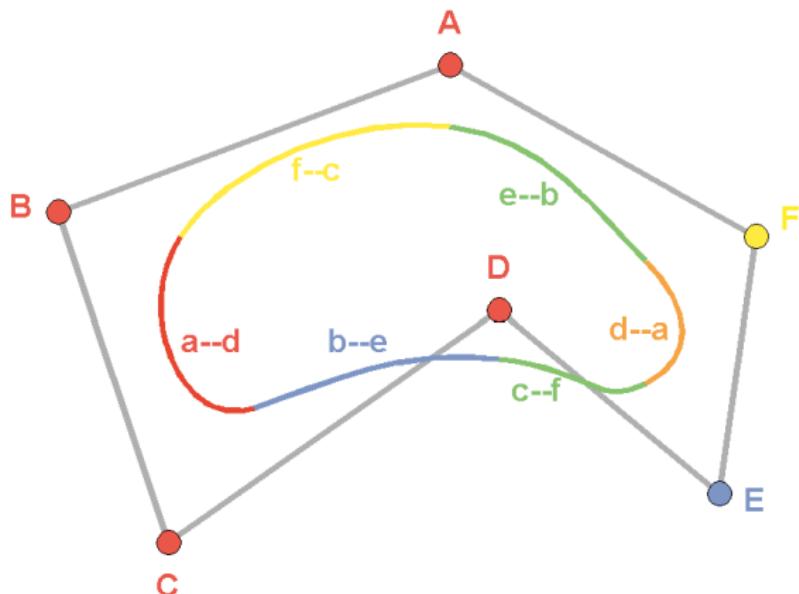
$$Q(t) = P_0 \cdot F_0(t) + P_1 \cdot F_1(t) + P_2 \cdot F_2(t) + P_3 \cdot F_3(t)$$

- $P$  ... řídící body,  $P_i = [x_i, y_i]$ .
- $F_i$  ... polynomiální funkce („váhuje příspěvky řídících bodů“).



# Spline křivka

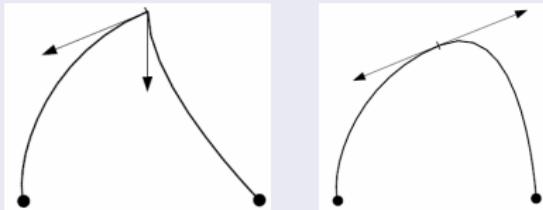
- Křivky spojované ze segmentů po částech polynomiální křivky.



# Spojitost křivek spojovaných ze segmentů

## Možnosti parametrické spojitosti $C^n$

- $C^0$  – totožnost navazujících koncových bodů.
- $C^1$  – totožnost tečných vektorů v navazujících bodech.
- $C^2$  – totožnost vektorů 2. derivace v navazujících bodech.

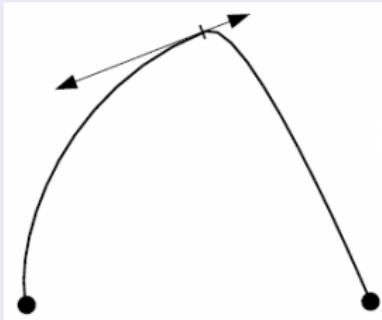


- Čím vyšší spojitost, tím déle se oba segmenty k sobě přimykají.

# Spojitost křivek, pokr.

## Oslabená podmínka spojitosti (též. *geometrická spojitost $G^n$* )

- $G^0$  – totožnost navazujících koncových bodů.
- $G^1$  – tečné vektory v navazujících bodech jsou lineárně závislé.
- $G^2$  – shoda první křivosti v navazujících bodech.

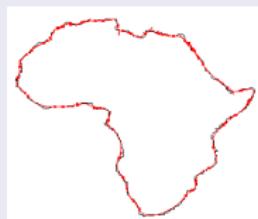
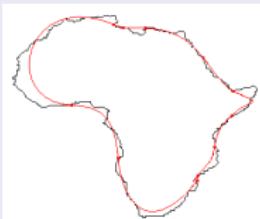


# Lepší definice spline křivky

- *Spline* – pružné křívítka, kovový pásek proložený body.
- Po částech polynomiální křivka.
- Spline křivka *stupně n* má *spojitost  $C^{n-1}$* .

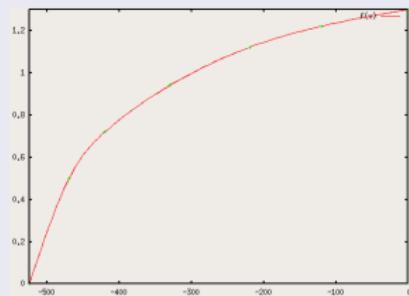
## Cíl použití spline křivek

- Minimalizovat křivost křivky (délku, energii).
- Efektivní řízení tvaru křivky (modelování).

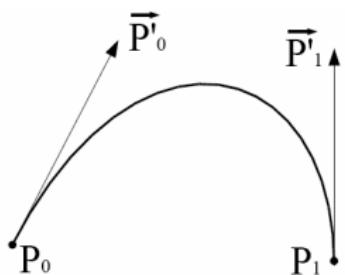


## Přirozený spline

- Spline, který interpoluje své řídící body.



# Fergusonova kubika



- Nejčastější interpolaci křivka.
- Určena dvěma koncovými body  $P_0, P_1$  (poloha) a dvěma tečnými vektory  $\vec{P}'_0, \vec{P}'_1$  (vyklenutí).

## Maticový zápis

$$Q(t) = T.M.P = [t^3 \ t^2 \ t^1] \cdot \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_0 \\ P_1 \\ \vec{P}'_0 \\ \vec{P}'_1 \end{bmatrix}$$

$$Q(t) = P_0.F_1(t) + P_1.F_2(t) + \vec{P}'_0.F_3(t) + \vec{P}'_1.F_4(t)$$

# Fergusonova kubika, pokr.

## Hermitovy polynomy

$$F_1(t) = 2t^3 - 3t^2 + 1$$

$$F_2(t) = -2t^3 + 3t^2$$

$$F_3(t) = t^3 - 2t^2 + t$$

$$F_4(t) = t^3 - t^2$$

+/-

- Neinteraktivní a neintuitivní řízení tvaru.
- Nelokální změna tvaru posunem jednoho bodu.

- Navazování *segmentů* – totožnost koncových bodů ( $C_0$ ) a shodnost tečných vektorů ( $C_1$ ).
- Mezi dvěma body jeden segment.
- *Přirozený spline* – maticové řešení celé křivky pro nulové koncové tečné vektory.

# Kochanek-Bartels spline

- Interpolační spline křivka.
- Pro interpolaci využívá Fergusonových kubik.
- Určeno *množinou N* interpolovaných bodů  $P_i$  a odpovídajícími koeficienty  $a_i$ ,  $b_i$  a  $c_i$ .
- Koeficienty určují chování křivky v daném bodě.

## Výpočet tečných vektorů

$$\vec{P'_i} = \frac{(1 - a_i)(1 + b_i)(1 + c_i)}{2}(P_i - P_{i-1}) + \frac{(1 - a_i)(1 - b_i)(1 - c_i)}{2}(P_{i+1} - P_i)$$
$$\vec{P'_{i+1}} = \frac{(1 - a_{i+1})(1 + b_{i+1})(1 - c_{i+1})}{2}(P_i - P_{i-1}) + \frac{(1 - a_{i+1})(1 - b_{i+1})(1 + c_{i+1})}{2}(P_{i+1} - P_i)$$

# Kochanek-Bartels spline, pokr.

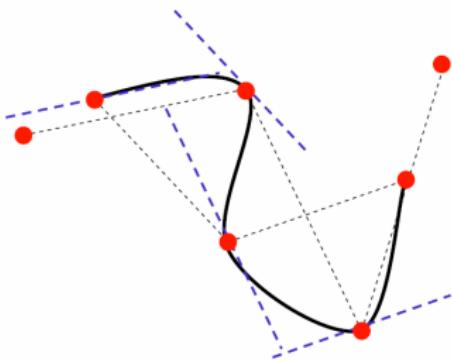
## Význam koeficientů

- $a_i$  ... tenze (jak ostře přiléhá křivka v bodě  $P_i$ ).
- $b_i$  ... šikmost křivky.
- $c_i$  ... spojitost křivky.

## Použití v animaci

- Definice dráhy pohybu objektů.
- Konstantní rychlosť bodu.

# Catmull-Rom spline



- Interpolační spline křivka určená množinou  $N$  bodů  $P_0, \dots, P_{N-1}$ .
- Tečný vektor v  $P_i$  je rovnoběžný s vektorem  $P_{i-1} - P_{i+1}$ .
- Kochanek-Bartels s nulovými koeficienty.

## Maticový zápis

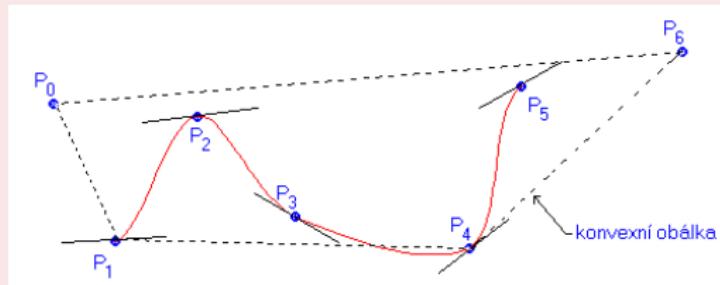
$$Q(t) = \frac{1}{2}[t^3 t^2 t^1] \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_{i-3} \\ P_{i-2} \\ P_{i-1} \\ P_i \end{bmatrix}$$

# Catmull-Rom spline, pokr.

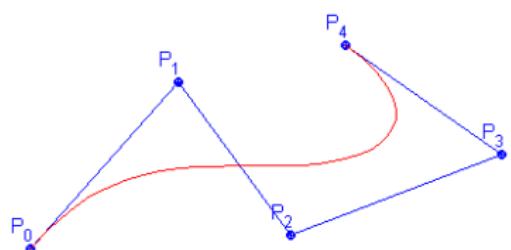
- Křivka vychází z bodu  $P_1$  a končí v bodu  $P_{N-2}$ .
- Interpolaci koncových bodů zajistí jejich opakování.

+/-

- Výsledný spline obecně neleží v konvexní obálce svých řídících bodů.



# Beziérový křivky



- Aproximační křivky (2D grafika, fonty, šablonování) 1960.
- Polynomiální křivka s použitím **Bernsteinových polynomů  $B_i^n$** .
- Křivka stupně  $n$  určena  $n + 1$  body.
- Prochází koncovými body.

## Definice křivky

$$Q(t) = \sum_{i=0}^n P_i \cdot B_i^n(t); \quad i = 0, 1, \dots, n$$

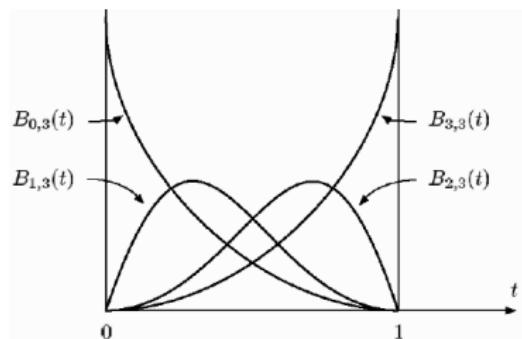
$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}; \quad t \in \langle 0, 1 \rangle$$

## Tečné vektory v koncových bodech

$$\vec{P'(0)} = 3(P_1 - P_0)$$

$$\vec{P'(1)} = 3(P_3 - P_2)$$

# Beziérový křivky, pokr.



## Rekurentní definice Bernsteinových polynomů

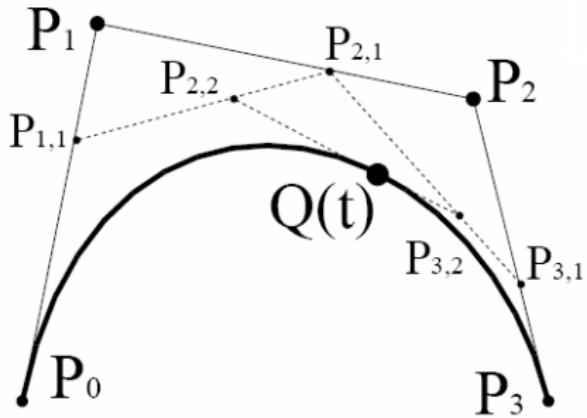
$$B_i^n(t) = (1-t) \cdot B_i^{n-1}(t) + t \cdot B_{i-1}^{n-1}(t)$$

- Využívá ji algoritmus *de Casteljau* pro vykreslení křivky (viz. dále).

## Další vlastnosti polynomů

- Mají nezápornou hodnotu.
- Mají jednotkový součet – křivka leží v konvexní obálce.
- Rekurentní definice.*

# Algoritmus de Casteljau



$$P_{i,j}(t) = (1-t) \cdot P_{j-1,i-1} + t \cdot P_{j,i-1}$$

$$P_{0,0} \downarrow$$

$$P_{0,1} \rightarrow P_{1,1} \downarrow$$

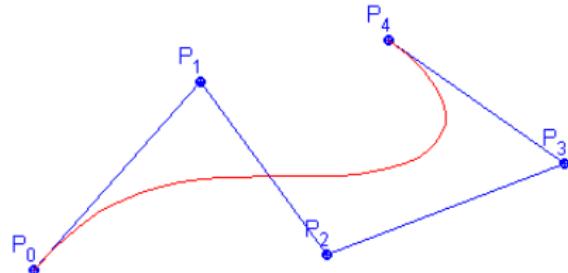
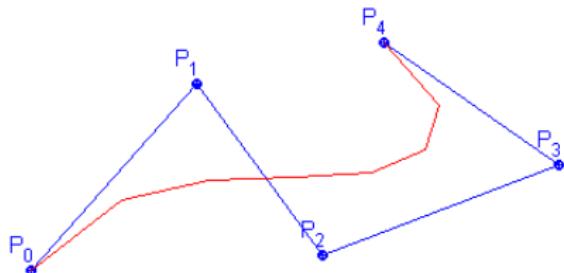
$$P_{0,2} \rightarrow P_{2,1} \rightarrow P_{2,2} \downarrow$$

$$P_{0,3} \rightarrow P_{3,1} \rightarrow P_{3,2} \rightarrow P_{3,3}$$

- Rekurzivní algoritmus vykreslování Beziérových křivek.
- Plyne z rekurentní definice pro Bernsteinovy polynomy.
- Úseky řídícího polynomu jsou děleny v poměru hodnot  $t$  a  $1 - t$ .

# Vykreslení křivky algoritmem de Casteljau

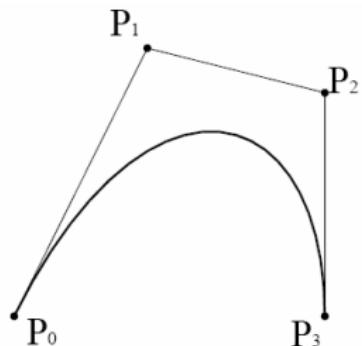
- Opakováný výpočet pro různé hodnoty  $t$  se zvoleným krokem.
- Spojování vypočtených bodů úsečkami.



+/-

- Jakou optimální velikost kroku zvolit?
- Rovnoměrně rozprostřené body (stejný počet na rovné i křivé části křivky)!

# Beziérový kubiky



- Segment popsán čtyřmi řídícími body  $P_0, P_1, P_2$  a  $P_3$ .
- Nelokální změna tvaru posunem jednoho bodu.
- Invariantní k lineárním transformacím.

