matfyz

Computer Graphics Charles University

Základy počítačové grafiky

NPGR 003

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/

Vector graphics

© 2014, Saylerman

Vector graphics

Interactive editing

- splines†, free-form drawing

Colors*

Vector image format*

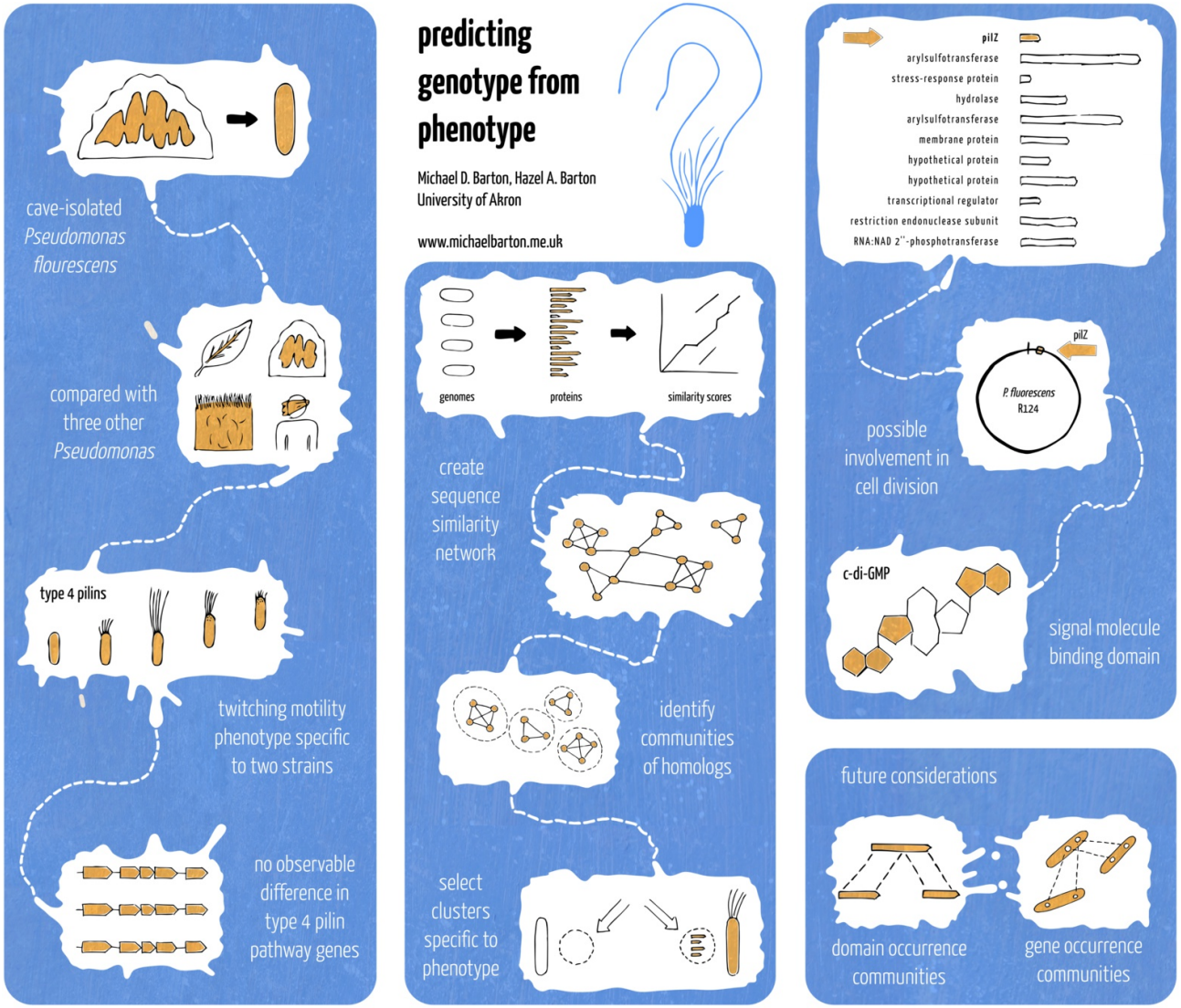
- SVG, PDF, EPS, DXF, AI

Transparency*

Vectorization tool

* … in this course
† … in other courses

Poster, billboard



Poster, billboard

© 1939, Charles Vershuuren

© DaveForYou

Digitized poster

Digital photography

Color balance†

Raster image*

- PNG, TIFF, JPEG

Image rotation†

Poster print:

Color conversion*

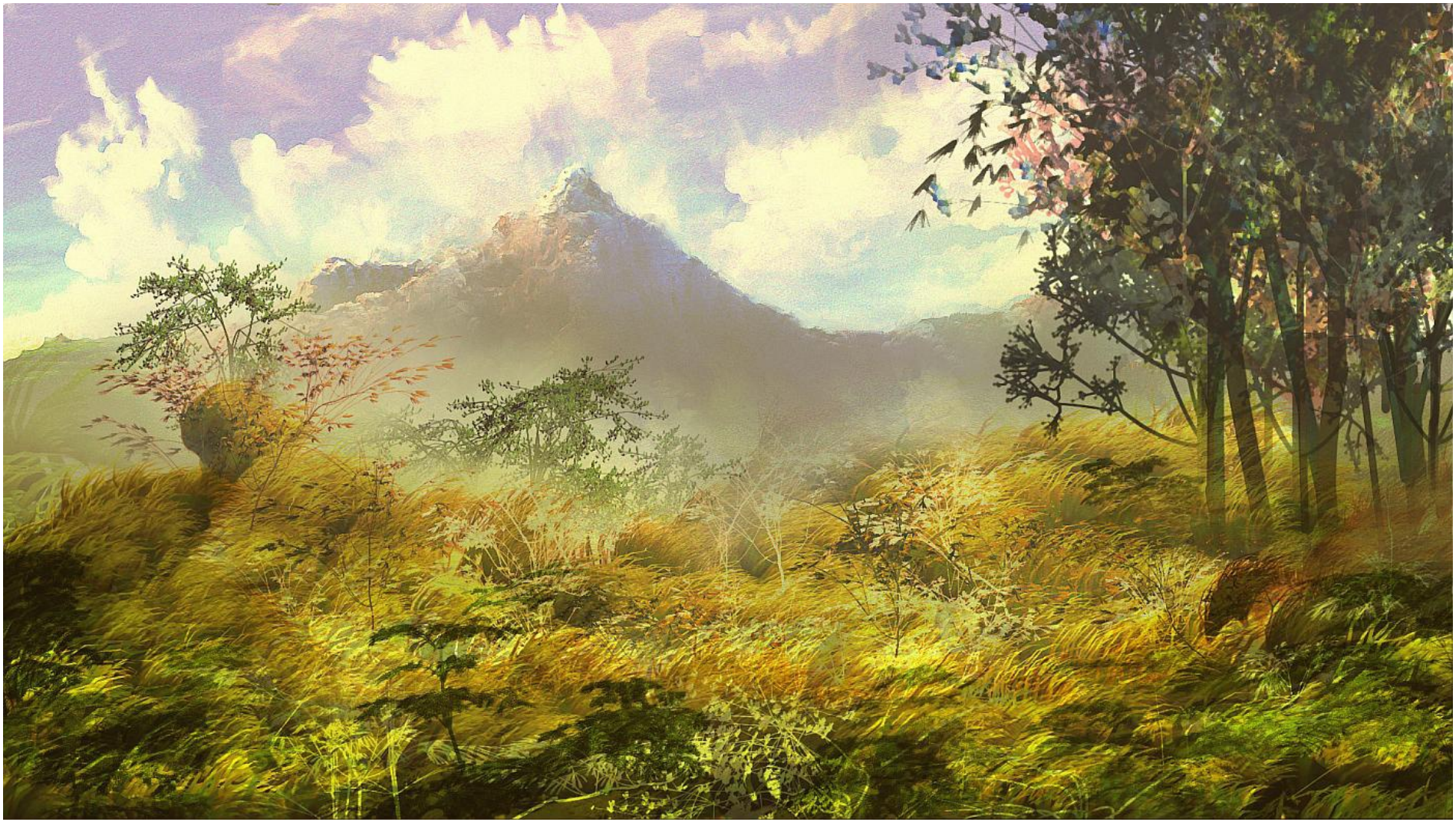
- RGB to CMYK

Digital halftoning*

Digital painting, 2D effects

© Corel Painter, Hahin

Digital painting tools



© Dan Ritchie (PD Particles)

Digital painting

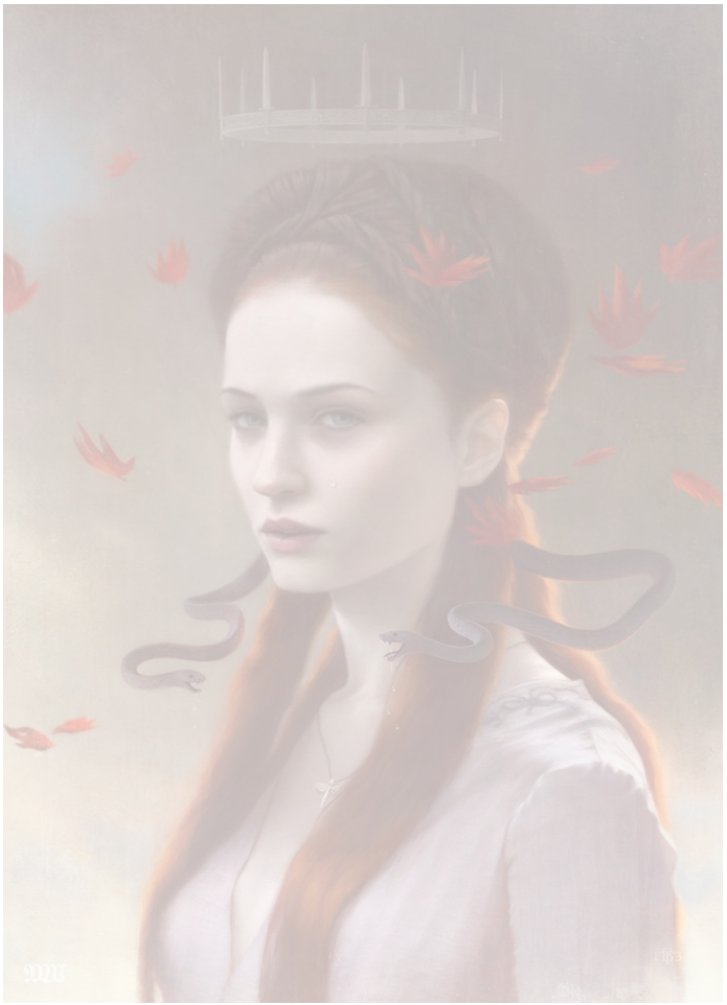
Interactive editing

- pens, brushes, special tools
- „undo“
- touchpad, touchpen, digitizer

Colors*

Transparency*

Painterly effects*



Digital photography

© 2016, DP Review

Digital photography

Autofocus

– edge-detection†

Colors*

– white balance

Raster image format*

– JPEG, RAW

Denoise†

HDR*

– super-bracketing

Digital effect – Photoshop, GiMP

Digital effect

Interactive editing

- pens, brushes, tools
- „undo“

Colors*

Raster image format*

- JPEG, PNG, TIFF

Special effect filters*†

- image enhancement, edge operators, histogram operation…
- color transforms (rebalance…)

HDR photography

© 2015, Andrea Baldwin

HDR photography

© 2013, Jimmy McIntyre

HDR photography

© Conor MacNeill (TheFella)

HDR photography

HDR acquisition*

- multiple exposure
- „super-bracketing“

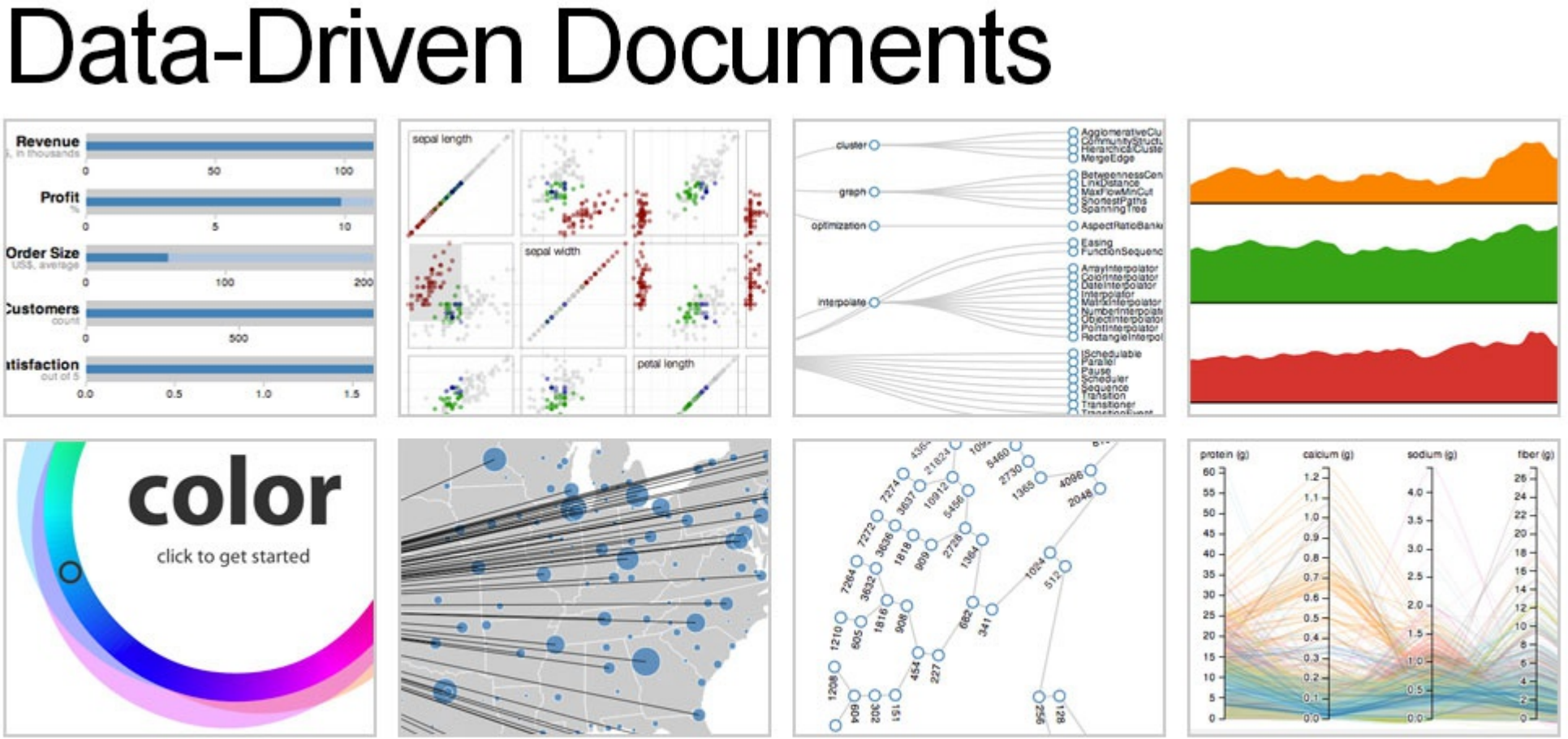
Colors*

HDR image format*

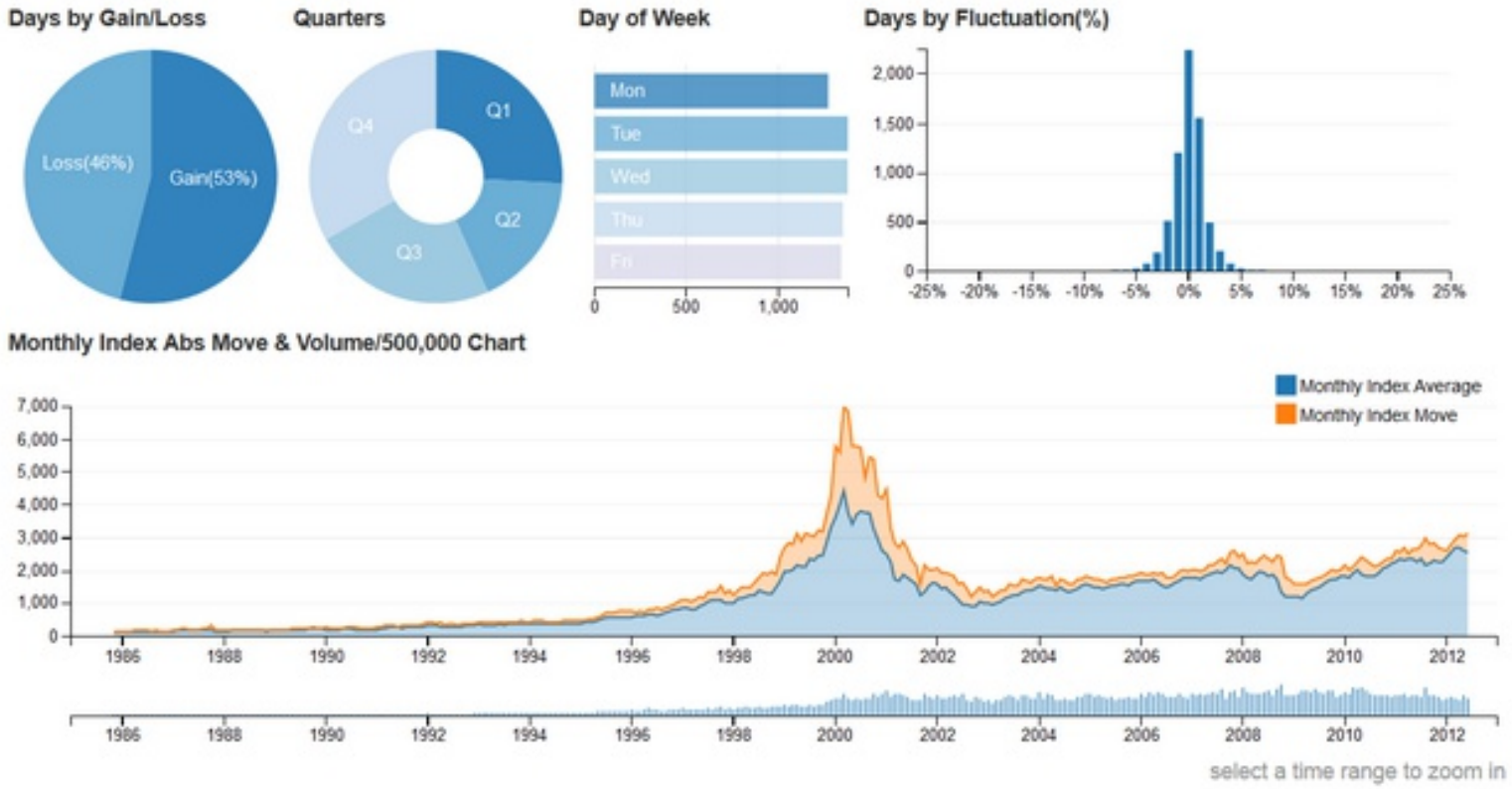
- HDR, EXR, PFM

Tone-mapping*

Web design, data visualization



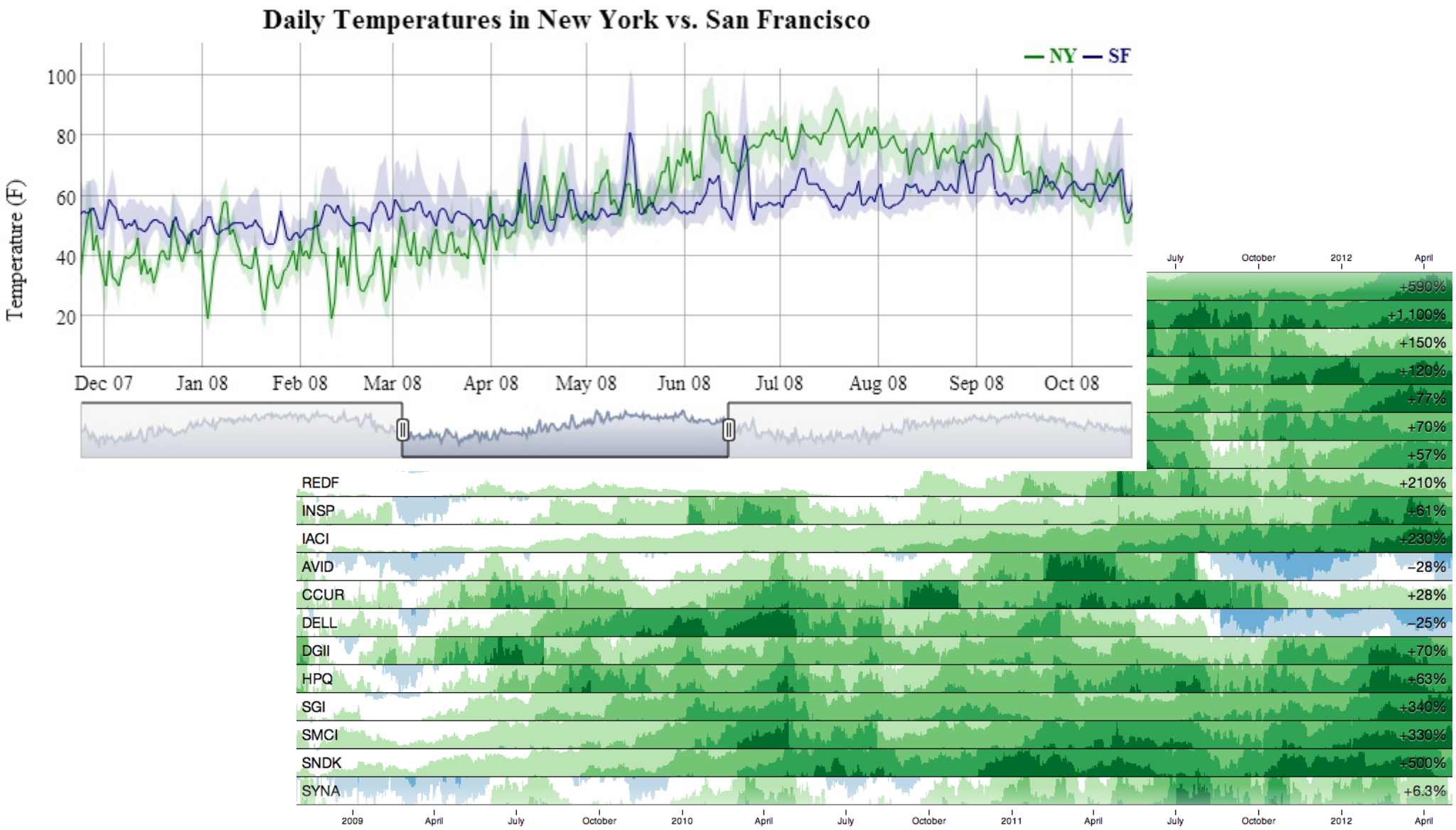
Web design, data visualization



6,724 selected out of 6,724 records | Reset All

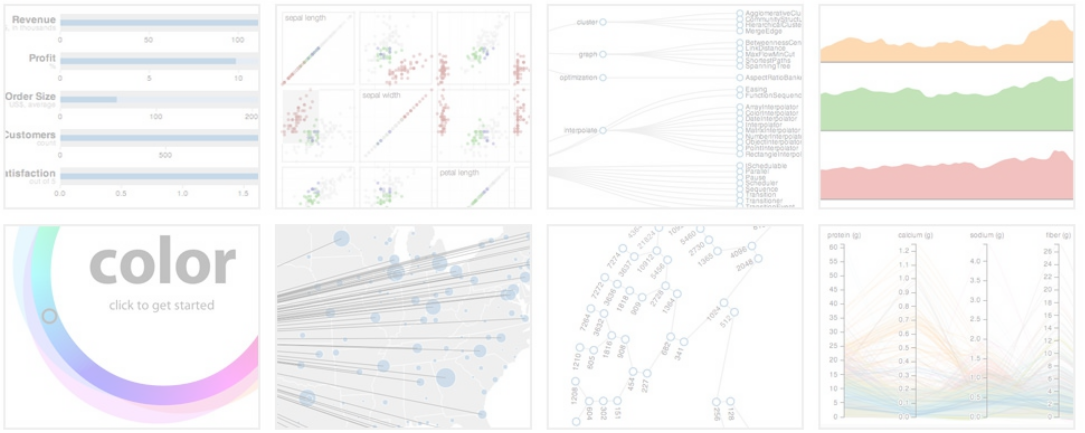
Date	Open	Close	Change	Volume
2012/06				
06/18/2012	2570.98	2592.52	21.54	15407330
06/19/2012	2606.43	2620.83	14.40	17714840

Interactive data on web



Modern web

Data-Driven Documents



HTML5†, CSS3†

- JavaScript
- templates, WordPress

Interactivity†

Data-Driven Documents†

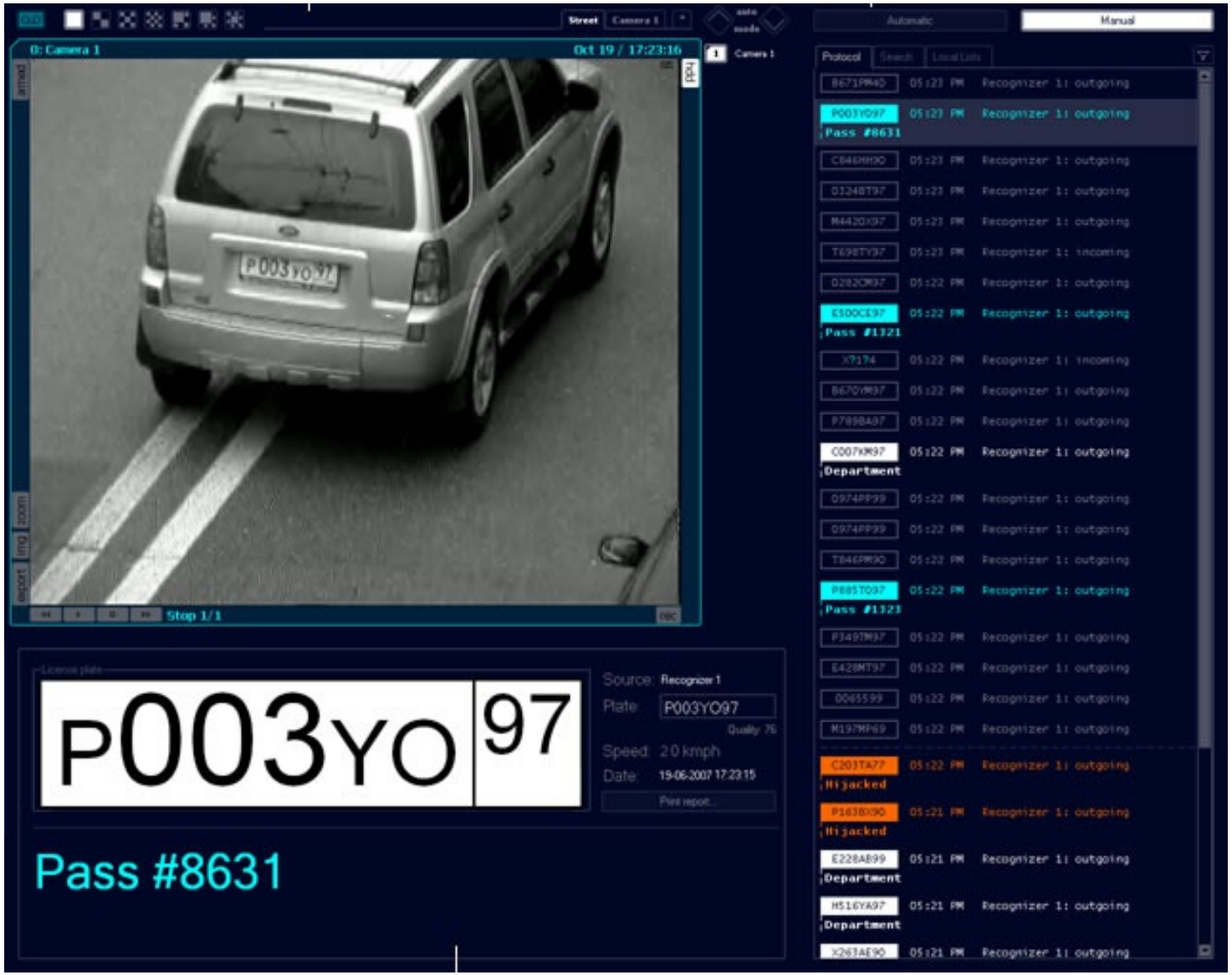
- d3.js library

WebGL for 3D†

- interactivity

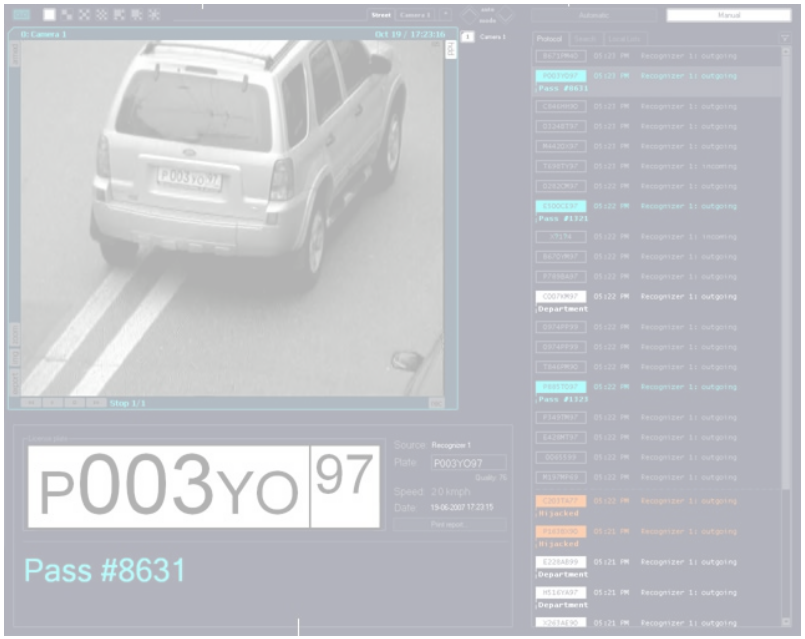
Video, 360-degree video

License-plate recognition



© Smart Security Camera, Inc.

License-plate recognition



Real-time image acquisition

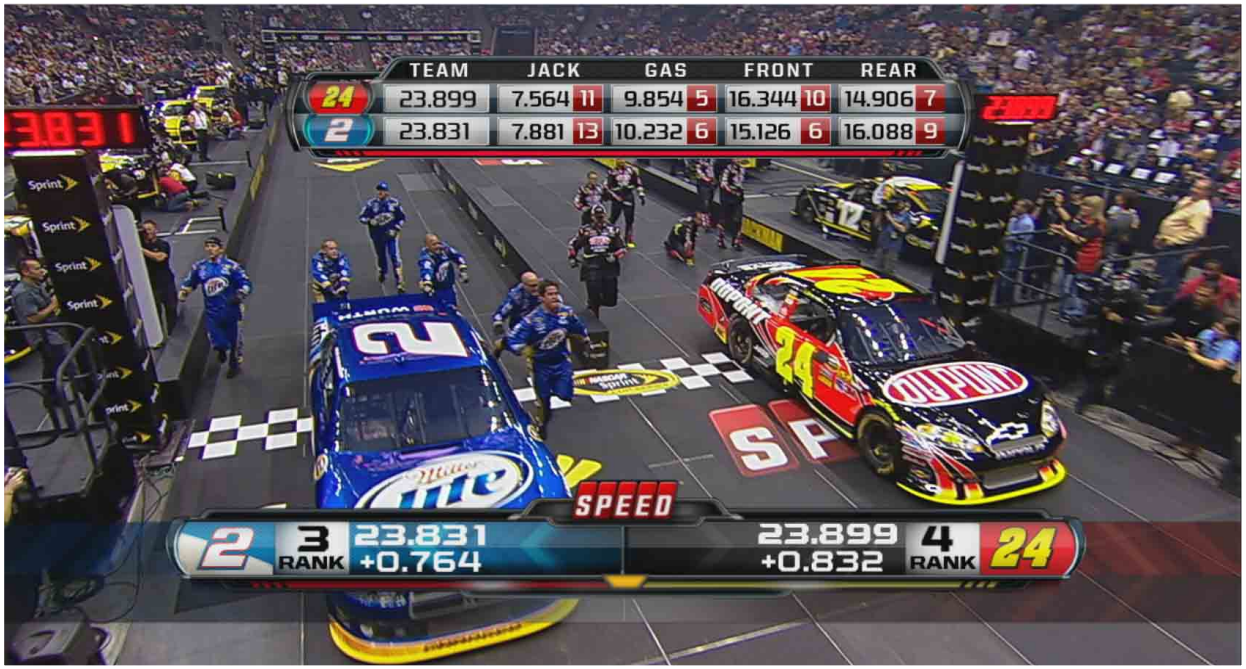
Plate segmentation†

Image warping†

Glyph recognition†

Speed measurement…?

Sport live on TV



Sport live on TV

Vector graphics*

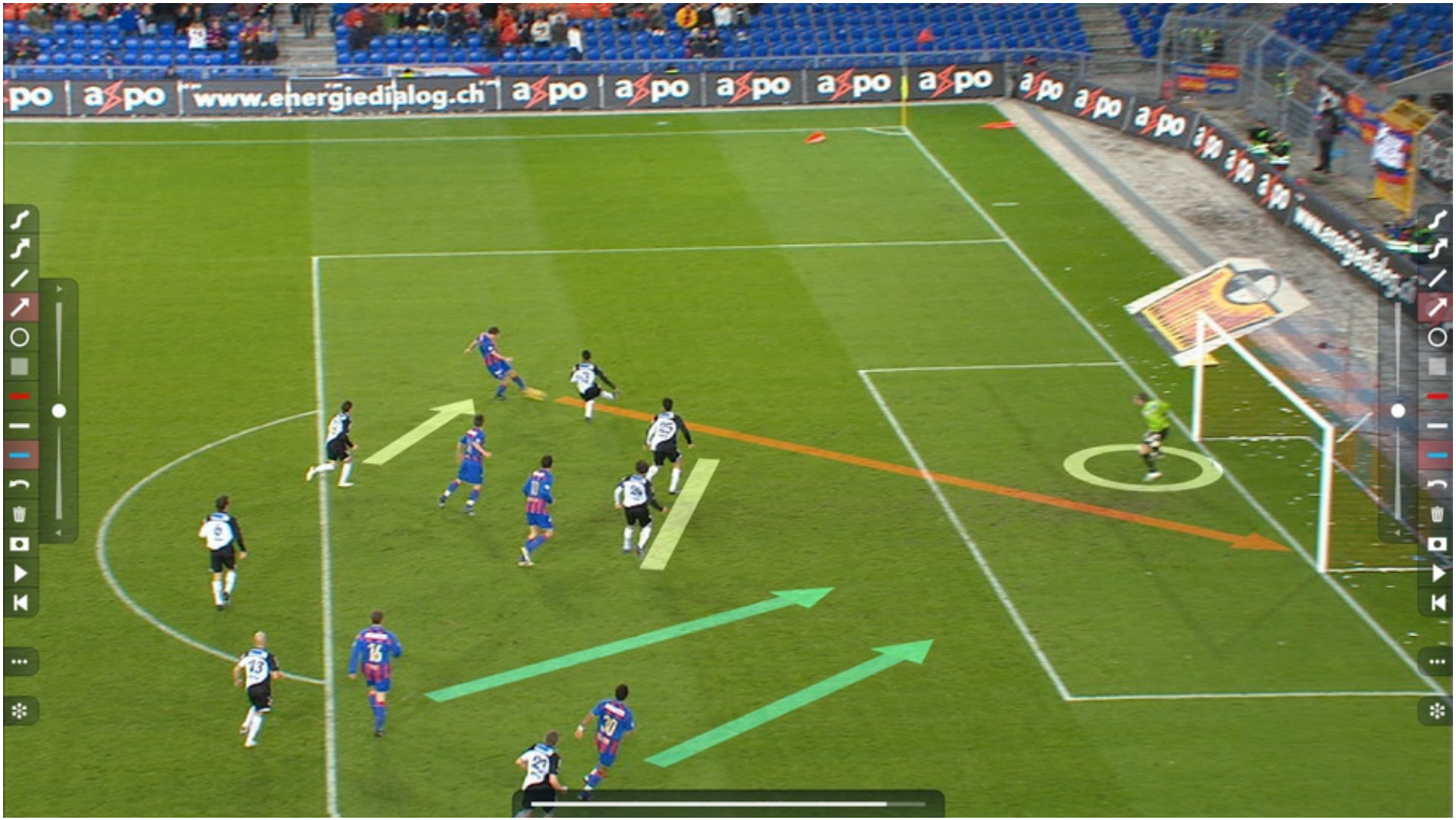
– real-time!