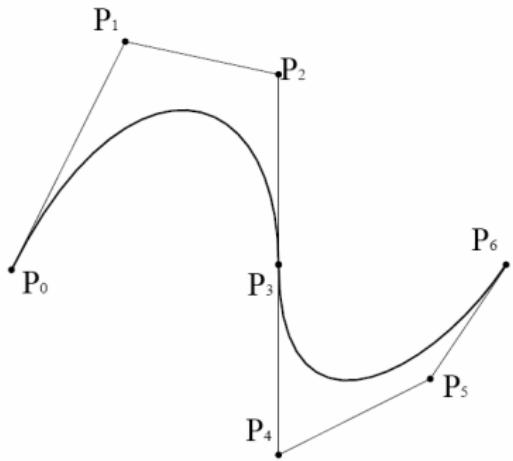
## Maticový zápis

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

## Zápis pomocí Bernsteinových polynomů stupně 3

$$\begin{aligned} P(t) &= P_0 B_0(t) + P_1 B_1(t) + P_2 B_2(t) + P_3 B_3(t) = \sum_{i=0}^3 P_i B_i(t) \\ B_0(t) &= (1-t)^3 \\ B_1(t) &= 3t(1-t)^2 \\ B_2(t) &= 3t^2(1-t) \\ B_3(t) &= t^3 \end{aligned}$$

# Navazování segmentů Beziérových kubik



## Tečné vektory v koncových bodech

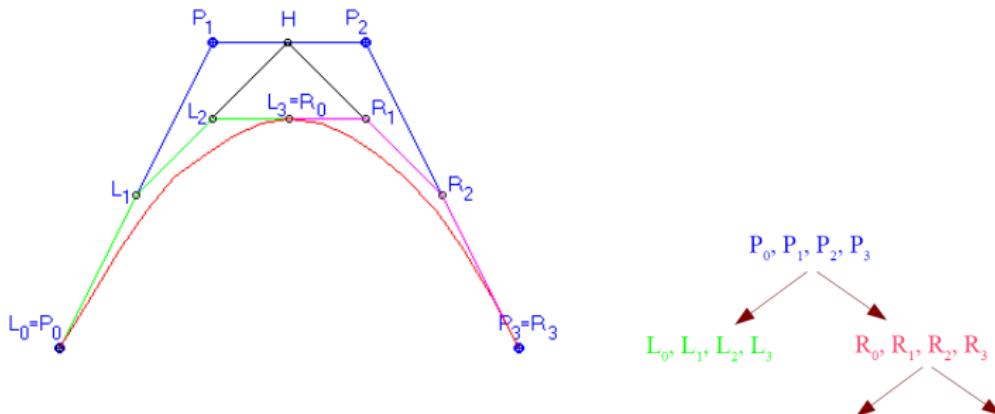
$$\vec{P'(0)} = 3(P_1 - P_0)$$

$$\vec{P'(1)} = 3(P_3 - P_2)$$

## Podmínky spojitosti $C^1$

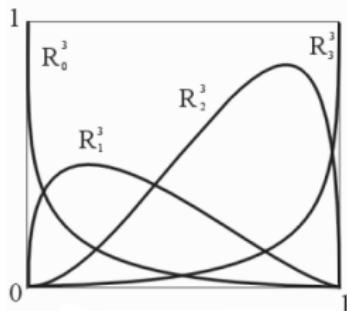
- Totožnost koncových bodů.
- Shodné tečné vektory – koncový bod úseku  $Q_i$  je středem úsečky předposledního bodu  $Q_i$  a druhého bodu  $Q_{i+1}$ .

# Algoritmus de Casteljau a Beziérový kubiky

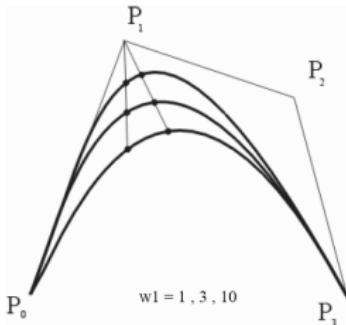


- "Divide and Conquer" – rekurzivní dělení na dvě podkřivky, volíme  $t = 0.5$ .
- Dostatečně rovná křivka není dále dělena, ale rovnou vykreslena.
- Minimální vzdálenost bodů na úrovni uhlopříčky pixelu.

# Racionální Beziérový křivky



- Místo neracionálních Bernsteinových polynomů  $B_i^n$  jsou použity *racionální polynomy*  $R_i^n$ .
- $w_i$  – váhový koeficient i-tého řídícího bodu.
- Pro  $w_i = 1$  jsou racionální polynomy  $R_i^n$  rovny neracionálním  $B_i^n$ .



## Definice křivky

$$Q(t) = \sum_{i=0}^n P_i \cdot R_i^n(t) = \frac{\sum_{i=0}^n w_i \cdot P_i \cdot B_i^n(t)}{\sum_{i=0}^n w_i \cdot B_i^n(t)}$$

$$R_i^n(t) = \frac{w_i \cdot B_i^n(t)}{\sum_{i=0}^n w_i \cdot B_i^n(t)}$$

# Racionální Beziérový křivky, pokr.

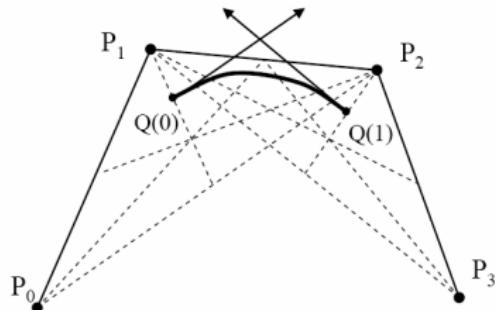
## Vlastnosti polynomů

- Mají nezápornou hodnotu.
- Mají jednotkový součet – křivka leží v konvexní obálce.
- *Nemají rekurentní definici!*

+/-

- Pro vykreslení nelze použít algoritmus *de Casteljau*!

# Coonsovy křivky (kubiky)



- Aproximační křivka (význam hlavně pro teorii spline křivek).
- Polynomiální křivka stupně  $n$ , určena  $n + 1$  řídícími body.
- Neprochází koncovými řídícími body.

## Maticový zápis

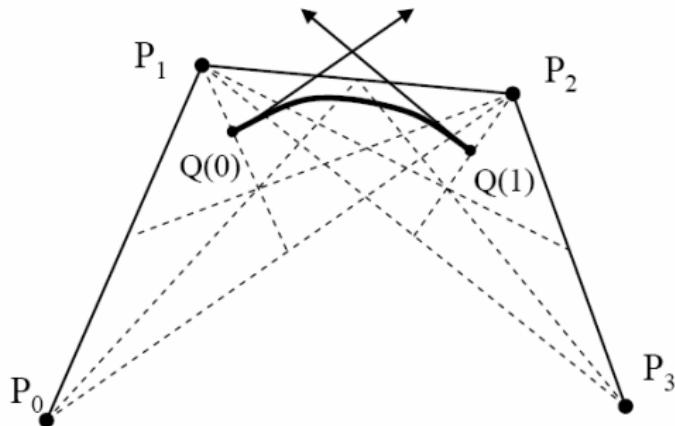
$$Q(t) = \frac{1}{6} \cdot T \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

## Tečné vektory

$$\vec{q'(0)} = \frac{\vec{P_2 - P_0}}{2}$$

$$\vec{q'(1)} = \frac{\vec{P_1 - P_3}}{2}$$

# Coonsovy křivky (kubiky), pokr.



## Koncové body

$$Q(0) = \frac{P_0 + 4 \cdot P_1 + P_2}{6}$$

$$Q(1) = \frac{P_1 + 4 \cdot P_2 + P_3}{6}$$

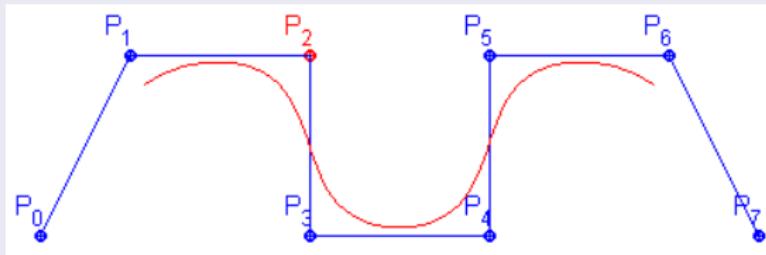
## Násobnost řídících bodů

- Jednonásobný → běžný bod  $P_n$ .
- Dvojnásobný →  $P_0 = P_1 \rightarrow Q(0)$  leží v  $1/6$  úsečky  $P_0P_1$ .
- Trojnásobný →  $P_0 = P_1 = P_2 \rightarrow Q(0)$  leží v  $P_0$ .

# Coonsovy křivky (kubiky), pokr.

## Navazování segmentů $C^2$

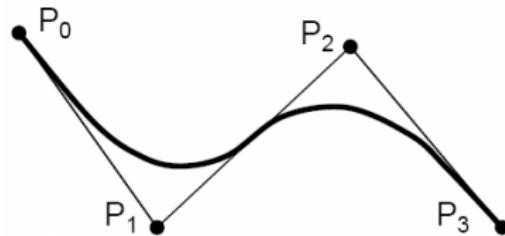
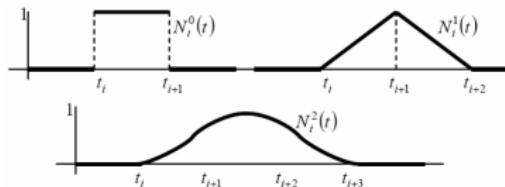
- Zopakováním posledních tří řídících bodů sousedního segmentu.



+/-

- S křivkou pracujeme po segmentech.
- Nelze přidat jeden bod, jedině celý segment.

# B-spline křivky

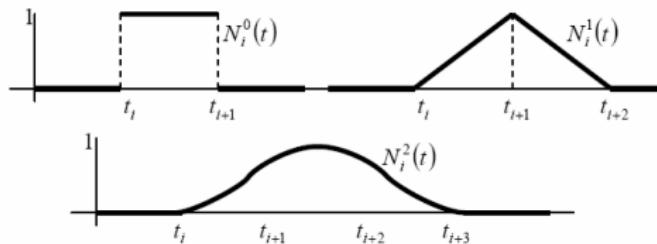


- Zobecnění Coonsových křivek.
- Křivka určena  $n + 1$  body.
- Křivka stupně  $k \rightarrow$  spojitost  $k + 1$ .
- Nejde o obyčejné navázání segmentů (nemusí platit  $t \in <0, 1>$ ).
- Uzlový vektor  $U_t$  – hodnoty parametru  $t$  v uzlech.

## Definice křivky

$$Q(t) = \sum_{i=0}^n P_i \cdot N_i^k(t); \quad t \in <0, t_m>$$

# Rekurentní definice polynomu B-spline křivky



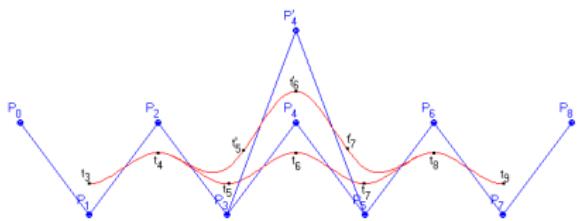
$$N_i^k(t) = \frac{(t - t_i)}{(t_{i+k} - t_i)} N_i^{k-1}(t) + \frac{(t_{i+k+1} - t)}{(t_{i+k+1} - t_{i+1})} N_{i+1}^{k-1}(t)$$

$$N_i^0(t) = 1, \text{ pro } t \in < t_i, t_{i+1} )$$

$$N_i^0(t) = 0, \text{ jinak}$$

- Polynomy mají nezápornou hodnotu.
- Jednotkový součet – křivka leží v konvexní obálce.
- Při dělení 0 je výsledek = 0.
- Vykreslování algoritmem *de Boor* (ala de Casteljau).

# Uzlový vektor $U_t$



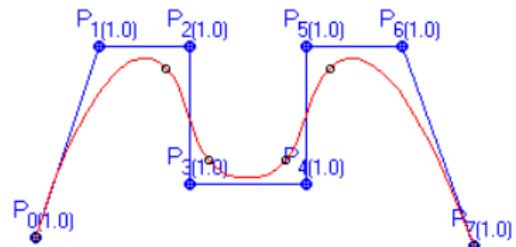
- Představuje hodnoty parametru  $t$  v uzlech.
- Délka  $m + 1$ , kde  $m = n + k + 1$ .
- Uniformní  $\rightarrow t_{i+1} - t_i = \text{konst.}$
- Násobné uzly (zdegenerování segmentu do jediného bodu).
- Lokální změna tvaru křivky.

## Příklad uzlového vektoru ( $k = 2$ )

$$U_t = (0, 0, 0, t_{k+1}, \dots, t_{m-k-1}, 1, 1, 1)$$

# NURBS (Non Uniform Rational B-Spline) křivky

- Zobecnění B-spline křivek.
- Racionální křivka → váhové koeficienty  $w_i$  i-tého řídícího bodu.
- Neuniformní uzlový vektor  $\rightarrow t_{i+1} - t_i \neq konst.$
- Umožňuje vkládání řídícího bodu při zachování tvaru!
- Přesné vyjádření kuželoseček apod.
- Invariantní vůči lineárním transformacím.



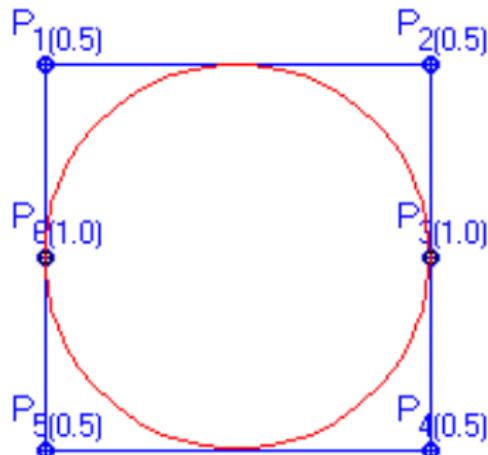
## Definice křivky

$$Q(t) = \sum_{i=0}^n P_i \cdot R_i^k(t) = \frac{\sum_{i=0}^n w_i \cdot P_i \cdot N_i^k(t)}{\sum_{i=0}^n w_i \cdot N_i^k(t)}$$

$$R_i^k(t) = \frac{w_i \cdot N_i^k(t)}{\sum_{i=0}^n w_i \cdot N_i^k(t)}$$

# Příklady NURBS křivek

## Definice kružnice



$$U_t = (0, 0, 0, 0.25, 0.5, 0.5, 0.5, 0.75, 1, 1, 1)$$

# Základy počítačové grafiky

## Ořezávání objektů ve 2D

Michal Španěl

Ústav počítačové grafiky a multimédií  
Fakulta informačních technologií  
Vysoké učení technické v Brně

Brno 2013

# Cíl přednášky

Zobrazovat to, co stejně není vidět je očividně zbytečné! Jak však poznat, co není vidět?

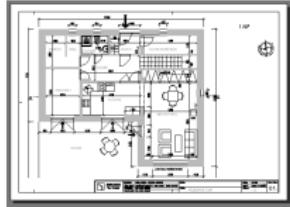
Seznámit se s algoritmy pro rychlé ořezávání vektorových primitiv (úsečka, polygon, atd.) ve 2D.

# Ořezávání objektů ve 2D

## Definice

Výběr zobrazovaných objektů vzhledem k aktuální poloze a rozměru *zobrazovacího okna*.

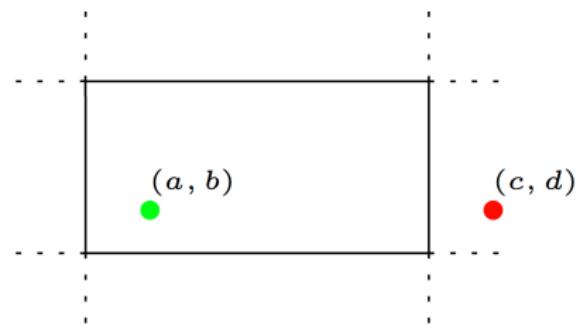
- Objekty mimo okno, budou vyloučeny.
- Objekty, které hranice okna protínají, budou rozdeleny (oříznuty).
- Výsledkem vždy objekt stejného typu (úsečka, polygon, atd.), jako původní.



# Základní přístupy

## Test polohy rasterizovaných bodů

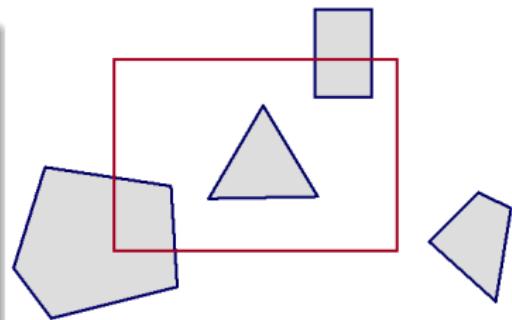
- Souřadnice pixelů získané rasterizací se porovnávají se zobrazovacím oknem a vykreslují se pouze vnitřní body.
- Nejprimitivnější způsob, neefektivní.
- Použití při HW implementaci.



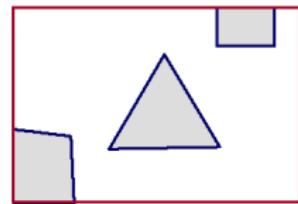
# Základní přístupy, pokr.

## Ořezávání vektorových entit

- Nejčastěji úsečky a polygony definované po částech lineární hranicí (mnohoúhelníky).
- Složitější vektorové entity převedeny na kombinaci úseček nebo mnohoúhelníků ...
- Zobrazovací okno je obecně definováno polygonem.
- Pro další výklad předpokládáme konvexní okno ve tvaru obdélníku se stranami rovnoběžnými se souřadnými osami.



Before Clipping

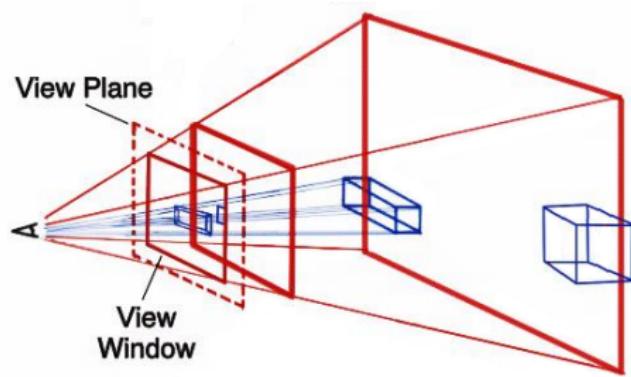


After Clipping

# Základní přístupy, pokr.

## Ořezání ve 2D

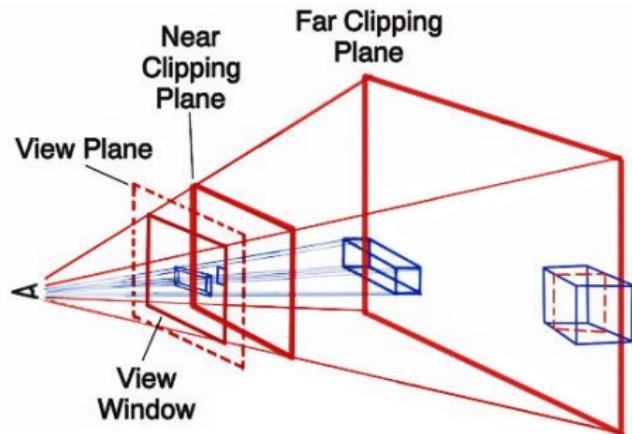
- 3D objekty jsou nejprve transformovány do roviny zobrazovacího okna a poté ořezány.
- Výrazně jednodušší a rychlejší algoritmy.
- Realizováno v HW grafické karty.



# Základní přístupy, pokr.

## Ořezání ve 3D

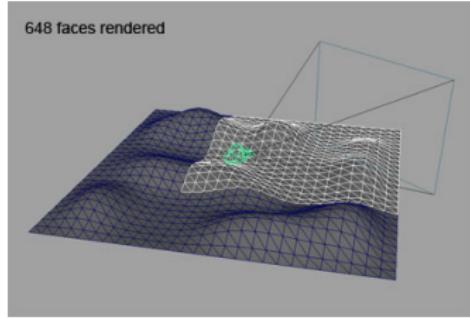
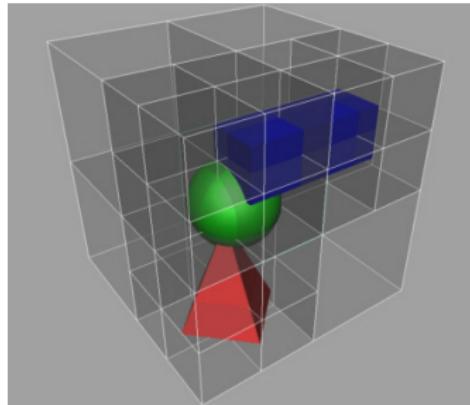
- Nejprve ořezání vzhledem k tzv. *pohledovému objemu* (angl. view frustum) přímo ve 3D, následně transformace do roviny pohledu a vykreslení.
- Výrazná redukce počtu entit přenášených do grafické karty.
- Náročnější implementace.



# Ořezávání vs. viditelnost objektů

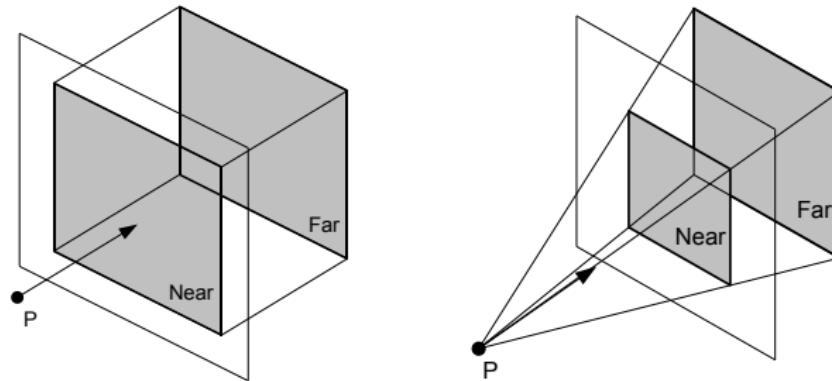
## Kombinace obou přístupů

- Ve 3D řešíme *viditelnost*. Entity rozdělíme na neviditelné a potencionálně viditelné (dělení prostoru scény, Octree, BSP stromy, ocludery, apod.).
- Ořezávání provádíme ve 2D.
- V praxi nejčastěji používaný přístup.



# Ořezání ve 3D

- *Pohledový objem* – oblast prostoru obsahující zobrazované objekty.
- Přední a zadní ořezávací stěna.
- *Hranol* u paralelní projekce (viz přednáška č. 9).
- *Komolý jehlan* u perspektivní projekce.

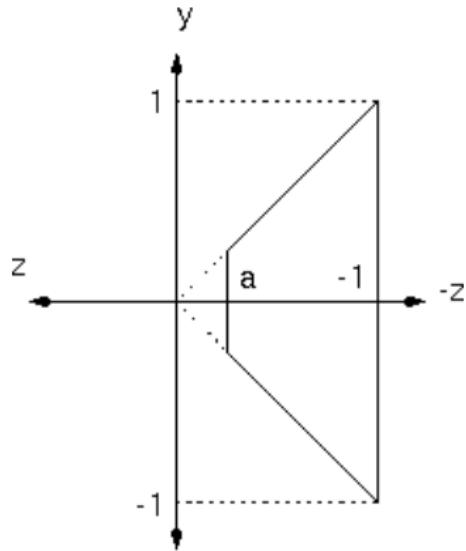


# Ořezání ve 3D, pokr

- Oříznutí objektů mimo hranice pohledového okna → *urychlení procesu zobrazení.*

## Optimalizace

- Transformace na jednotkovou krychli nebo jehlan pro snadnější ořezání (před projekcí).



# Ořezávání úsečky

- Nejjednodušší vektorová entita.
- Definovana souřadnicemi koncových bodů  $P_1 = (x_1, y_1)$  a  $P_2 = (x_2, y_2)$ .

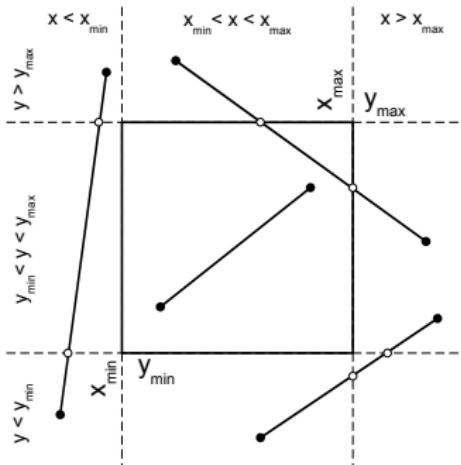
## Test polohy úsečky vůči oknu

- Úsečka leží zcela mimo okno → *vyloučit* ze zpracování.
- Celá úsečka uvnitř okna → *vykreslit*.
- Protíná-li úsečka hranice okna → *rozdělit* na hranicích okna na části a znova testovat.

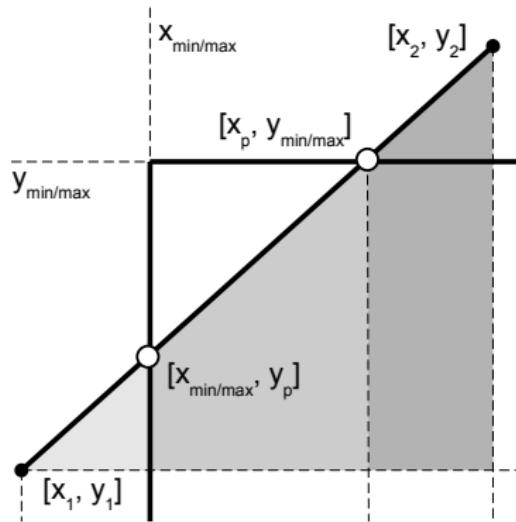
# Primitivní test polohy úsečky

Testování polohy koncových bodů:

- Oba koncové body úsečky uvnitř okna → celá úsečka uvnitř okna.
- Oba koncové body na vnější straně stejné hranice → úsečka leží vně okna.
- Zbývající případy → analyzovat průsečíky, rozdělit na části a opětovně testovat.



# Výpočet průsečíku úsečky s hranicemi okna



- Vztah pro výpočet průsečíku  $[x_p, y_p]$  odvodíme z podobnosti trojúhelníků.

$$\frac{x_2 - x_1}{y_2 - y_1} = \frac{x_p - x_1}{y_p - y_1}$$

# Výpočet průsečíku úsečky s hranicemi okna, pokr.

- Dosadíme polohy hranice okna  $x_p = x_{min/max}$  pro svislé nebo  $y_p = y_{min/max}$  pro vodorovné hranice okna.

$$x_p = x_{min/max}$$

$$y_p = \frac{(x_{min/max} - x_1)(y_2 - y_1)}{x_2 - x_1} + y_1$$

$$y_p = y_{min/max}$$

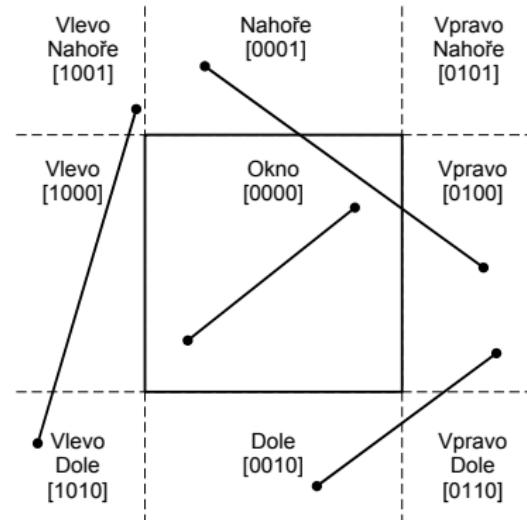
$$x_p = \frac{(x_2 - x_1)(y_{min/max} - y_1)}{y_2 - y_1} + x_1$$

# Algoritmus Cohen-Sutherland

- Rozšíření primitivního testování polohy úsečky vůči oknu.
- Koncové body  $P_1, P_2$  označeny binárními kódy  $C_1, C_2$  oblastí.
- Devět kódů oblastí  $\rightarrow 4\text{-bity}$ .

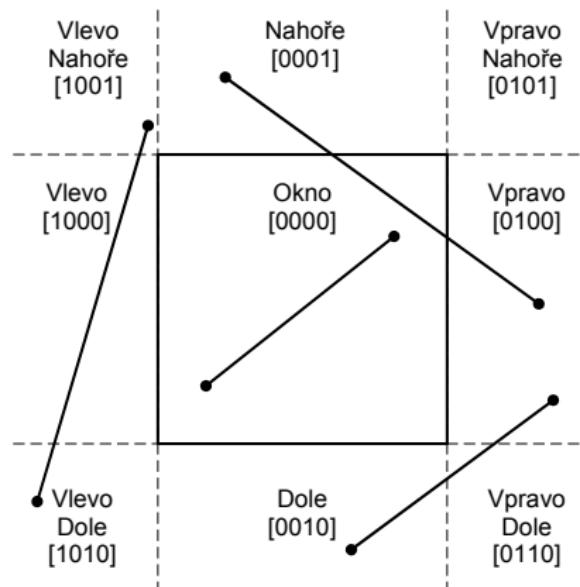
## Rozhodování

- $C_1 = C_2 = 0 \rightarrow$  celá úsečka uvnitř okna.
- $C_1 \text{ and } C_2 \neq 0 \rightarrow$  celá úsečka mimo okno.
- $C_1 \text{ and } C_2 = 0 \rightarrow$  úsečka potenciálně protíná hranice.



# Úsečka potencionálně protíná hranice

- Určit průsečíky, rozdělit úsečku na části a znova testovat polohu.
- Nastavené bity v hodnotě binárního výrazu  $C_1$  or  $C_2$  odpovídají zasaženým hranicím okna.