Transparency*

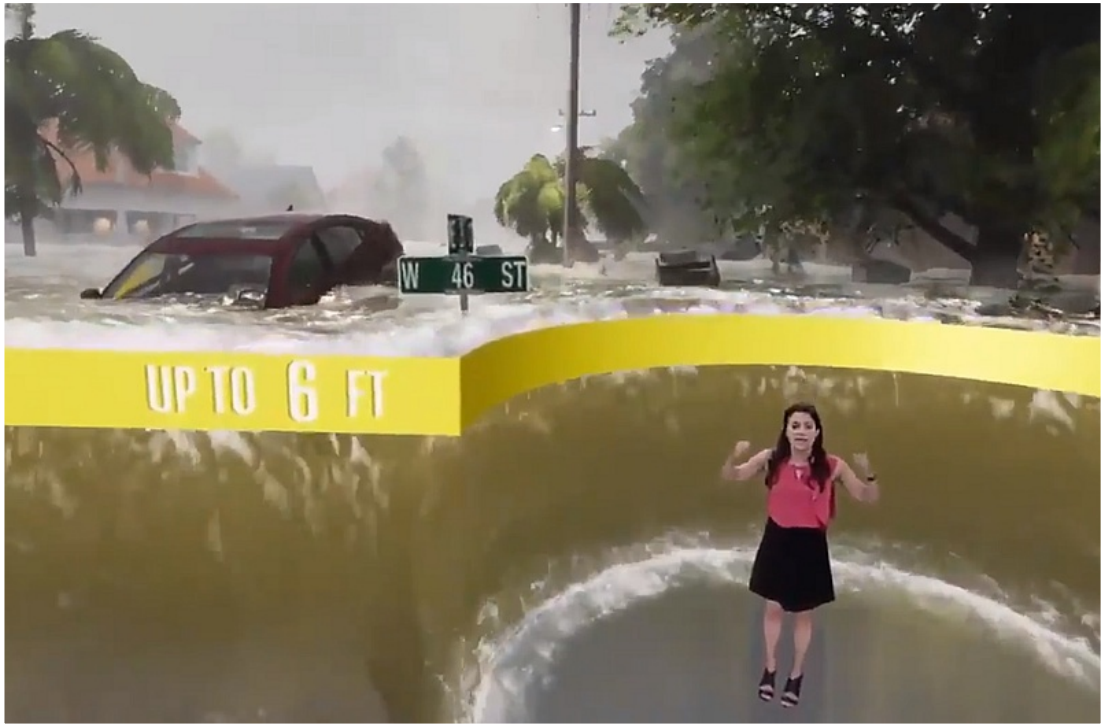
Real-time video signal composition

– real-time video compression†

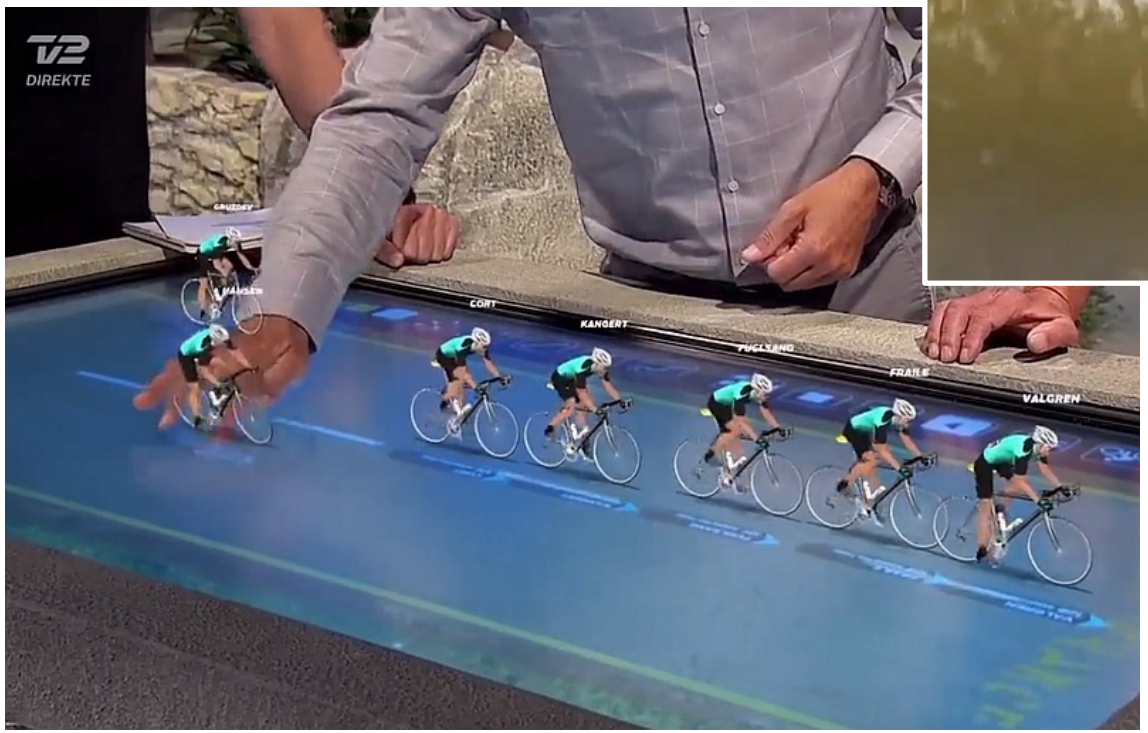
„Next-generation“ sport TV



Augmented reality broadcasting



© The Weather Channel



© TV 2

„Next-gen“ sport TV

3D computer vision†

- camera calibration
- object recognition, segmentation

3D „extra“ model*

- augmented reality

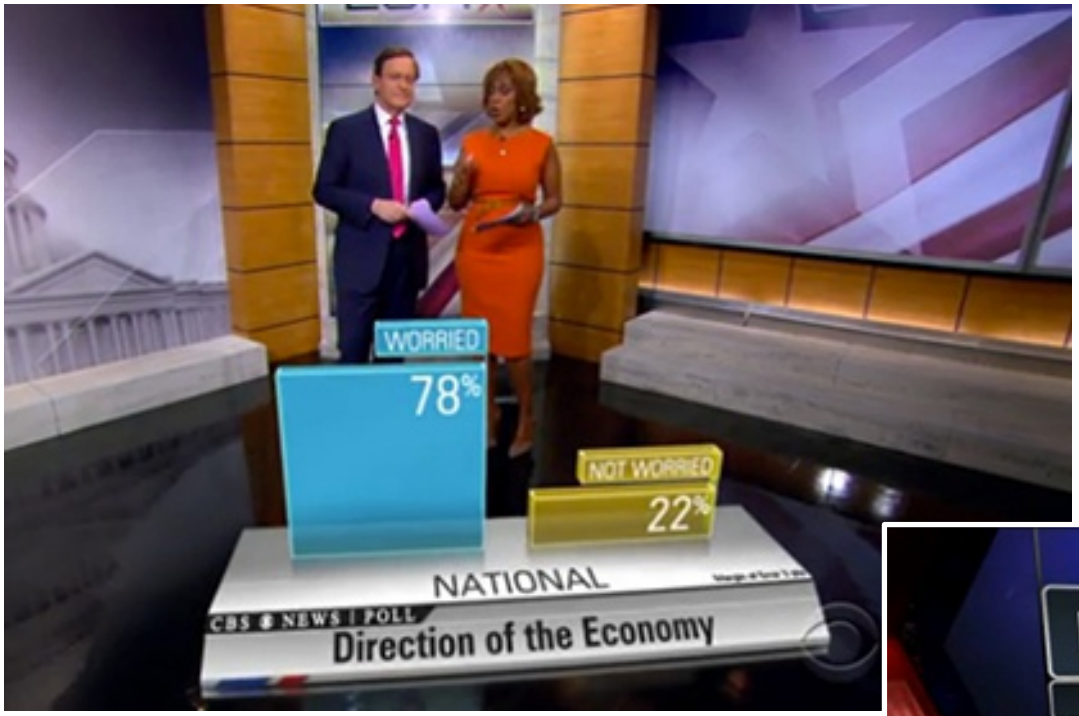
Real-time interaction?

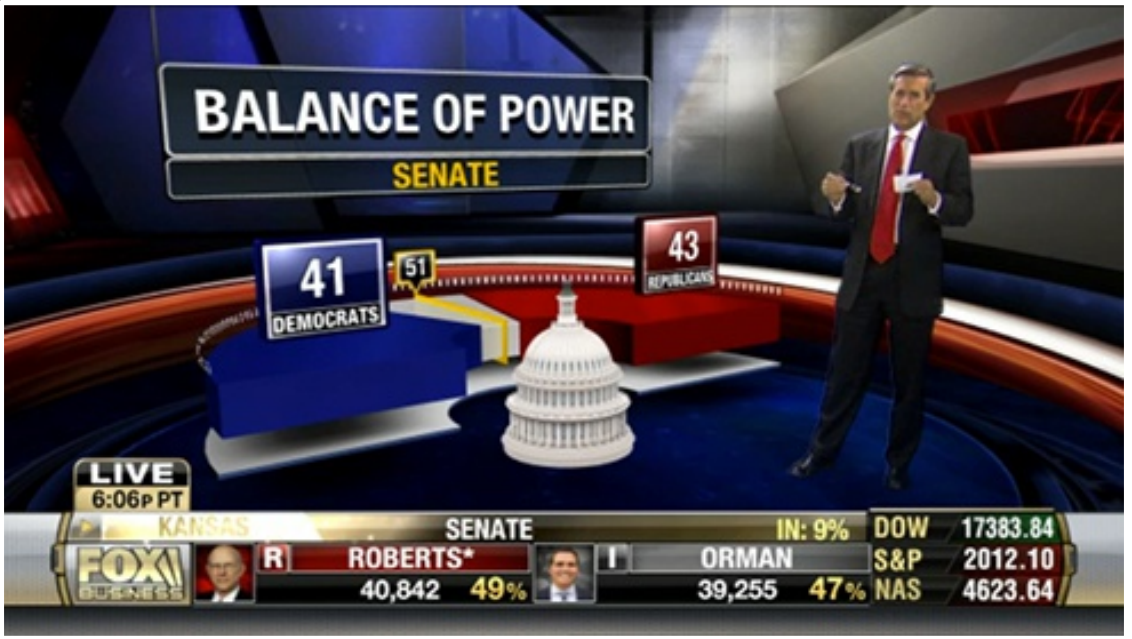
- reporter in a studio…

Real-time video composition

- layers, transparency*
- video compression†

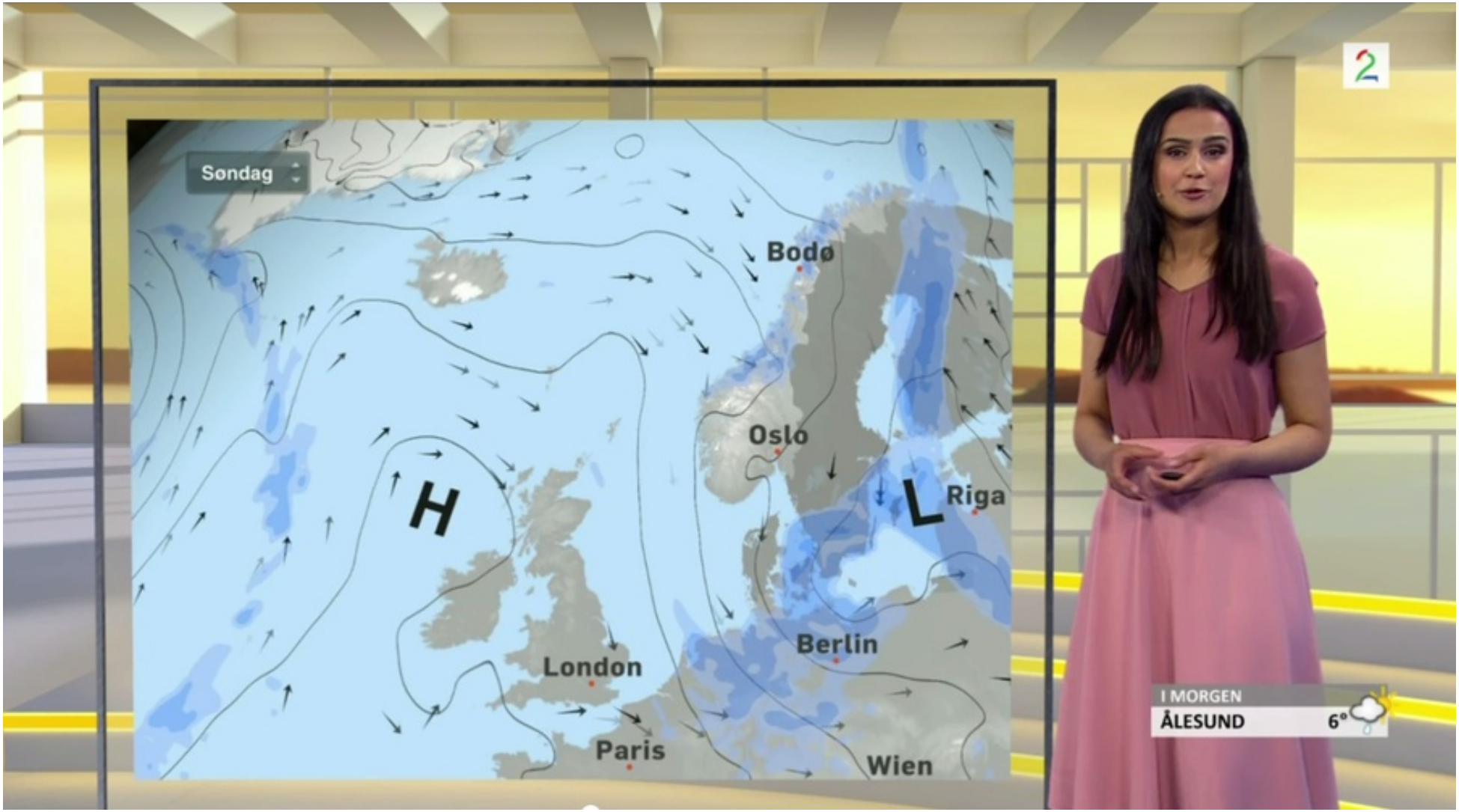
Virtual TV studio





© CBS, FOX TV

Virtual TV studio

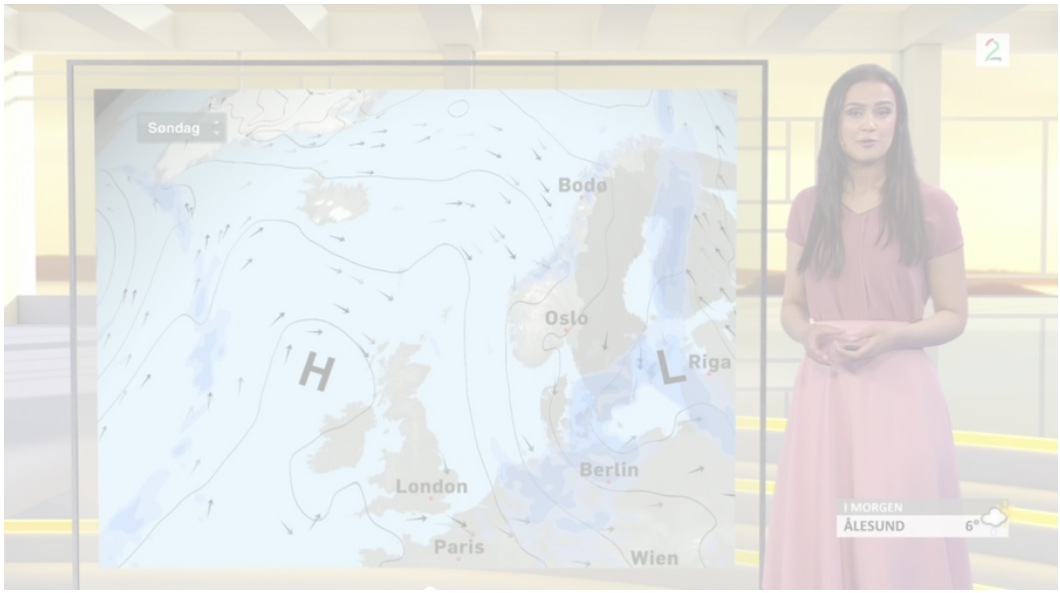


© TV 2

Green screen („virtual studio“)

© 2009, vr3 virtual production oHG

Virtual TV studio



„Green-screen“

- keying in hardware

3D virtual model*

- can be dynamic (animations, additional video channels…)

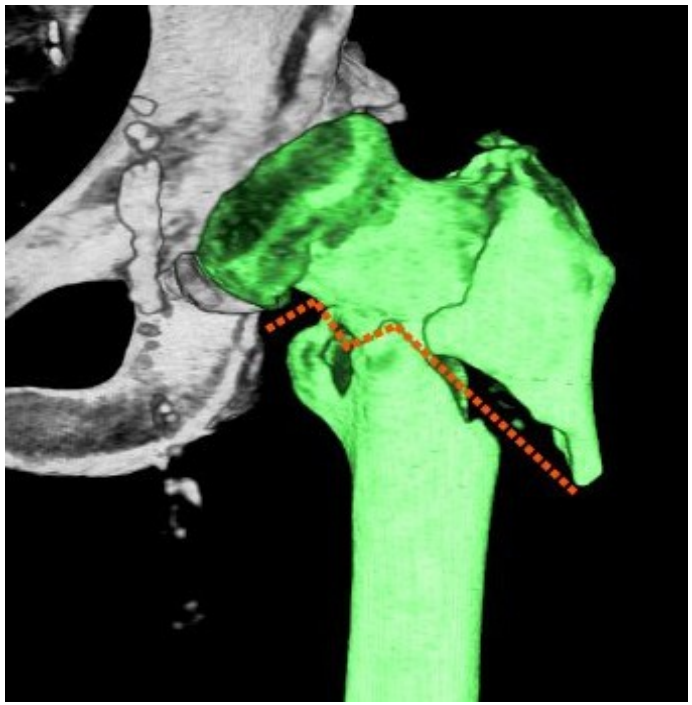
Real-time video composition

- layers, transparency*

Video compression†

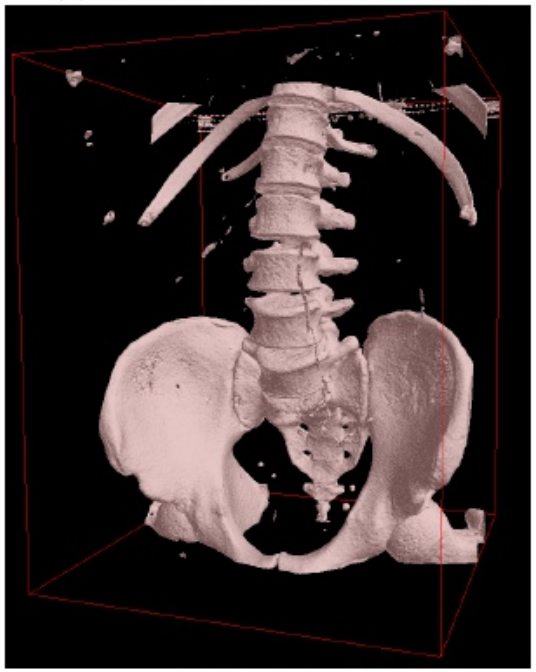
- all in real-time

Medical data

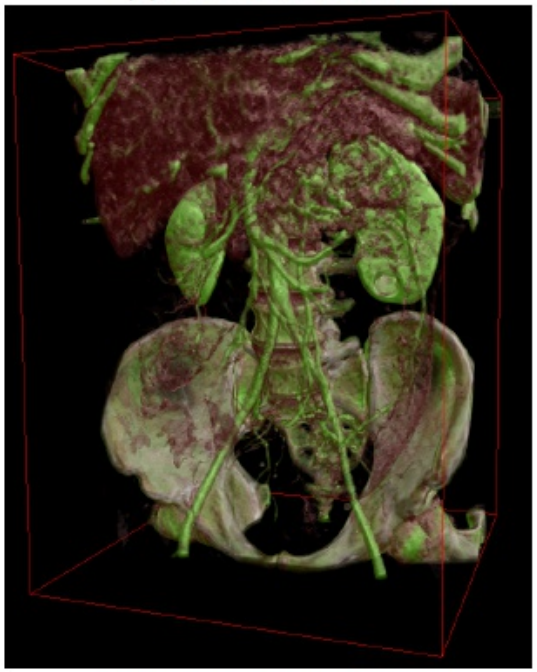


(a) Maximum intensity projection

(b) Density integration

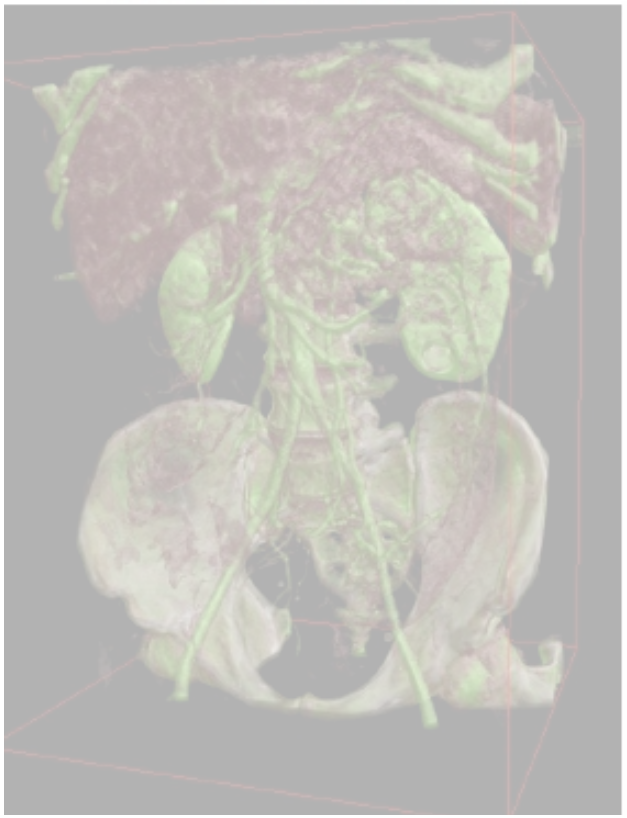


(c) Isosurfaces



(d) 1D transfer function

Medical data



Volume data acquisition†

- Computer Tomography
- Magnetic Resonance Imaging…

Data enhancement†

- de-noise, contrast (CUDA†, GPU†)

Segmentation†

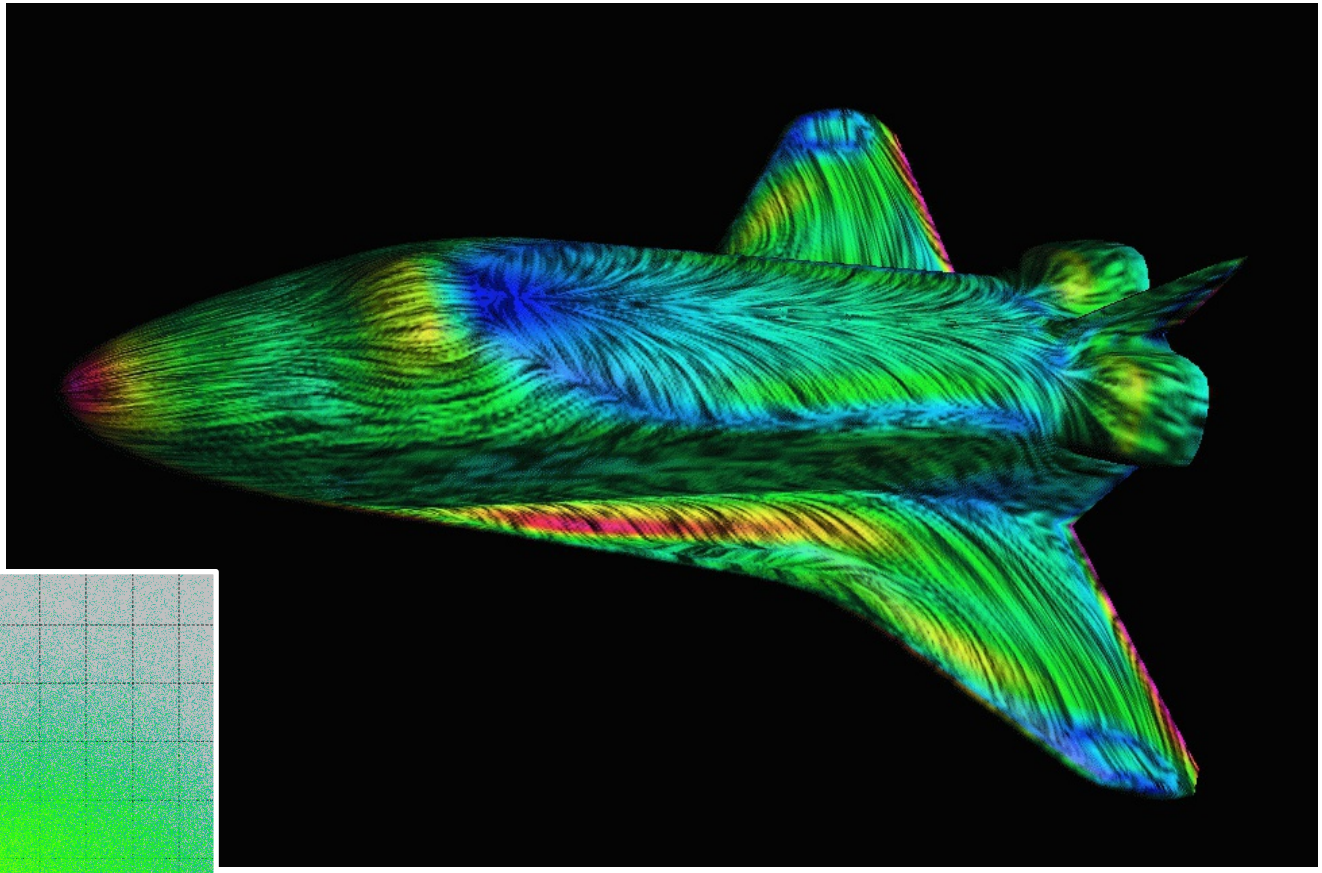
- organs, vessels, bowels (CUDA†, GPU†)

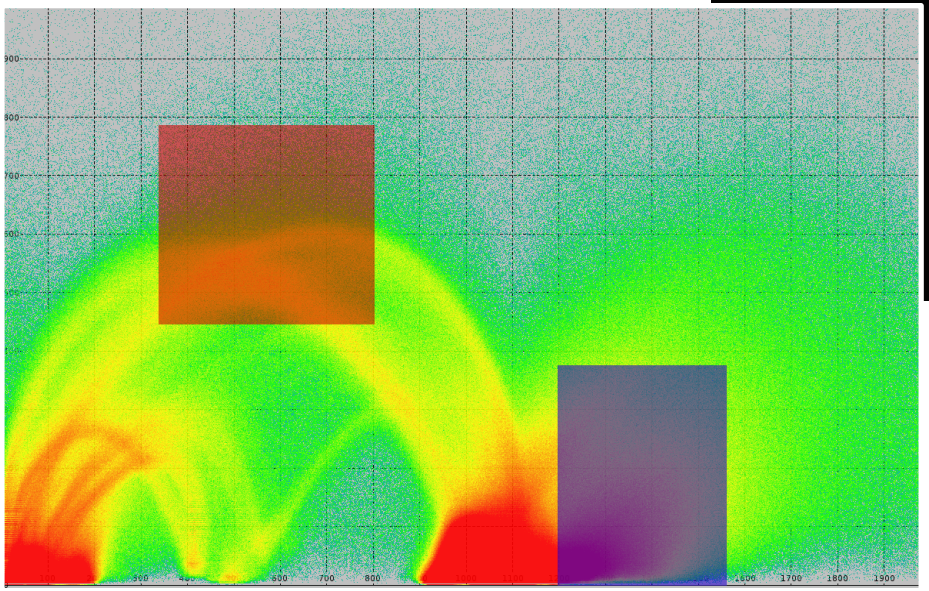
Real-time volume rendering†

- ray-casting on GPU

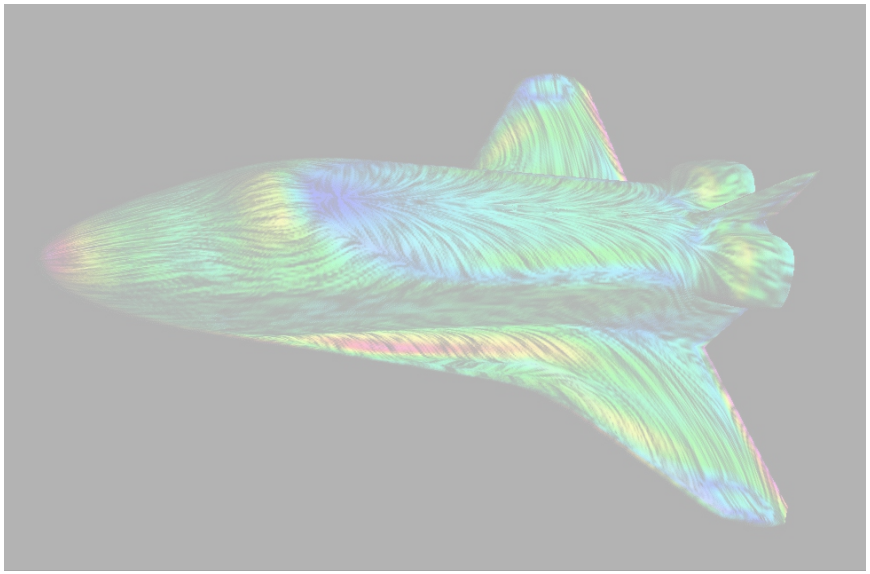
Measurements…

Scientific visualization





Scientific visualization



Data acquisition†

- numeric simulation
- measurements…

Visualization primitives†

- streamlines, arrows…

Real-time rendering†

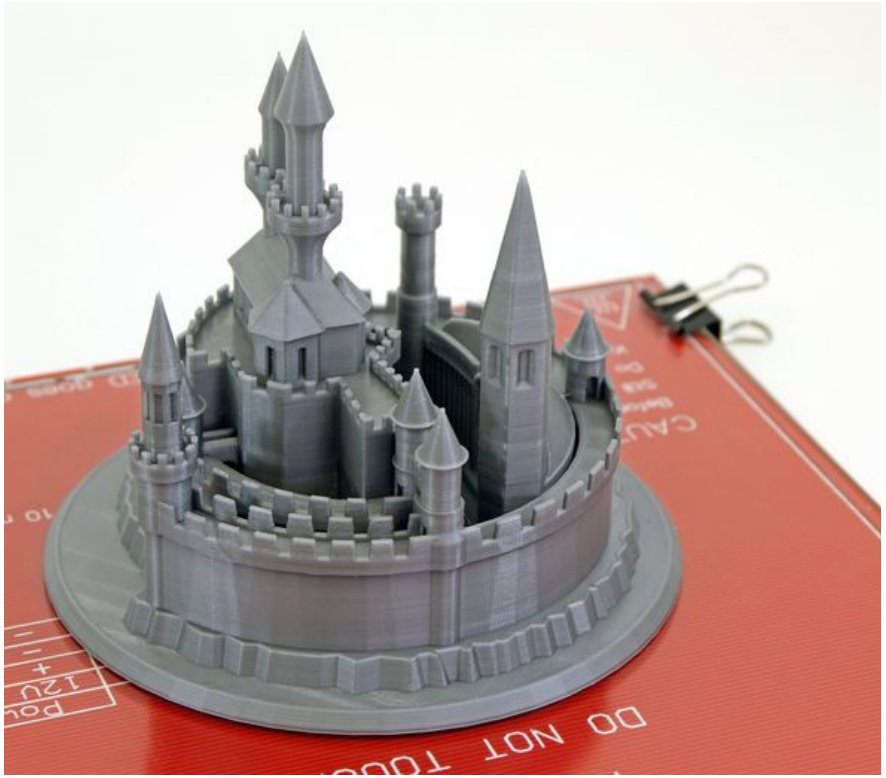
- vanilla 3D or full volume rendering (CUDA†, GPU†)

Interaction†

- „steering“

Measurements…

3D printing



© 2016, Prusa Research

3D printing

3D model editor†

- CSG, triangle-mesh…

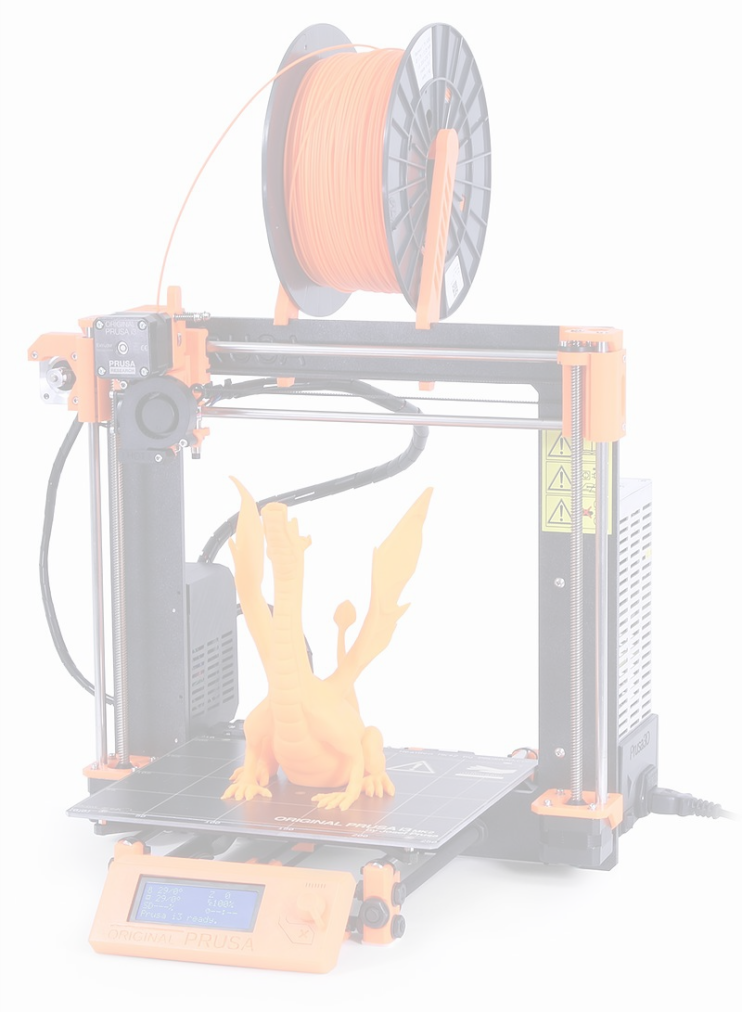
„Rendering“, rasterization

- similar to 2D rasterization*

Geometric optimization

- stiffness simulation?

Appearance modeling



Realistic rendering – Corona

© Bertrand Benoit, Pavel Stavila

Realistic rendering

3D scene model*

3D editing†

- 3DS Max, Blender, Rhinoceros

Materials*†

- surface appearance, textures†

Lighting†

- primary light sources + global illumination (GI) simulation†

HDR results*

Computer animation

© 1995, Pixar Animation Studios

Computer animation

© 2007, DreamWorks Animation SKG

Computer animation

© 2007, DreamWorks Animation SKG

Computer animation

© 2015, Pixar Animation Studios,
Walt Disney Pictures

Computer animation

3D scene model*

3D/animation editing†

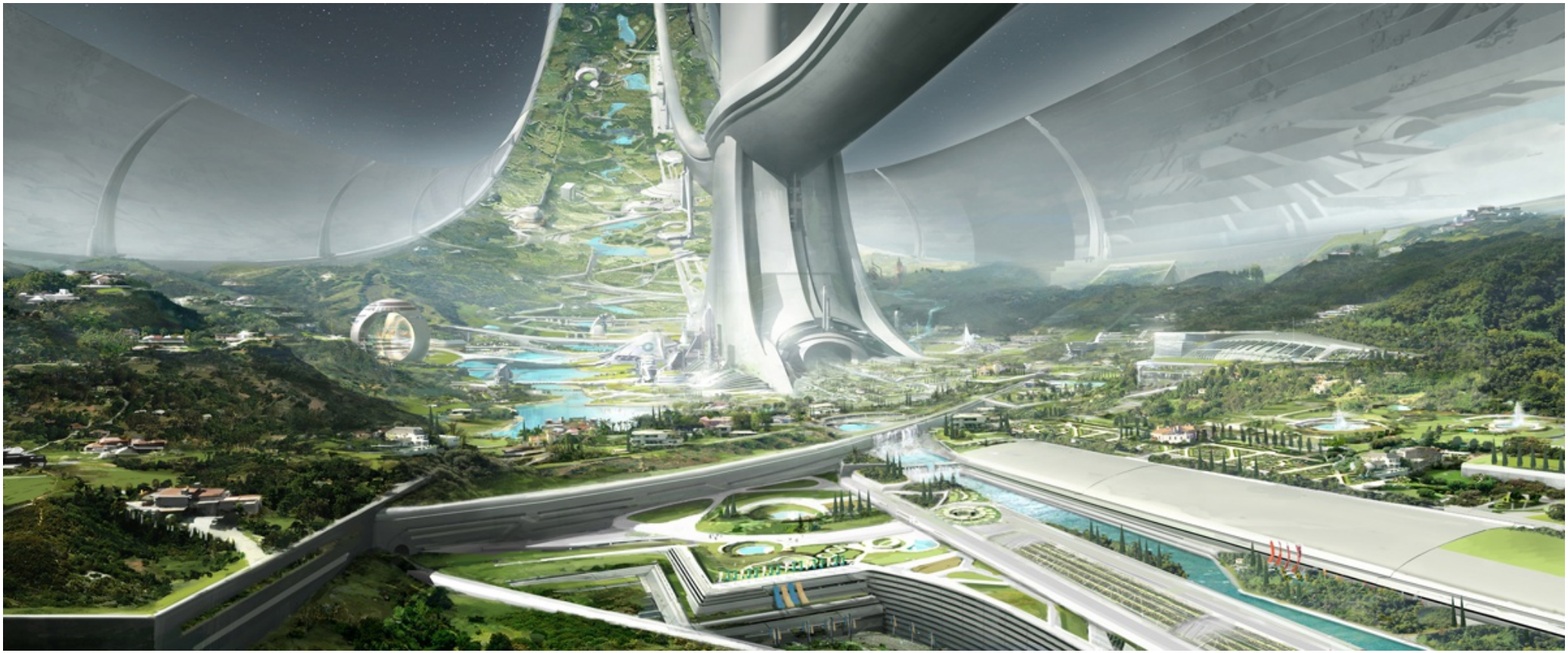
realistic rendering*†

- off-line (CUDA†, GPU†)
- materials, textures, appearance models
- lighting with GI

video-compression†

- off-line

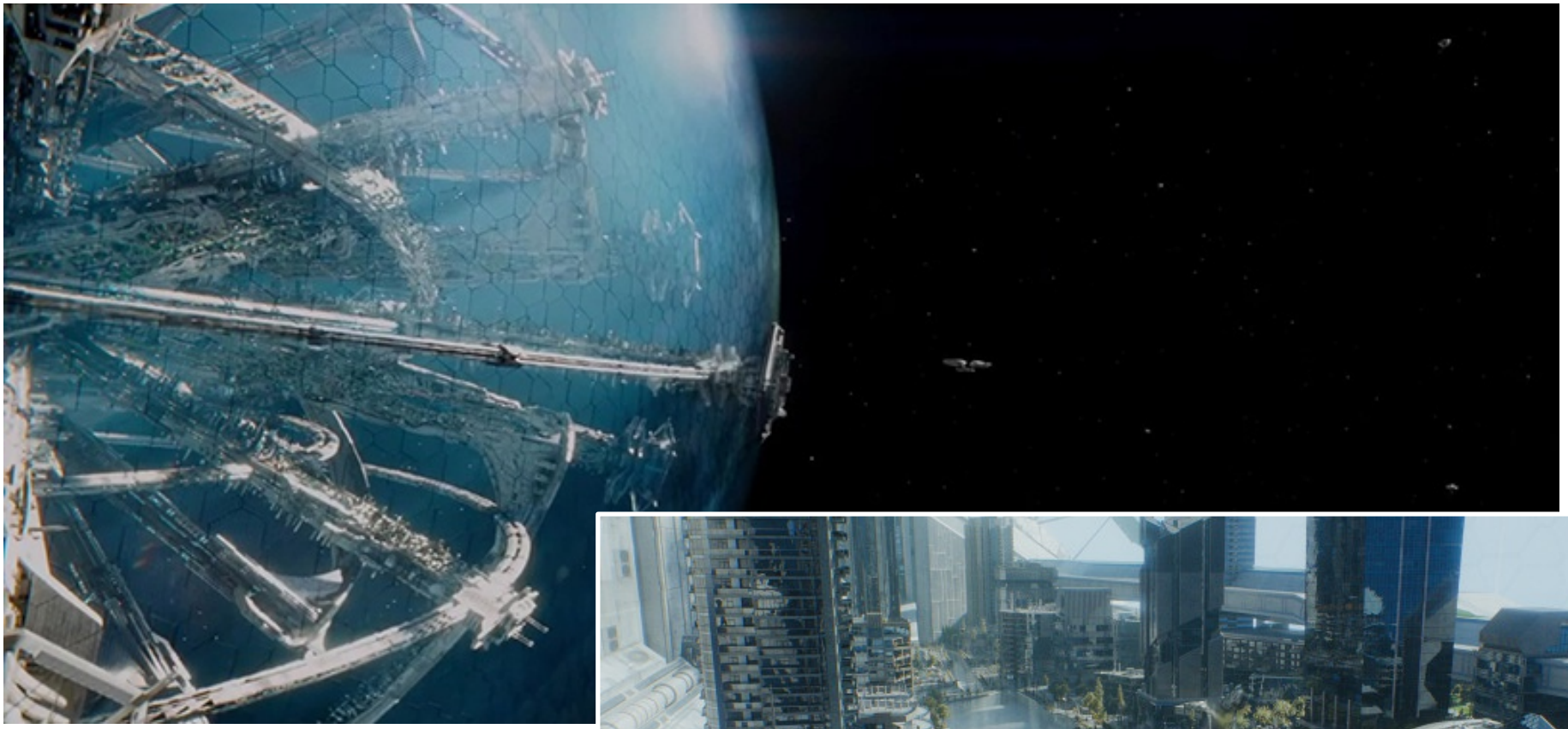
CGI in film – Elysium



© 2013, TriStar Pictures

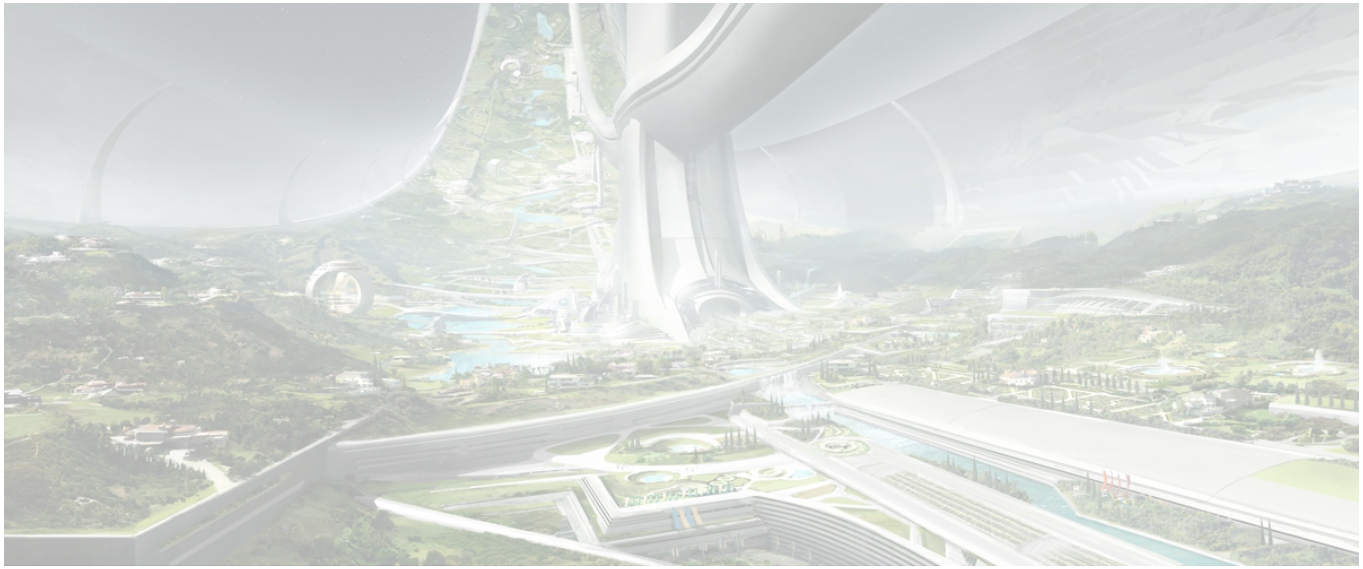
CGI in film – Star Trek into Darkness

CGI in film – Star Trek Beyond



© 2016, Double Negative, Paramount Pictures

CGI in film



3D scene model*

3D/animation editing†

Photo-realistic rendering*†

- off-line (CUDA†, GPU†)
- materials, textures, appearance + global illumination

Video-compression†

- off-line

VFX – The Perfect Storm

The Perfect Storm

Numeric ocean-water model!

- including realistic rendering of water

3D/animation editing†

Video composition and compression†

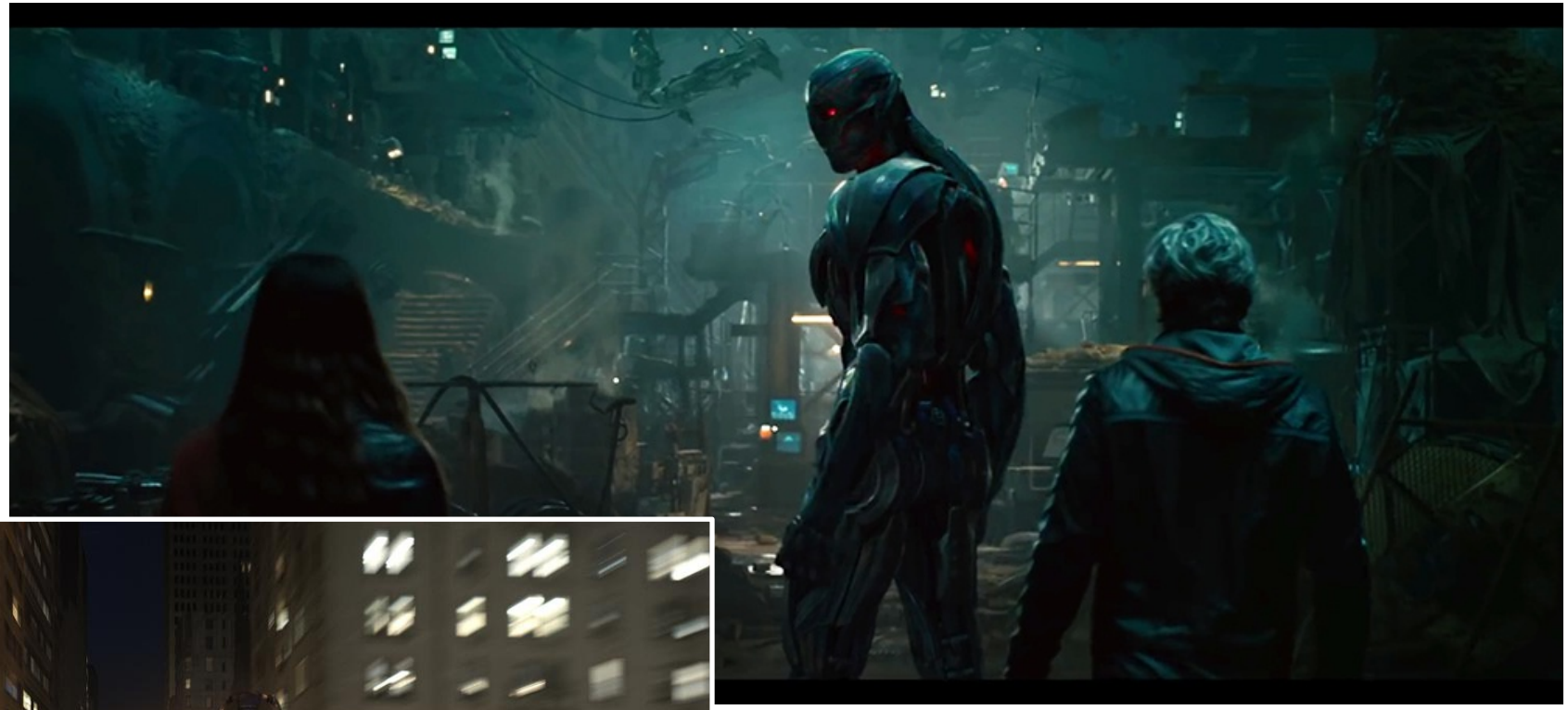
- off-line

VFX – The Perfect Storm

VFX – Marvel

© Marvel Studios, Paramount Pictures, IL&M…

VFX – Marvel



© Marvel Studios, IL&M…

VFX – Tron Legacy

VFX – Tron Legacy

© 2010, Disney Enterprises, Inc.

VFX – Tron Legacy (color scheme)

VFX – Tron Legacy

Motion capture!

- incl. green-screen keying

3D scene model*

3D/animation editing†

Photo-realistic rendering*†

- off-line (CUDA†, GPU†)
- materials, textures, appearance + global illumination

Video-compression†

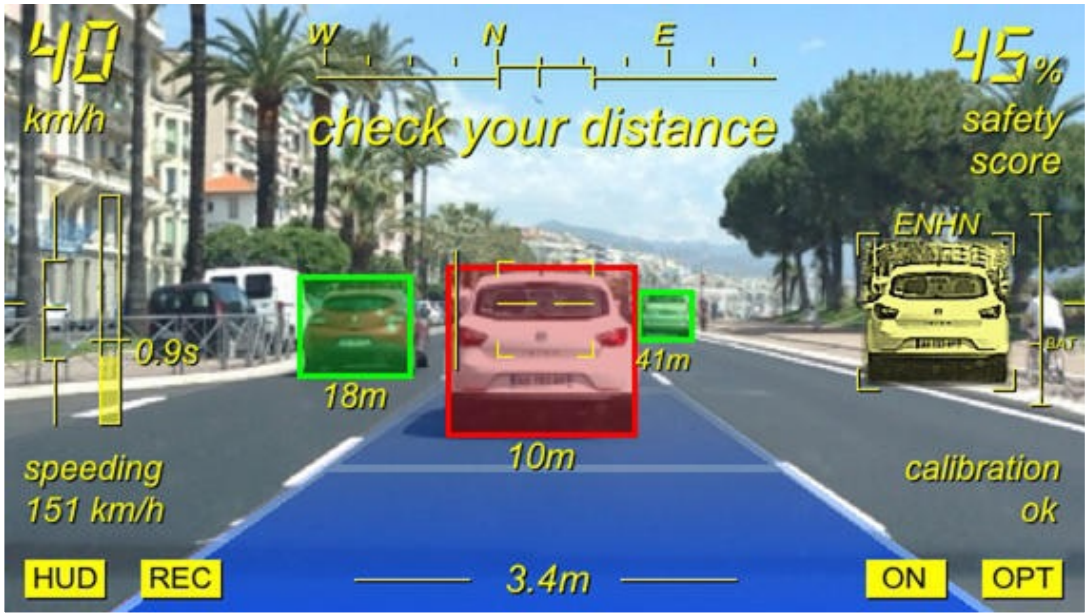
- off-line

Self-driving car

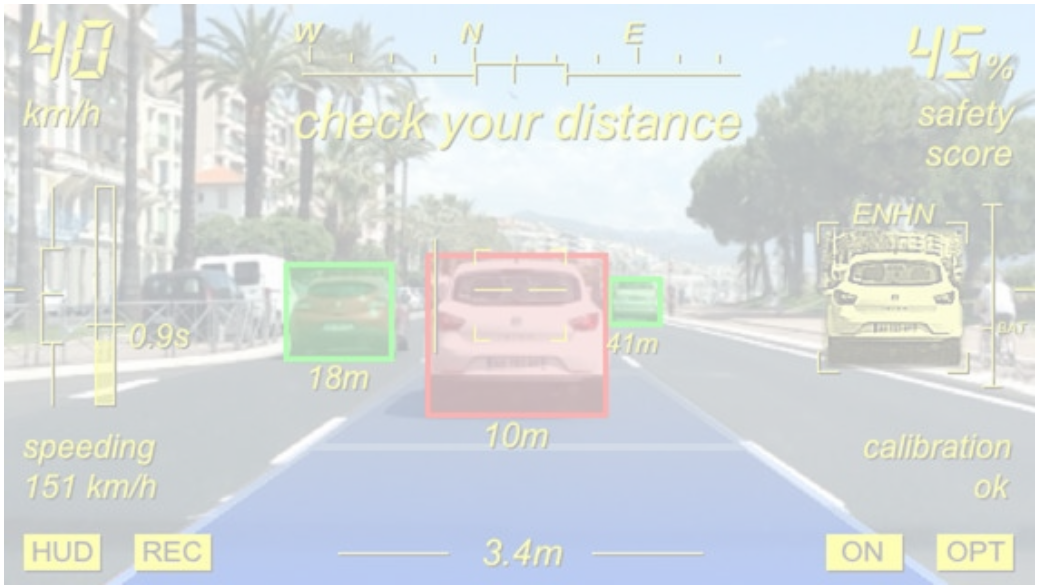
© Tesla Motors ?

Self-driving car

© Volvo



Self-driving car



Real-time camera data

Camera calibration†

- as accurate 3D context as possible

3D computer vision†

- robust!
- real-time! (no lags)

Prediction, planning

- artificial intelligence

Actual steering

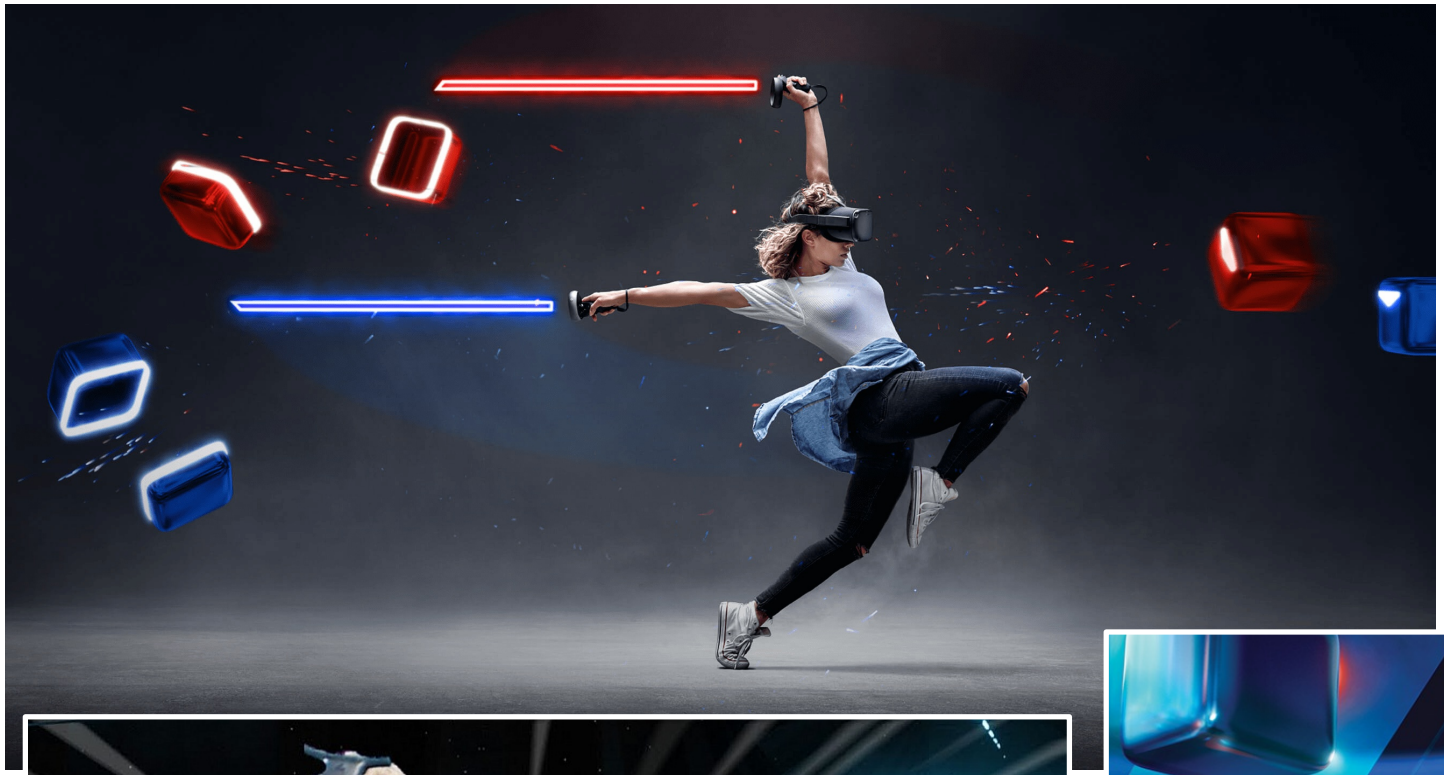
Videogame – DayZ (Arma II mod)

Videogame – Kingdom Come: Deliverance

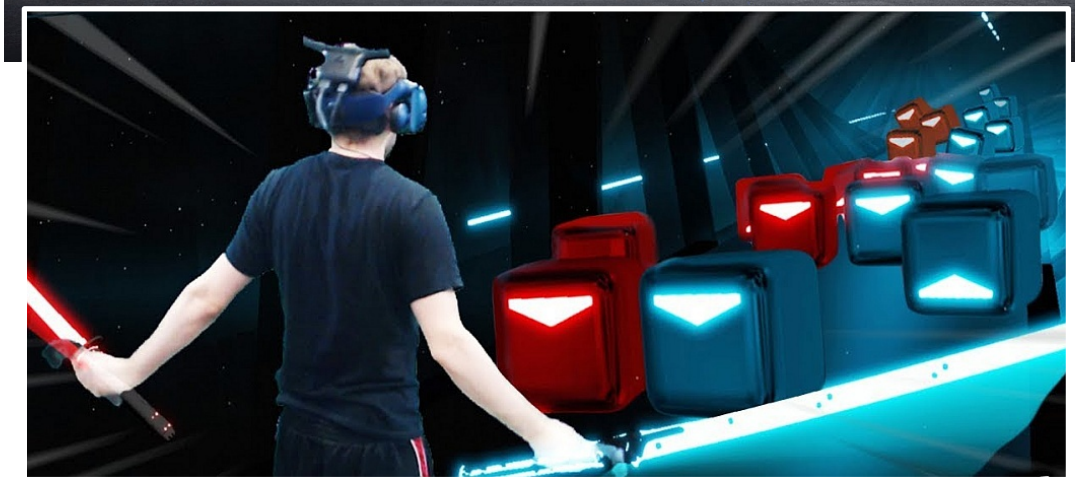
© 2016-2017 Warhorse Studios

Videogame – Overwatch

Videogame – Beat Saber



© Oculus, Beat Games



Videogames

3D editing, tools

Game logic†

- interaction among virtual objects

User interaction†

Real-time rendering†*

- constant FPS, textures, LoD, GPU shaders†
- scene virtualization (potentially infinite scene)…

Agents, AI players†

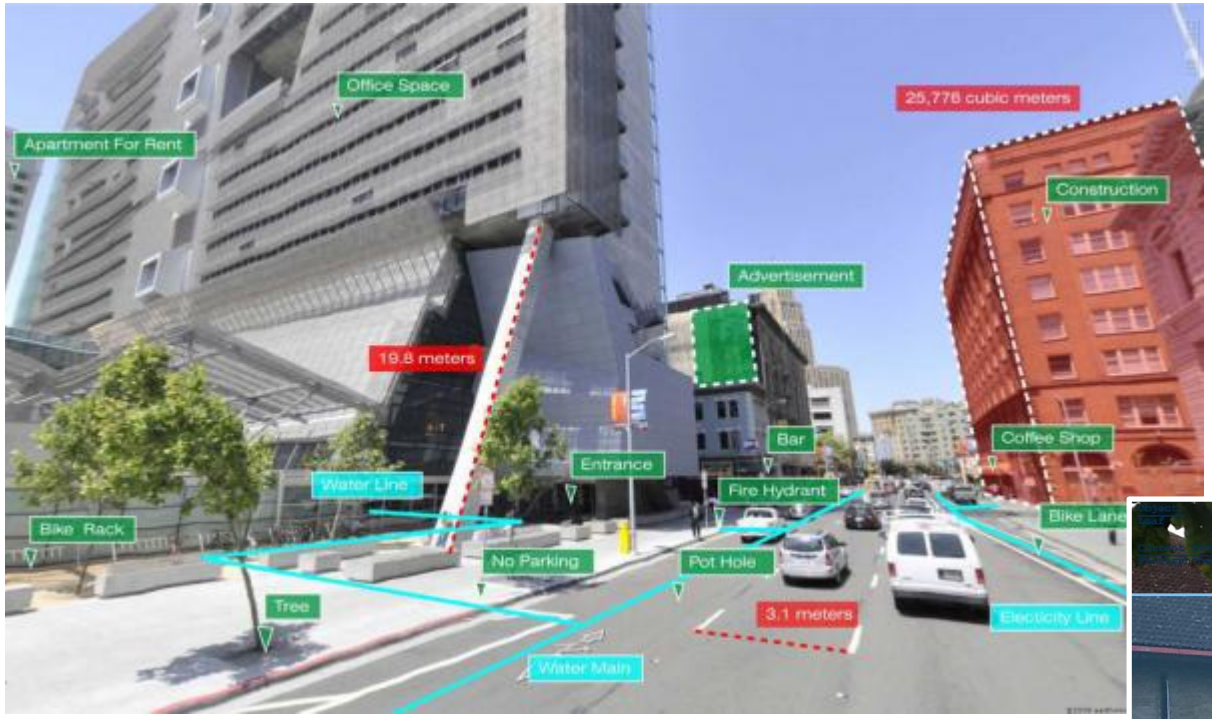
Multiplayer

- LAN layer, lag compensation…

Virtual reality – „Cave“

© 2011, Land Rover

Augmented reality – „Smart glasses“



© Google, Stormy's Media Mountain

Augmented reality – „Smart glasses“

© 2016, Epson (Moverio BT 300)

Augmented reality – military

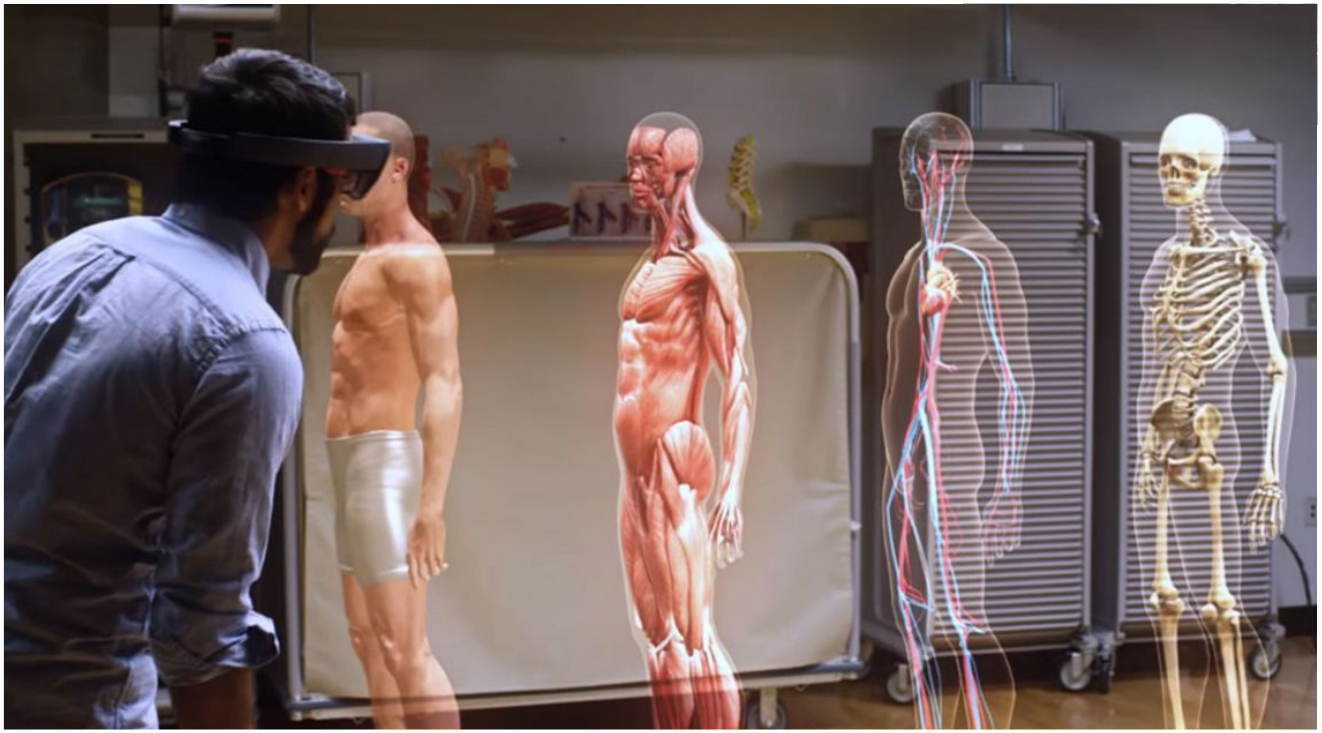
Augmented reality – phone

© 2012, JP

Augmented reality – tablet

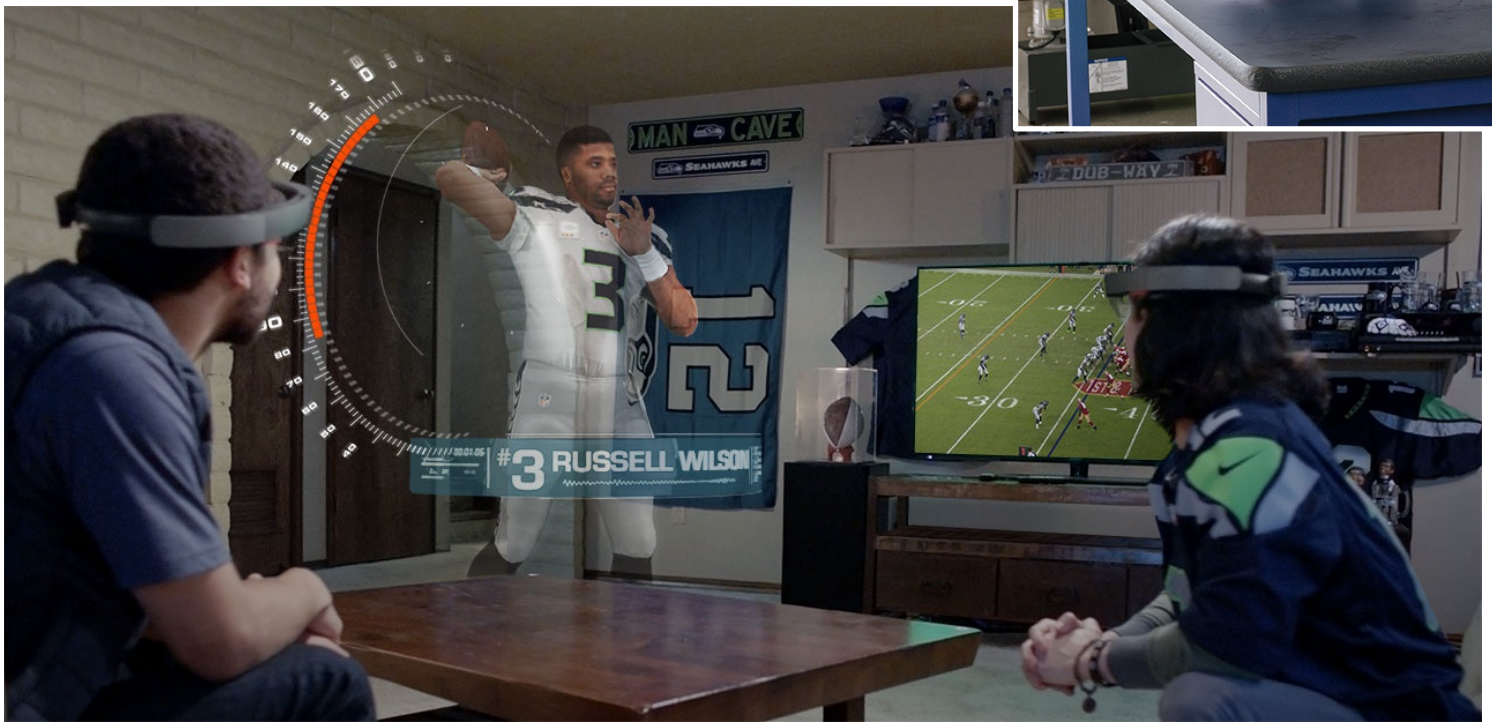
© RE'FLEKT GmbH

Augmented reality – HoloLens



© 2016 Microsoft

Augmented reality – HoloLens



Augmented reality

Virtual 3D scene*

3D position†

- computer vision („inside-out tracking“…)
- accelerometers, gyroscopes, magnetometers

Real-time rendering*†

- GPU, shaders†
- no lags can be tolerated!

Interactivity†

- computer vision† (hand gestures), discrete controllers…

Základy počítačové grafiky
NPGR 003

Zimní semestr 2/2 Z, Zk

Obsah a forma

Základy 2D i 3D grafiky

- navazují na ni
  - » Fotorealistická grafika (NPGR004) v LS
  - » Realtime grafika na GPU (NPGR019) v LS

2/2 Z, Zk

- přednáška i cvičení jednou týdně
- cvičení – ukázky a úlohy v C#
- komplexní hodnocení písemné zkoušky dohromady s výsledky zápočtových úloh

Stručný plán přednášky 2D

Rastrová a vektorová grafika (2)

- rastrový obraz, průhlednost, HDR grafika, operace s rastrovými obrázky, vektorový formát SVG

Barvy, jejich vnímání a zobrazování (2)

- barevné vidění, barevné prostory (RGB, CMYK, HSV), zobrazování barev, půltónování a rozptylování

Kódování rastrových obrázků (1)

- kódování, grafické formáty (JPEG+JFIF, GIF, PNG…)

Rastrové kreslení (1)

- kreslení úseček, křivek, vyplňování, ořezávání…

Stručný plán přednášky 3D

Matematika pro 3D grafiku (1)

– lineární transformace, homogenní souřadnice, projekce

Reprezentace 3D scén (1)

– výčtové, objemové a povrchové reprezentace, hierarchie

Úvod do OpenGL (1+lab)

Základy animace (1)

– rotace, trajektorie, interpolační křivky

Zobrazování 3D scén, stínování, viditelnost (2)

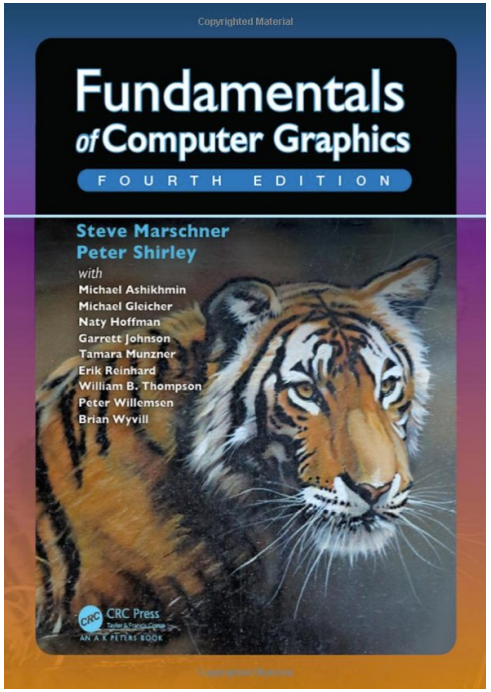
– příklady algoritmů na viditelnost, základy stínování a renderingu, vrhání a sledování paprsku (ray-tracing)

Literatura (CZ)

Jiří Žára, Bedřich Beneš, Jiří Sochor, Petr Felkel:
Moderní počítačová grafika, 2. vydání,
Computer Press, 2005, ISBN: 80-251-0454-0

Literatura (US)

S. Marschner, P. Shirley: ***Fundamentals of Computer Graphics***, 4rd edition, A K Peters / CRC Press, 2015



J. Foley, A. van Dam, S. Feiner, J. Hughes: ***Computer Graphics, Principles and Practice***, 2nd edition in C, Addison-Wesley, 1995

Předpoklady

Základní kurs programování

- algoritmy, datové struktury

Základy programování v jazyku C#

- nejsou potřeba detaily jazyka ani knihovny
- na cvičeních budete mít připraveny šablony pro vaši práci

Základní kurs matematické analýzy a lineární algebry

Užitečné adresy

Aktuální informace na WWW

- https://cgg.mff.cuni.cz/prednasky.cz.php
- https://cgg.mff.cuni.cz/~pepca/

Podpora pro cvičení

- https://cgg.mff.cuni.cz/~pepca/grcis/
- svn://cgg.mff.cuni.cz/grcis/trunk/

Facebook CGG

- https://www.facebook.com/CGGMFF

Další vhodné grafické předměty (ZS)

Seminář z počítačové grafiky a vidění: 0/2, NPGR005

Introduction to Colour Science: 2/0, NPGR025 (Alexander Wilkie)

Geometrické modelování: 2/2, NPGR021 (Zbyněk Šír)

Digitální zpracování obrazu: 3/0, NPGR002 (Jan Flusser, ÚTIA AV ČR)

3D počítačové vidění: 2/2, NPGR001 (Václav Hlaváč, CIIRC)

Strojové učení v počítačovém vidění: 2/2, NPGR035 (Elena Šikudová)

Animovaná a grafická tvorba: 1/1, NPGR039 (Ondřej Javora, FF UK)

Doporučení pro nadšence

HiVisComp

Konference **HiVisComp**

- každoročně v zimě (s lyžováním), setkávají se tam počítačoví grafici a fanoušci příbuzných oborů z ČR, Slovenska, Německa, Rakouska…
- **https://www.hiviscomp.cz/**

Studentská konference **CESCG**

CESCG

- prezentují se studentské příspěvky a projekty
- Slovensko, Rakousko, ČR, Německo, Francie…
- **https://www.cescg.org/**

matfyz

Computer Graphics Charles University

Barevné vidění

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

http://cgg.mff.cuni.cz/~pepca/

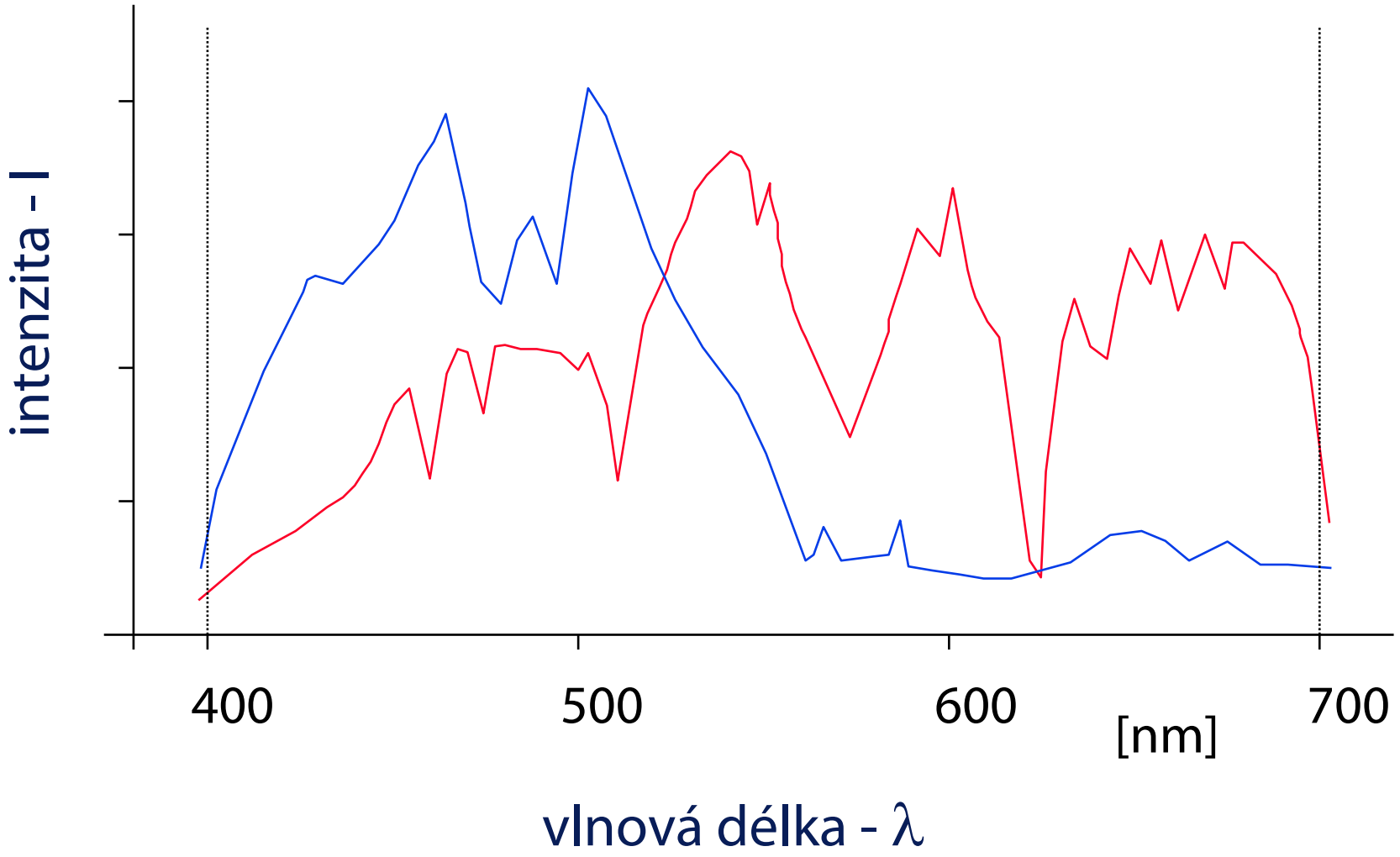
Co je světlo?