# Liang-Barsky

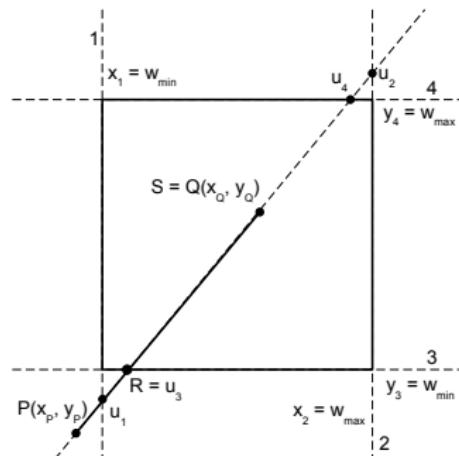
- Řešení průsečíku úsečky s oknem v *parametrické oblasti*.
- Úsečka dána body  $P[x_P, y_P]$ ,  $Q[x_Q, y_Q]$  a rovnicemi:

$$x = x_P + (x_Q - x_P) \cdot u = x_P + \Delta x \cdot u$$

$$y = y_P + (y_Q - y_P) \cdot u = y_P + \Delta y \cdot u, \quad 0 \leq u \leq 1$$

## Základní myšlenka

- Hledáme část úsečky ležící uvnitř okna, nebo-li hodnoty parametru  $u$ .



# Ořezání úsečky v parametrickém prostoru

- Uvnitř okna platí nerovnice

$$x_{w_{min}} \leq x_P + \Delta x \cdot u \leq x_{w_{max}}$$

$$y_{w_{min}} \leq y_P + \Delta y \cdot u \leq y_{w_{max}}$$

- Přepíšeme do tvaru

$$u \leq \frac{q_k}{p_k}, \quad k = 1, 2, 3, 4$$

$$p_1 = -\Delta x, \quad q_1 = x_P - x_{w_{min}}$$

$$p_2 = \Delta x, \quad q_2 = x_{w_{max}} - x_P$$

$$p_3 = -\Delta y, \quad q_3 = y_P - y_{w_{min}}$$

$$p_4 = \Delta y, \quad q_4 = y_{w_{max}} - y_P$$

# Pravidla určování polohy úsečky vůči oknu

$$p_k = 0$$

- Úsečka rovnoběžná s hranicí  $k$ .

$$p_k = 0 \text{ a } q_k < 0$$

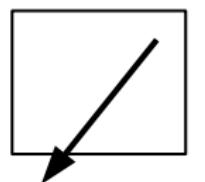
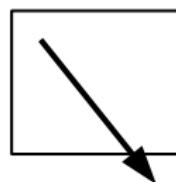
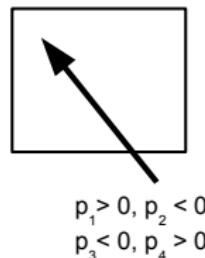
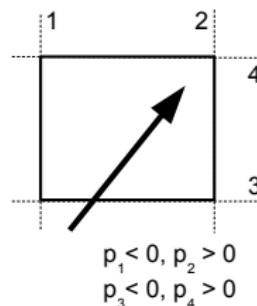
- Úsečka zcela mimo okno.

$$p_k < 0$$

- Přímka "vniká" do okna od/do bodu  $P$  přes hranici  $k$ .

$$p_k > 0$$

- Přímka "opouští" okno od/do bodu  $Q$ .



# Ořezání úsečky v parametrickém prostoru, pokr.

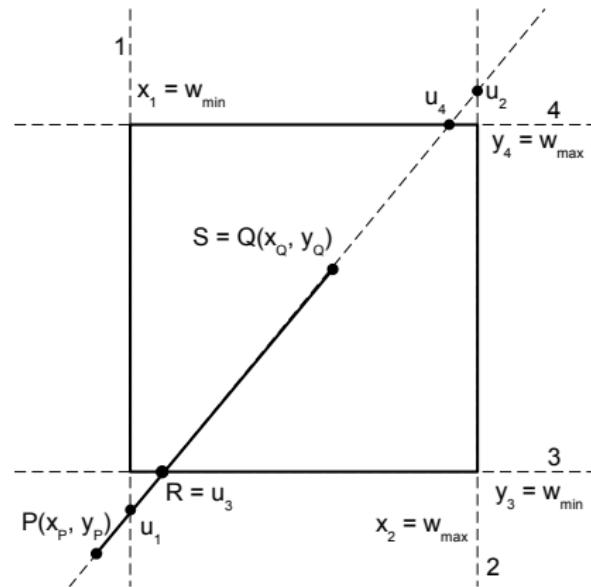
- Uvnitř okna musí platit  $0 \leq u_{R,S} \leq 1$ .

## Nalezení parametrů $u_R$ a $u_S$

$$u_R = \max ( u_k (p_k < 0), 0 )$$

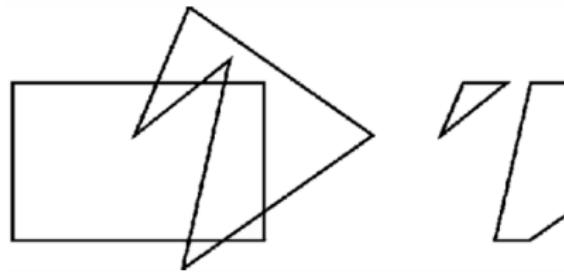
$$u_S = \min ( u_k (p_k > 0), 1 )$$

$$u_k = \frac{q_k}{p_k}, \quad k = 1, 2, 3, 4$$



# Ořezávání oblastí

- Ořezávají se uzavřené oblasti, polygony (mnohoúhelníky s po částech lineární hranicí = hranice z úseček).
- Musí se zachovat uzavřené oblasti → **nelze pouze ořezat hranice oblasti.**

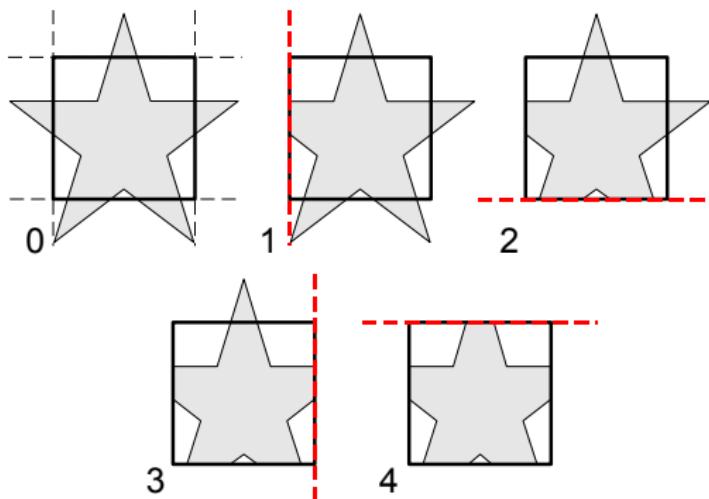


# Sutherland-Hodgman

- Ořezání polygonů obdélníkovým oknem se stranami rovnoběžnými se osami X a Y.
- Jednoduchý a snadno implementovatelný algoritmus.

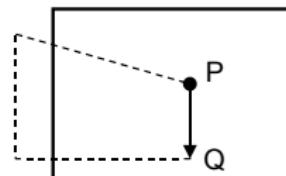
## Princip

- Ořezání celým oknem zjednodušeno na ořezání pouze *jednou stranou*.
- Po oříznutí jednou stranou provedeme oříznutí druhou, třetí a čtvrtou stranou.

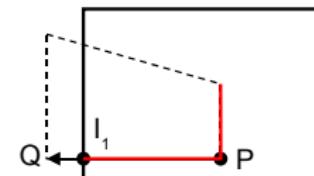


# Postup ořezání polygonu jednou stranou okna

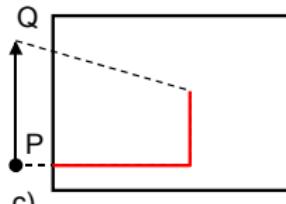
- Seřazení vrcholů → orientovaný seznam.
- Postupně pro každou hranu
  - Otestuj polohu úsečky (koncové body).
  - Je-li celá uvnitř → ulož do výsledného seznamu oba vrcholy.
  - Protíná stranu → vypočítej průsečík a ulož do seznamu.
  - Vnější hrany zahazujeme.



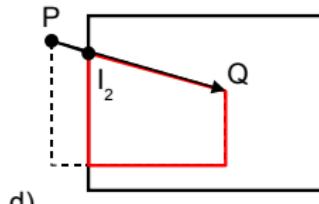
a)



b)



c)



d)

# Častý způsob implementace

- Ořezáváme přímo jednou souřadnou osou.
- Postupně otáčíme a posouváme ořezávaný polygon.

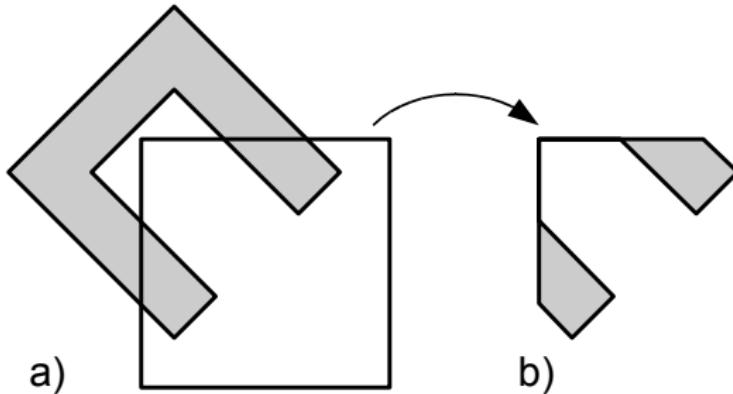
## Otočení polygonu o $90^\circ$

- Prohození souřadnic:  $x' = y$ ,  
 $y' = -x$ .

# Sutherland-Hodgman, pokr.

+/-

- Generování hran ořezaného polygonu podél hranic a rohů okna.
- Komplikovaná realizace polygonů s vnitřními otvory.
- Ořezávaný polygon může být definován pouze jednou uzavřenou smyčkou hraničních úseček.

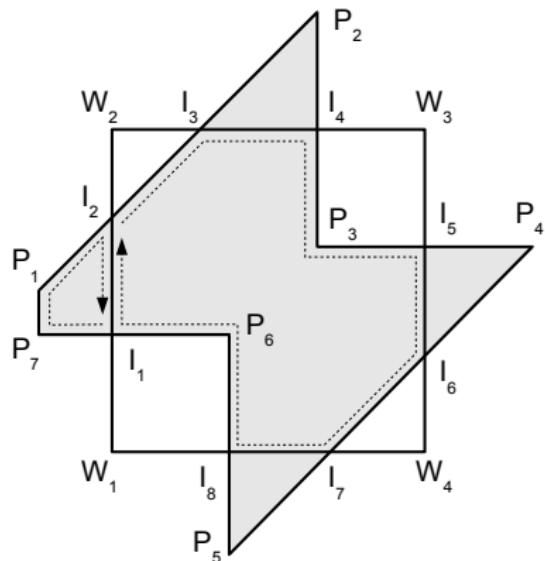


# Weiler-Atherton

- Ořezání obecných polygonů i s vnitřními otvory (vnitřní smyčky mají opačnou orientaci) obecným oknem definovaným polygonem.

## Základní myšlenka

- Výpočet průsečíků s ořezávacím oknem (alg. Liang-Barsky).
- Pomocné orientované seznamy z vrcholů a průsečíků.
- Vhodný průchod seznamy a nalezení vnitřních nebo vnějších částí polygonu.



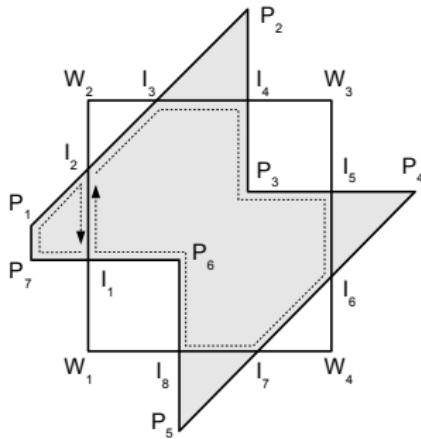
# Pomocné seznamy

- P – seznam vrcholů a průsečíků po ořezávaném polygonu.
- W – seznam vrcholů a průsečíků po ořezávacím okně.
- I – seznam průsečíků na vstupu do okna.
- O – seznam průsečíků na výstupu z okna.
- C – seznam ořezaných polygonů uvnitř okna.
- R – seznam odřezků vně okna.

# Příklad

## Příprava seznamů

- P –  $P_1, I_2, I_3, P_2, I_4, P_3, I_5, P_4, I_6, I_7, P_5, I_8, P_6, I_1, P_7, P_1$
- W –  $W_1, I_1, I_2, W_2, I_3, I_4, W_3, I_5, I_6, W_4, I_7, I_8, W_1$
- I –  $I_2, I_4, I_6, I_8$
- O –  $I_1, I_3, I_5, I_7$



# Postup ořezání polygonu

## Polygony uvnitř okna, ukládáme do C

- Start v P na prvním vrcholu z I.
- Jsme-li v P nebo W na vrcholu z I, pokračujeme po P.
- Jsme-li v P nebo W na vrcholu z O, pokračujeme po W.
- Pokračujeme do uzavření obrazce.

## Odřezky vně okna, ukládáme do R

- Start v P na prvním vrcholu z O.
- Jsme-li v P nebo W na vrcholu z O, pokračujeme po P.
- Jsme-li v P nebo W na vrcholu z I, pokračujeme po W.
- Pokračujeme do uzavření obrazce.

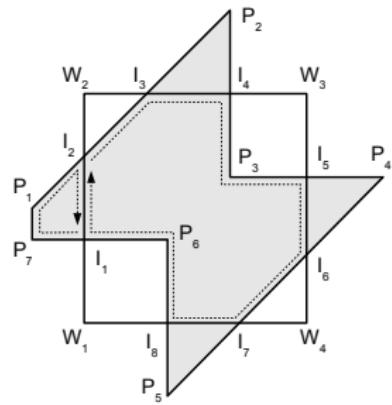
# Příklad, pokr.

## Orientované seznamy

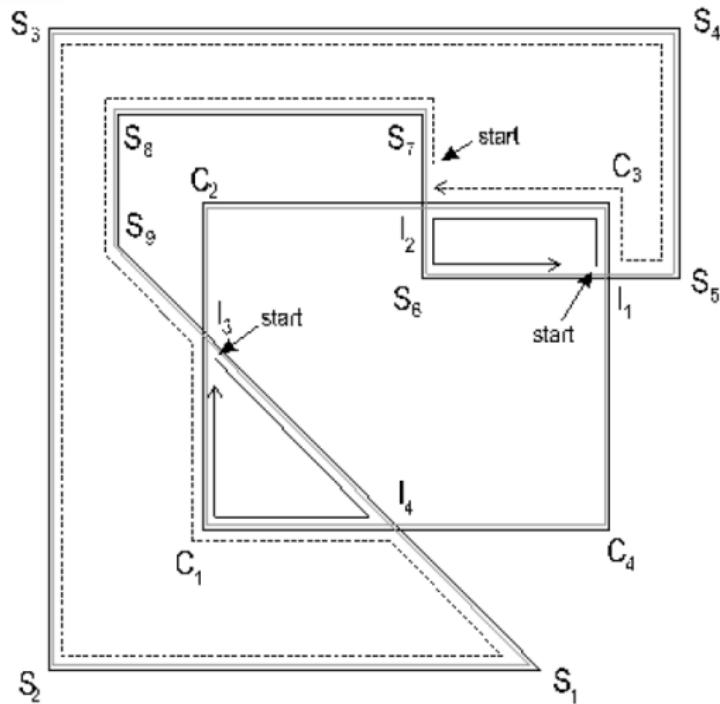
- P –  $P_1, I_2, I_3, P_2, I_4, P_3, I_5, P_4, I_6, I_7, P_5, I_8, P_6, I_1, P_7, P_1$
- W –  $W_1, I_1, I_2, W_2, I_3, I_4, W_3, I_5, I_6, W_4, I_7, I_8, W_1$
- I –  $I_2, I_4, I_6, I_8$
- O –  $I_1, I_3, I_5, I_7$

## Výsledek

- C –  $I_2, I_3, I_4, P_3, I_5, I_6, I_7, I_8, P_6, I_1, I_2$
- $R_1$  –  $I_1, P_7, P_1, I_2, I_1$
- $R_2$  –  $I_3, P_2, I_4, I_3$
- $R_3$  –  $I_5, P_4, I_6, I_5$
- $R_4$  –  $I_7, P_5, I_8, I_7$



## Příklad 2



# Základy počítačové grafiky

## Základní principy 2D grafických API

Michal Španěl

Ústav počítačové grafiky a multimédií  
Fakulta informačních technologií  
Vysoké učení technické v Brně

Brno 2014

# Cíl přednášky

Seznámit se s principy 2D grafických API a základy tvorby 2D grafických aplikací.