Špatnota bludy tvořitelská:

Newton stvořil blud, že Sluno vysílá ze sebe jemné částečky proti Huyghensovým ukám, že světlo jsou chvěje tenýra zrakovým čivem pojaté …

Jakub Hron: *„Skutky lidské, čili Jeden tisíc špatnot žijby a konby lidské"*, 1907

Viditelné světlo, spektrum



Barevný vjem

Prostor všech spekter má **nekonečnou dimenzi**

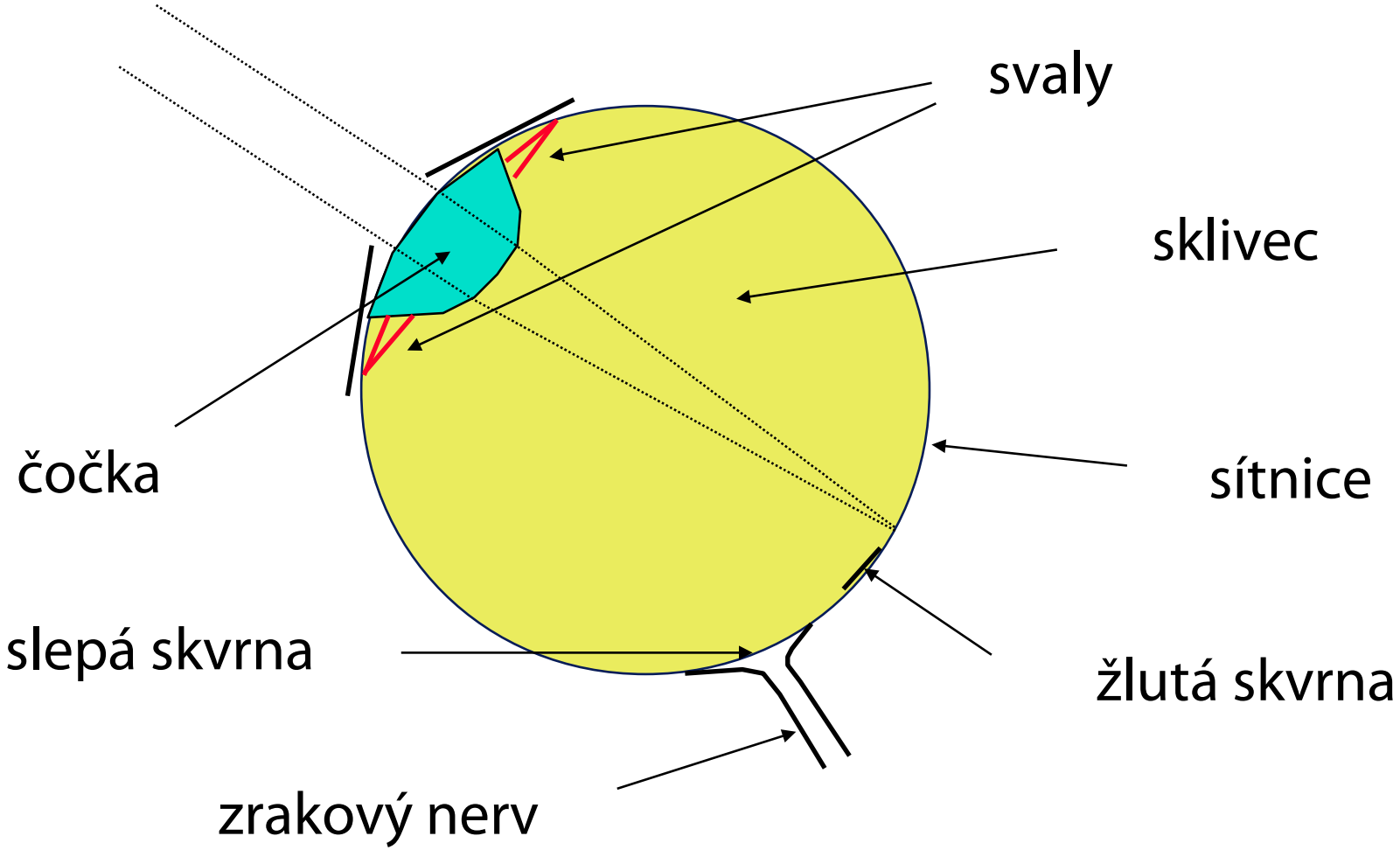
- systém lidského vidění je však nedokáže všechny rozeznat („metamers")

Grassmanovy zákony (1854) – lidské oko vnímá:

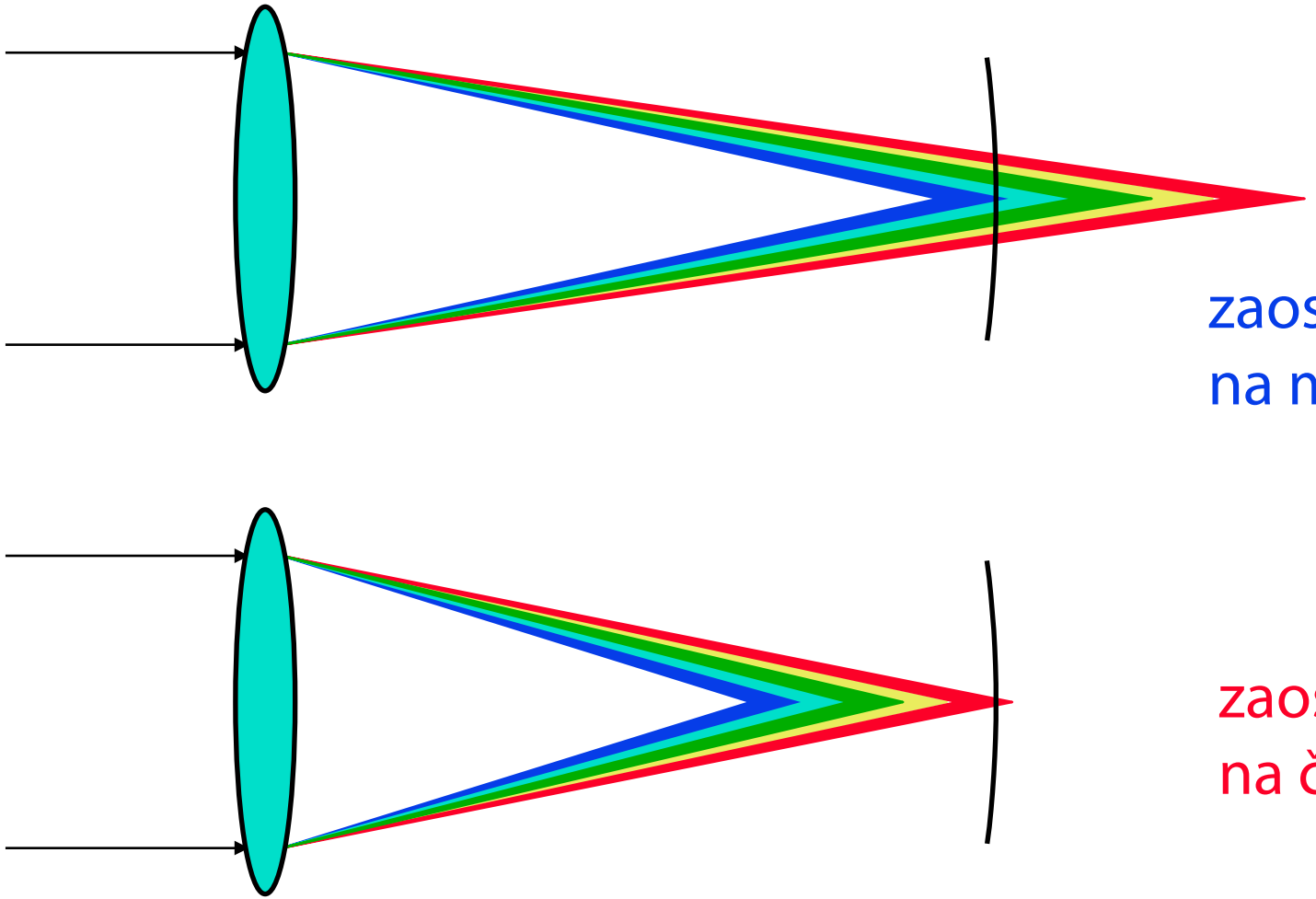
- **dominantní vlnovou délku** (odstín, „hue")
- **čistotu barvy** (sytost, „saturation")
- **intenzitu** (jas, „brightness")

Barvy lze aditivně skládat ($\mathbf{A = B, C = D \Rightarrow A + C = B + D}$)

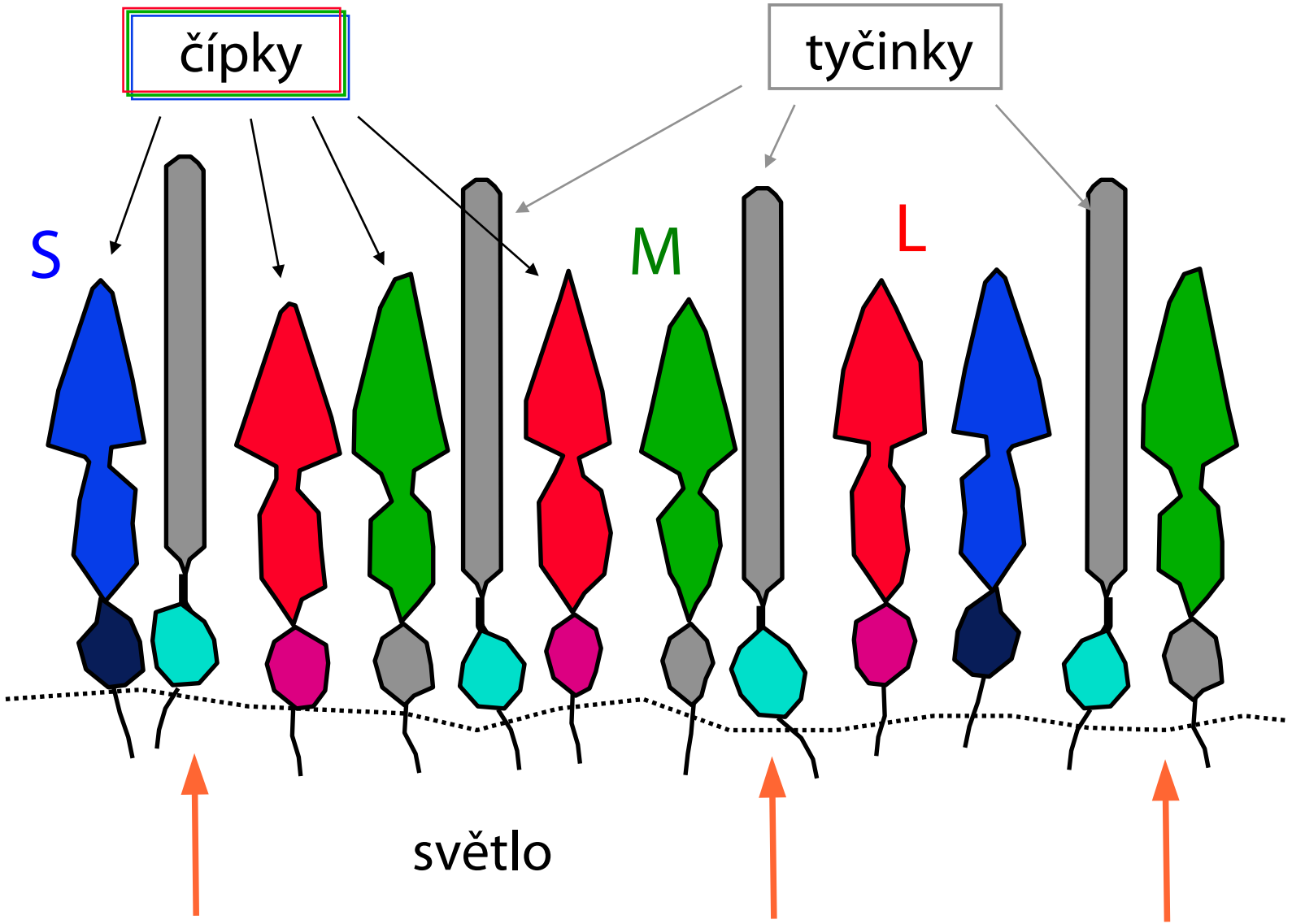
Lidské oko


svaly
sklivec
čočka
sítnice
slepá skvrna
žlutá skvrna
zrakový nerv

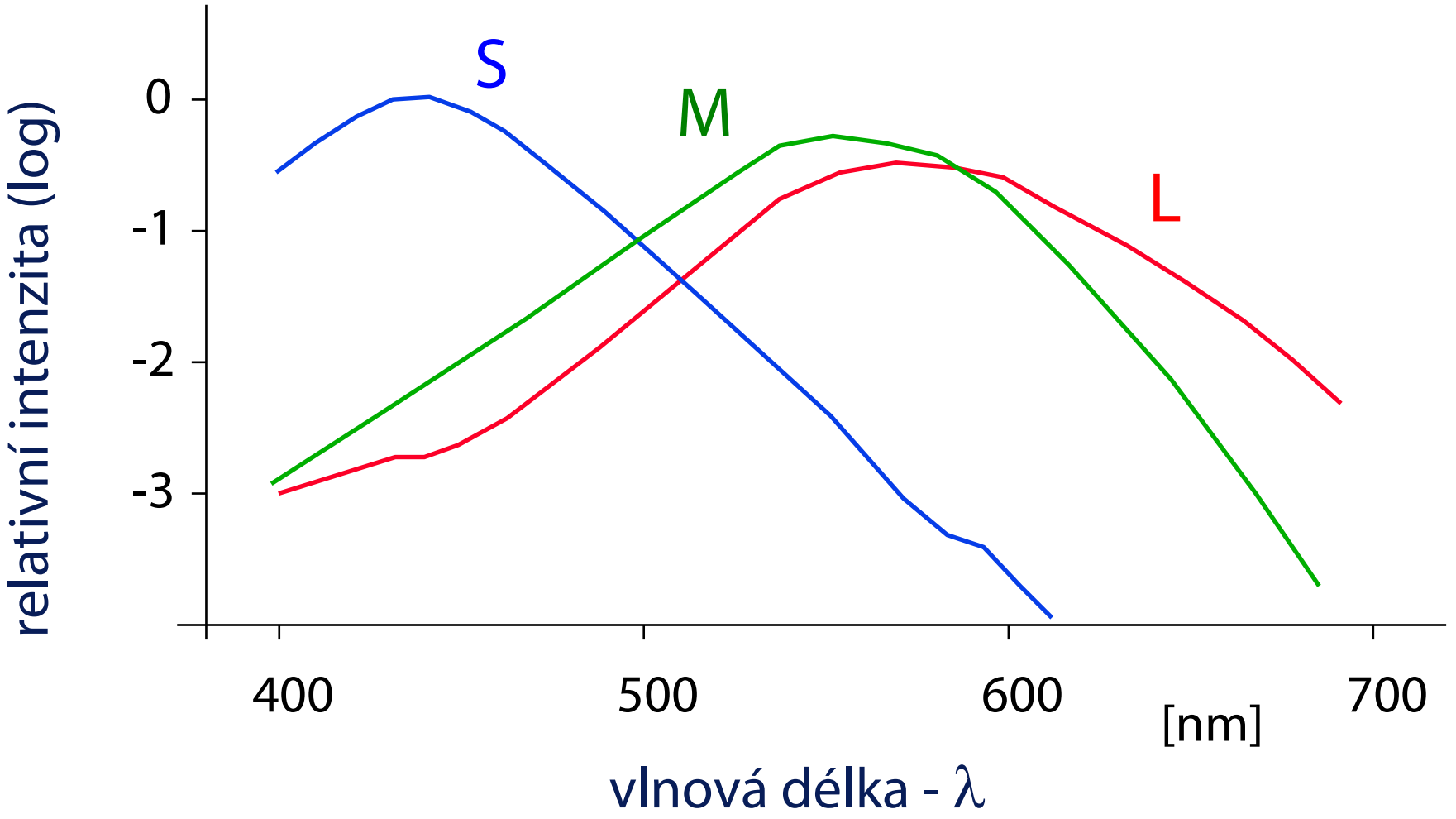
Barevná aberace



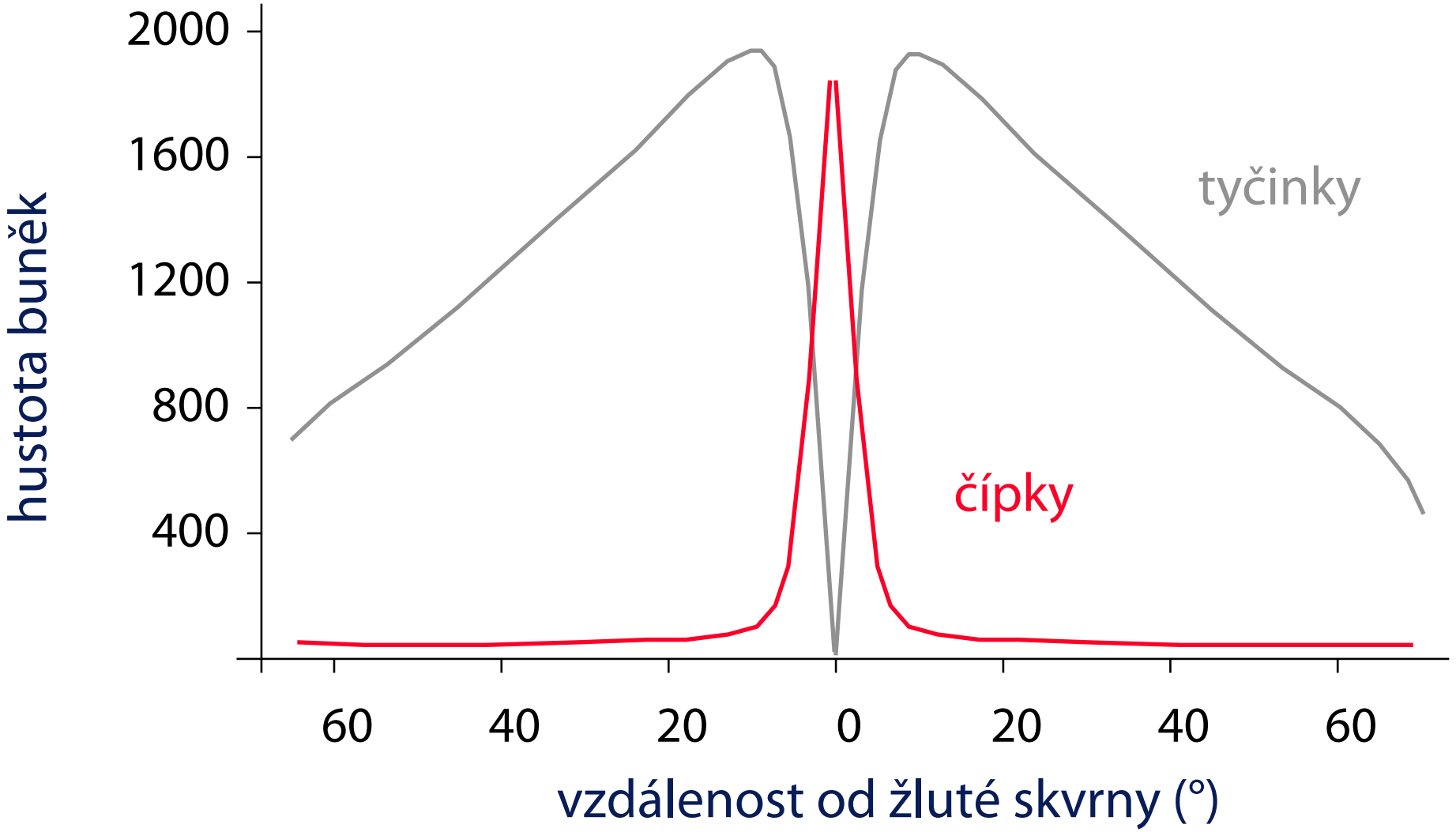
Sítnice



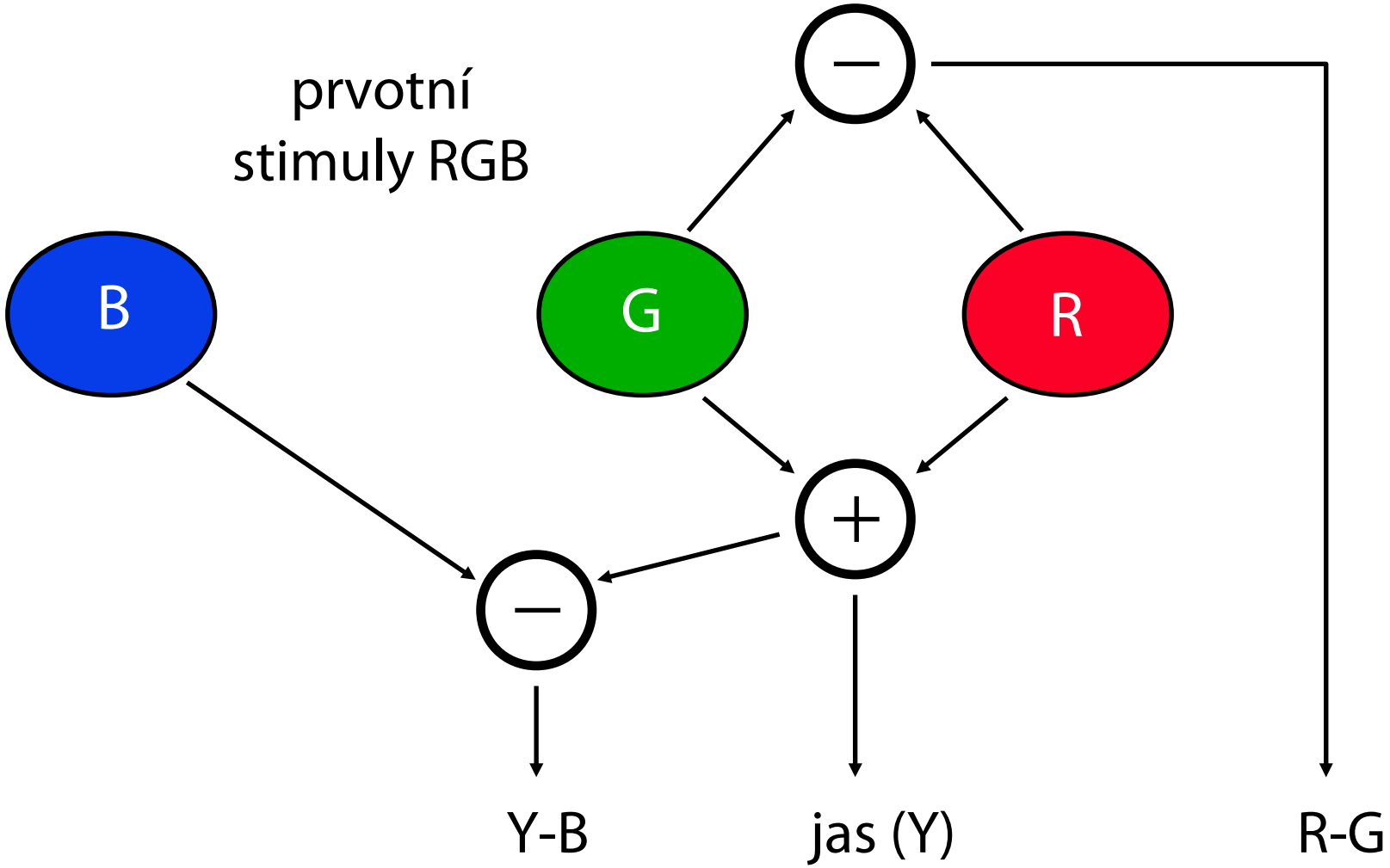
Tři fotopigmenty



Rozložení fotoreceptorů



Předzpracování barev



Vlastnosti systému vidění I

Různá citlivost na **červenou** (0.3), **zelenou** (0.6) a **modrou** (0.1) barvu

- navíc střed žluté skvrny téměř neobsahuje „modré" čípky

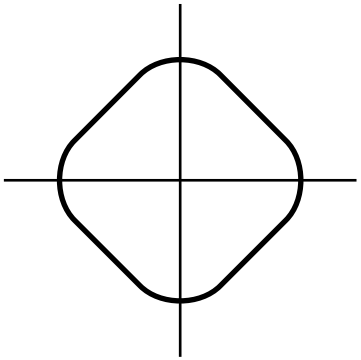
Zaostřuje se podle **jasové složky** (Y = R + G)

- nelze dobře zaostřit na rozdíly v modré složce

Integrační schopnost sítnice

- vnímáme samostatné tečky a zároveň jejich hustotu
- umožňuje použít rozptylovací metody (dithering)

Vlastnosti systému vidění II



Větší rozlišovací schopnost ve svislém a vodorovném směru

- v šikmých směrech asi o 30% menší

Přeostřování na barvy vzdálené ve spektru

Setrvačnost („afterimage”)

- [chemické] vyčerpání některých receptorů

Očekávání („expectation”)

- psycho-fyziologická vlastnost

Vlastnosti systému vidění III

Vliv okolí („surround")

- vjem barvy závisí na okolních barvách/intenzitách
- hnědá barva „neexistuje"

Čočka a sklivec se zbarvují stále více do žluta

- ve stáří klesá schopnost vidět krátké vlnové délky

Vady barevného vidění

- splynutí „červeného" a „zeleného" pigmentu (nebo absence jednoho z nich) – nejčastější vada
- chybí „modrý" pigment
- chybějí čípky vůbec („monochromats")

Doporučení

Používat barvy střízlivě

- maximálně 4-6 různých barev, odstínů může být víc

Nekreslit modrou barvou malé objekty a tenké čáry

- málo „modrého” pigmentu ve středu žluté skvrny

Na pozadí nepoužívat červenou a zelenou

- modrá i žlutá vyhovují

Nekreslit vedle sebe syté barvy daleko ve spektru

Používat barvy logicky a konzistentně

Literatura

G. Murch: ***Human Factors of Color Displays***, in Advances in Computer Graphics II, Springer, 1986, 1-27

D. Pritchard: *U.S. Color Television Fundamentals - A Review*, IEEE Transactions on Consumer Electronics, vol. CE-23, #4, 467-478

J. Foley, A. van Dam, S. Feiner, J. Hughes: ***Computer Graphics, Principles and Practice***, 574-579

matfyz

Computer Graphics Charles University

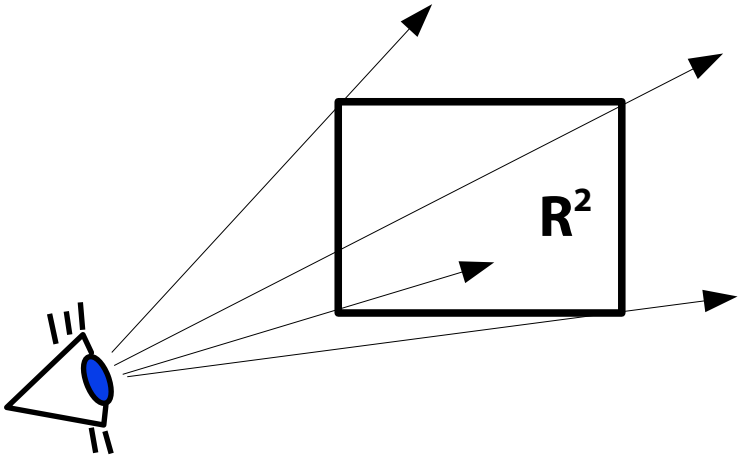
Rastrová a vektorová grafika

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

http://cgg.mff.cuni.cz/~pepca/

Obrazová funkce



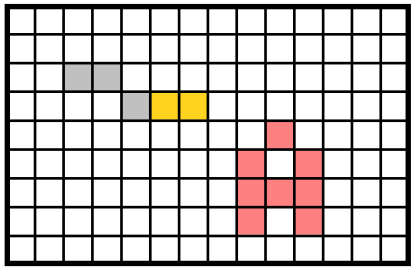
„Okno“ do reálného spojitého světa

- zobrazení $\mathbf{R^2} \rightarrow$ „barva”
- nekonečně zvětšovatelný obraz

Diskretizace obrazu

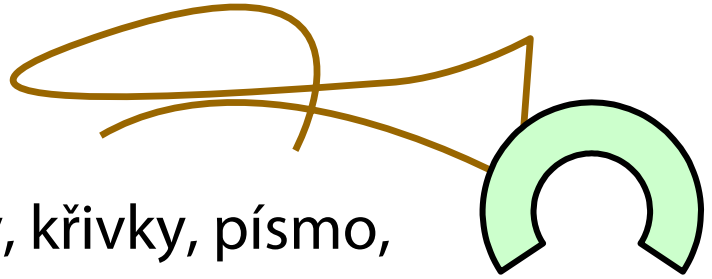
- vzorkování roviny v pravidelné mřížce
- matice pixelů
- praxe: snímací sensor fotoaparátu, kamery
- druhá diskretizace – hodnoty pixelů (viz později)

Rastrový vs. vektorový přístup



Rastrový výstup

- jsou přímo ovládány (adresovány) jednotlivé **pixely**
- data jsou závislá na rozlišení (nelze jednoduše škálovat)



Vektorový výstup

- zobrazují se přímo složitější **objekty** (čáry, křivky, písmo, plošné útvary)
- data nejsou závislá na rozlišení (lze je škálovat až v zobrazovacím zařízení)

Grafický výstup

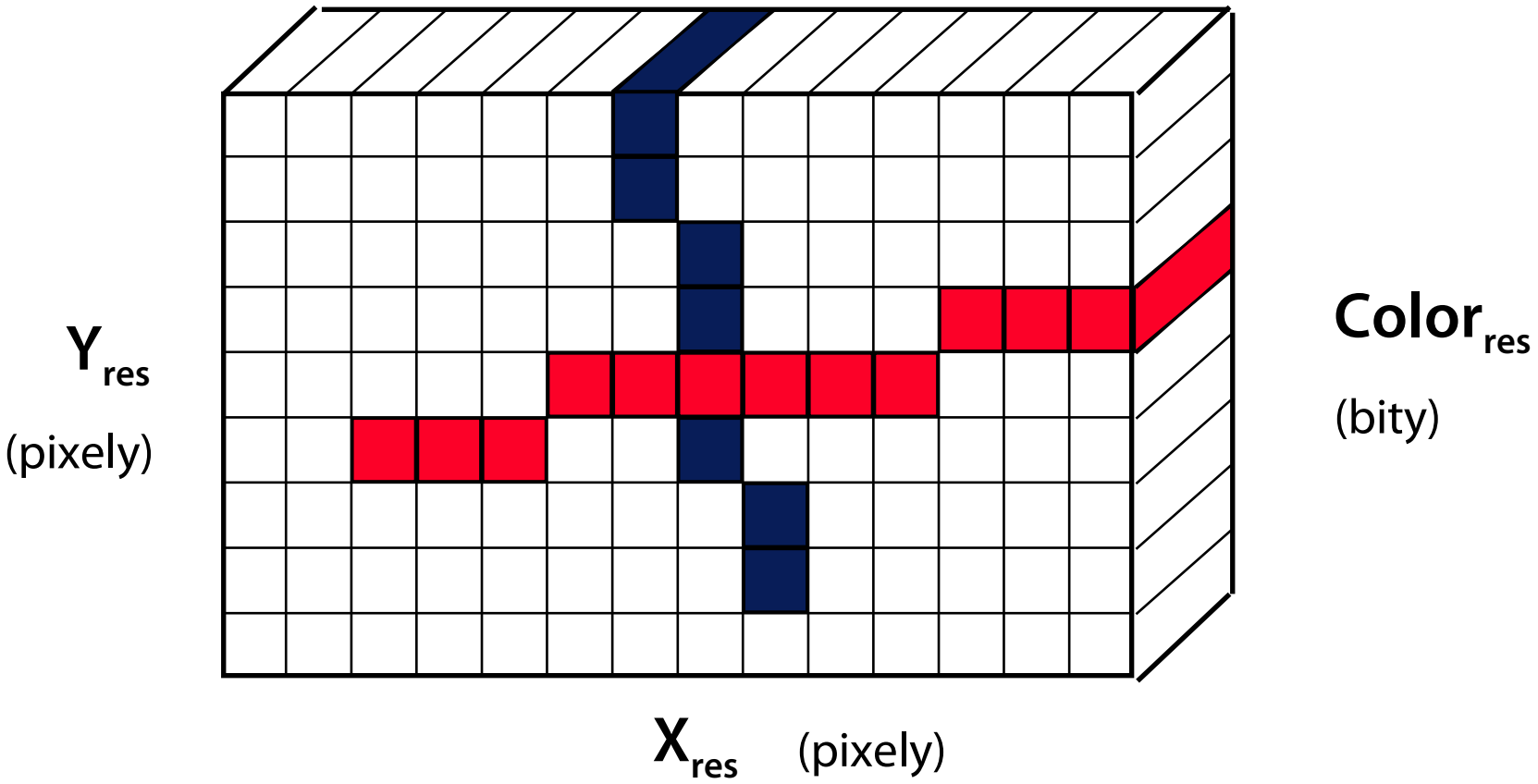
Podle **technologie výstupu**

- vektorový výstup (staré displeje, stolní plotter, starší osvitové jednotky?)
- rastrový výstup (displeje, tiskárny, plottery)

Podle **komunikace**

- vektorové zařízení (video-karty = GPU, SVG standard /W3C/, Adobe PDF, PostScript®)
- rastrové zařízení (běžné video-adaptéry, tiskárny v grafickém režimu)

Rastrový obraz



Např: **720×1280×8** bitů, **1920×1200×24** bitů,
3840×2160×24 bitů

Formát pixelu

Celočíselné hodnoty

- starší, klasický přístup
- obyčejně [8 bit (s paletou)], **3×8 bit** nebo **4×8 bit**

Plovoucí desetinná čárka

- HDR grafika („High Dynamic Range“)
- obyčejně **3× float** (96bit) nebo **3× half** (48bit)
- bez problémů se ztrátou přesnosti

Vektorové kreslení

Sada **vektorových příkazů** pro kreslení jednotlivých grafických primitiv

- čára („`moveto`“, „`lineto`“), křivka („`curveto`“)
- základní tvary („`rect`”, „`circle`“, „`polygon`“ …)

Definice **barev a vzorků pro vyplnění**

- základní přístupy: „`fill`“, „`stroke`”

Vykreslení **textu**

- všechny běžné typografické atributy (font, velikost, mezery, „kernings“ …)

Vektorový formát SVG

W3C standard

- všechny běžné současné WWW prohlížeče (HTML5)
- podpora pro animace
- uživatelský souřadný systém, 2D transformace, ořezávání …

Založen na **XML syntaxi**


```
<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 100 100">
 <path d="M30,1h40l29,29v40l-29,29h-40l-29-29v-40z" stroke="#000" fill="none"/>
 <path d="M31,3h38l28,28v38l-28,28h-38l-28-28v-38z" fill="#a23"/>
 <text x="50" y="68" font-size="48" fill="#FFF" text-anchor="middle"><![CDATA[410]]>
 </text>
</svg>
```

Barevný systém RGB

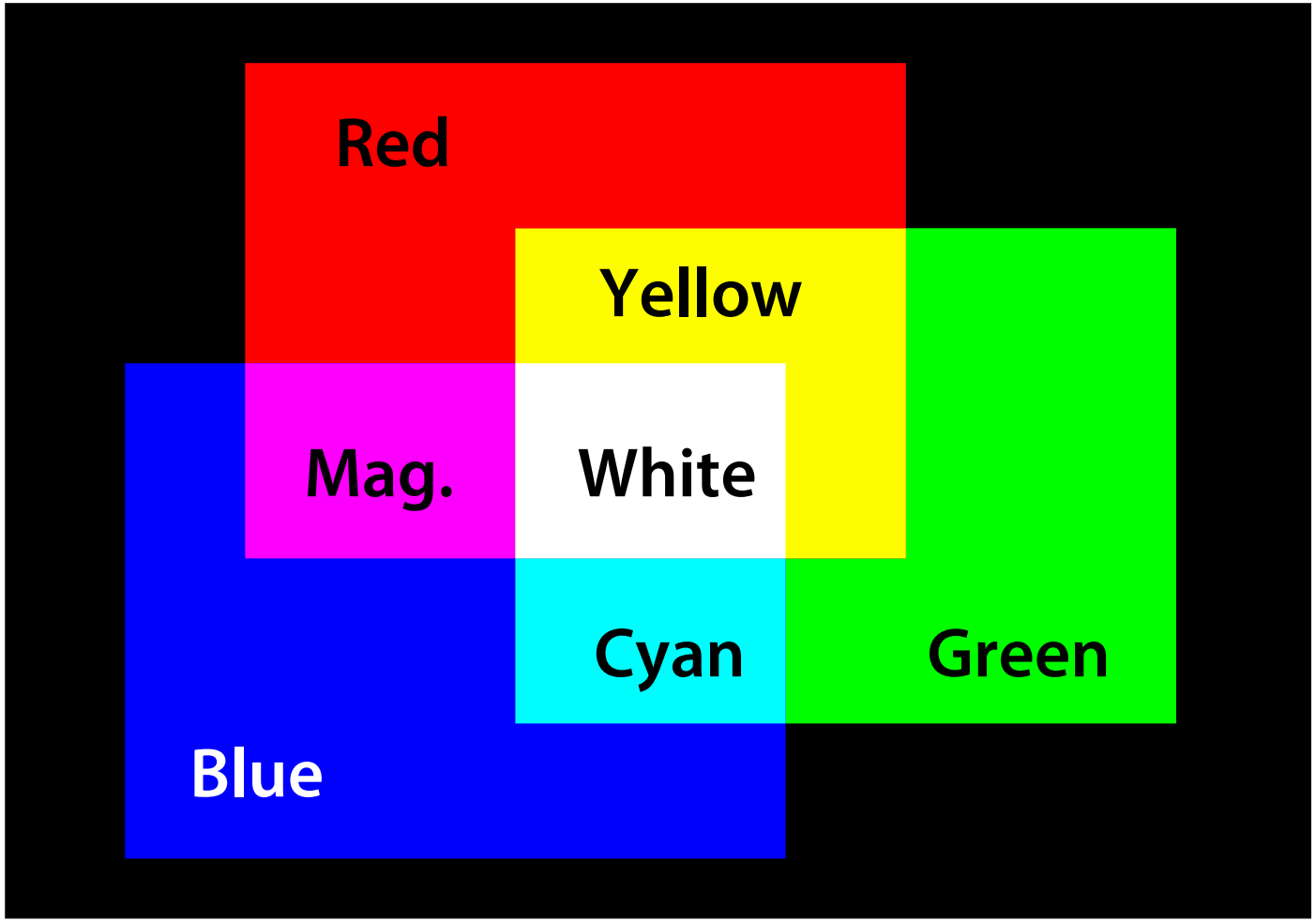
Základní **barevné složky**: červená, zelená, modrá

- vychází z aktivního zobrazování (staré CRT monitory)
- lidský zrakový systém vnímá podobně

Aditivní skládání barev

- černé pozadí (nulová barva, vypnutý displej)
- např. bílou dostaneme složením maxim všech tří složek

Aditivní skládání barev



Literatura

Další informace:

J. Foley, A. van Dam, S. Feiner, J. Hughes: ***Computer Graphics, Principles and Practice***, 8-15, 145-199

Jiří Žára a kol.: ***Počítačová grafika, principy a algoritmy***, 29-49

`https://www.w3.org/Graphics/SVG/`

matfyz

Computer Graphics Charles University

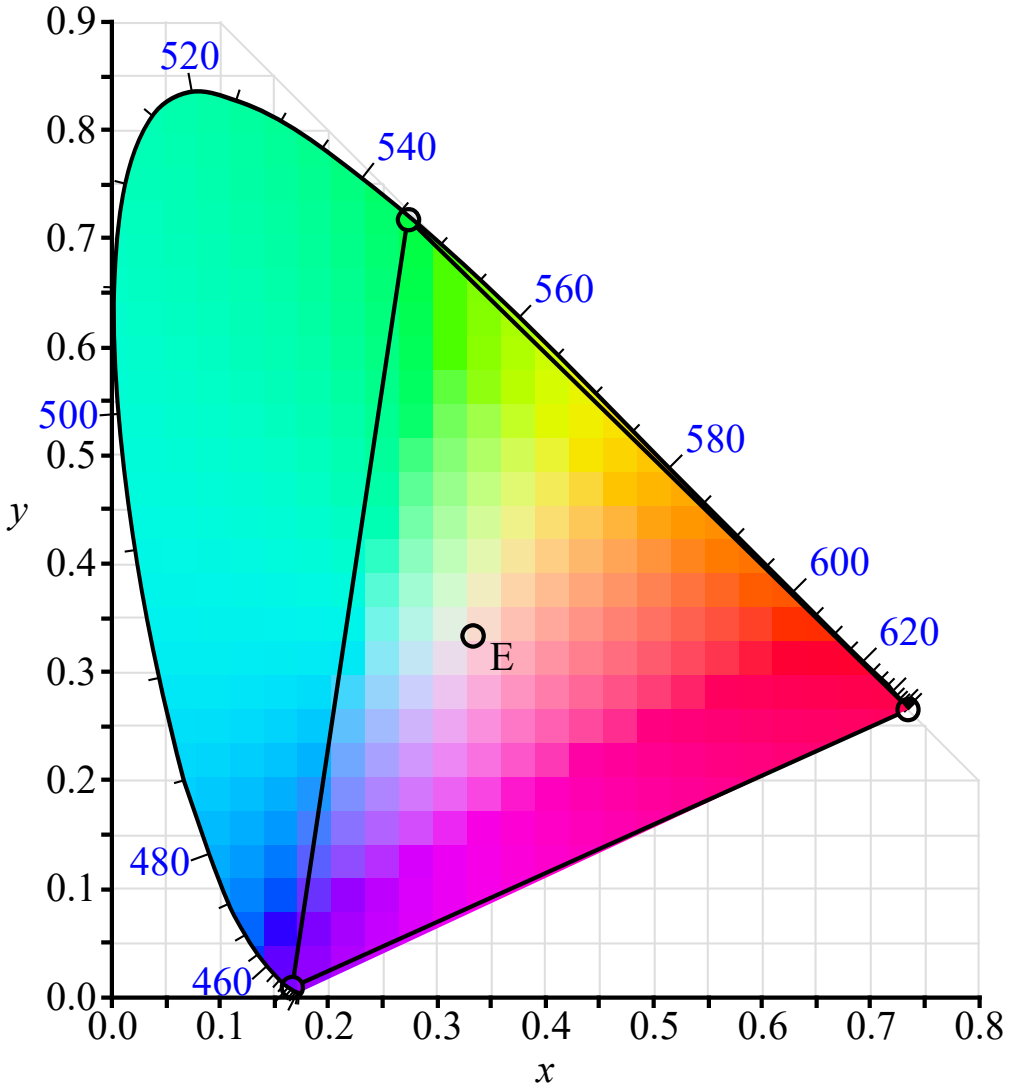
Barevné systémy

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

http://cgg.mff.cuni.cz/~pepca/

Viditelné barvy



CIE RGB primitiva:

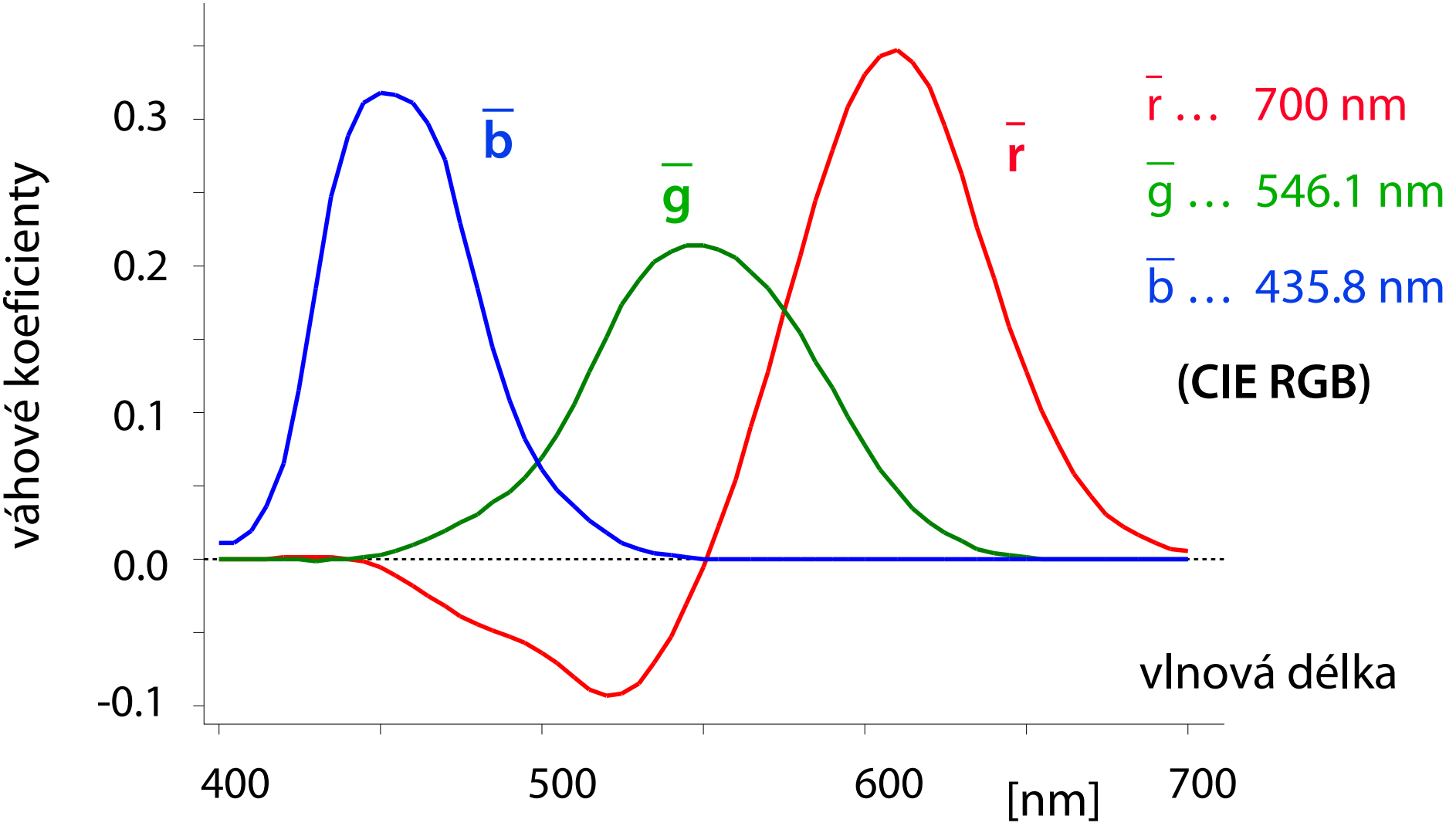
700 nm

546.1 nm

435.8 nm

© BenRG, Wiki

Rozklad spektrálních barev



Virtuální barevná primitiva X,Y,Z

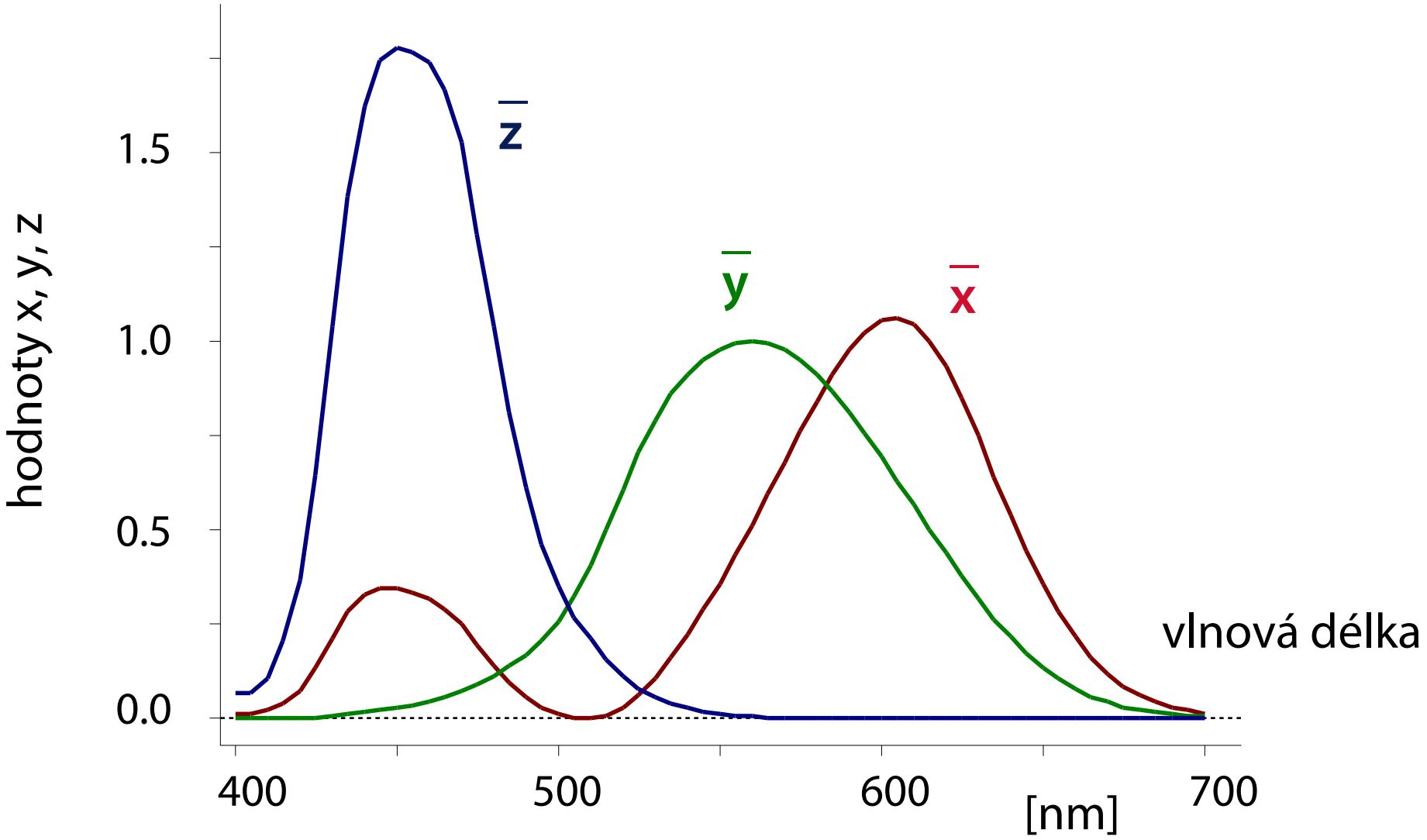
Commision Internationale de l'Éclairge (CIE) v roce 1931 definovala tři virtuální barvy X, Y, Z, jejichž konvexní kombinací již vytvoříme libovolnou viditelnou barvu

- $\underline{X}, \underline{Y}, \underline{Z}$ jsou definovány pomocí svých spektrálních charakteristik $\underline{\mathbf{x}}, \underline{\mathbf{y}}, \underline{\mathbf{z}}$ (tabelovaných po 1nm)
- **Y** … jas
- **Z** ~ modrý stimulus („S“ čípky)
- **X** … pozitivita

Závislost mezi složkami R,G,B a X,Y,Z je **lineární**

- převodní matice 3×3

Srovnávací funkce CIE



Barevný prostor CIE-xyY

Normalizované barevné složky x, y, z

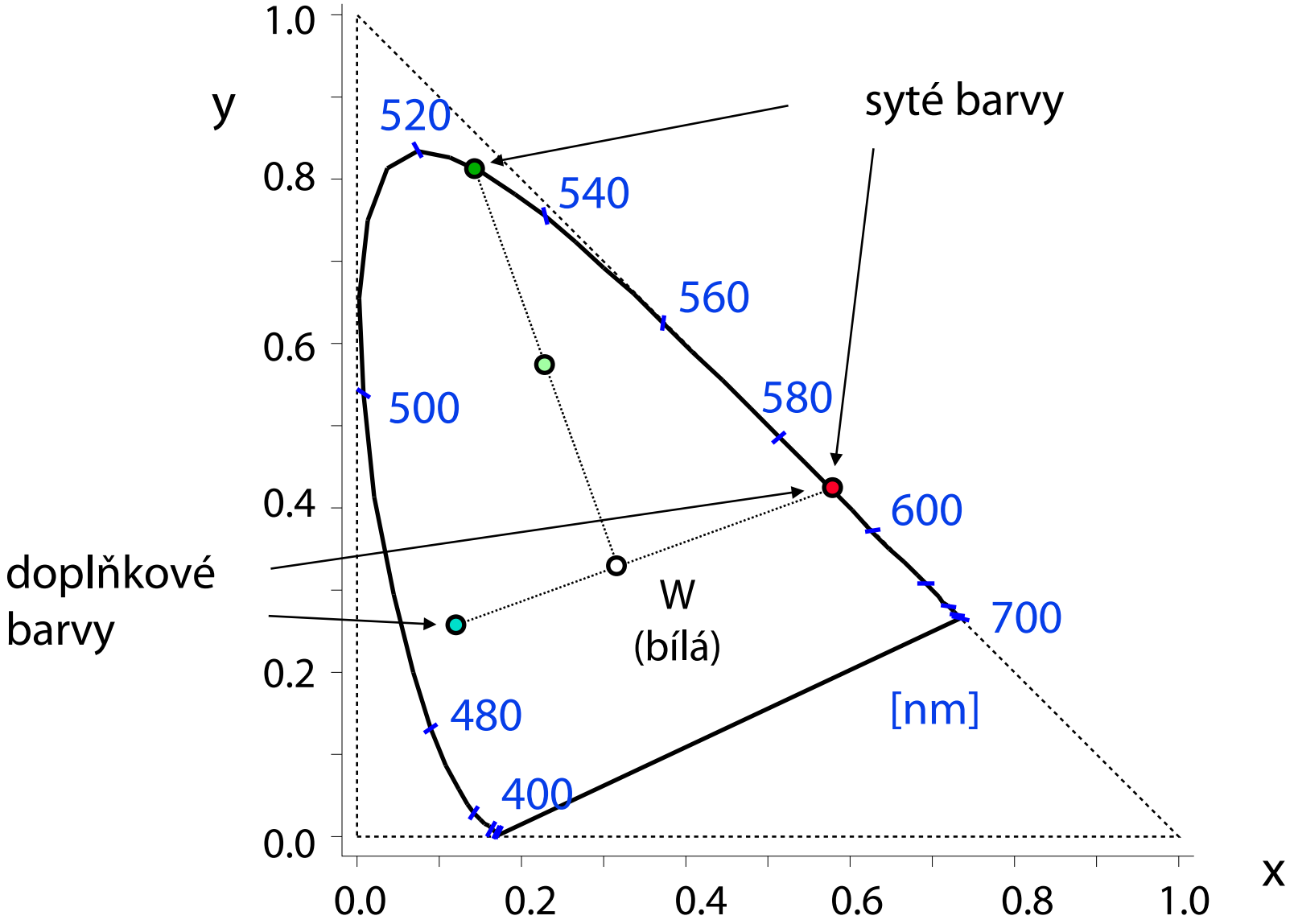
- $x = X/(X+Y+Z)$, $y = Y/(X+Y+Z)$, $z = Z/(X+Y+Z)$
- **x, y, z** nesou informace o odstínu a sytosti (barva) i jasu, složka **Y** byla návržena jako měřítko jasu

Barevný diagram **CIE-xyY** nepoužívá složku **z**

- je závislá na předchozích dvou ($z = 1 - x - y$)

Systém **CIE-xyY** nezohledňuje **subjektivní citlivost** na barevné rozdíly (k tomu slouží CIE-uv)

Barevný diagram CIE-xy



Barevná primitiva RGB

Odpovídají poloze **tří typů barevných luminoforů**

R = [**0.628, 0.346**]

G = [**0.268, 0.588**]

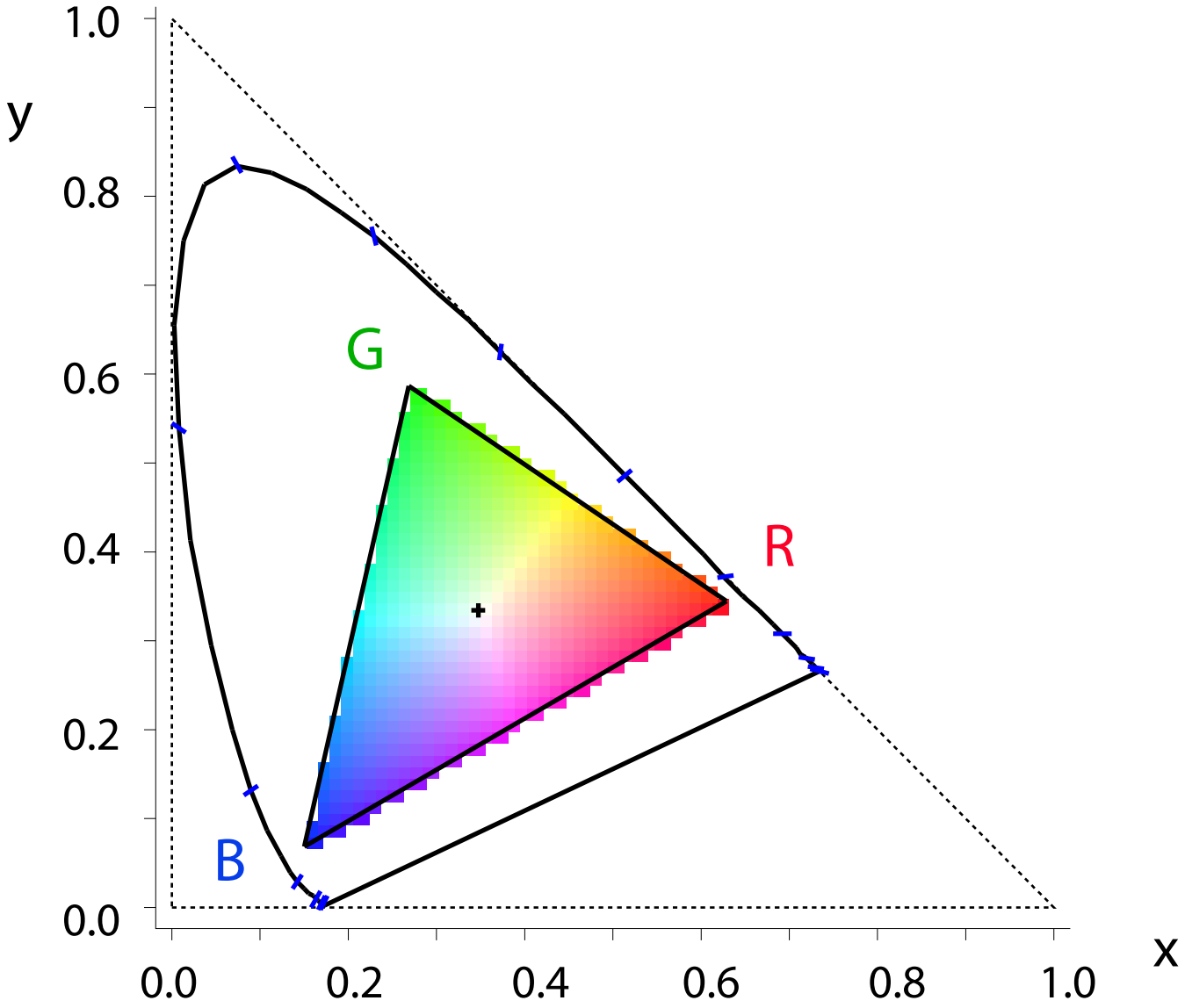
B = [**0.150, 0.070**]

- bílá (Planck) **W**($\mathbf{D_{6500}}$) = [**0.313, 0.329**]

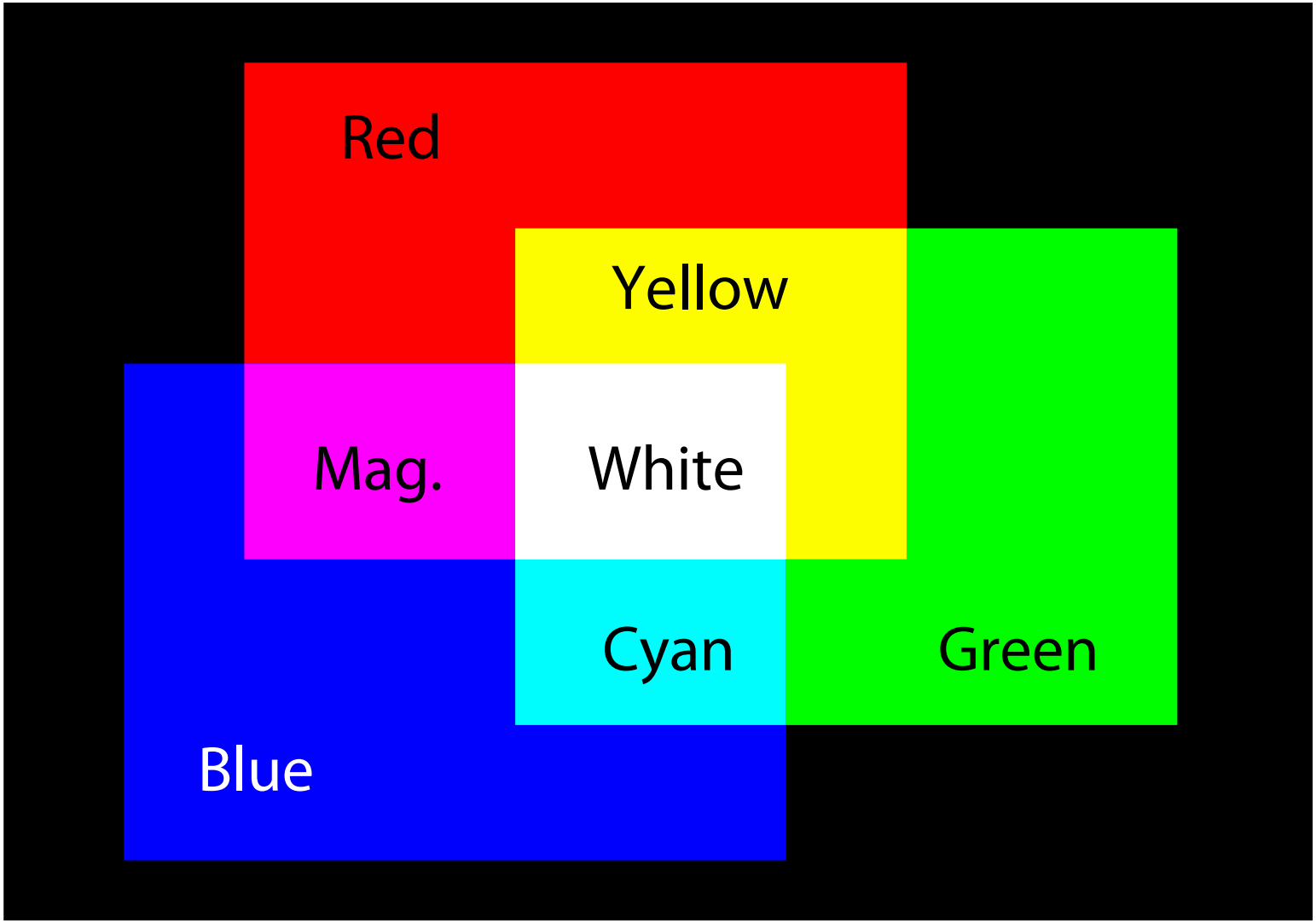
Izoenergetická bílá W má souřadnice [**1/3, 1/3**]

Bílá **R** podle televizní **NTSC** normy [**0.31, 0.316**]

Gamut konkrétního monitoru v CIE-xy



Aditivní skládání barev (RGB)



Barevný systém CMY(K)

Používá se při **tisku** a ve fotografii

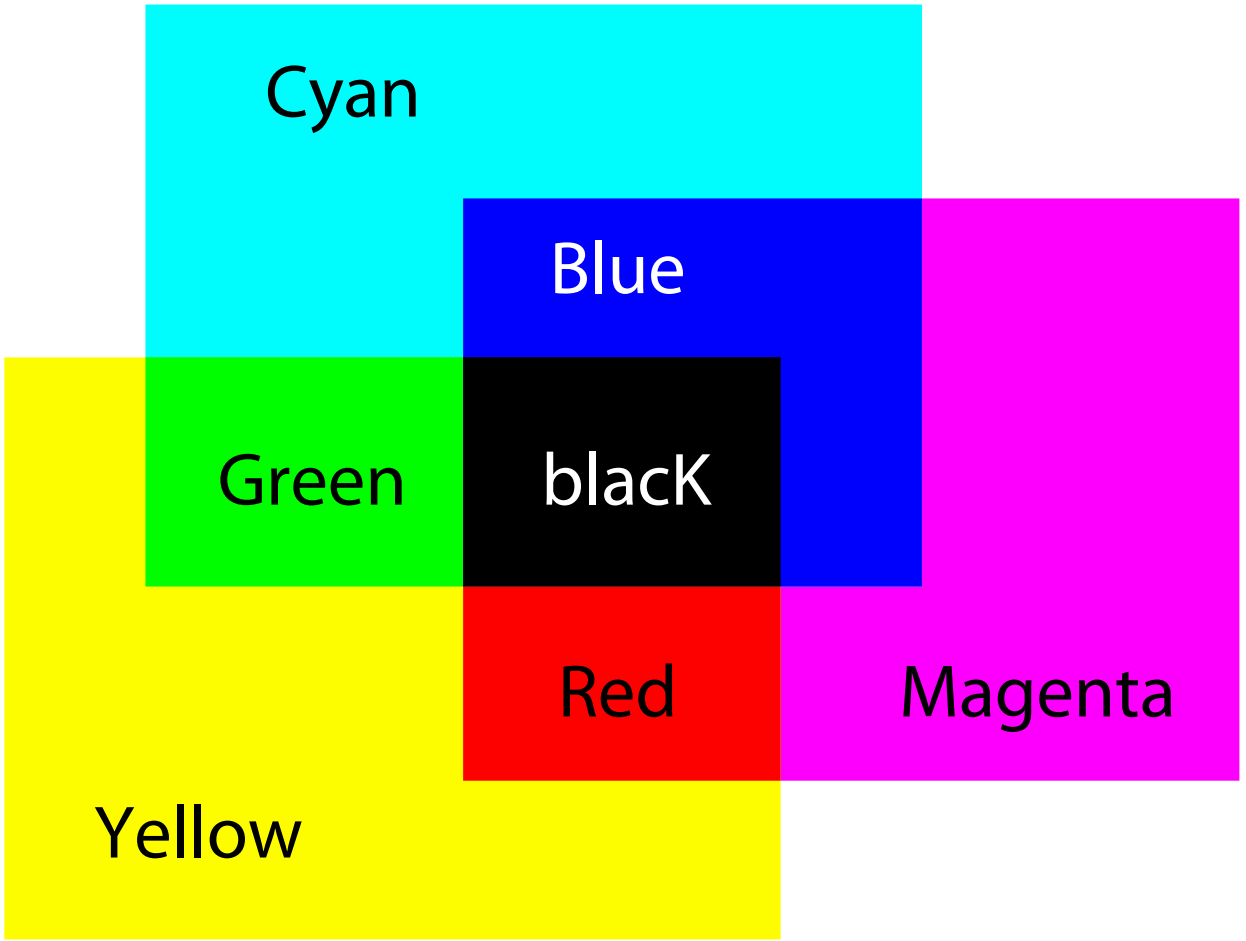
- tam, kde barevný dojem vzniká **pohlcením** některých složek bílého světla

Barvy se skládají **subtraktivně**

Základní barevná primitiva **C** (cyan), **M** (magenta), **Y** (yellow) odpovídají tiskařským barvám

- **C, M, Y** jsou doplňkové k **R, G, B**

Subtraktivní skládání barev (CMY)



Barevný systém CMY(K)

Převody mezi CMY a RGB

- $C = 1 - R,\ M = 1 - G,\ Y = 1 - B$

Ke třem složkám **C, M, Y** se ještě často přidává černá **K**:

- černá barva složená z C, M a Y není dostatečně kvalitní
- černý inkoust (toner) je mnohem levnější než barevný

$$\mathbf{K' \approx min(C,M,Y)}$$

$$C' \approx C–K,\ M' \approx M–K,\ Y' \approx Y–K$$

Barevný systém YIQ

Používá se při **barevném televizním vysílání**

- zaveden komisí **NTSC** v roce 1953
- kompatibilita s černobílými TV přijímači

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.528 & 0.311 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Barevné rozdílové složky (**I, Q**) jsou pro lidské oko méně důležité

- menší rozlišení nebo užší přenosové pásmo

Barevný systém HSV

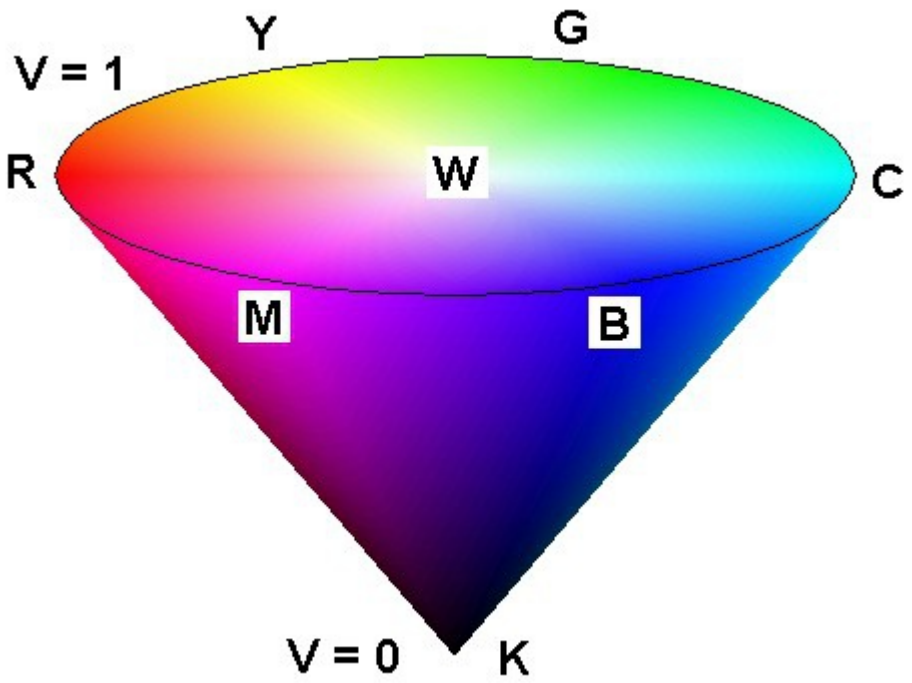
Orientovaný na **uživatele**

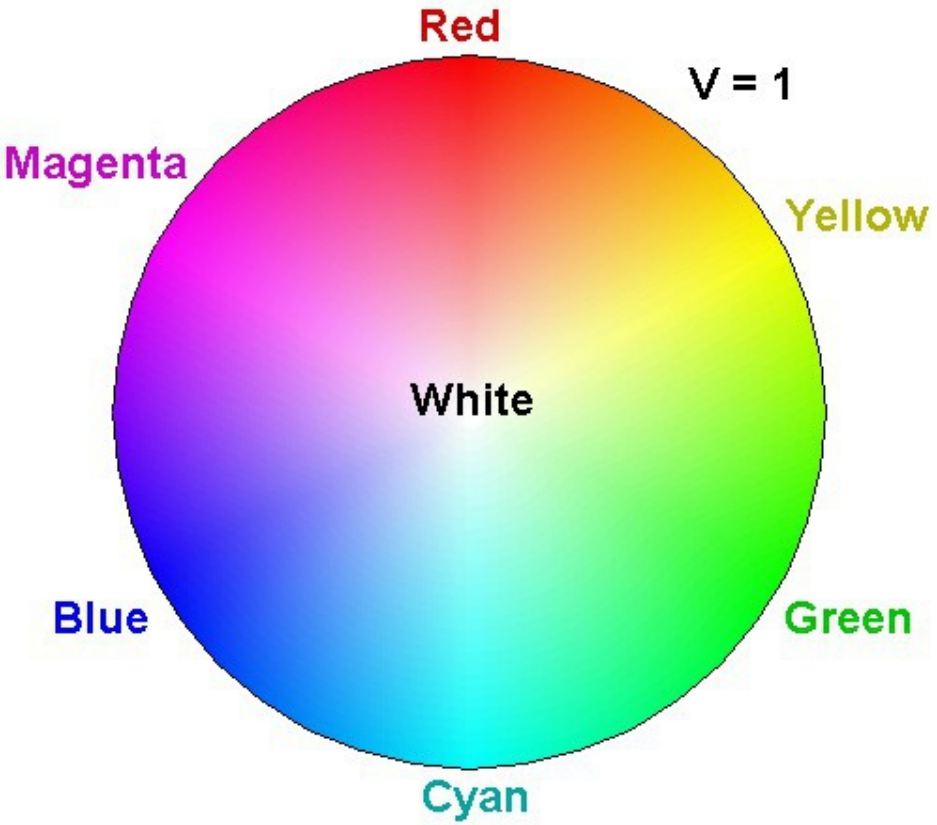
- intuitivní veličiny: **barevný odstín** („hue”), **sytost** („saturation”) a **jas** („value”)

Význam jednotlivých složek

- **H**: základní spektrální barva (dominantní vlnová délka) – rozsah **0°** až **360°**
- **S**: sytost, čistota barvy (poměr čisté barvy a bílé) – rozsah **0** (bílá) až **1** (spektrální barva)
- **V**: jas, intenzita – rozsah **0** (černá) až **1**

Barevný kruh





Převod RGB → HSV

```
procedure RGB2HSV ( R,G,B : real; var H,S,V : real );
var min, max, delta : real;
begin
  min := minimum(R,G,B); max := maximum(R,G,B);
  V := max; delta := max - min;
  if max <> 0.0 then S := delta/max
                else S := 0.0;
  if delta <> 0.0 then
    begin                          { chromatický případ }
      if R = max then H := (G - B)/delta else
      if G = max then H := 2 + (B - R)/delta
                 else H := 4 + (R - G)/delta;
      H := H * 60.0;   { převod na stupně }
      if H < 0.0 then H := H + 360.0;
    end;
end;
```

Převod HSV → RGB

```
procedure HSV2RGB ( H,S,V : real; var R,G,B : real );
var i, f, p, q, t: real;
begin
  if S = 0.0 then
    begin                          { achromatický případ }
      R := V; G := V; B := V;
    end       else
    begin                          { chromatický případ }
      if H = 360.0 then H := 0.0;
      H := H/60.0;      { 0 <= H < 6 }
      i := trunc(H);    { číslo výseče: 0 <= i <= 5 }
      f := H-i;                    { 0 <= f < 1 }
      p := V * (1.0 - S);
      q := V * (1.0 - S*f);
      t := V * (1.0 - S*(1.0 - f));

      ...
```

Převod HSV → RGB

```
        ...

        case i of                    { šest výsečí: }
          0: (R,G,B) := (V,t,p);
          1: (R,G,B) := (q,V,p);
          2: (R,G,B) := (p,V,t);
          3: (R,G,B) := (p,q,V);
          4: (R,G,B) := (t,p,V);
          5: (R,G,B) := (V,p,q);
          end;
      end;                           { chromatický případ }
  end;
```

Další barevné systémy

HLS („hue", „lightness", „saturation")

- podobný jako **HSV**, dvojitý kužel

Firemní systémy

- např. **TekHVC** (Tektronix)

Vzorníky a katalogy barev

- **PANTONE®** (Pantone Inc.)
- **Munsellův systém** (tiskařství) - klasifikace barev „odstín jas/sytost" (např. žlutá barva „5Y 7/4")
- Ostwaldův systém (1931)

Literatura

G. Murch: ***Human Factors of Color Displays***, in Advances in Computer Graphics II, Springer, 1986, 1-27

J. Foley, A. van Dam, S. Feiner, J. Hughes: ***Computer Graphics, Principles and Practice***, 579-599

V. Skala: ***Algoritmy počítačové grafiky III***, skriptum ZČU, 1992, 23-65

Literatura

Další informace:

Jiří Žára a kol.: ***Počítačová grafika, principy a algoritmy***, 316-328

Přednáška **A. Wilkie:** ***Introduction to Colour Science*** (NPGR025)

matfyz

Computer Graphics Charles University

Gamma a nelinearita grafického výstupu

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

http://cgg.mff.cuni.cz/~pepca/

Vnímání jasových úrovní

Šedý odstín má jediný atribut

- **intenzita** (fyzika, vyzařovaná energie, hustota fotonů)
- **jas** (subjektivní vjem člověka, „brightness“)

Vztah mezi intenzitou a jasem **není lineární**

- člověk vnímá intenzity **relativně** (zdravé oko rozezná ~1% rozdílu)
- pro rovnoměrně odstupňované jasové odstíny je třeba mít **logaritmickou stupnici** intenzit

Pravidelné jasové stupně

Minimální zobrazitelná intenzita

- záleží na výstupním zařízení (dynamika)
- $I_0 = 10^{-4}$ až 10^{-2}

Ostatní stupně intenzity

- $I_j = I_0 \cdot r^j$ ($r \approx 1.018$ pro kontrast 100:1 a 256 odstínů)
- měly by tvořit pravidelné stupně jasu (vnímaného člověkem)

CRT monitor

Intenzita vyzařovaného světla **nezávisí lineárně** na hodnotě napětí přiváděného do monitoru

- nelinearitu zavádí katodové dělo

$$I = K\,(V + \varepsilon)^{\gamma}$$

- **V** – napětí přiváděné do CRT (hodnota pixelu)
- **K** – proměnná – ovládací prvek kontrast („picture")
- **ε** – proměnná – ovládací prvek „jas" („black level")
- **γ** – **konstanta** – gamma exponent (2.35 až 2.55)

Praktické důsledky nelinearity

Nelinearita CRT monitoru je **přibližně inverzní** k funkci vnímání našeho zraku

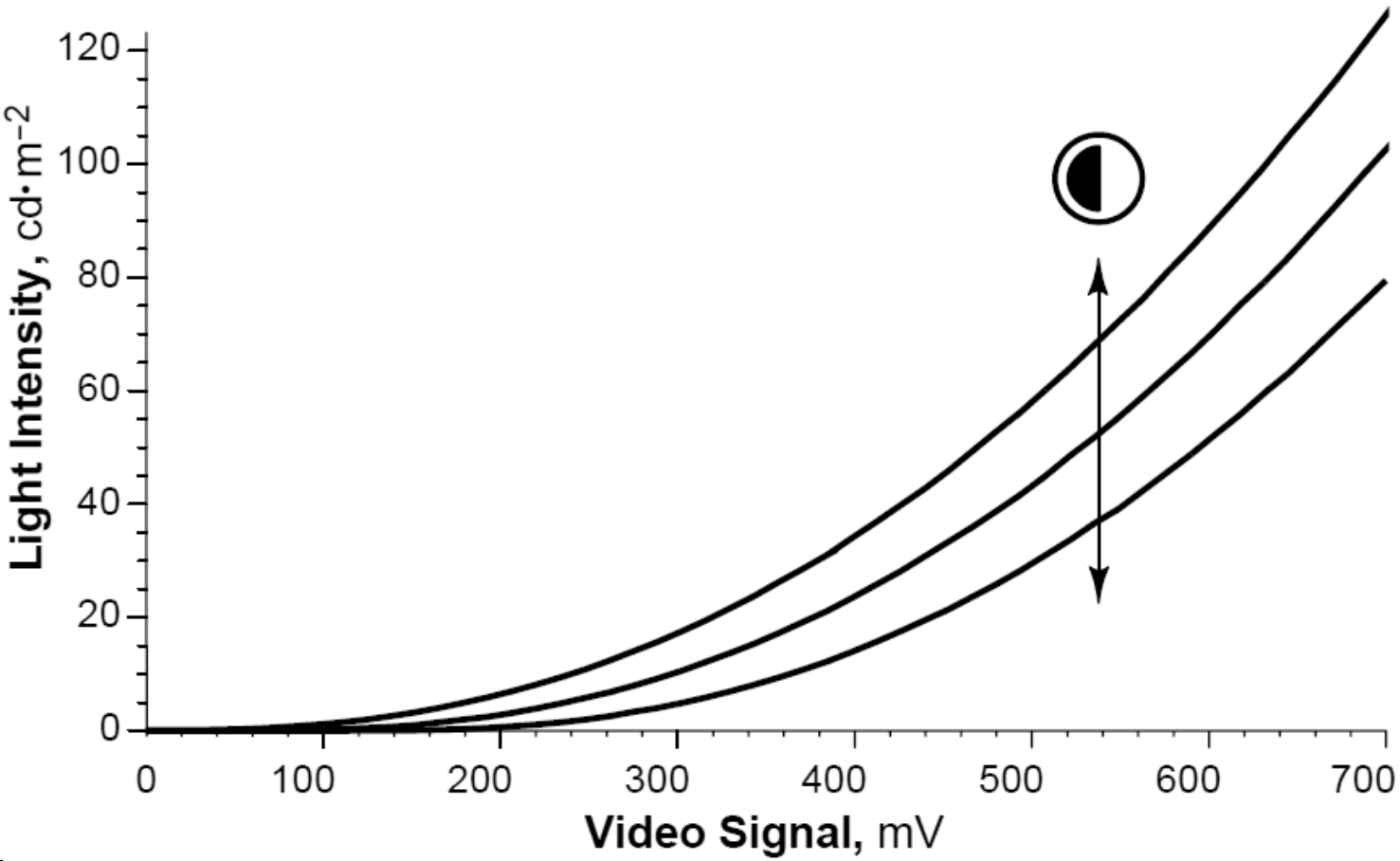
- je tedy dobře, že má výstup takový průběh!
- měli bychom toho využívat

Korekce „gamma” se provádějí

- pro efektivní využití omezeného rozsahu pixelu (0 ÷ 255)
- pro zobrazení na zařízeních s jinou charakteristikou než CRT (např. tisk)
- pozor na skryté konverzní funkce (SGI, Macintosh..)!