Jak využít znalosti jednotlivých principů 2D grafiky a ze střípků poskládat aplikaci?

# Knihovny pro 2D grafiku

- Knihovny výrobců OS pro různé platformy (Windows, Mac OS X, Android, Windows Phone, atd.)
- Multiplatformní knihovny, které poskytují další abstrakci...
- Knihovny pro webové aplikace (Javascript, atd.)

## Příklady

- Cairo (GTK+)
- SDL (znáte z projektu...)
- Skia (Google, použito v Chrome)
- Direct 2D (Windows)
- Quartz 2D (Mac)
- Java 2D

# Cairo (<http://cairographics.org/>)

- Multiplatformní open source knihovna (Windows, Mac OS X, Android, Windows Phone, ...)
- Používá se v GTK+ a Gecko (renderovací jádro pro webové prohlížeče, např. Mozilla)
- Podporuje výstup do obrázku, PDF i SVG
- Pouze kreslení, neřeší interakci s uživatelem...
- Lze snadno propojit s SDL knihovnou...



# Příklad použití Cairo

```
cairo_set_source_rgb (cr, 0, 0, 0);
cairo_move_to (cr, 0, 0);
cairo_line_to (cr, 1, 1);
cairo_move_to (cr, 1, 0);
cairo_line_to (cr, 0, 1);
cairo_set_line_width (cr, 0.2);
cairo_stroke (cr);

cairo_rectangle (cr, 0, 0, 0.5, 0.5);
cairo_set_source_rgba (cr, 1, 0, 0, 0.80);
cairo_fill (cr);

cairo_rectangle (cr, 0, 0.5, 0.5, 0.5);
cairo_set_source_rgba (cr, 0, 1, 0, 0.60);
cairo_fill (cr);

cairo_rectangle (cr, 0.5, 0, 0.5, 0.5);
cairo_set_source_rgba (cr, 0, 0, 1, 0.40);
cairo_fill (cr);
```



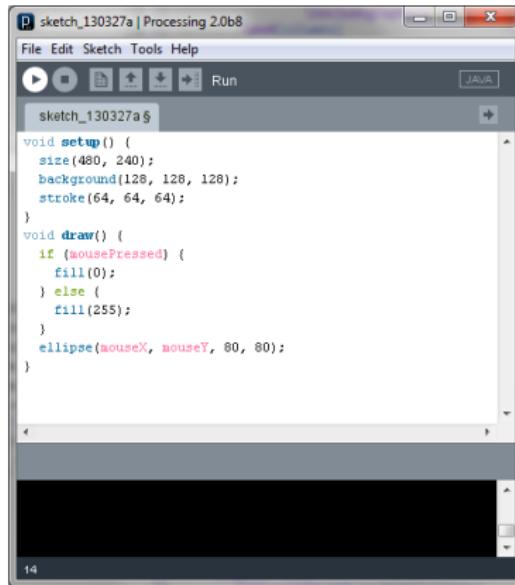
# Processing (<http://www.processing.org/>)

- Jednoduchý programovací jazyk a prostředí (open source, Java)
- 2D a 3D grafika, interaktivní aplikace, animace,...
- Rychlé prototypování řešení
- Aplikaci lze „exportovat“ pro spuštění na Windows, Linux, apod.
- Ale i do Javascriptu pro použití na webu
- **Budeme používat v přednášce pro ilustrační příklady**



# Processing - základní prostředí

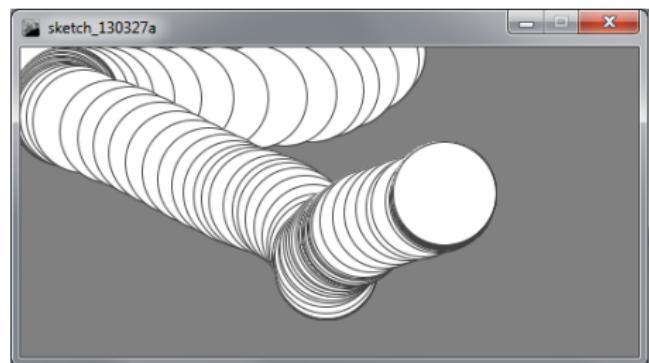
- Stahujte z <http://www.processing.org/download/>
- Rozbalte ZIP archiv
- A spusťte *processing.exe*



The screenshot shows the Processing 2.0b8 IDE interface. The title bar says "sketch\_130327a | Processing 2.0b8". The menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with various icons. The main area contains the following Java-like pseudocode:

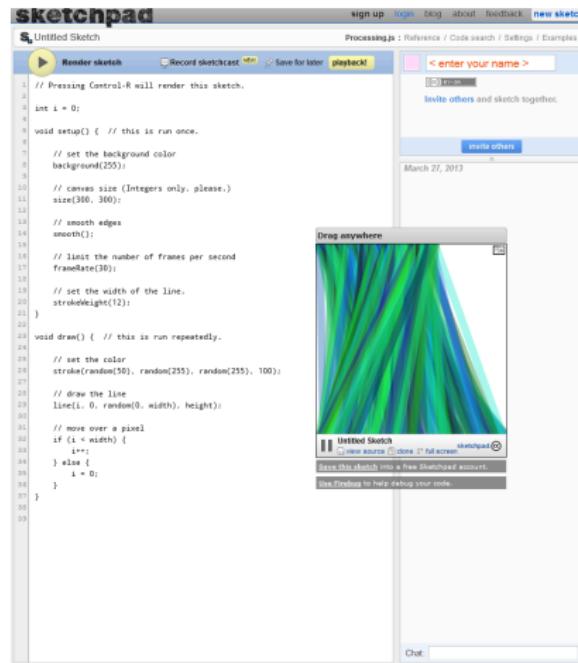
```
void setup() {
    size(480, 240);
    background(128, 128, 128);
    stroke(64, 64, 64);
}
void draw() {
    if (mousePressed) {
        fill(0);
    } else {
        fill(255);
    }
    ellipse(mouseX, mouseY, 80, 80);
}
```

The status bar at the bottom left shows the number 14.



# Processing - Studio Sketchpad

- Webové prostředí pro Processing...
- <http://sketchpad.cc/>



# Processing, pokr.

## Ukázky

- <http://studio.sketchpad.cc/sp/padlist/all-portfolio-sketches>
- <http://www.processing.org/exhibition/>

## Dokumentace

- <http://www.processing.org/learning/>
- <http://www.processing.org/reference/>

# Cíl přednášky

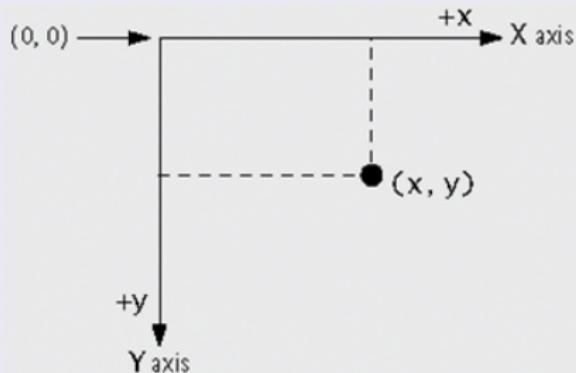
Seznámit se s obecnějšími principy 2D grafických API, které se v různé podobě vyskytují ve všech knihovnách.

# Lokální souřadný systém

- 2D souřadný systém používaný při kreslení "v rámci okna"
- Orientace a pozice počátku  $(0, 0)$  dána konkrétním API
- Většinou v jednotkách pixelů (floating-point hodnoty)

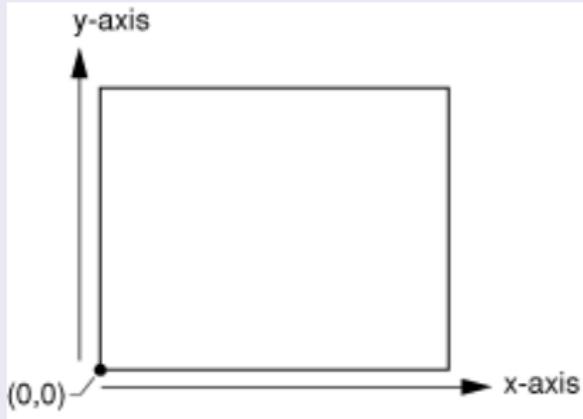
## Počátek nahoře

- např. Cairo, SDL, Java 2D a Processing



## Počátek dole

- např. Quartz 2D

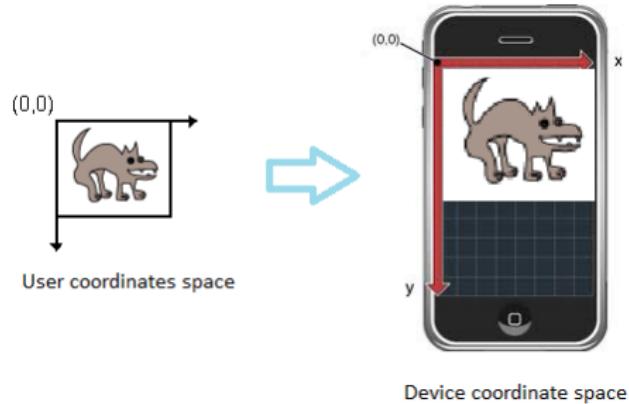


# Souřadný systém nezávislý na zařízení

- Různá zařízení (monitor, tiskárna, telefon) mají různé grafické možnosti (rozlišení, apod.).
- Většina API definuje:
  - **souřadný systém zařízení** (*device coordinate system*)
  - **uživatelský souřadný systém** (*user coordinate space*)

## Přepočet mezi souřadnicemi

- Objekty se definují v uživatelském souř. systému.
- Před vykreslením se aplikuje transformační matice.
- Typicky změna měřítka.

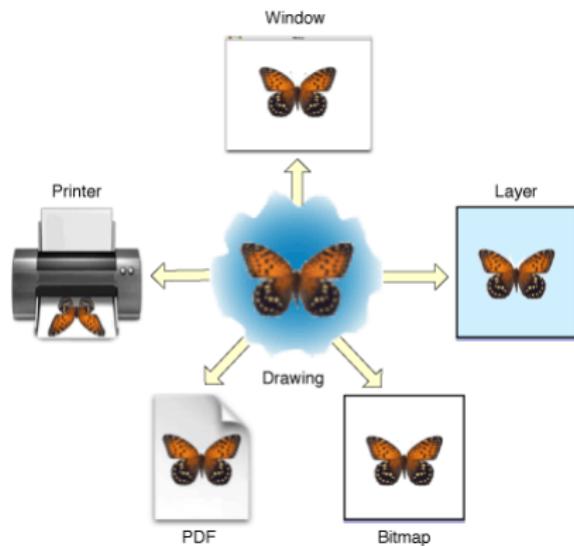


# Co je to grafický kontext?

- „Cíl kreslení“ (též. graphics context, surface, render target, ...)

## Grafický kontext

- Datová struktura, která drží specifické informace potřebné pro kreslení na různá výstupní zařízení (display, bitmapa, PDF soubor, atd.).



# Vytvoření kontextu v knihovně Cairo

```
cairo_surface_t *surface;
cairo_t *cr;

surface = cairo_image_surface_create( CAIRO_FORMAT_ARGB32,
                                     120, 120 );
cr = cairo_create( surface );

cairo_set_line_width( cr, 30.0 );
cairo_set_source_rgb( cr, 1, 0.2, 0.2 );
cairo_set_line_cap( cr, CAIRO_LINE_CAP_BUTT );
cairo_move_to( cr, 64.0, 50.0 );
cairo_line_to( cr, 64.0, 200.0 );
cairo_stroke( cr );
```

# Co grafický kontext obsahuje?

- Parametry výstupního zařízení (formát obrázku, atd.)
- Šířka a výška kreslící plochy (řeší i ořezávání)
- Transformace výstupu (device-independent kreslení)

## „Statefull“ grafický kontext (Cairo, SDL, Quartz 2D, ...)

- Stavové proměnné pro různá nastavení kreslení
- Kreslící barva
- Tloušťka čáry
- atd.

## „Stateless“ grafický kontext (Skia, Direct 2D, ...)

- Nastavení kreslení není součást kontextu
- Samostatné struktury...

# „Stateless“ grafický kontext v Direct2D

## Vytvoření štětce

```
// Create a gray brush.  
m_pRenderTarget->CreateSolidColorBrush(  
    D2D1::ColorF(D2D1::ColorF::LightSlateGray),  
    &m_pLightSlateGrayBrush  
) ;
```

## Kreslení...

```
m_pRenderTarget->DrawLine(  
    D2D1::Point2F(0.0f, 0.0f),  
    D2D1::Point2F(0.0f, 50.0f),  
    m_pLightSlateGrayBrush,  
    0.5f  
) ;
```

# Co jsou to události?

- Umožňují interakci s uživatelem i reakci na změny ve stavu aplikace, operačního systému, apod.

## Události pocházejí od

- Klávesnice, myši, joysticku, ...
- OS – změna velikosti okna, překreslení okna, ...
- Vlastní události (změna ve stavu aplikace, hotovo načtení dat, ...)
- Časovač (např. animace)

# Realizace událostí

- Jak se mechanismus událostí prakticky realizuje?

## Zasílání zpráv (WinAPI)

```
LRESULT CALLBACK WndProc(...)  
{  
    switch(Msg)  
    {  
        case WM_CREATE: break;  
        case WM_SIZE: break;  
        case WM_DESTROY:  
            PostQuitMessage(WM_QUIT);  
            break;  
        default:  
            return DefWindowProc(...);  
    }  
    return 0;  
}  
  
LRESULT SendMessage(...);  
BOOL GetMessage(...);
```

## Callback funkce (GLUT, časovač v SDL)

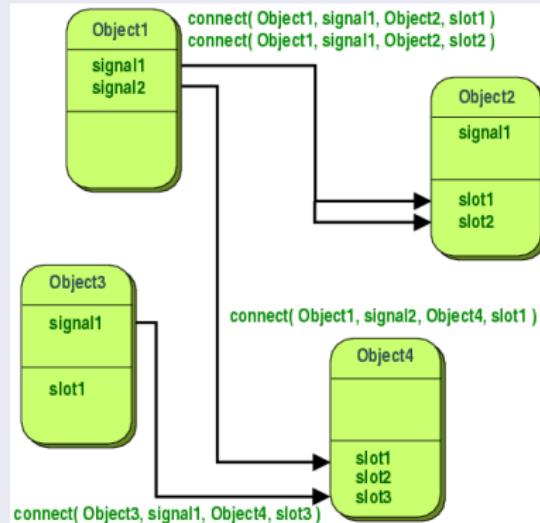
- Callback je ukazatel na fci

```
Uint32 onTimer(...)  
{  
    SDL_Event event;  
    event.type = SDL_USEREVENT;  
    ...  
    SDL_PushEvent(&event);  
  
    return interval;  
}  
  
// registrace callback fce  
SDL_AddTimer(80, onTimer, NULL);
```

# Realizace událostí, pokr.

## Signály a sloty (Qt)

- Objektové jazyky (C++, ...)
- Lepší typová kontrola



# Smyčka událostí/zpráv

- Typická součást kódu GUI knihoven pro zpracování událostí

## Smyčka událostí v SDL

```
SDL_Event event;

while( SDL_PollEvent (&event) )
{
    switch (event.type) {
        case SDL_MOUSEMOTION:
            printf("Mouse moved by %d,%d to (%d,%d)\n",
                   event.motion.xrel, event.motion.yrel,
                   event.motion.x, event.motion.y);
            break;
        case SDL_QUIT:
            exit(0);
    }
}
```

# Kdy kreslit?

- Pokud reagujeme na vstup uživatele
- Je-li třeba překreslit okno (změna velikosti, maximalizace, apod.)
- Něco ve scéně se změnilo...