Monitor – kontrast („picture“)

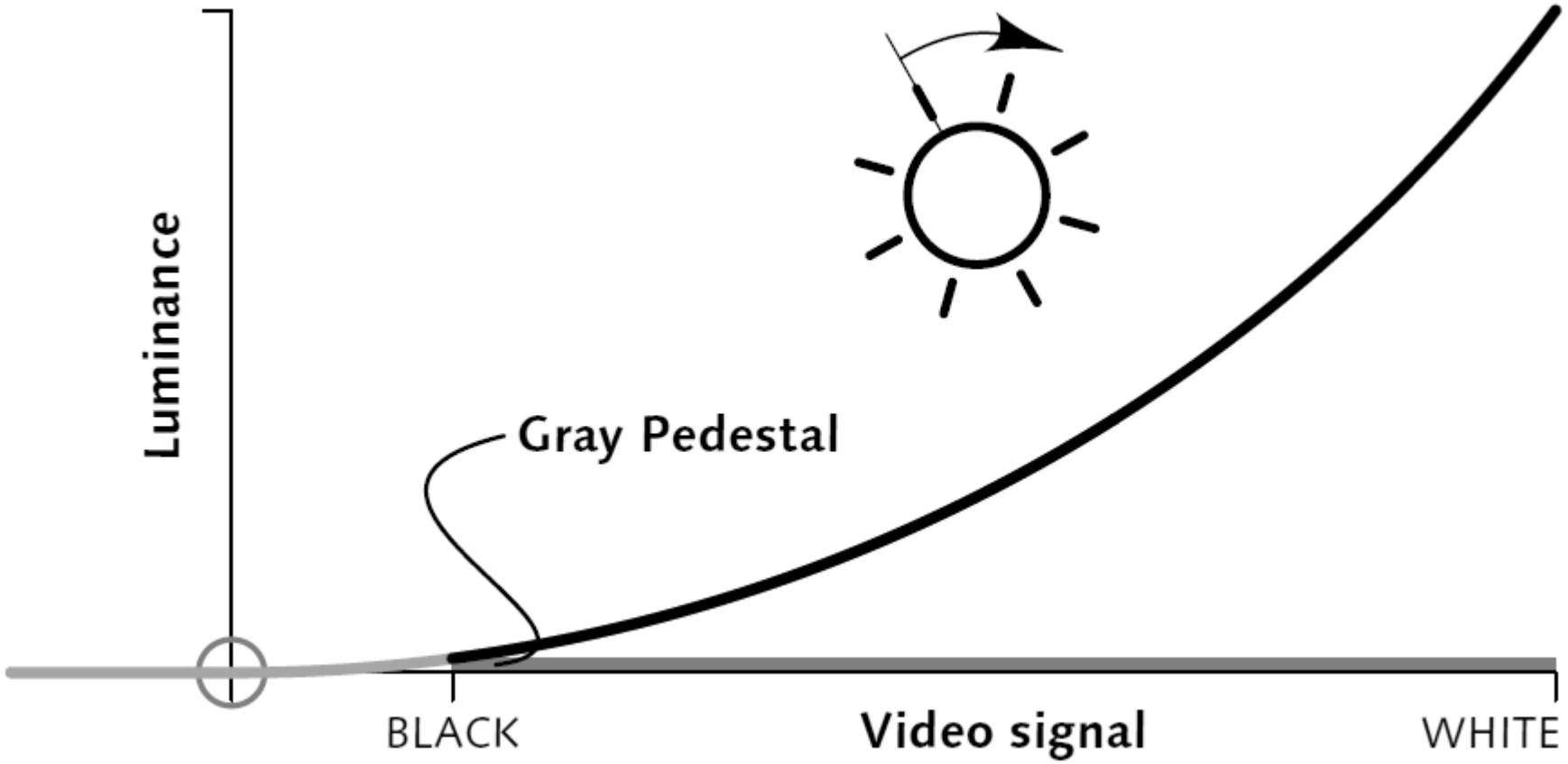
Multiplikativní konstanta K



© Ch. Poynton

Monitor – jas („black level“)

Posunutí vstupního argumentu ε

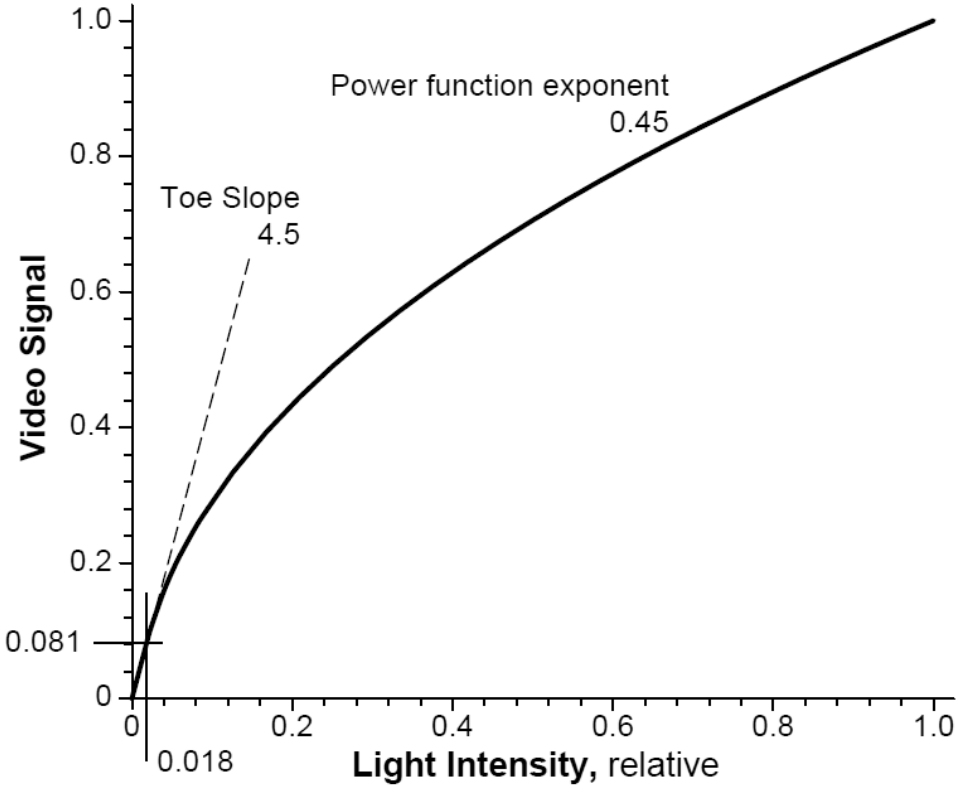


© Ch. Poynton

Vstupní „Gamma korekce“

Transformace intenzity před uložením do RAM

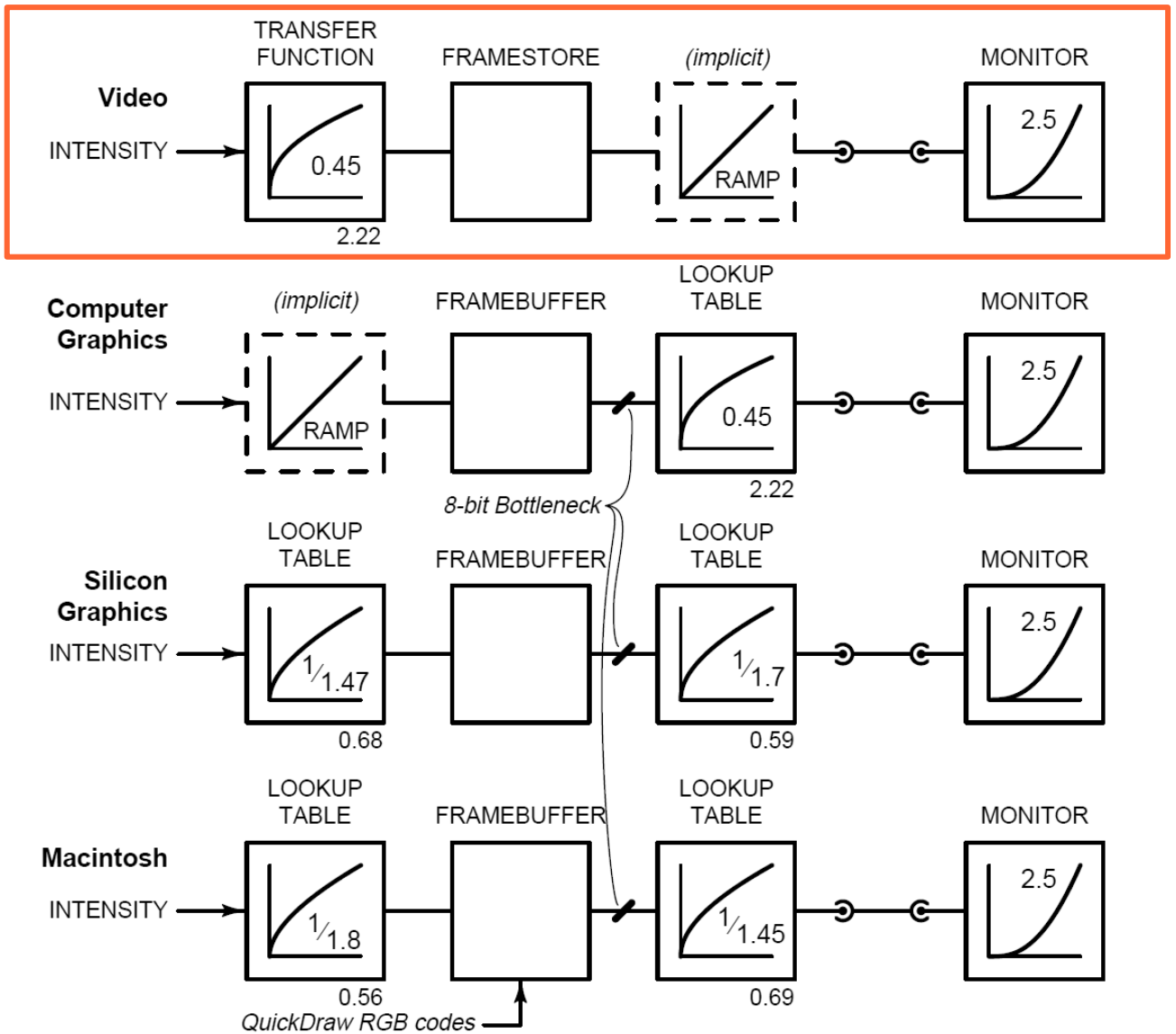
~ inverzní k nelinearitě CRT monitoru



© Ch. Poynton

Zpracování video-signálu

optimální



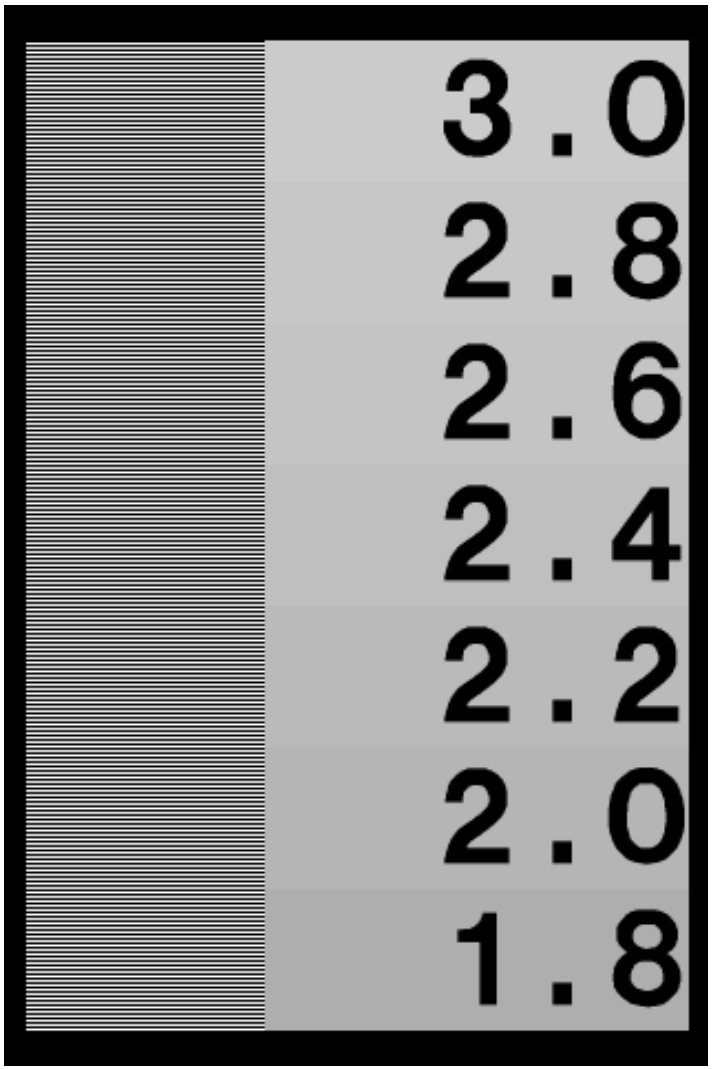
© Ch. Poynton

Testovací obrázky

Vedle sebe zobrazují ½ odstín získaný půltóno-vacím rastrem a běžné šedé odstíny

- hledáme shodu

Můžeme odečíst exponent aktuálního zobrazení

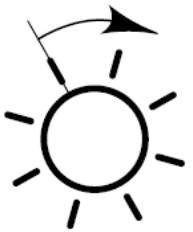


Ladění monitoru – postup

1. **kontrast** na minimum

2. zobrazit absolutně černý obrázek

3. **jas** doladit tak, aby černá vůbec nesvítila (těsně)
   - jasovým prvkem už později nehýbat !



4. **kontrast** nastavit zpět na úroveň vhodnou pro světelné podmínky okolí

Literatura

Další informace:

Ch. Poynton: ***The rehabilitation of gamma,***
www.poynton.com, 2004

Ch. Poynton: ***FAQ about gamma***, www.poynton.com, 1998

matfyz

Computer Graphics Charles University

Rastrový obraz, grafické formáty

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

http://cgg.mff.cuni.cz/~pepca/

Snímání rastrového obrazu

Digitální fotoaparát / kamera

- **plošný snímač** (CCD, CMOS)
- barvy → nejčastěji Bayerova maska
- náročné vyčítání dat
- syrová data (RAW) se přímo v přístroji zpracují
  - » specializovaný digitální obrazový procesor (DIGIC …)

Scanner (filmový, stolní, kopírka)

- obvykle **lineární snímač** (1D)
- jednodušší vyčítání, ale nutnost mechanického pohybu

Plošný snímač obrazu

Rozměry a rozlišení

- větší fyzické rozměry → méně šumu (fyzika)
  … i optika bývá kvalitnější
- větší rozlišení (více Mpix) → více šumu

Citlivost snímače (ISO)

- pouze nastavuje **zesílení** při ADC konverzi
- větší citlivost (zesílení) → více šumu

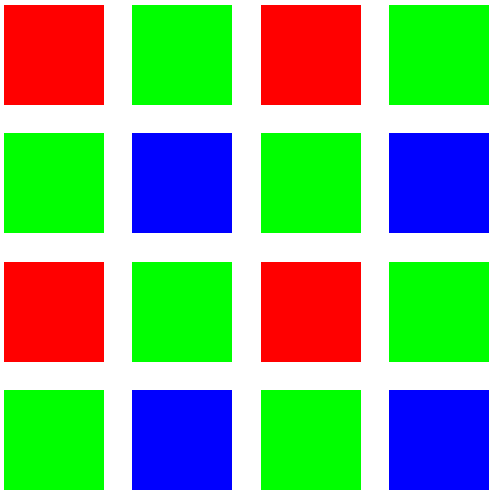
Snímání barev

- Bayerova maska, přímé uložení do RAW

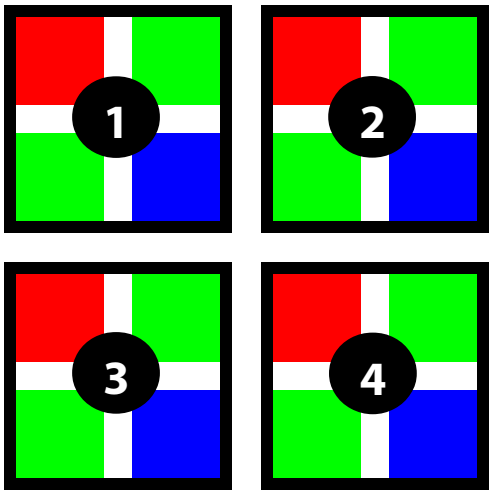
Bayerova maska

Barevné filtry pro RGB

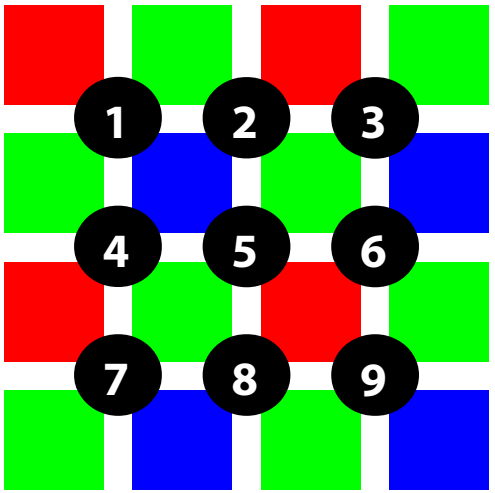
- jednotlivé složky se snímají odděleně (vedle sebe)
- menší efektivita snímače, ale jednoduchá výroba



maska senzoru



přepočet na pixely



efektivnější přepočet

Barevné formáty pixelu

Barevná paleta (8 bitů)

- globální tabulka barev (paleta, „colormap“)
- pixel obsahuje **index barvy** v paletě

Černobílé / šedé pixely (1 bit / 8-16 bitů)

- 1-bitová „bitmask“ (např. při faxovém přenosu)
- odstíny šedé, korigované na „gamma“ koeficient

Plná barva, „true-color“ (24-48 bitů)

- nejběžnější uložení barev (RGB), korekce na „gamma“

„Hi-color“ (15-16 bitů)

- „ošizená“ plná barva, 5-5-5 nebo 5-6-5 bitů (RGB)

Grafické formáty

Rastrové

- obdélníková **matice pixelů** („bitmapa”)
- MS-Windows Bitmap (BMP), Portable Network Graphics (PNG), CompuServe GIF, Interchange File Format (IFF), JFIF (JPG), PBM/PGM/PPM/PFM, Macintosh (PICT), Targa (TGA), Tagged Image File Format (TIFF) …

Vektorové

- **posloupnost objektů nebo příkazů** (škálování)
- CorelDraw!™ (CDR), Scalable Vector Graphics (SVG), AutoCAD™ (DXF), Adobe Illustrator™ (AI), Adobe PDF™, PostScript™, Windows Metafile (WMF) …

Rastrové grafické formáty

Formát uložení barev

- barevná paleta, šedá škála, „true-color”, kanál „α”

Komprese

- bezeztrátová / ztrátová
- RLE: TGA, BMP; LZ*: PNG, GIF, TIFF; JPEG: JFIF, TIFF

Rozklad obrázku

- prokládané/progresivní režimy (PNG, GIF, TGA, JFIF …)

Negrafické info – metadata (popisky, copyright, datum …)

- všechny moderní formáty (TIFF, PNG, GIF, JFIF …)

PBM / PGM / PPM

Velmi jednoduchý **rastrový formát**

Jednoduchá textová hlavička + txt nebo bin data

- bez komprese
- pixelové formáty: B/W (P1/4), gray (P2/5), RGB (P3/6)

Příklad šedého obrázku 16×8

```
P2
16 8 255
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 90 90 0 0 0 0 0 0 0
0 0 0 0 0 0 90 128 128 90 0 0 0 0 0 0
0 0 0 0 0 90 180 255 255 180 90 0 0 0 0 0
0 0 0 0 90 180 255 255 255 255 180 90 0 0 0 0
0 0 0 90 180 255 255 255 255 255 255 180 90 0 0 0
0 0 90 180 255 255 255 255 255 255 255 255 180 90 0 0
0 90 180 255 255 255 255 255 255 255 255 255 255 180 90 0
```

Targa formát (Truevision Inc.)

Jednoduchý **rastrový formát**

Původně HW orientovaný

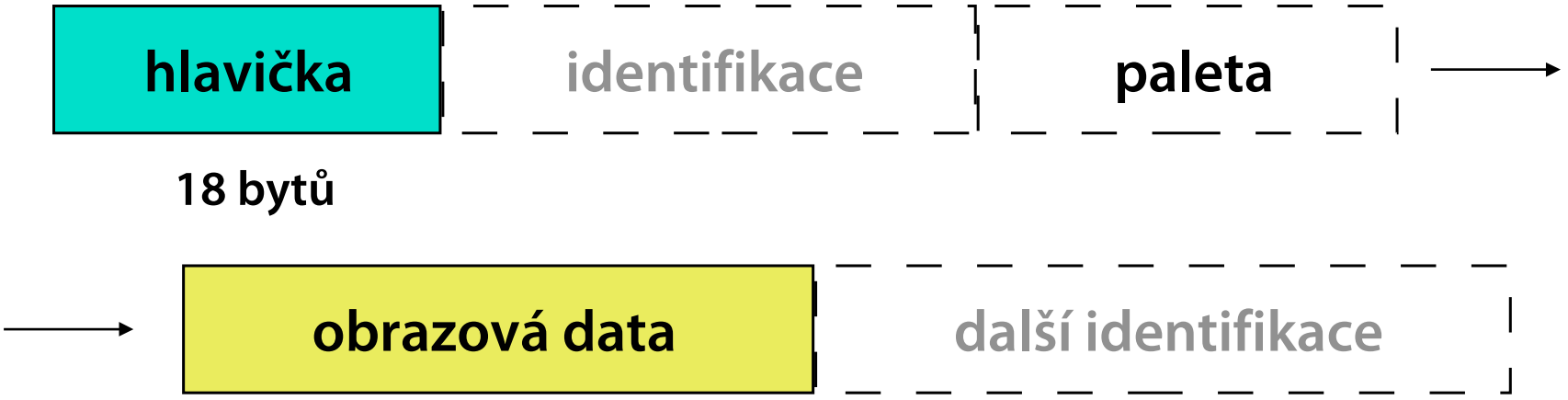
- video-adaptéry Targa (Targa 16, Targa 24, ..)

Několik různých **barevných formátů**

- RGB, RGBα, šedé obrázky, obrázky s paletou, atributové bity
- různé metody **komprese** (RLE komprese je pixelově orientovaná)

Různé typy **prokládání** (přenos po síti)

Struktura TGA souboru



Hlavička souboru

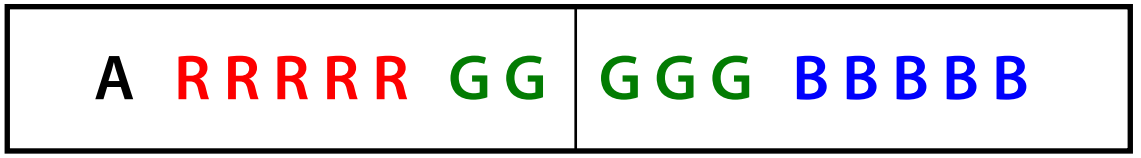
- barevný formát (paleta, RGB, RGBα, šedý obrázek)
- délka identifikace (ASCII text, maximálně 256 znaků)
- typ komprese: bez, RLE, Huffman, delta-modulace
- velikost obrázku: $[\mathbf{X_0},\mathbf{Y_0}]$, šířka, výška
- orientace (shora, zdola), typ prokládání (1, 2, 4 fáze)

Formáty pixelu v TGA

Paleta, šedý obrázek

8 nebo 16 bitů

RGB 16



atribut 16 bitů

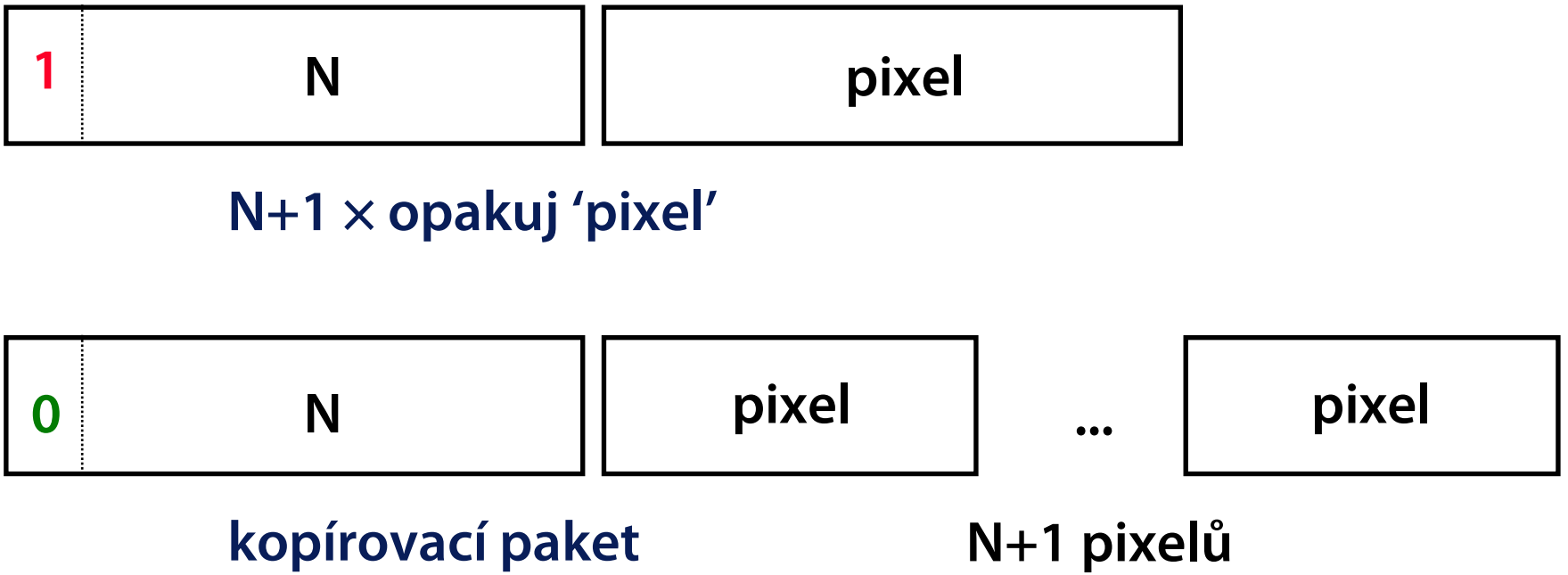
RGB 24

24 bitů

RGB 32

32 bitů

RLE komprese v TGA



Maximální délka paketu je 128 pixelů

– prodloužení je v nejhorším případě 0.8 % délky souboru

GIF formát (CompuServe Inc.)

Graphics Interchange Format (verze 87a, 89a)

- rastrový formát relativně nezávislý na HW
- pouze obrázky s paletou (max. 256 barev)

LZW komprese s dynamickou délkou kódu

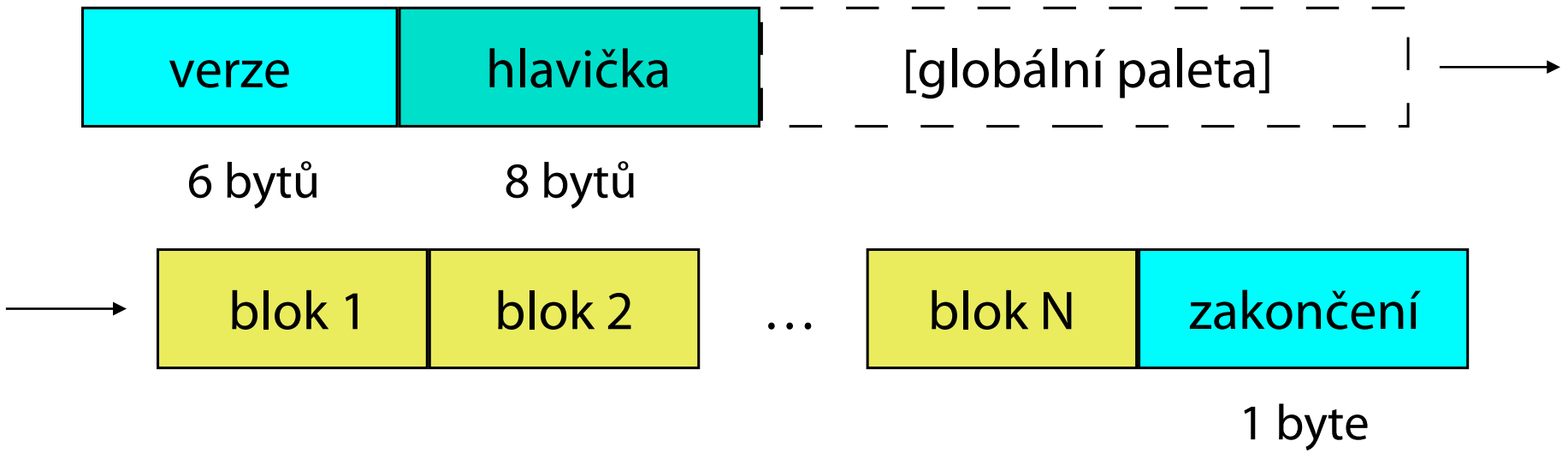
- patenty UniSys Inc. (licenční poplatky do 2003-2004)

Volitelné 4-fázové **prokládání** (pro přenos po síti)

Další rozšíření

- více obrázků v jednom souboru (**animace**)
- aplikační neobrazové informace (**metadata**)
- definice „průsvitné barvy”, interakce uživatele, výpis textu …

Struktura GIF souboru



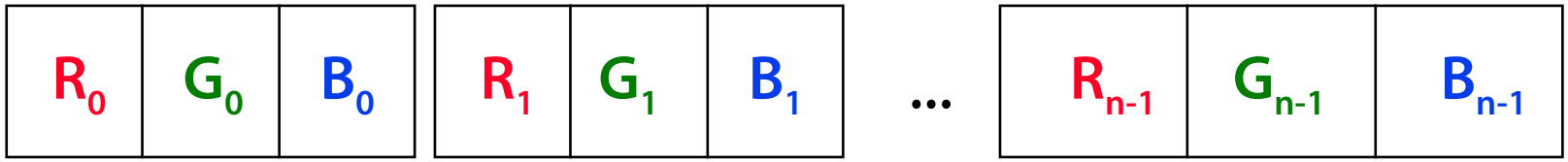
Verze: 'GIF87a' nebo '**GIF89a**'

Globální hlavička

- výška a šířka virtuální obrazovky, počet bitů na pixel, barva pozadí, „pixel aspect ratio" (4/1 až 1/4)
- globální paleta: délka, třídění (významné barvy na začátku)?

Struktura GIF souboru

Paleta



(n×3) byty

Bloky

- obrazová nebo jiná data (poznámky, aplikační data, řídící bloky)
- **jednotný vnější formát**
  - » jiná verze dekodéru může neznámé bloky přeskakovat

Obrazový blok

Geometrie obrázku

- $[X_0,Y_0]$, šířka, výška

Nepovinná **lokální paleta**

- počet barev, třídění (podle důležitosti)

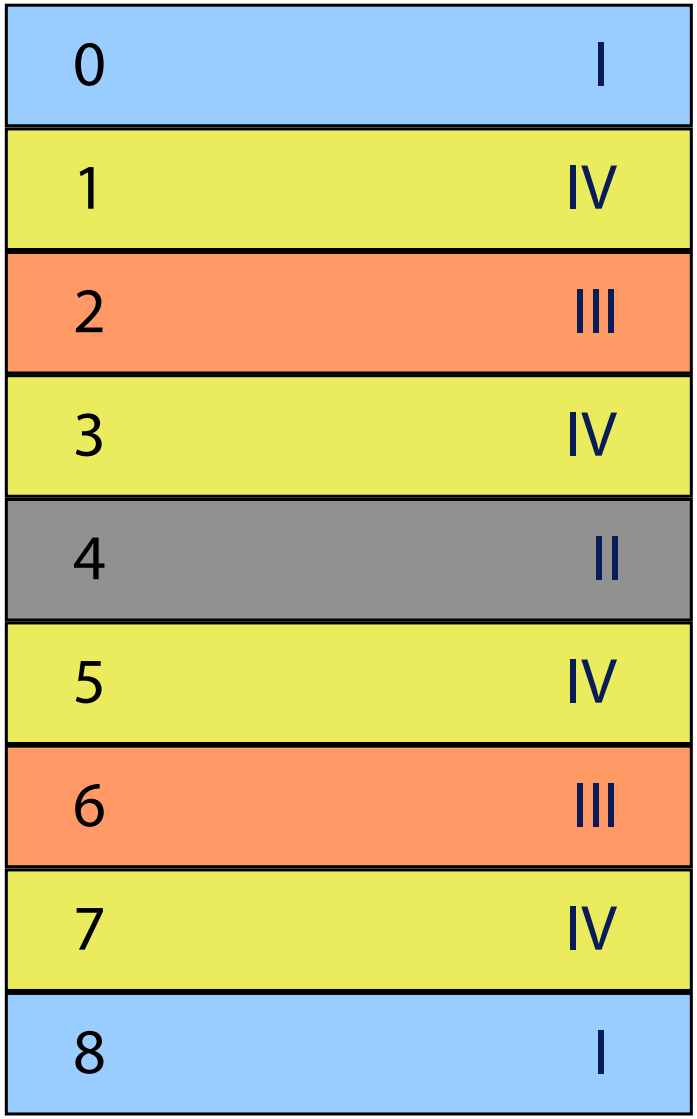
Volba – **prokládání**

- 8 fází kreslení obrázku

Obrazová data

- počáteční délka LZW kódu
- vlastní kódovaná data

Prokládání



I fáze: řádky 8i

II fáze: řádky 4 + 8i

III fáze: řádky 2 + 4i

IV fáze: řádky 1 + 2i

Rozšiřující bloky (verze 89a)

Blok řízení grafiky

- uvolnění grafiky (nechat/smazat/obnovit)
- interakce uživatele, prodleva (pro animace)
- definice transparentní barvy

Blok komentáře

- jakýkoli text – pro uživatele

Blok textu

- výpis textu na obrazovku (neproporcionální font)

Aplikačně závislý blok

- libovolná binární data (např. FractInt)

LZW komprese (Lempel-Ziv-Welch)

Slovníková kompresní metoda (1984)

- vychází z LZ78 (Lempel-Ziv, 1978)
- slovník obsahuje přiřazení „fráze → kód"
- fráze = posloupnost pixelů („znaků abecedy")
- kód = n-bitové číslo ($3 \leq n \leq 12$)

V průběhu kódování se mění

- **slovník**
  - » adaptivní přizpůsobení kódovaným datům
- **délka kódového slova „n"**
  - » zvětšuje se po jedné až do **12**

Schema kódovacího algoritmu

1. inicializace
   - do slovníku všechny jednopixelové (jednoznakové) fráze
   - Act := [ ] (prázdná posloupnost)
2. přečti další pixel ze vstupu do K
3. je fráze „Act + K“ ve slovníku?
   - Ano: Act := Act + K
   - Ne: zapiš na výstup kód fráze Act
   - přidej Act + K do slovníku
   - Act := K
4. dokud neskončí vstup, opakuj kroky 2. a 3.
5. zapiš na výstup kód fráze Act

Práce se slovníkem

Inicializace slovníku

- kódy **0 ÷ 2^p - 1** … jednopixelové (jednoznakové) fráze
- kód **2^p** … „reset“ (inicializace přeplněného slovníku)
- kód **2^p + 1** … ukončovací znak (EOF)
- první volný kód fráze … **2^p + 2**
- počáteční délka kódového slova **n = p + 1** bitů

Pokud má přidaná fráze kód **2p**, **zvětším n o 1**

- maximální hodnota **n** je **12** (4094 frází)
- při přeplnění zakonzervuji slovník (méně často) nebo pošlu **„reset” kód** (reinicializace slovníku)

PNG formát (Portable Network Graphics)

Konsorcium W3C (1995)

Rastrový formát navržený pro WWW

Několik formátů pixelu

- paleta, gray, **true-color**, **spojitá průhlednost**
- 8 ÷ 16 bitů na kanál

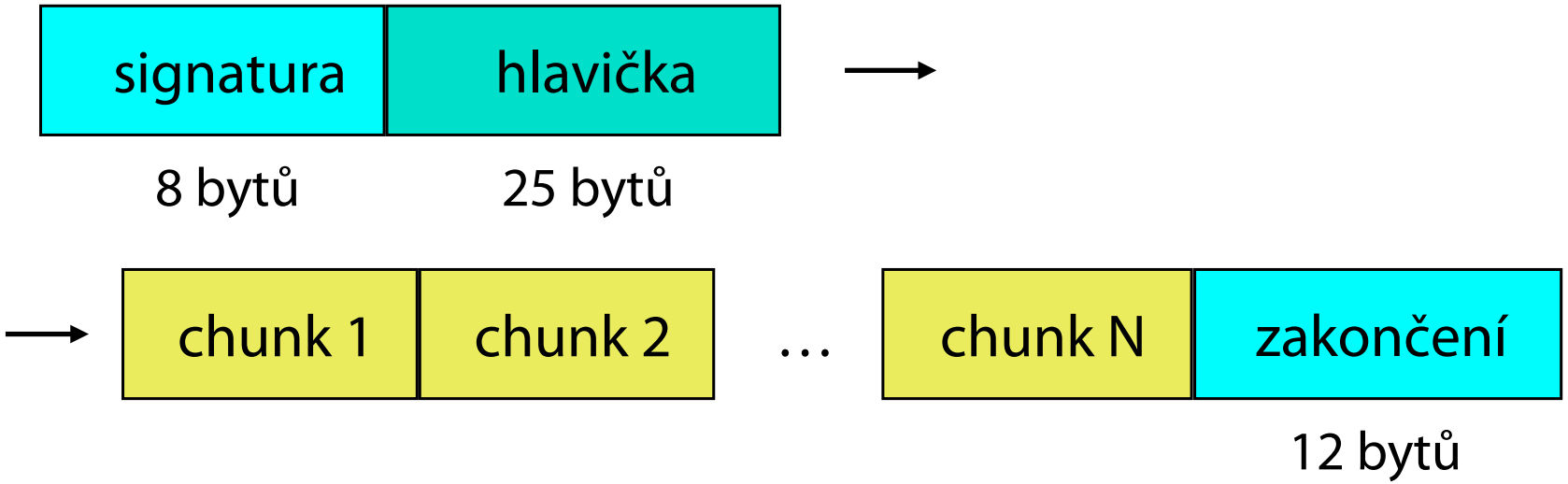
Informace pro kompenzaci HW odchylek

- gamma, gamut, bílý bod

Komprese DEFLATE (bez patentů!) založená na LZ77

Vylepšené volitelné prokládání v 7 fázích

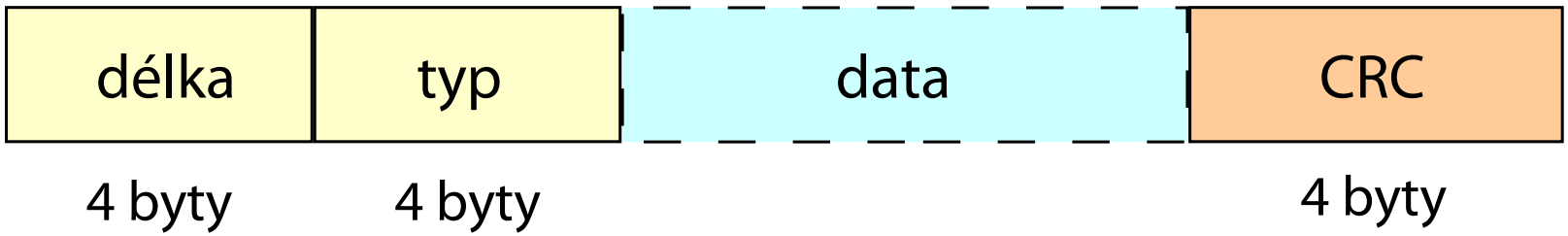
Struktura PNG souboru



Hlavička

- výška a šířka obrázku, bitová hloubka
- barevný formát (paleta, gray, true-color, průhlednost)
- komprese, predikce, prokládání

PNG chunk



Obrazová nebo jiná data

– paleta, průhlednost, HW kompenzace, dodatečné textové informace …

Jednotný vnější formát

– neznámé chunky může dekodér přeskakovat

LZ77 komprese (Lempel-Ziv)

Bezeztrátová slovníková kompresní metoda

- s posuvným oknem (typicky desítky KB)
- data v okně bývají zpracována do vhodné datové struktury pro velmi rychlé vyhledávání
- nesymetrický algoritmus (dekódování je rychlejší)

Kóduje se sekvence dat

- fráze = posloupnost znaků (pixelů)

Kódem je trojice [**offset, délka, znak**]

- offset = relativní poloha začátku fráze (v okně)
- délka = délka fráze v pixelech
- znak = pixel, který následuje za frází

DEFLATE komprese v PNG

Dvě fáze

- LZ77 po řádkách
- Huffmanovo kódování
  - » offset
  - » délka, znak

Volitelná predikce

- standard definuje pět predikčních filtrů
- mohou se přepínat dynamicky na začátku každé řádky
  - » prostor pro optimalizaci (adaptace na charakter obrázku)

Prokládání v PNG

7-fázové, v první fázi se přenese **1/64 pixelů**

1	6	4	6	2	6	4	6
7	7	7	7	7	7	7	7
5	6	5	6	5	6	5	6
7	7	7	7	7	7	7	7
3	6	4	6	3	6	4	6
7	7	7	7	7	7	7	7
5	6	5	6	5	6	5	6
7	7	7	7	7	7	7	7

Komprese JPEG

Joint Photographic Experts Group (1990)

Rastrová komprese se ztrátou dat

Vhodná pro **spojité obrázky** (fotografie, rendering)

Nevhodná pro písmo, diskrétní grafiku, screenshots

- zřetelné artefakty, menší kompresní poměr než LZ*

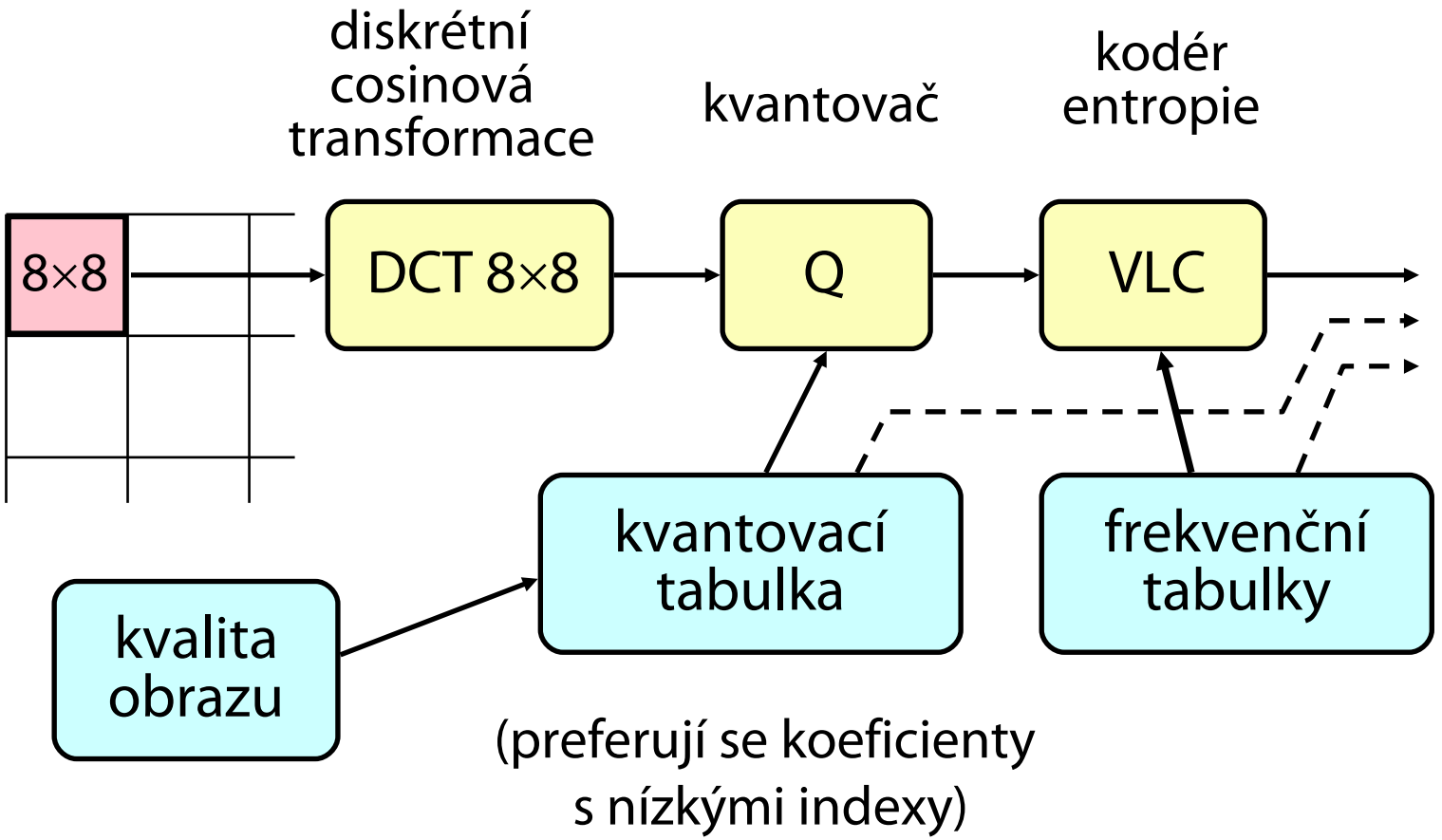
Volitelná kvalita výsledku (0-100, ovlivní kompresní poměr)

Další volby: progresivní režim, hierarchické kódování

Standardní formát souboru: JFIF (přípona JPEG, JPG)

- JPEG File Interchange Format

Ztrátová komprese JPEG



Barvy v JPEG

Podle doporučení CCIR 601

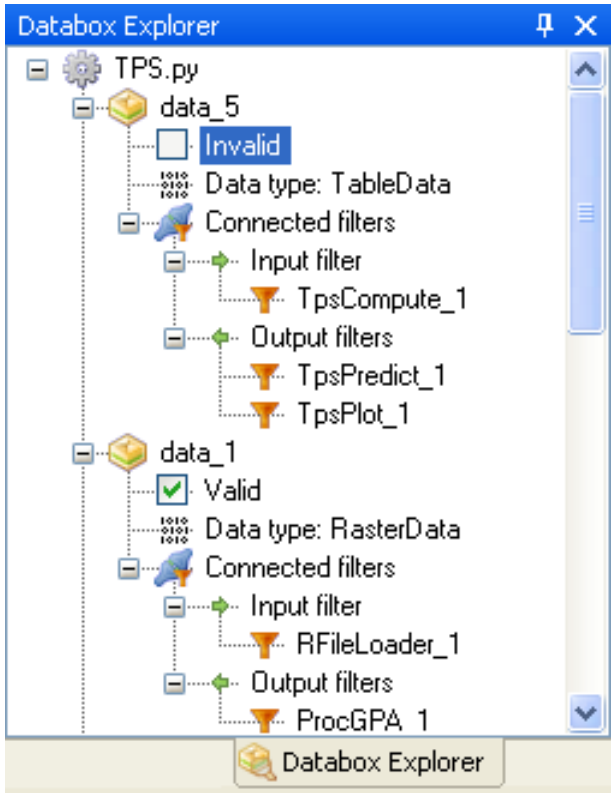
- používá se i ve video-technice

8 bitů na každou složku (existuje i 12-bitová varianta)

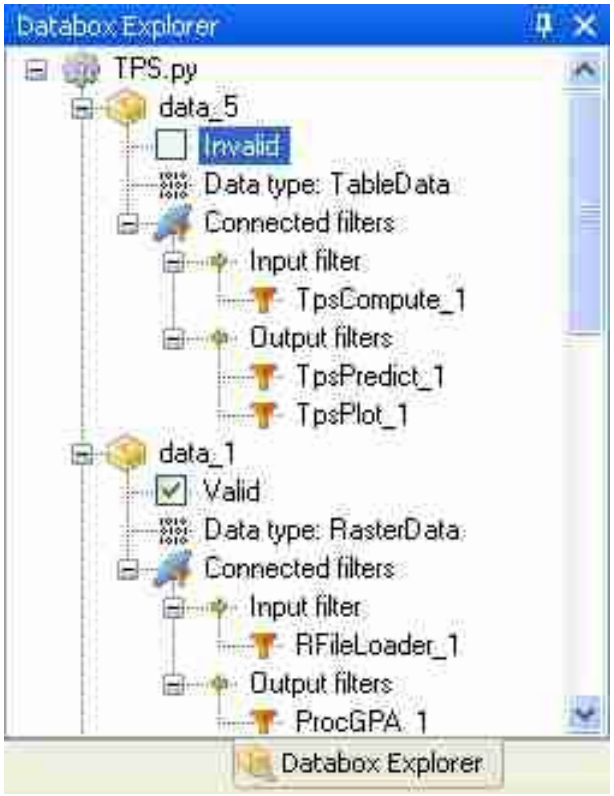
- Y … jasová složka (odstín šedi)
- C_b a C_r … barevné rozdílové složky (reprezentují převážně B a R)

$$
\begin{aligned}
Y &= 0.299\,R + 0.587\,G + 0.114\,B \\
C_b &= -0.1687\,R - 0.3313\,G + 0.5\,B + 128 \\
C_r &= 0.5\,R - 0.4187\,G - 0.0813\,B + 128
\end{aligned}
$$

Artefakty JPEG komprese

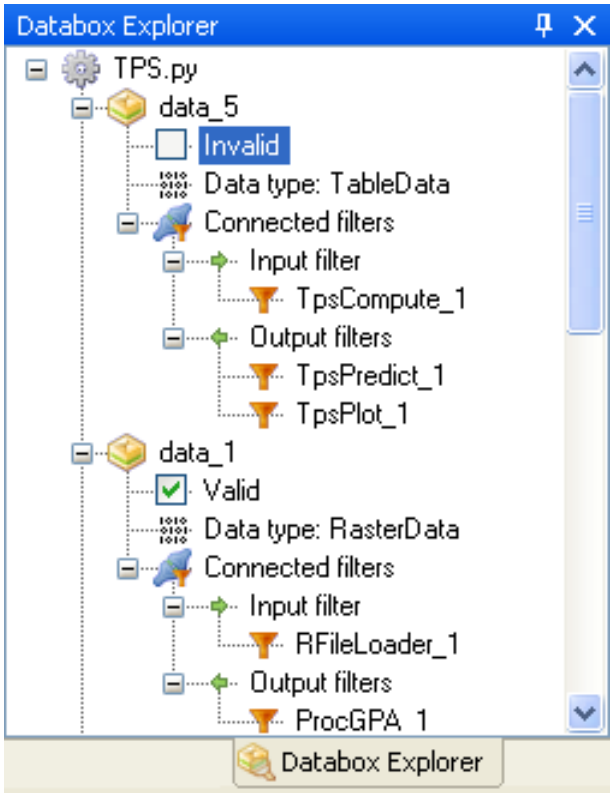


PNG (bezeztrátově)
12.3 KB



JPEG (kvalita 20%)
8.4 KB

Efektivita komprese – screenshot

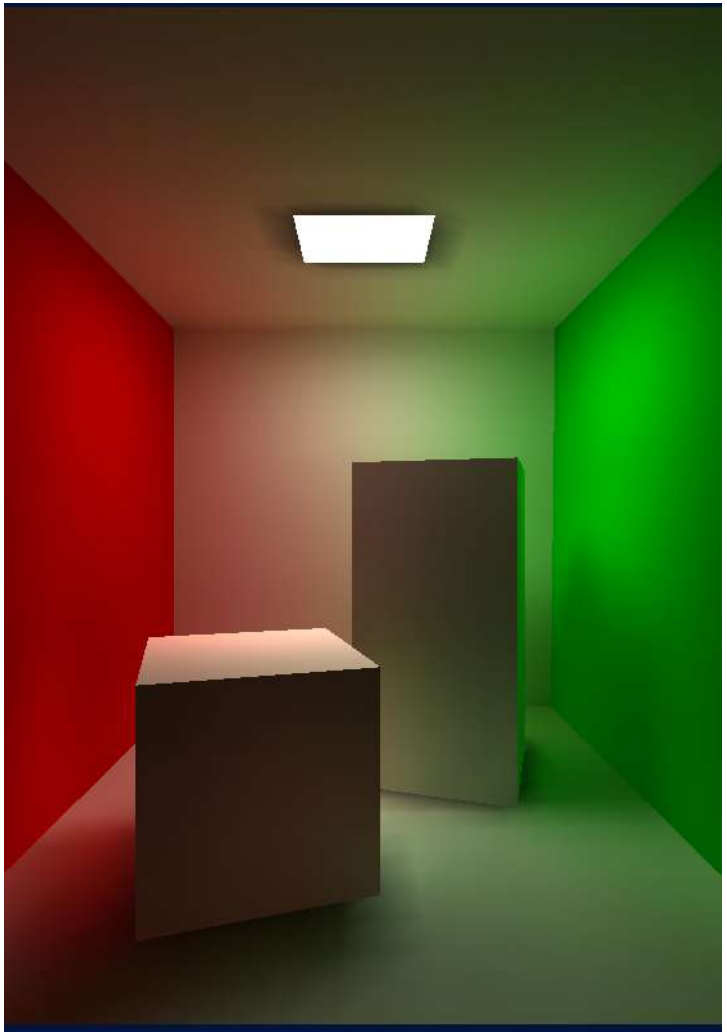


PNG (8 bit)	**5.8 KB**
JPEG (24 bit, q=20%)	8.4 KB
GIF (8 bit)	8.7 KB
PNG (24 bit)	**12.3 KB**
JPEG (24 bit, q=60%)	15.6 KB
JPEG (24 bit, q=90%)	26.5 KB
JPEG (24 bit, q=100%)	45.0 KB
PPM (24 bit)	**242.0 KB**

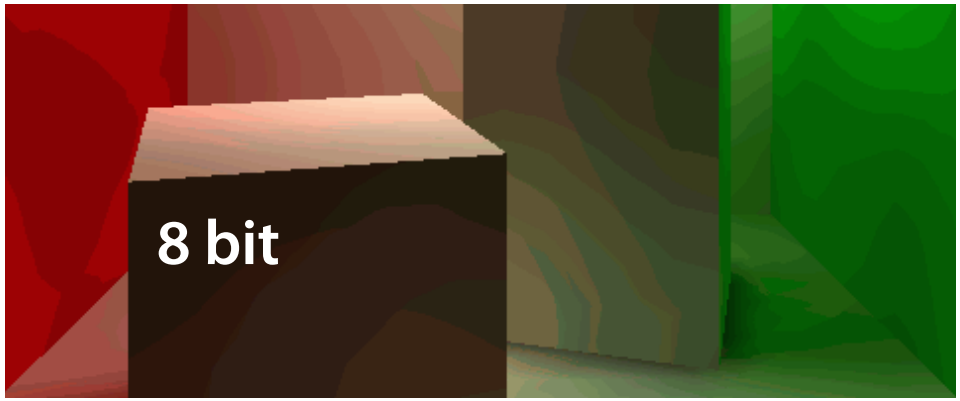
Efektivita komprese: fotografie

JPEG (24 bit, q=20%)	16 KB
JPEG (24 bit, q=60%)	**37 KB**
JPEG (24 bit, q=90%)	**87 KB**
PNG (8 bit)	158 KB
GIF (8 bit)	191 KB
JPEG (24 bit, q=100%)	245 KB
PNG (24 bit)	488 KB
PPM (24 bit)	**1052 KB**

Efektivita komprese: rendering



JPEG (24 bit, q=20%)	9 KB
JPEG (24 bit, q=60%)	**17 KB**
PNG (8 bit)	26 KB
JPEG (24 bit, q=90%)	**39 KB**
GIF (8 bit)	59 KB
JPEG (24 bit, q=100%)	136 KB
PNG (24 bit)	140 KB
PPM (24 bit)	**1876 KB**



Literatura

Další informace

Kay D. C., Levine J. R.: ***Graphics file formats***, MGWH, 1994

Wikipedia: ***Image_file_formats***

matfyz

Computer Graphics Charles University

Vektorový formát SVG

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/

Zobrazování grafiky v HTML5

SVG

- Scalable Vector Graphics
- také pod patronátem W3C

Grafické objekty (primitiva)

- rect, circle, line, ..
- snadno přístupné parametry formou XML atributů

```
<circle cx="250" cy="25" r="25"/>
```

Možnost použití **CSS** pro definici vzhledu (stylu)

```
<circle cx="25" cy="25" r="22" class="pumpkin"/>
```

Standardy – HTML5

HTML5

- mnoho zdrojů pro studium

Minimální platný HTML5 dokument

```
<!DOCTYPE html><title/>x
```

Stručný HTML dokument

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8">
  <title>Simple valid HTML5 page</title>
 </head>
 <body>
  <p>Paragraph..</p>
 </body>
</html>
```

Standardy – SVG

Scalable Vector Graphics

- W3C standard, založen na XML
- http://www.w3.org/Graphics/SVG/

Stručná **HTML5 stránka se SVG grafikou**

```
<!DOCTYPE html>
<meta charset="utf-8">
<title>SVG hello</title>
<svg width="800" height="400">
 <text y="12">
  Hello, world!
 </text>
</svg>
```

Standardy – CSS

Cascading Style Sheets

- W3C standard (CSS 2.2)
- http://dev.w3.org/csswg/css2/
- nepřidává obsah, pouze definuje styly zobrazení

Stručná **HTML5 stránka s CSS stylem**

```
<!DOCTYPE html>
<meta charset="utf-8">
<title>CSS hello</title>
<style>
body { background: steelblue; }
</style>
<body>
Hello, world!
</body>
```

SVG tvary

Rectangle <rect>

Circle <circle>, **ellipse** <ellipse>

Line <line>, **polygon** <polygon>, **polyline** <polyline>

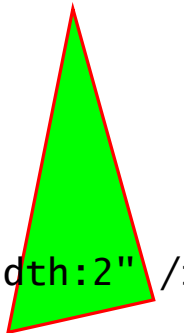
Text <text>

Path

- komplikovanější popis tvaru
- vlastní jednoduchý jazyk
- lomené čáry, splines …
- vyplnění uzavřené cesty a/nebo obkreslení čárou

SVG samples


```
<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 100 100">
 <path d="M30,1h40l29,29v40l-29,29h-40l-29-29v-40z" stroke="#000" fill="none"/>
 <path d="M31,3h38l28,28v38l-28,28h-38l-28-28v-38z" fill="#a23"/>
 <text x="50" y="68" font-size="48" fill="#FFF" text-anchor="middle"><![CDATA[410]]>
 </text>
</svg>
```



```
<svg xmlns="http://www.w3.org/2000/svg" height="210" width="500">
 <polygon points="200,10 250,190 160,210" style="fill:lime;stroke:red;stroke-width:2" />
</svg>
```

```
<svg xmlns="http://www.w3.org/2000/svg" height="150" width="500">
 <ellipse cx="240" cy="100" rx="220" ry="30" style="fill:purple" />
 <ellipse cx="220" cy="70" rx="190" ry="20" style="fill:lime" />
 <ellipse cx="210" cy="45" rx="170" ry="15" style="fill:yellow" />
</svg>
```

Rendering model

SVG grafika se kreslí **odzadu-dopředu**

- postupné vykreslování přes sebe
- „malířův algoritmus“

Možnost **poloprůhledné kresby** (alpha-channel)

- attribute **opacity** ('opacity=“0.5“')

Vyplňování (**„fill“**) a/nebo **obkreslení** (**„stroke“**)

- style=“fill:<color>“
- style=“stroke:<color>;stroke-width:<number>“
- …

SVG shapes I

`<rect>`

- x, y, width, height, rx, ry

`<circle>`

- cx, cy, r

`<ellipse>`

- cx, cy, rx, ry

`<line>`

- x1, y1, x2, y2

SVG shapes II

<polygon>

- points="100,100 50,100 55,80 ..."

<polyline>

- points

<path>

- d="**M** 10 10 **L** 100 100" (MoveTo, LineTo – absolute)
- d="M 10 10 **l** 90 90" (LineTo – relative)
- d="M 10 10 l 90 90 l -40 10 l -10 -70 **z**" (ClosePath)

Relative positioning: **lower case letters**

SVG path details

All path elements

- **M** (moveto 'x y'), **L** (lineto 'x y'), **H** (horizontal lineto 'x'), **V** (vertical lineto 'y')
- **C** (curveto 'x1 y1 x2 y2 x y'), **S** (smooth curveto 'x2 y2 x y')
- **Q** (quadratic Bèzier curve 'x1 y1 x y'), **T** (smooth quadratic Bèzier curveto 'x y')
- **A** (elliptical arc 'rx ry x-rot large? sweep? x y'), **Z** (closepath)

Simplifications

- white-space can be omitted, ',' can be used instead of ' '
- command letter can be omitted if equal to previous one

d="M30,1h40l29,29v40l-29,29h-40l-29-29v-40z"

Grouping <g>

Common attributes

- style, fill, stroke …

Coordinate transformations

- <g **transform="translate(50,0)"**> … </g>
- **scale**(s), **scale**(sx,sy)
- **rotate**(angle), **rotate**(angle,x,y) [all angles in **degrees**]
- **skewX**(angle) … "x += y*tan(angle)"
- **skewY**(angle)
- **matrix**(a,b,c,d,e,f)
- transform="scale(1.5),rotate(45),translate(10,0)"

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

Links <use>

Shared components, shapes

- define once, use multiple times..

'id' attribute = label

- <g **id="tree"** …> … </g>
- <path **id="arrow"** d="M0,0l-30-10 …" …>

<use> element = reference

- <**use xlink:href="#tree"** transform="translate(20,0)"/>
- <**use xlink:html="#arrow"** opacity="0.8"/>

Clipping

<clipPath> element

„clip-path“ attribute

… not yet

Text <text>

„font-family“

„font-style“

- normal, italic, oblique

„font-variant“

- normal, small-caps

„font-weight“

- normal, bold, bolder, lighter, 100, 200, 300, 400, 500, 600, 700, 800, 900

```
<text x="50" y="68" font-family="Verdana" font-size="48"
 fill="#FFF" text-anchor="middle">The answer is 42</text>
```

Animation

<animate>

- animation of an attribute value during defined time-interval

<set>

- sets an attribute value at a specific time

Specific subclasses: **<animateTransform>**, **<animateColor>**

Repetitions, animation curves …

```
<rect x="20" y="10" width="120" height="40" fill="#501080">
 <animate attributeName="width" from="120" to="40" begin="0s" dur="8s" fill="freeze"/>
 <animate attributeName="height" from="40" to="82" begin="6s" dur="7s" fill="freeze"/>
</rect>
```

Resources

SVG homepage

- https://www.w3.org/Graphics/SVG/

SVG 2 recommendation

- https://www.w3.org/TR/SVG/

David Duce et al.: *SVG Tutorial*

- https://www.w3.org/2002/Talks/www2002-svgtut-ih/hwtut.pdf

W3schools: *SVG tutorial*

- https://www.w3schools.com/graphics/svg_intro.asp

Local example page

- https://cgg.mff.cuni.cz/~pepca/lectures/examples/SVG/

matfyz

Computer Graphics Charles University

HDR obraz (High Dynamic Range)

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

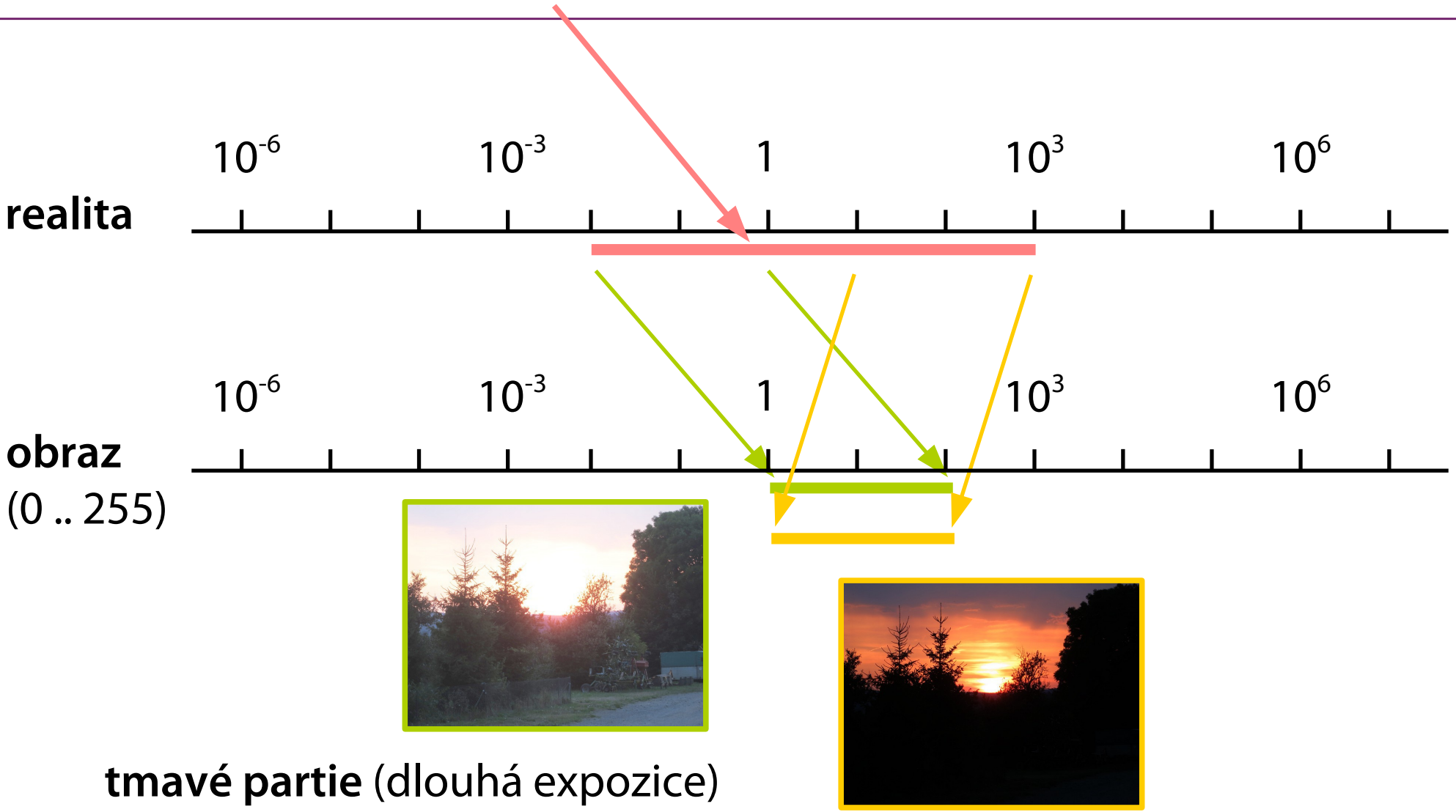
https://cgg.mff.cuni.cz/~pepca/

Velká dynamika obrazu

světlé partie
(krátká expozice)

tmavé partie
(dlouhá expozice)

Velká dynamika obrazu



tmavé partie (dlouhá expozice)

světlé partie (krátká expozice)

HDR grafika

High Dynamic Range

- velký dynamický rozsah
- pixely – čísla v **plovoucí desetinné čárce**
- např. **float[3]** pro RGB (96bpp)

Pořizování HDR dat

- výpočet (rendering)
- fotografování (vícenásobná expozice)

Zobrazování na LDR zařízení

- převod do normální škály („tone-mapping“)

Formát pixelu RGBE (Radiance)

Formát souboru **.hdr (Radiance)**

- úsporné uložení (jen 4 byty na pixel)
- individuální mantisa [RGB], společný exponent [E]

Mantisa [RGB]

- typ **float**, normalizován mezi ½ a 1 (maximální složka)

Exponent [E]

- binární exponent v doplňkovém kódu (8-bitové číslo)

Příklad: [**0.3, 0.02, 0.1**]

$= [\ 0.6,\ 0.04,\ 0.2\ ] \cdot 2^{-1} \rightarrow$ [**153, 10, 51, 127**]

Další HDR formáty

OpenEXR (.exr)

- Industrial Light & Magic (G. Lucas 1975, Star Wars etc.)
- zcela otevřený, knihovny jsou open-source
- různé typy komprese (ZIP, wavelets), používá často typ **half** (fp16)
- uživatelsky **rošiřitelný formát pixelu**

Portable Float Map (PFM)

- analogický PPM / PGM / PBM
- pixely jsou tři čísla typu **float**
- bez komprese