## Fce pro vykreslení scény (viz projekt v SDL)

```
SDL_Surface * screen = NULL;

// funkce zajistujici prekresleni obsahu okna
void draw(void)
{
    // nejake to kresleni
    ...

    // vymena zobrazovaneho a zapisovaneho bufferu
    SDL_Flip(screen);
}
```

# Kdy kreslit, pokr.

## Smyčka zpráv (viz projekt v SDL)

```
int main(int argc, char *argv[])
{
    SDL_Event event;

    if( SDL_Init(SDL_INIT_VIDEO) == -1 ) exit(-1);
    screen = SDL_SetVideoMode(DEFAULT_WIDTH, DEFAULT_HEIGHT, ...);

    while( !quit ) {
        while( SDL_PollEvent(&event) ) {
            switch( event.type ) {
                case SDL_VIDEORESIZE: // změna velikosti okna
                    onResize(&event.resize);
                    break;
                case SDL_QUIT: // SDL_QUIT event
                    quit = 1;
                    break;
                ...
            }
        }

        // vykreslení po jakékoli události
        draw();
    }
}
```

# „OnPaint“ (Passive Rendering)

- Událost/zpráva generovaná GUI když je třeba překreslit okno.
- WM\_PAINT, QPaintEvent, wxPaintEvent, ...

## Zpráva WM\_PAINT v D2D

```
LRESULT MainWindow::HandleMessage(....)
{
    switch (uMsg)
    {
        case WM_PAINT:
            OnPaint();
            return 0;
        ...
    }
    return DefWindowProc(m_hwnd, uMsg, wParam, lParam);
}
```

# Kreslení v Processing (<http://sketchpad.cc/>)

- Automaticky volané vestavěné funkce
- Globalní proměnné (pozice myši, velikost okna, atd.)

## Reakce na myš

```
void setup() {  
    size(480, 240);  
    background(128, 128, 128);  
    stroke(64, 64, 64);  
}  
  
void draw() {  
    if (mousePressed) {  
        fill(0);  
    } else {  
        fill(255);  
    }  
    ellipse(mouseX, mouseY, 80,  
           80);  
}
```

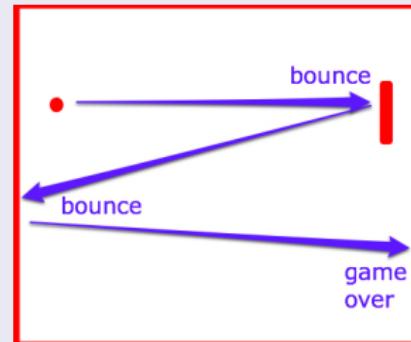
- `setup()` ... zavolá se jednou (inicializace)
- `draw()` ... volá se opakováně (kreslení)
- `mousePressed` ... stav myši
- `mouseX, mouseY` ... pozice kurzoru v okně

# Hra „Pong“

Allan Alcorn (Atari), 1976



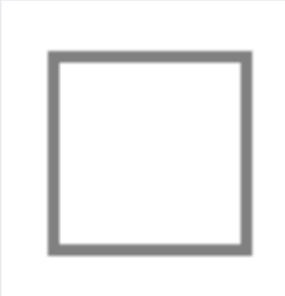
Naše verze pro jednoho hráče...



# Stroke vs. Fill

## Tah/Stroke

- Čáry a obrys objektů:
  - kreslící barva,
  - šířka čáry,
  - styl čáry (plná, čárkovaná, ...),
  - typ štětce/pera, apod.



## Výplň/Fill

- Vnitřní část objektu:
  - barva,
  - gradient,
  - vzory, apod.



# Různé přístupy k implementaci...

## Processing

```
size(100, 100);
background(255);
...
strokeWeight(3);
// barva
stroke(128);
// zadani a kresleni tvaru
rect(25, 25, 50, 50);
...
// bez hranice
noStroke();
// barva vyplne
fill(128);
// kresleni
rect(25, 25, 50, 50);
```

## Cairo

```
// barva
cairo_set_source_rgb(cr, 0.5, 0.5, 0.5);
// zadani tvaru
cairo_rectangle(cr, 25, 25, 50, 50);
...
cairo_set_line_width (cr, 3.0);
// kresleni obrysu tvaru
cairo_stroke(cr);
...
// kresleni vyplneneho tvaru
cairo_fill(cr);
```

# Hra „Pong“ (1)

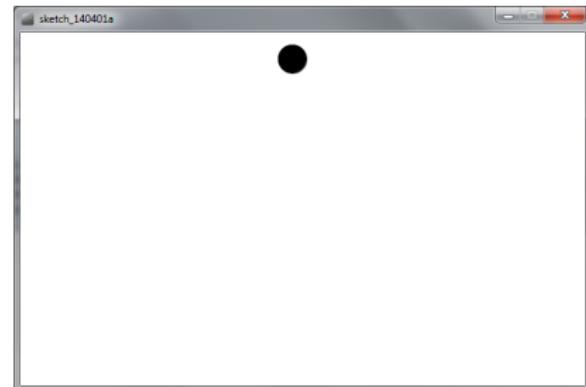
## Letící mýček v ose X

```
float posX = 30;
float posY = 30;
float ballR = 16;

void ball()
{
    fill(0);
    ellipse(posX, posY, 2 * ballR, 2 * ballR);
}

void setup() {
    size(640, 400);
}

void draw() {
    background(255, 255, 255);
    ball();
    posX = posX + 1;
}
```



# Hra „Pong“ (2)

## Odráz od stěny

```
float posX = 30;
float posY = 30;
float ballR = 16;
float dX = 1;

void ball() {
    fill(0);
    ellipse(posX, posY, 2 * ballR, 2 * ballR);
}

void setup() {
    size(640, 400);
}

void draw() {
    background(255, 255, 255);
    ball();
    posX = posX + dX;
    if( posX > width || posX < 0 ) {
        dX = -dX;
    }
}
```

## Co dál?

- Podobně vyřešit osu Y
- Ošetřit správný okamžik odrazu
- Náhodná inicializace

# Path

- Definice složitějších tvarů - polygony, křivky, atd.

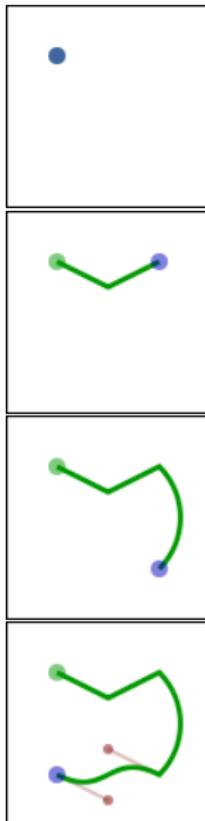


## Cairo

- „Aktivní“ cesta je součástí kontextu
- Zadávání stylem propojování bodů

```
cairo_stroke(); // vykreslí hranici  
cairo_fill(); // vyplní tvar
```

# Příklad v Cairo (<http://cairographics.org/tutorial/>)



```
cairo_move_to (cr, 0.25, 0.25);

// propojovani bodu - cary
cairo_line_to (cr, 0.5, 0.375);
cairo_rel_line_to (cr, 0.25,
                   -0.125);

// oblouk
cairo_arc (cr, 0.5, 0.5, 0.25 *
           sqrt(2), -0.25 * M_PI, 0.25
           * M_PI);

// Bezierova kubika
cairo_rel_curve_to (cr, -0.25,
                     -0.125, -0.25, 0.125, -0.5,
                     0);

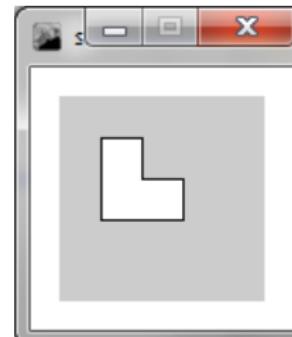
// uzavreni tvaru
cairo_close_path (cr);
```

# Shapes – alternativa v Processing

- Definice složitějších tvarů
- beginShape() a endShape()

```
void setup()
{
    size(100, 100);

    beginShape();
    vertex(20, 20);
    vertex(40, 20);
    vertex(40, 40);
    vertex(60, 40);
    vertex(60, 60);
    vertex(20, 60);
    endShape(CLOSE);
}
```

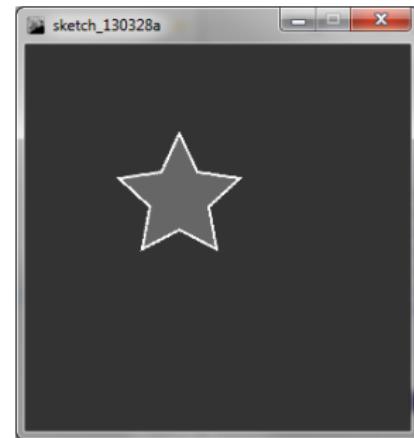


# Shapes – objektově

```
PShape star;

void setup() {
    size(300, 300, P2D);
    star = createShape();
    star.beginShape();
    star.fill(102);
    star.stroke(255);
    star.strokeWeight(2);
    star.vertex(0, -50);
    star.vertex(14, -20);
    star.vertex(47, -15);
    star.vertex(23, 7);
    star.vertex(29, 40);
    star.vertex(0, 25);
    star.vertex(-29, 40);
    star.vertex(-23, 7);
    star.vertex(-47, -15);
    star.vertex(-14, -20);
    star.endShape(CLOSE);
}

void draw() {
    background(51);
    translate(mouseX, mouseY);
    shape(star);
}
```



# 2D transformace

- Afinní transformace
- Zadávají se pomocí fcí typu
  - translate(tx, ty)
  - rotate(rads)
  - scale(sx, sy)
- Akumulují se v „globální transformační matici“
- Nastavené transformace se aplikují při kreslení objektů

```
void setup() {
    size(200, 200);
    background(255);
    noStroke(); // no outline

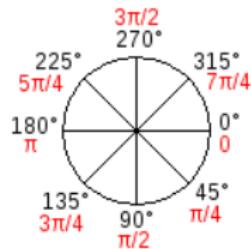
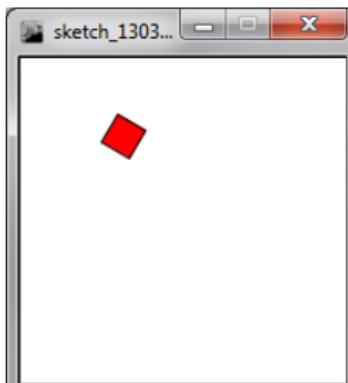
    // draw the original position
    fill(192);
    rect(20, 20, 40, 40);

    // changing the coordinates
    fill(255, 0, 0, 128);
    rect(20 + 60, 20 + 80, 40, 40);

    // and using transformations
    fill(0, 0, 255, 128);
    translate(60, 80);
    rect(20, 20, 40, 40);
}
```

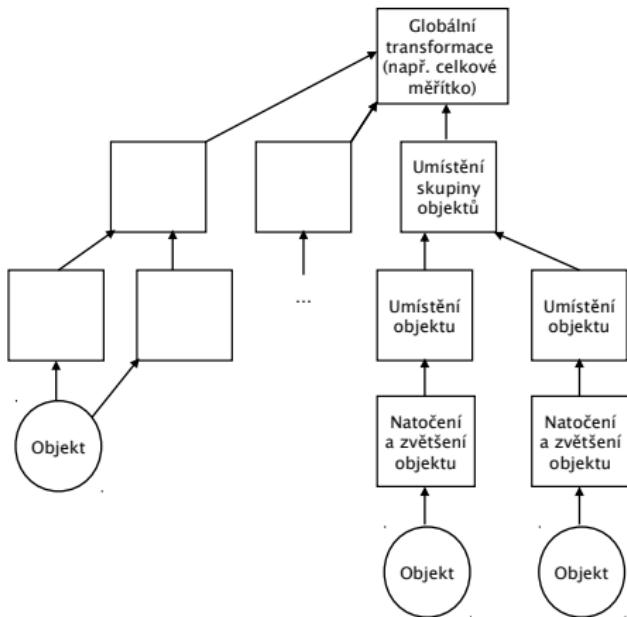
# Na pořadí transformací záleží...

- V jakém pořadí se transformace aplikují?



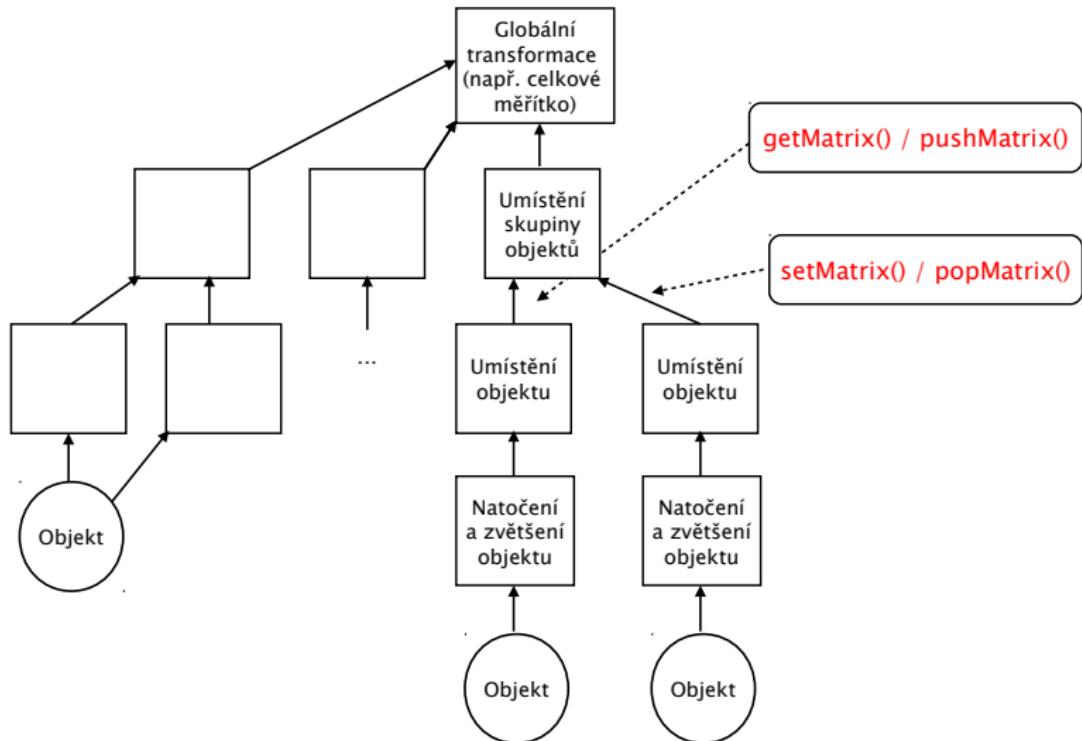
```
void setup() {  
    size(200, 200);  
    background(255);  
    smooth(); // antialiased edges  
    line(0, 0, 200, 0); // draw axes  
    line(0, 0, 0, 200);  
  
    fill(255, 0, 0); // red square  
    rotate(radians(30));  
    translate(70, 0);  
    rect(0, 0, 20, 20);  
}
```

# „Strom transformací“

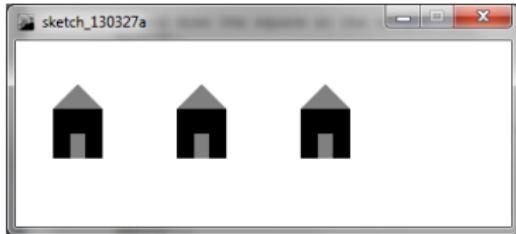


- Hierarchická představa transformací objektů
  - Uzly ... transformace
  - Listy ... kreslení objektu
  - Skládáme transformace v uzlech po cestě ke kořenu
- Jak realizovat prakticky?

# (get,set)Matrix nebo (push,pop)Matrix



# Příklad v Processing



```
void setup() {
    size(400, 150);
    background(255);
    noStroke();

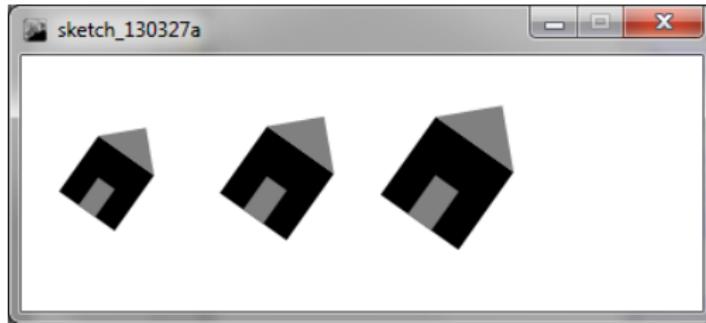
    translate(50, 75);
    for( int i = 0; i < 3; ++i )
    {
        pushMatrix();
        translate(i * 100, 0);
        scale(20);
        house();
        popMatrix();
    }
}

void house() {
    fill(0);
    rect(-1, -1, 2, 2);
    fill(128);
    triangle(-1, -1, 1, -1, 0, -2);
    rect(-0.3, 0, 0.6, 1);
}
```

# Vyzkoušejte si...

Upravit příklad tak, aby

- Domečky byly různě velké
- a otáčely se na místě...



# Hra „Pong“ (3)

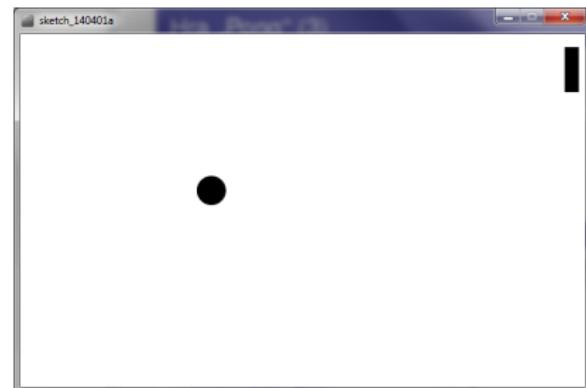
## Přidáme transformace a pálku

```
float pposX, float pposY = 15;
float paddleW = 15; float paddleH = 50;

void ball() {
    fill(0);
    ellipse(0, 0, 2 * ballR, 2 * ballR);
}
void paddle() {
    fill(0);
    rect(0, 0, paddleW, paddleH);
}
void setup() {
    size(640, 400);
    pposX = width - 1.5 * paddleW;
}
void draw() {
    background(255, 255, 255);

    pushMatrix();
    translate(pposX, pposY);
    ball();
    popMatrix();

    translate(pposX, pposY);
    paddle();
    ...
}
```



## Co dál?

- Jak na pohyb pálkou...

# Hra „Pong“ (4)

## Pohyb pálkou

```
void draw() {  
    ...  
    if (keyPressed) {  
        if (key == CODED) {  
            if (keyCode == UP) {  
                if (pposY >= 0) {  
                    pposY = pposY - 5;  
                }  
            }  
            if (keyCode == DOWN) {  
                if (pposY <= height - paddleH) {  
                    pposY = pposY + 5;  
                }  
            }  
        }  
    }  
}
```

## Co dál?

- Kolize...
- Konec hry...

# Hra „Pong“ (5)

## Kolize a konec hry

```
void draw() {  
    ...  
    if( collision() ) {  
        dX = -dX;  
    }  
    if( posX + ballR > width ) {  
        fill(255, 0, 0, 100);  
        rect(0, 0, width, height);  
        noLoop();  
    }  
    ...  
}  
boolean collision() {  
    if (posX + ballR >= pposX) {  
        if ((posY >= pposY) && (posY <= pposY +  
            paddleH)) {  
            return true;  
        }  
    }  
    return false;  
}
```

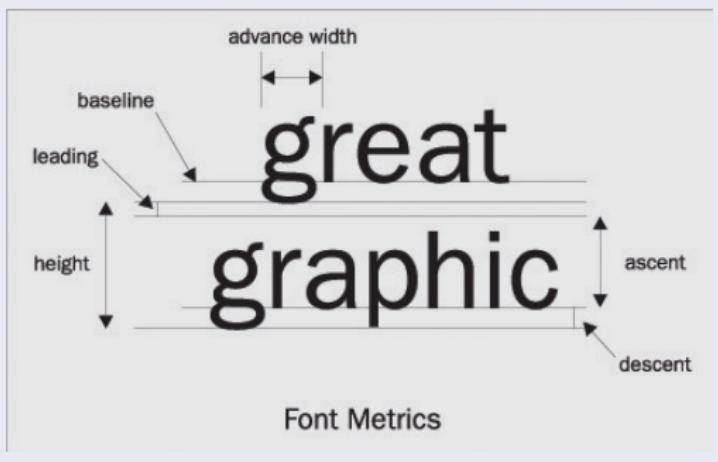
## Co dál?

- Výpis skóre...

# Použití fontů

- Fonty definovány vektorově
- Text se chápe jako další tvar (cesta)
- Lze kreslit hranici nebo vyplnit
- Výběr fontu
  - Serif, Arial, ...
  - Styl fontu (plain, bold, italic)
  - Velikost fontu
  - atd.

## Definice a parametry fontů



# Fonty v Processing

```
PFont f;  
  
void setup()  
{  
    size(200, 200);  
    background(255);  
  
    // Arial, 16 point, anti-aliasing on  
    f = createFont("Arial", 36, true);  
  
    // font to be used  
    textFont(f, 36);  
  
    // font color  
    fill(128);  
  
    // draw text  
    text("Hello world!", 10, 100);  
}
```



# Hra „Pong“ (6)

## Výpis skóre

```
PFont f;  
int score = 0;  
  
void draw() {  
    ...  
    if( collision() ) {  
        dX = -dX;  
        score = score + 1;  
    }  
    ...  
    textFont(f, 32);  
    fill(150, 0, 0);  
    text("Score: " + str(score), 150, 50);  
}
```

## Co dál?

- Lepší grafika...

# Co je to Sprite?

- Grafika, založená na bitmapových obrázcích
- Animace pohyblivých objektů
- Typicky 2D hry, apod.

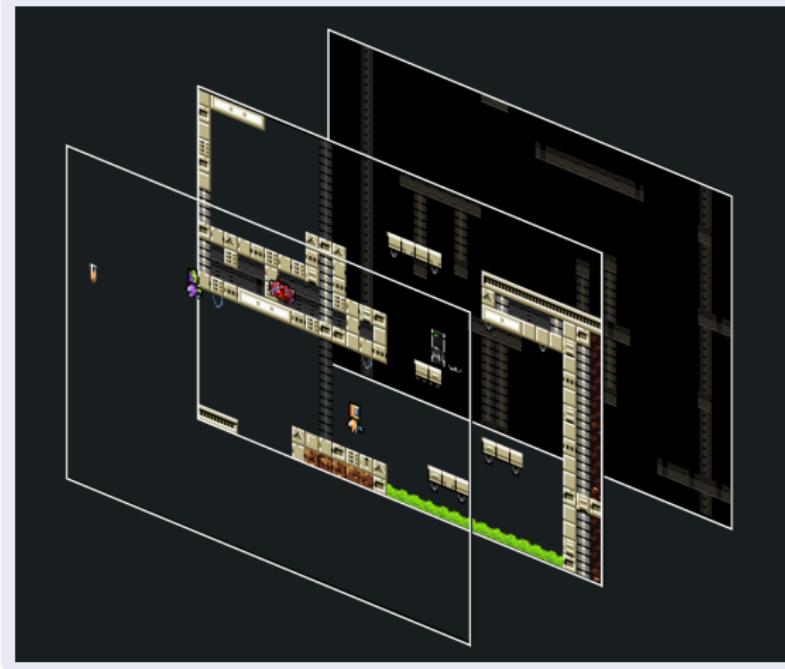
Příklad spritů pro postavičku...



# Průhlednost (transparency)

## Blending grafiky ve vrstvách

- RGBA obrázky/pixely
- Kreslení bitmap/spritů „přes sebe“



# Hra „Pong“ (7)

Obrázek na pozadí, míček a  
pálka

```
Plimage court, ball, paddle;  
  
void setup() {  
    ...  
    court = loadImage("court.jpg");  
    ball = loadImage("ball.png");  
    paddle = loadImage("paddle.png");  
}  
void draw() {  
//    background(255, 255, 255);  
    background(court);  
    ...  
}  
void ball() {  
    image(ball, -ballR, -ballR);  
}  
void paddle() {  
    image(paddle, 0, 0);  
}
```



# Základy počítačové grafiky

Základy vykreslování 3D scény, perspektivní a paralelní projekce

Michal Španěl

Ústav počítačové grafiky a multimédií  
Fakulta informačních technologií  
Vysoké učení technické v Brně

Brno 2013

# Cíl přednášky

Seznámit se s principy zobrazování 3D grafiky...

Co tvoří 3D scénu?

Jak se zobrazí 3D scéna na 2D obrazovce?

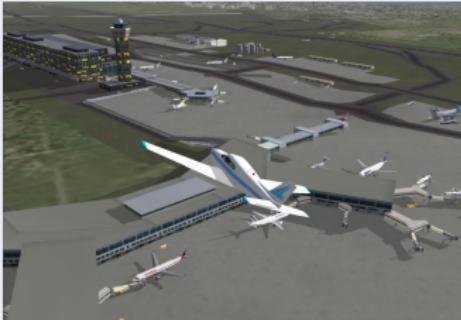
Co je to perspektivní a paralelní projekce?

# Motivační příklady

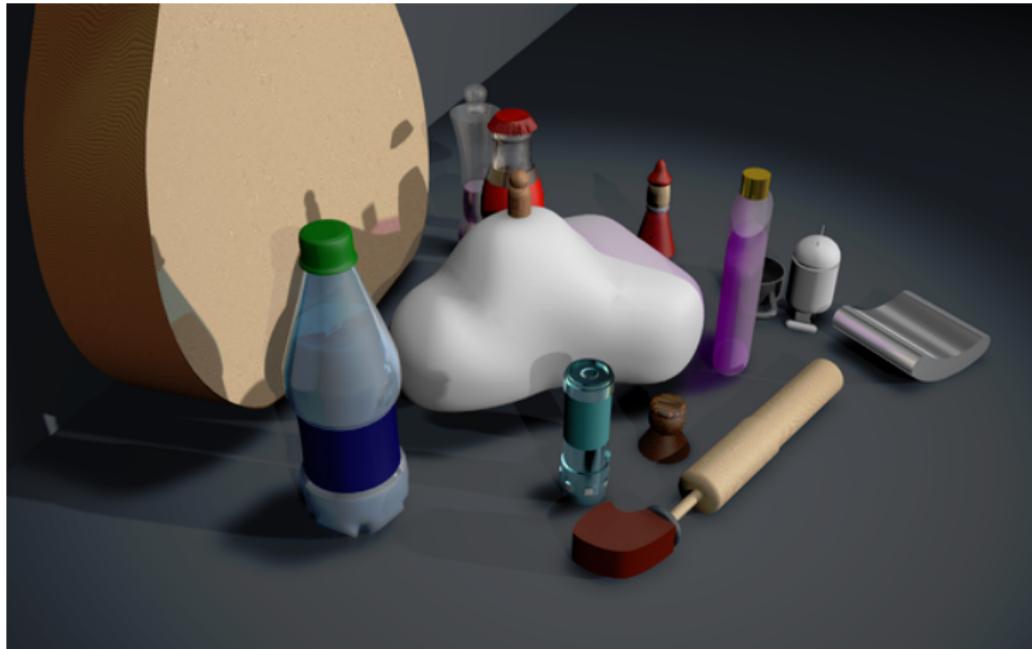
## Architektura



## Simulátory, hry, atd.



# Co potřebujeme pro zobrazení 3D scény?



# Prvky 3D scény

## Objekty

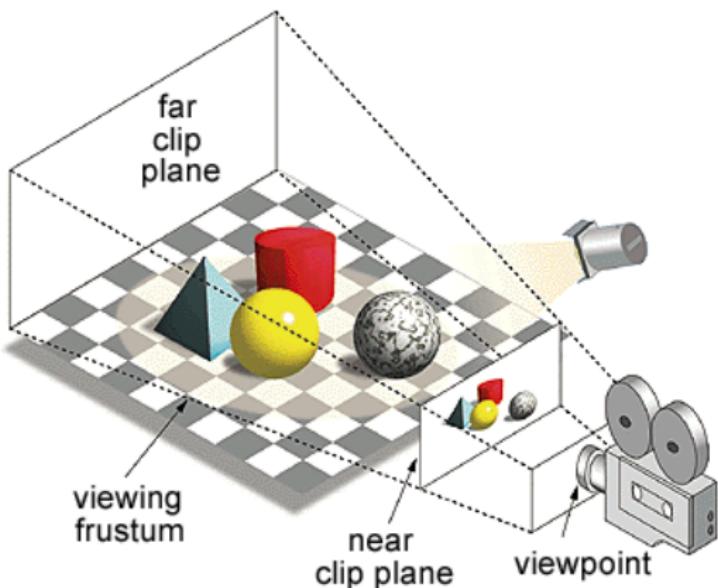
- Geometrie,  
materiál,  
průhlednost, atd.

## Zdroj světla

- Bodový, plošný, ...

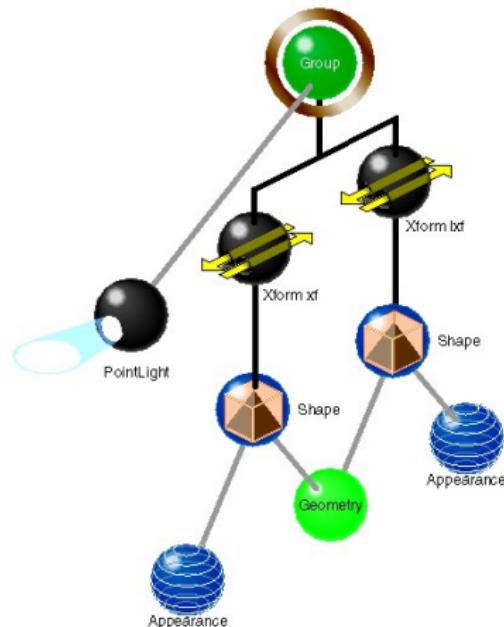
## Kamera

- Pozice, orientace,  
FOV, ...



# Graf scény

- Příklad moderní reprezentace 3D scény
- Knihovny OpenSceneGraph, Ogre3D a další

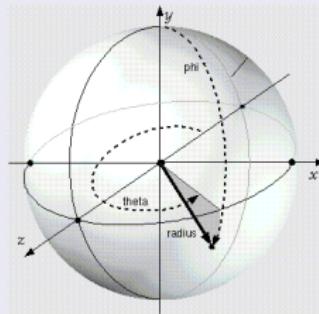
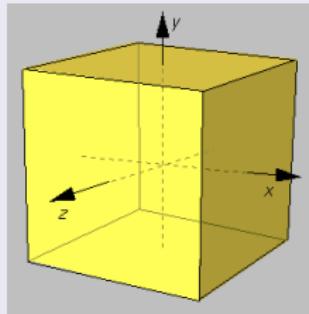


# Tvoříme svět...

- 3D modely jednotlivých objektů.
- Model scény (rozmístění objektů, poloha kamery, apod.).

## 3D model objektu

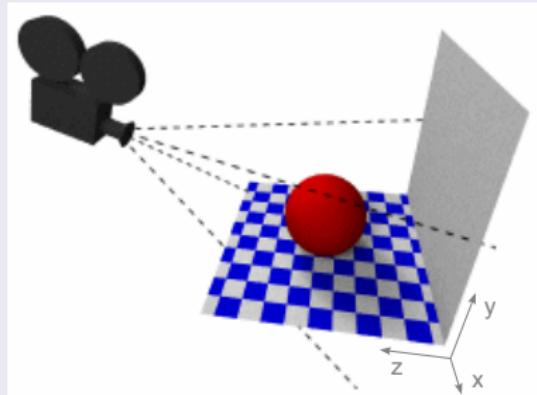
- Nejčastěji obecný model (koule, židle, atd.)
- Vhodně **zvolený souřadný systém**.



# Tvoříme svět...

## 3D scéna

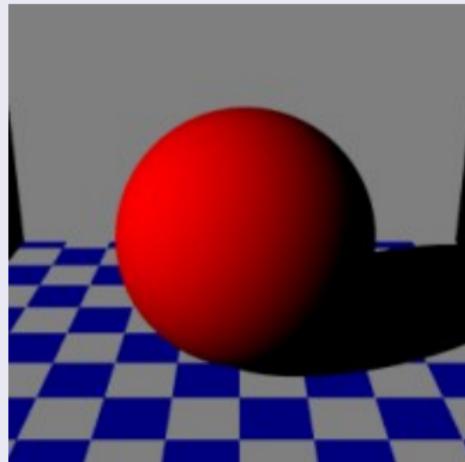
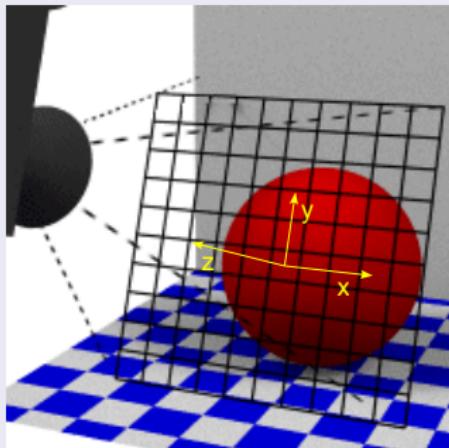
- Souřadný systém scény (angl. *world coordinates*).
- Umístění objektu do scény → **transformace do prostoru scény** (posunutí, rotace, apod.).
- Definice kamery (poloha, směr, úhel).



# Tvoříme svět...

## Zobrazení scény

- Transformace do souřadného systému kamery.
- (Perspektivní) projekce.

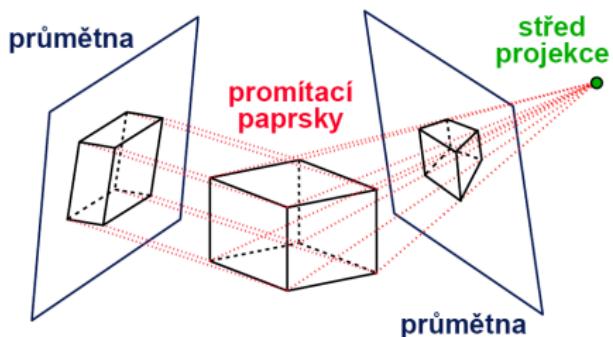


# Projekce

- 3D objekty zobrazujeme na 2D výstupu.
- Projekce (též. promítání) = *transformace ze 3D do 2D prostoru*.
- Dochází ke ztrátě informace.
- Projekční paprsek promítá body na průmětnu.

## Základní druhy projekcí

- Paralelní (rovnoběžná).
- Perspektivní (středová).



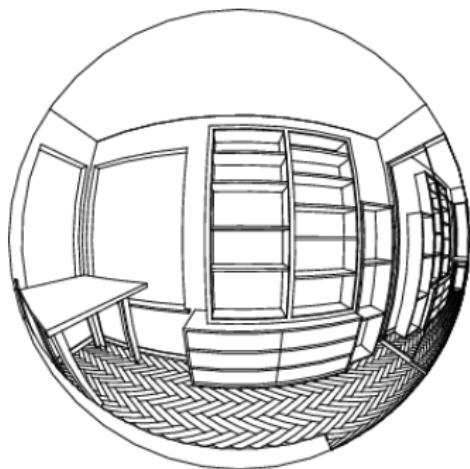
# Projekce, pokr.

## Promítání na rovinnou průmětnu

- *Úsečky se projektují na úsečky.*
- Projekce pouze vrcholů úseček.

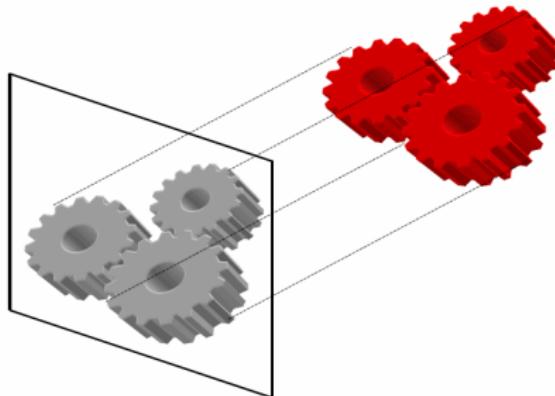
## Promítání na zakřivenou průmětnu

- *Úsečky se projektují na křivky (panorama, rybí oko).*
- Projekce všech *rastrovaných bodů objektu.*



# Paralelní projekce

- Lineární projekce prostřednictvím rovnoběžných paprsků – *rovnoběžné promítání*.
- **Zachovává rovnoběžnost hran!**
- Použití pro technické aplikace (CAD/CAM/CAE). Výkresová dokumentace, zobrazení 3D objektů, technická schémata, apod.
- *Vzdálenost od průmětny neovlivňuje velikost průmětu!*
- Nejčastěji tzv. *kolmé promítání* – paprsky kolmé na průmětnu.

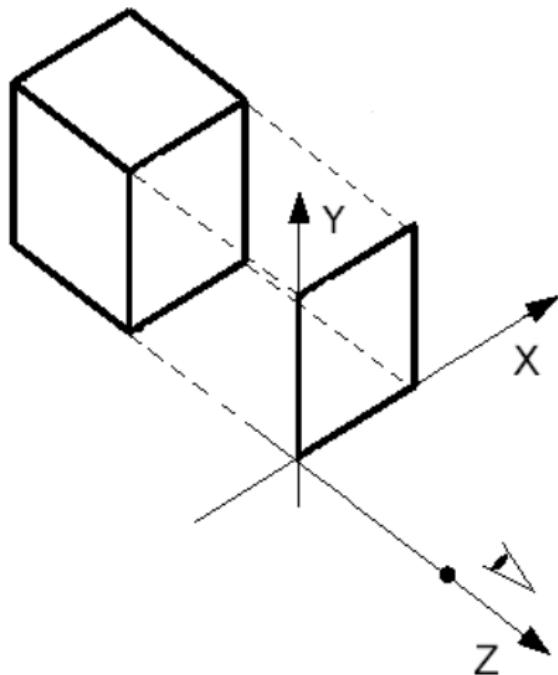


# Paralelní projekce, pokr.

Kolmá projekce, průmětna  
rovnoběžná s rovinou XY

- Promítnutí do roviny XY v  
ose Z = zanedbání (= 0)  
souřadnice Z.

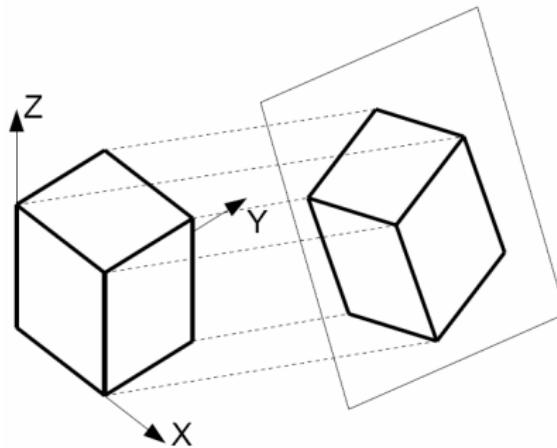
$$P_{XY} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



# Paralelní projekce, pokr.

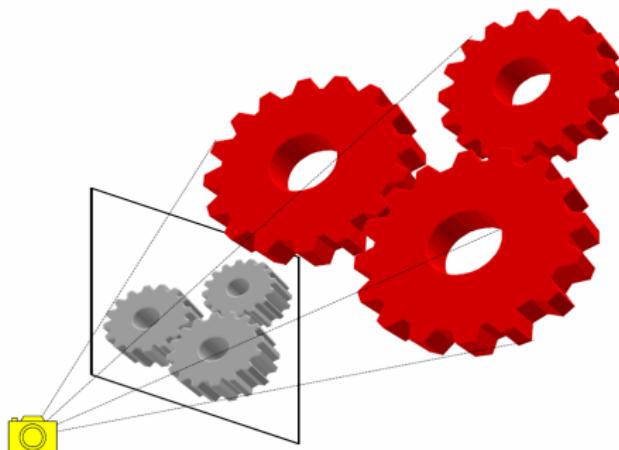
## Obecná kolmá projekce

- Přenést objekty do souřadného systému, kde je průmětna rovnoběžná s rovinou XY (složená lineární transformace).
- Provést promítnutí do roviny XY.



# Perspektivní projekce

- *Nelineární středová projekce* – paprsky vycházejí z jednoho bodu (střed projekce, místo pozorovatele).
- **Nezachovává rovnoběžnost hran!**
- Použití pro virtuální realitu, architekturu, hry, atd.
- *Vzdálenost od středu projekce ovlivňuje velikost průmětu!*
- Odpovídá promítání v realitě.



# Transformační matice pro perspektivní projekci

- Jednobodová perspektiva, průmětna rovnoběžná s rovinou XY.

## Parametrické vyjádření usečky $PS$

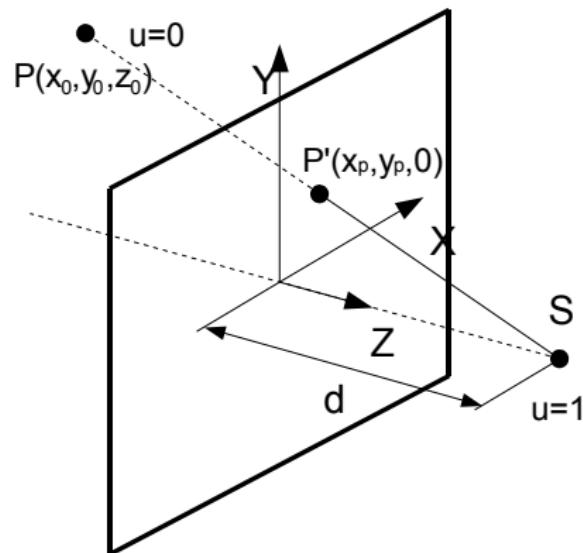
- Body  $P = (x_0, y_0, z_0)$  a  $S = (0, 0, d)$ ,  $u \in \langle 0, 1 \rangle$ .

$$x = x_0 - x_0 u$$

$$y = y_0 - y_0 u$$

$$z = z_0 - (z_0 + d)u$$

- V rovině XY platí  
 $z = 0 \rightarrow u = z_0 / (z_0 + d)$



# Transformační matice pro perspektivní projekci, pokr.

## Odvození matice

- Po dosazení  $u = z_0/(z_0 + d)$

$$x_{P'} = x_0 \frac{1}{1 + z_0/d}, \quad y_{P'} = y_0 \frac{1}{1 + z_0/d}$$

- Maticový zápis

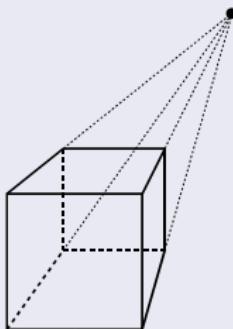
$$(x_P, y_P, z_P, w_P) = (x, y, z, w) \cdot P_{per}, \quad w_P = 1 + z/d$$

$$P_{per} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Vícebodová perspektivní projekce

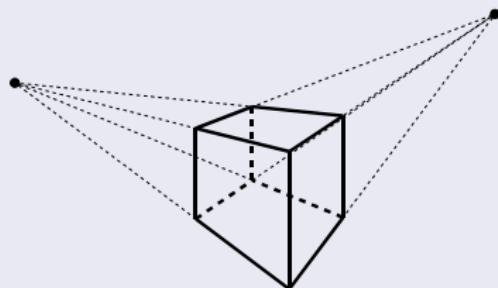
## Jednobodová perspektiva

- Průmětna protíná jedinou souřadnicovou osu.
- Úsečky kolmé na průmětnu se sbíhají do jednoho bodu - *úběžníku*.



## Dvoubodová perspektiva

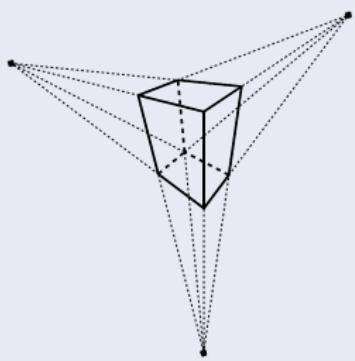
- Průmětna protíná dvě ze souřadných os.
- Hrany rovnoběžné s těmito osami směřují do dvou úběžníků.



# Vícebodová perspektivní projekce, pokr.

## Tříbodová perspektiva

- Nejobecnější případ, průmětna protíná všechny tři osy.
- Hrany rovnoběžné s osami se sbíhají do tří úběžníků.



## Transformační matice pro tříbodovou projekci

$$P_{per} = \begin{bmatrix} 1 & 0 & 0 & 1/d_x \\ 0 & 1 & 0 & 1/d_y \\ 0 & 0 & 1 & 1/d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$