```
PF
1024 768
1
<binary data>
```

Fotografování HDR

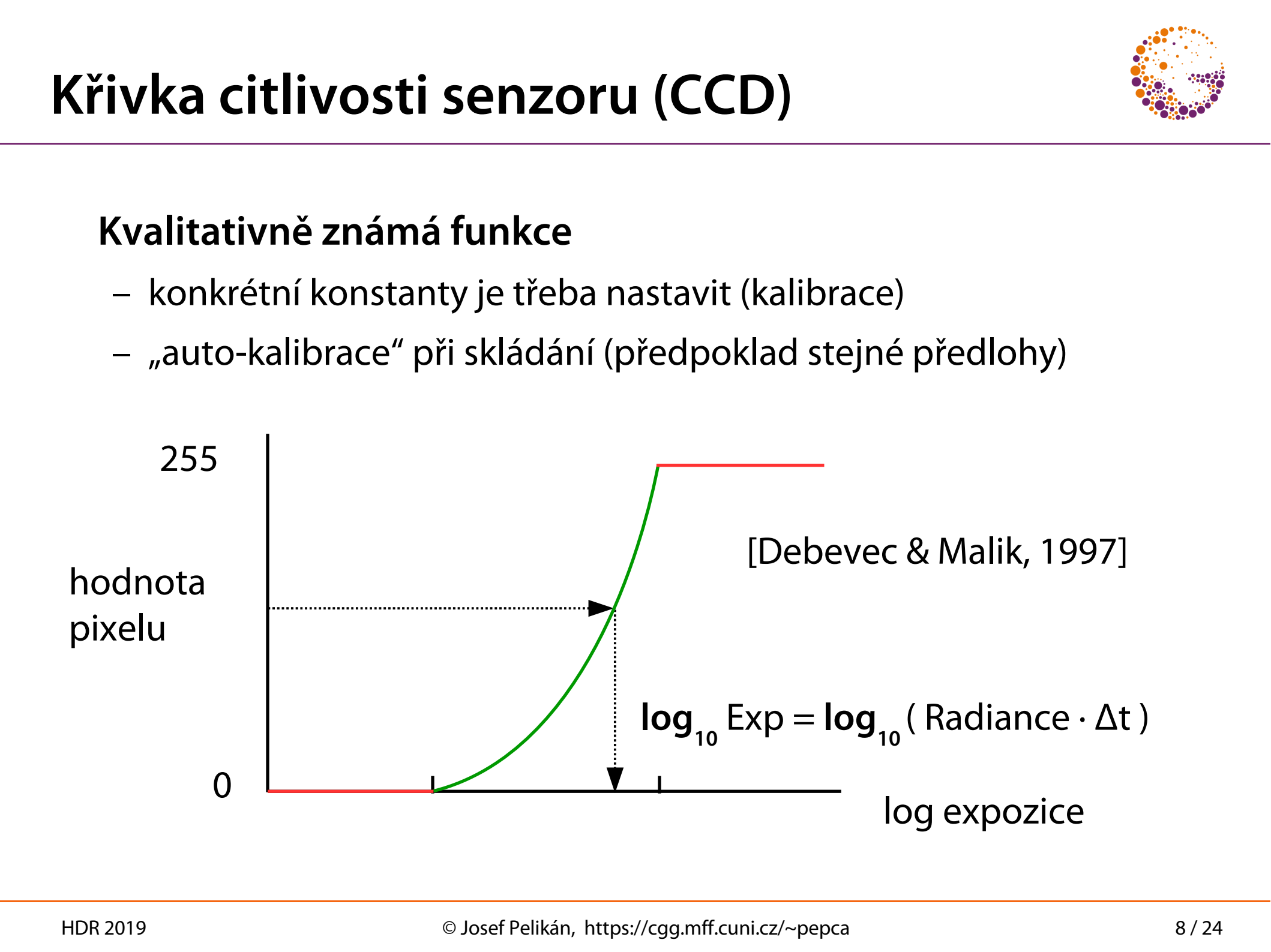
Vícenásobná expozice

- statická scéna
- konstantní clona, proměnlivý čas
- sekvence např. od **1/1000s** do **2s**
- vestavěný „bracketing“ (–2 EV, 0, +2 EV)
- „super-bracketing“ (5-7 rychlých expozic za sebou, ≥ EOS 80D i víc)

Zpracování sekvence obrázků do jednoho HDR

- běžné programy na zpracování foto (PhotoShop, GiMP …)
- Picturenaut (http://www.hdrlabs.com/picturenaut)
- užitečné funkce: registrace snímků, auto-kalibrace

Křivka citlivosti senzoru (CCD)



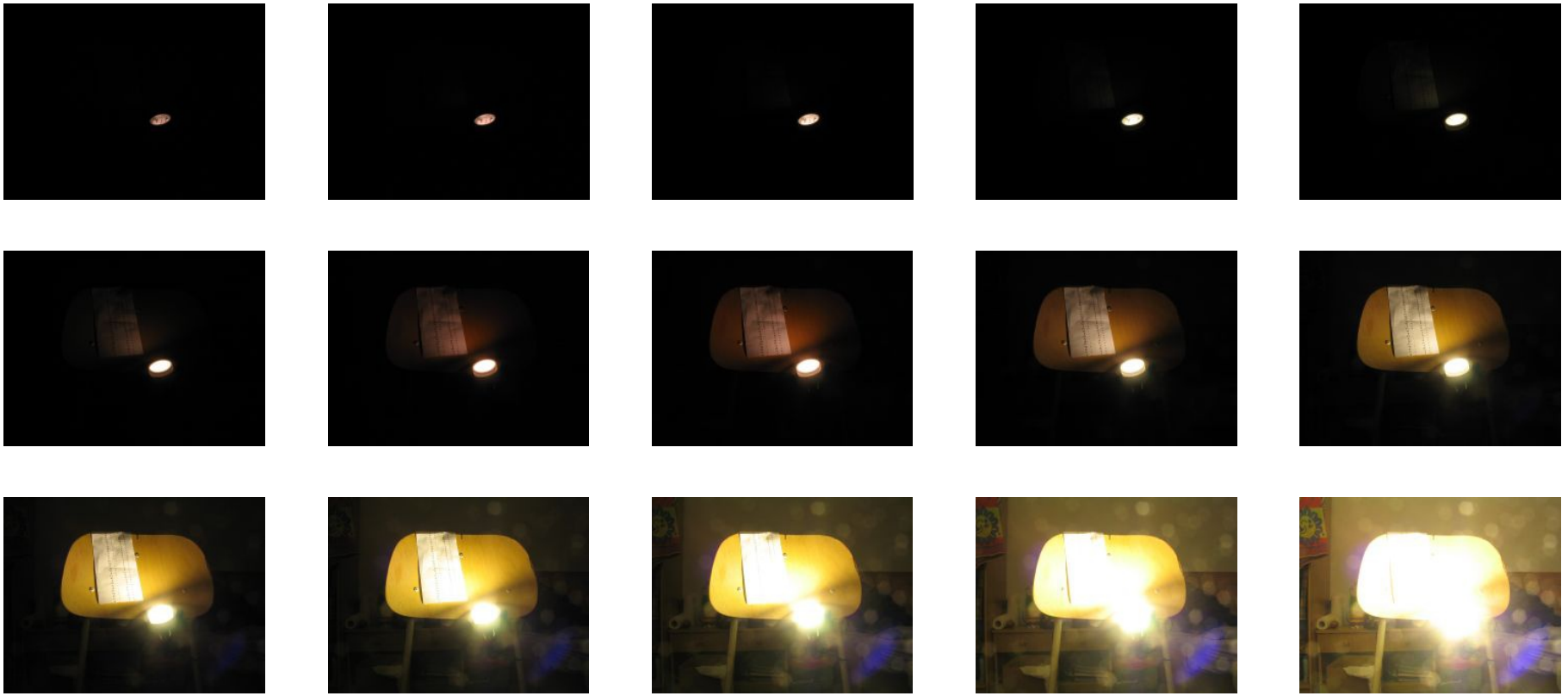
Kvalitativně známá funkce

- konkrétní konstanty je třeba nastavit (kalibrace)
- „auto-kalibrace“ při skládání (předpoklad stejné předlohy)

Příklad pořízení HDR

15 snímků mezi 1/2000s a 8s (rozestup 1 EV)

Sestavení … HDR Shop



Reprodukce HDR

Jednoduché **ořízntutí dynamiky**

- přetečení → přezáření (bílá nebo nějaké „glare“ efekty)

„Tone mapping“

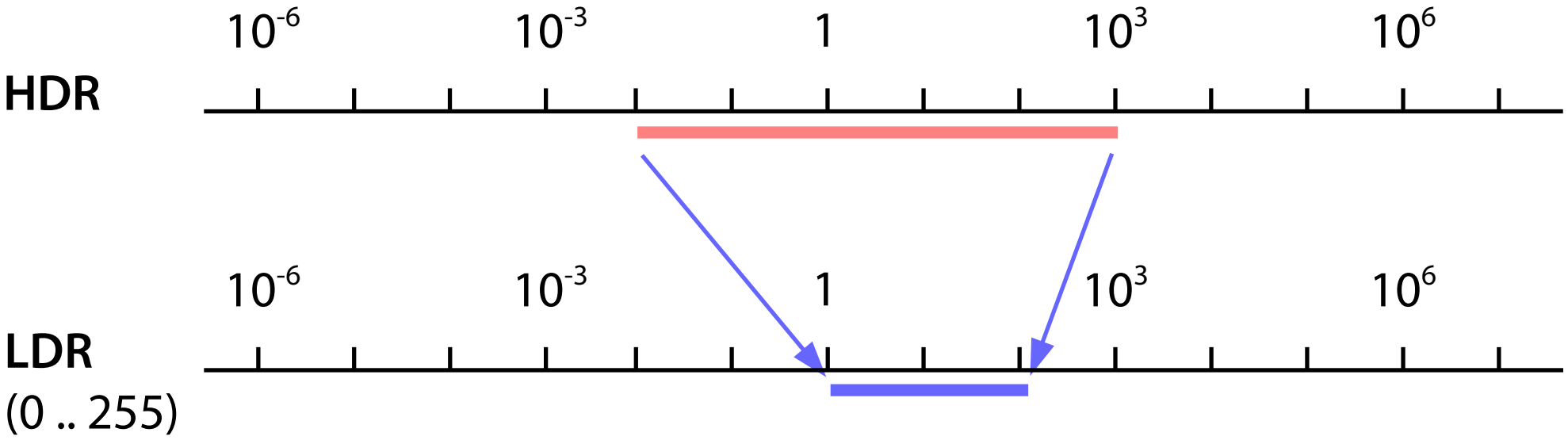
- obecně: transformace **celé** originální škály do LDR
- globální vs. lokální zobrazení
- lokální zachování kontrastu, apod.

„Tone mapping“

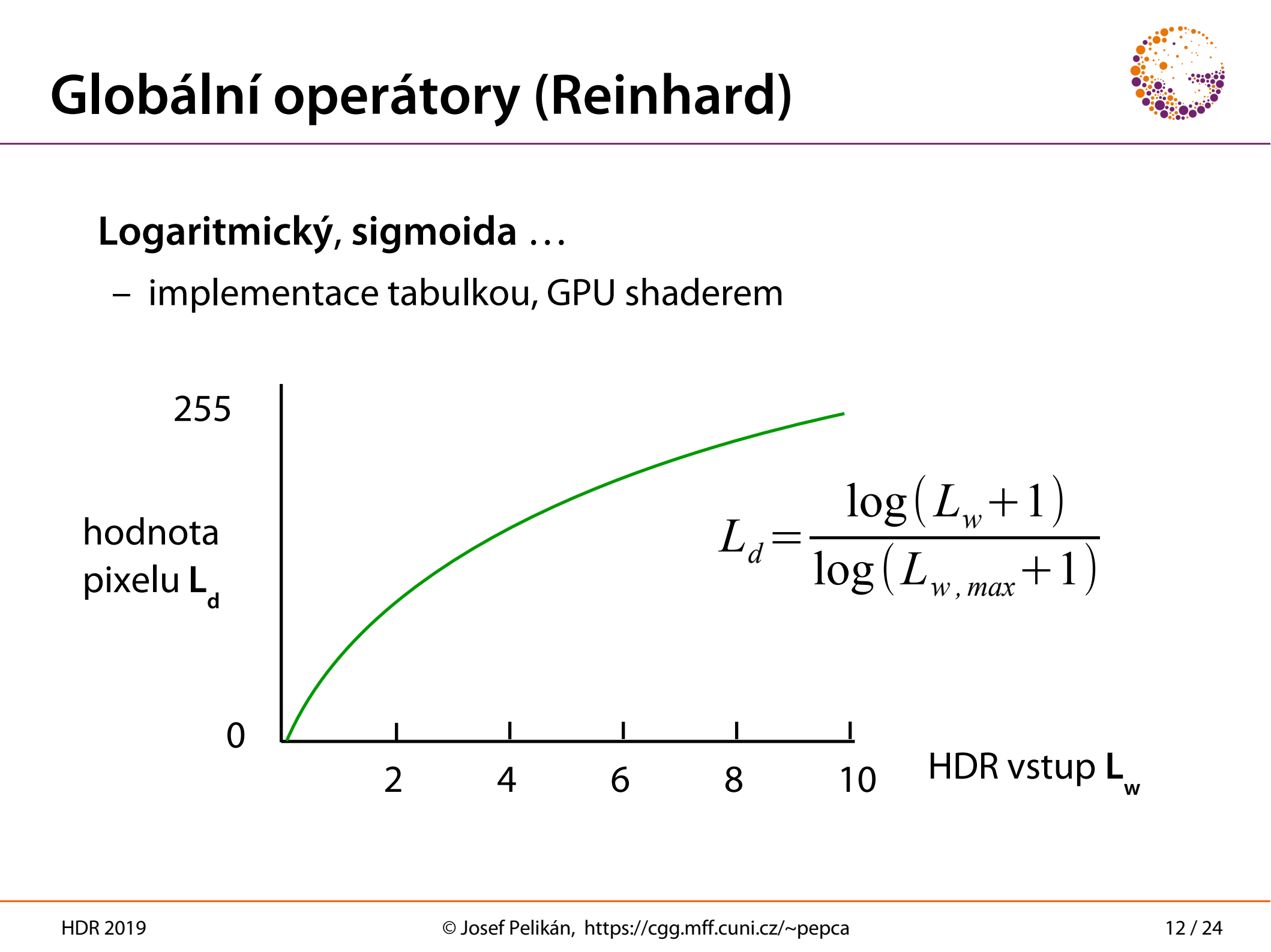
Transformace HDR rozsahu do LDR

- zachování kresby (kontrastu) v tmavých i světlých partiích

Globální vs. **lokální** zobrazení



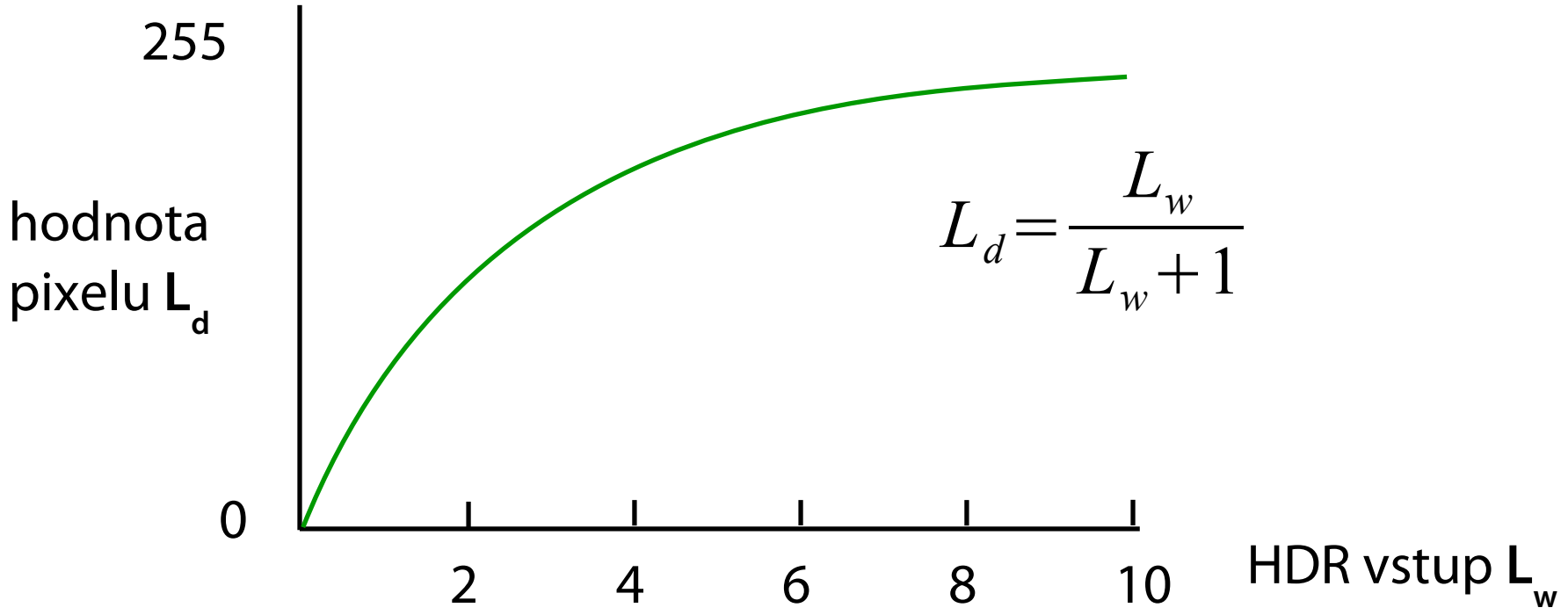
Globální operátory (Reinhard)



Logaritmický, sigmoida …

– implementace tabulkou, GPU shaderem

Sigmoida



Gamma komprese

Již existující mechanismus

- nevýhoda: útlum barev

$$L_d = L_w^{\gamma}$$

Gamma komprese intenzity

- barevná informace zůstává
- intenzita se komprimuje jako výše

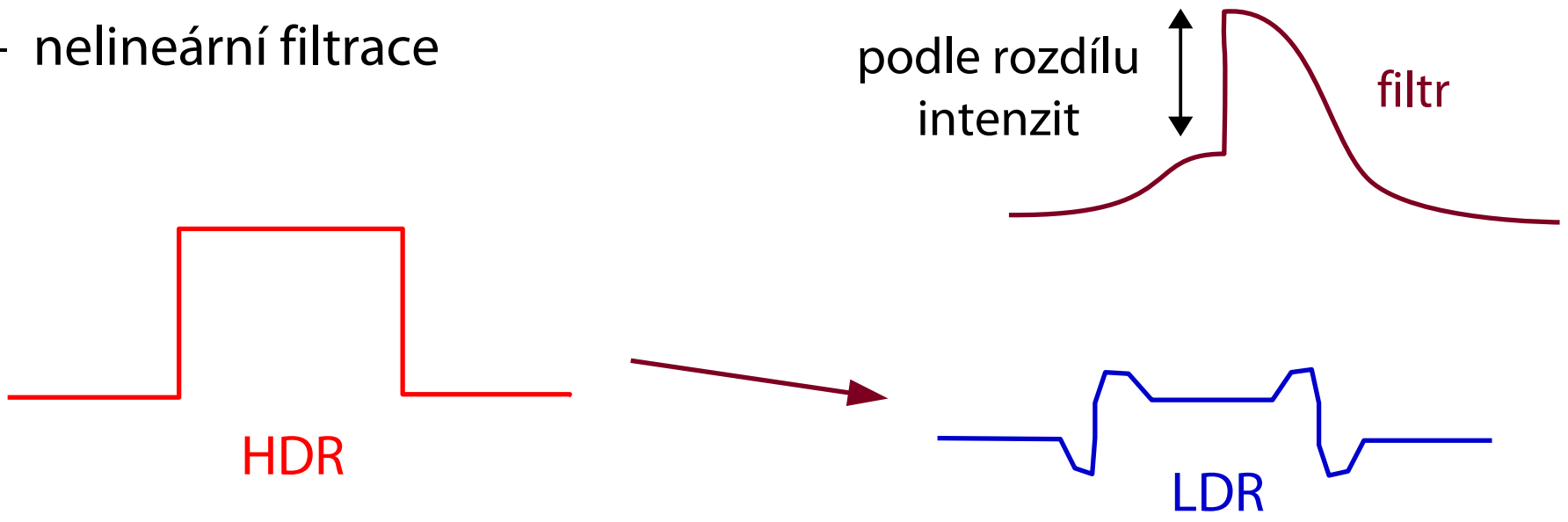
Lokální operátory

Lokálně zachovávají kontrast

- detekce hran
- lokální analýza jasu (průměr jasu, lokální histogram …)

Bilateral filtering (1998+)

- nelineární filtrace



Aplikace HDR

Fotografie

- lepší reprodukce přirozené dynamiky scény
- citlivý „tone mapping“, výsledek nepůsobí nepřirozeně
- HDR panoramata (slunce v záběru, obloha vs. terén)

CGI (počítačem generovaná grafika)

- dobře použitelná data pro **„environment mapping“** (světelná mapa okolí)
- všechny interní výpočty a mezivýsledky jsou „HDR“
- realisticky vypadající lesklé odrazy, rozmazání pohybem …

Příklady – „tone mapping“

LDR

Tone-mapped HDR

Příklady – „tone mapping“

Příklady – „tone mapping“

Lokální „tone mapping“

Příklad s odrazem světla

Environment-map (360/180° latitude/longitude mapping)

Příklad s odrazem světla

Environment map („cube-map“)

Reference

Další informace:

Erik Reinhard: ***Tone Reproduction*** (slides, Bristol)

http://www.pauldebevec.com/

https://cgg.mff.cuni.cz/~pepca/hdr/

http://www.hdrlabs.com/

https://www.openexr.com/

http://www.mpi-inf.mpg.de/resources/hdr/

http://www.hdrshop.com/

matfyz

Computer Graphics Charles University

Kompozice rastrových obrázků

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/

Kompozice obrázků

Montáž několika reálných obrázků

- vkládání objektů do jiného pozadí …

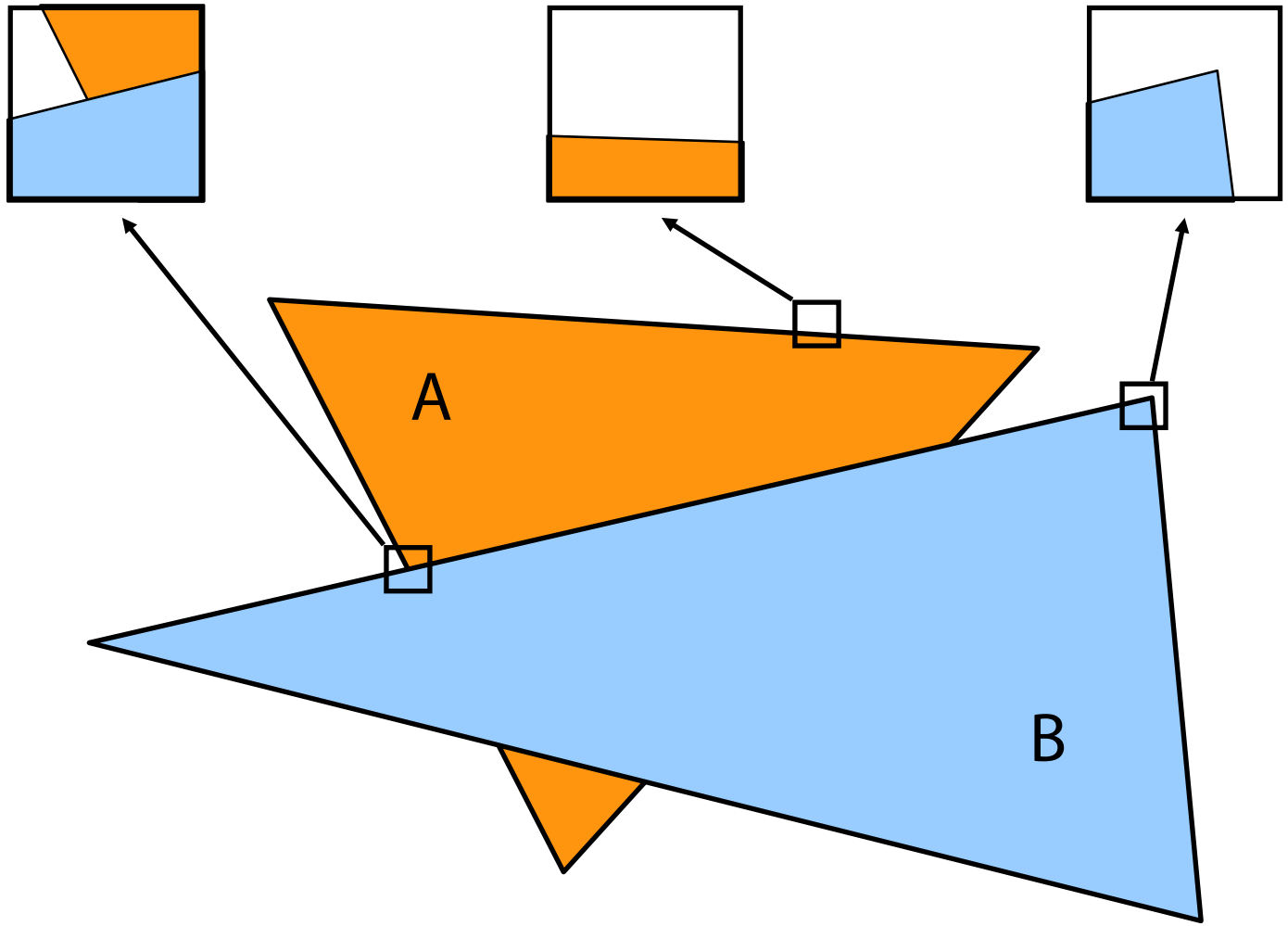
Prolínání obrázků, **„fade-in”**, **„fade-out”**

- animace, střih

Syntéza obrazu

- skládání umělého obrázku z několika samostatně vyrobených dílů (např. pozadí, popředí, plameny, mlha …)

Pokrytí plochy pixelu



Kanál alfa

Procentuální **pokrytí pixelu** neprůsvitnou barvou

- doplněk **průhlednosti**
- $\alpha = \mathbf{0}$ … zcela průhledný pixel (nemá vliv na výsledek)
- $\alpha = \mathbf{1}$ … neprůhledný pixel („nic za ním neprosvítá")

Ukládání hodnoty $\boldsymbol{\alpha}$ v každém pixelu

- často celočíselná reprezentace $(0 \div 255)$
- čtveřice $[\ \mathbf{R, G, B,}\ \alpha\ ]$
- ještě častější reprezentace $[\ \mathbf{R}\alpha, \mathbf{G}\alpha, \mathbf{B}\alpha,\ \alpha\ ]$

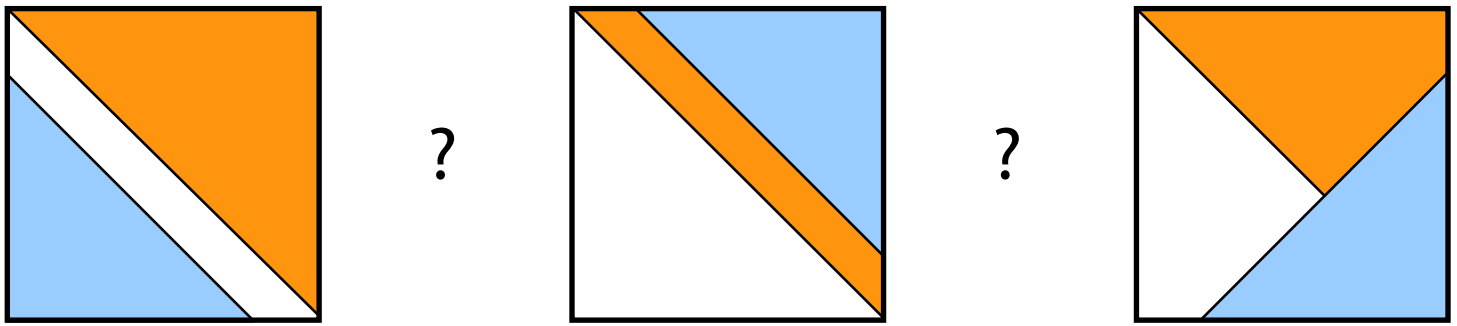
Skládání dvou obrázků

Dva skládané pixely **[A, α_A]** resp. **[B, α_B]**

– potřebuji určit výslednou hodnotu **[C, α_C]**

? **model pro skládání pixelů** ?

$\alpha_A = 0.5$ $\alpha_B = 0.4$

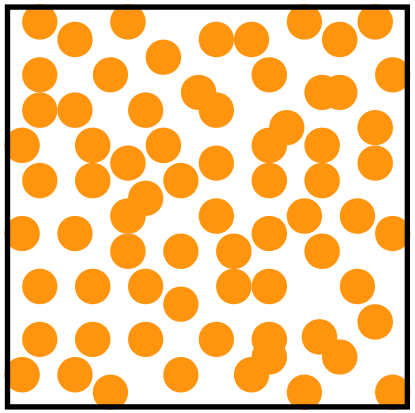


Model pokrytí pixelu

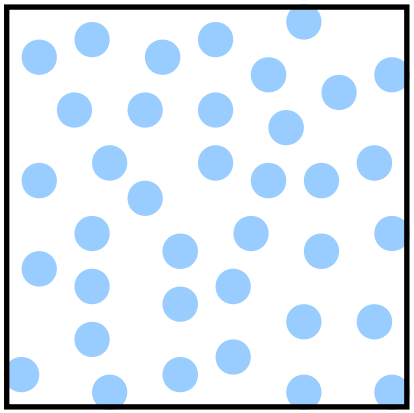
Pixel [**A,** α_A] je **náhodně pokryt** barvou **A**
s rovnoměrně rozloženou pravděpodobností α_A

- skládání geometricky nezávislých tvarů
- vyhovuje ve většině případů

$\alpha = 0.5$

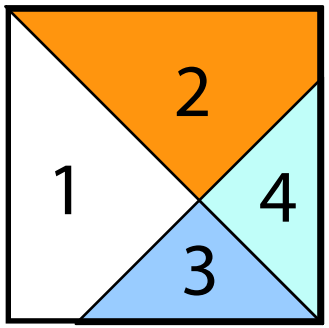


$\alpha = 0.2$



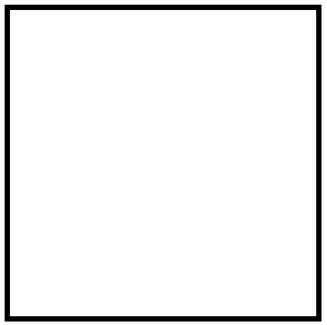
Překrytí dvou pixelů

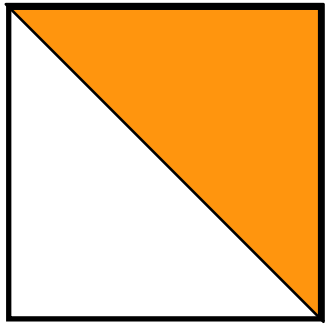
oblast		plocha	vybarvení
1	nic	$(1 - \alpha_A)(1 - \alpha_B)$	0
2	A	$\alpha_A(1 - \alpha_B)$	0, A
3	B	$\alpha_B(1 - \alpha_A)$	0, B
4	A i B	$\alpha_A\alpha_B$	0, A, B

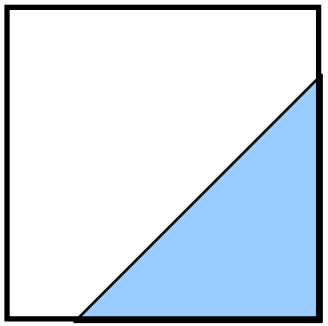


celkem **12 možností**

Operace skládání dvou pixelů

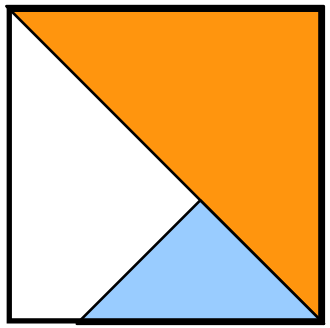


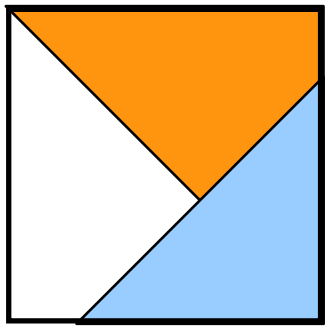


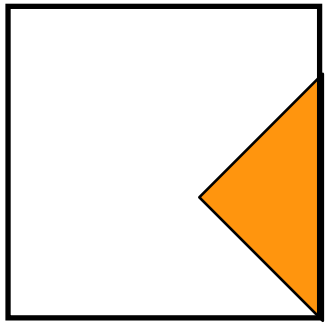


	nic clear	A	B
barvy	(0,0,0,0)	(0,A,0,A)	(0,0,B,B)
F_A	0	1	0
F_B	0	0	1

Operace skládání dvou pixelů

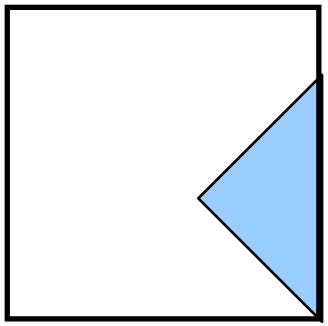


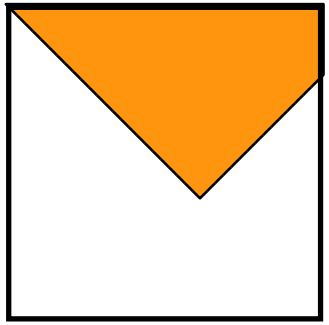


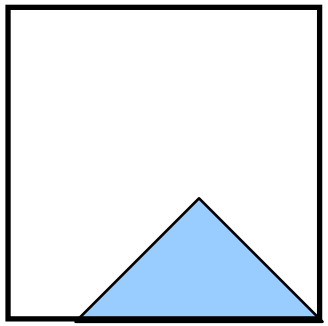


	A přes B A over B	B přes A B over A	A v B A in B
barvy	(0,A,B,A)	(0,A,B,B)	(0,0,0,A)
F_A	1	$(1 - \alpha_B)$	α_B
F_B	$(1 - \alpha_A)$	1	0

Operace skládání dvou pixelů

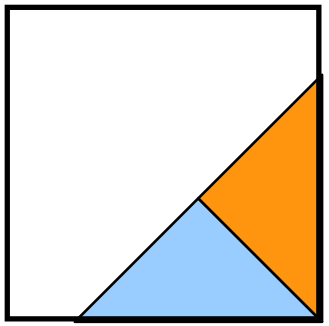


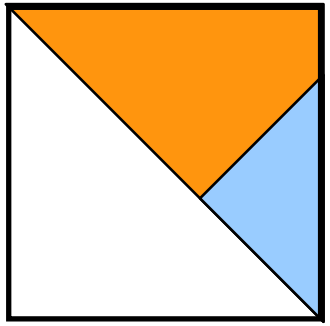


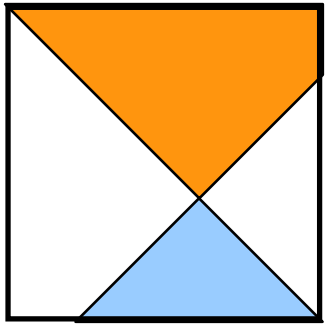


	B v A B in A	A mimo B A out B	B mimo A B out A
barvy	(0,0,0,B)	(0,A,0,0)	(0,0,B,0)
F_A	0	$(1 - \alpha_B)$	0
F_B	α_A	0	$(1 - \alpha_A)$

Operace skládání dvou pixelů







	A na povrchu B A atop B	B na povrchu A B atop A	A xor B
barvy	(0,0,B,A)	(0,A,0,B)	(0,A,B,0)
F_A	α_B	$(1 - \alpha_B)$	$(1 - \alpha_B)$
F_B	$(1 - \alpha_A)$	α_A	$(1 - \alpha_A)$

Implementace

Čtveřice **RGBα** se **ukládají** jako [**Rα, Gα, Bα,** α]

- při skládání se stejně vždy barva násobí alfa-kanálem

Při **zpětném převodu** do RGB by se barevné složky vydělily alfa-kanálem

- nedělá se to často
- pouhé odstranění čtvrté složky dává lepší výsledek

Při operaci **skládání dvou pixelů** se násobí všechny čtyři složky faktorem $\mathbf{F}_x$

- operace lineární kombinace na čtveřicích (SSE, GPU)

Operátory – shrnutí

Binární operace **A op B** (F_A, F_B podle tabulky)

$$[\, F_A R_A + F_B R_B,\ F_A G_A + F_B G_B,\ F_A B_A + F_B B_B,\ F_A \alpha_A + F_B \alpha_B \,]$$

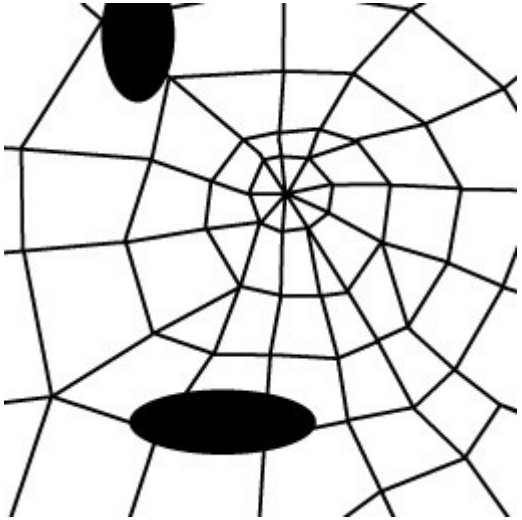
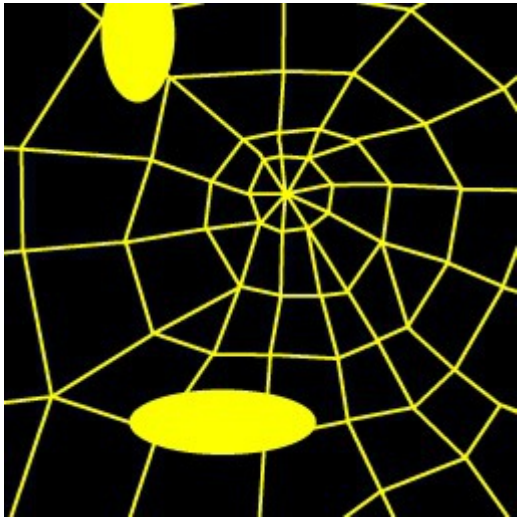
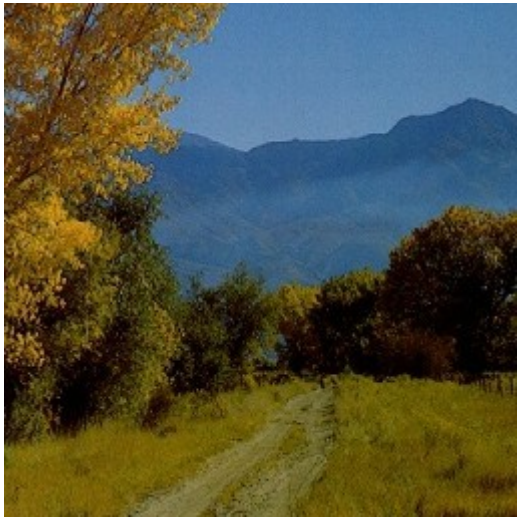
darken (A, ρ **)** $= [\, \rho R_A,\ \rho G_A,\ \rho B_A,\ \alpha_A \,]$

fade (A, δ **)** $= [\, \delta R_A,\ \delta G_A,\ \delta B_A,\ \delta \alpha_A \,]$

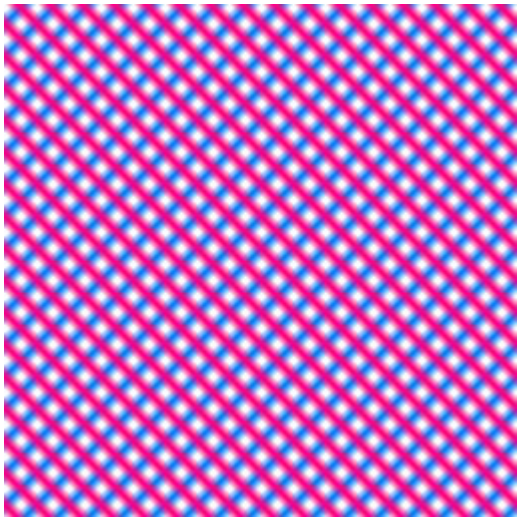
opaque (A, ω **)** $= [\, R_A,\ G_A,\ B_A,\ \omega \alpha_A \,]$

Ukázka – vstupy

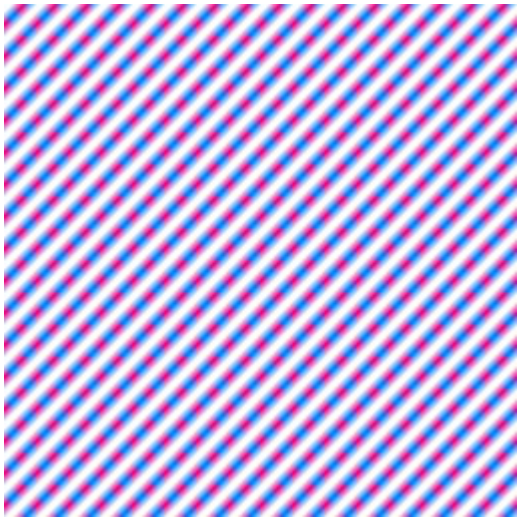


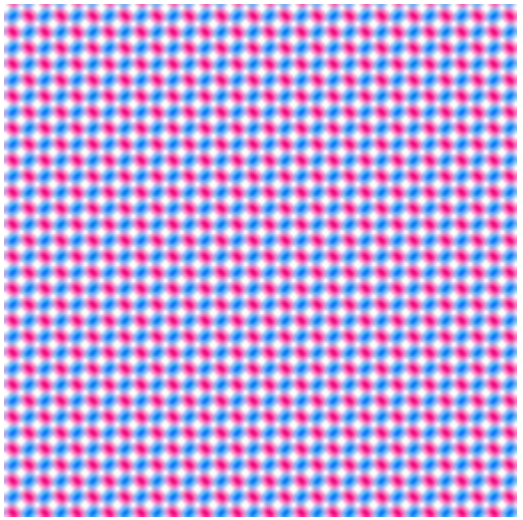
Ukázka – binární operace I



1 over 2



1 atop 2

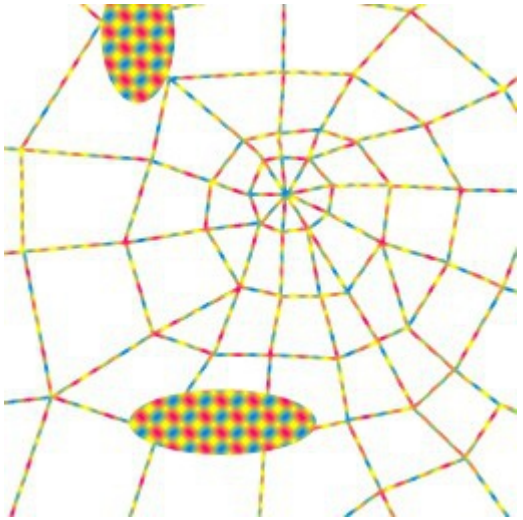


1 xor 2

Ukázka – binární operace II

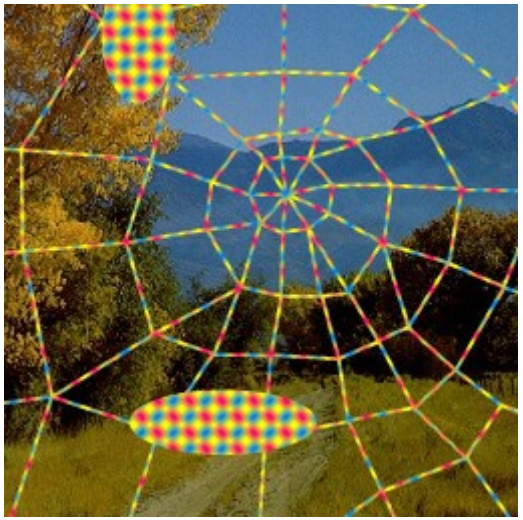
1 in 2

1 out 2



(1 xor 2) atop W

Ukázka – binární operace III

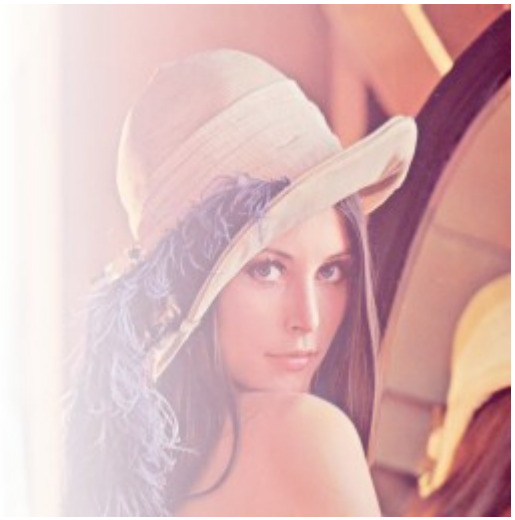


((1 xor 2) atop W) over V

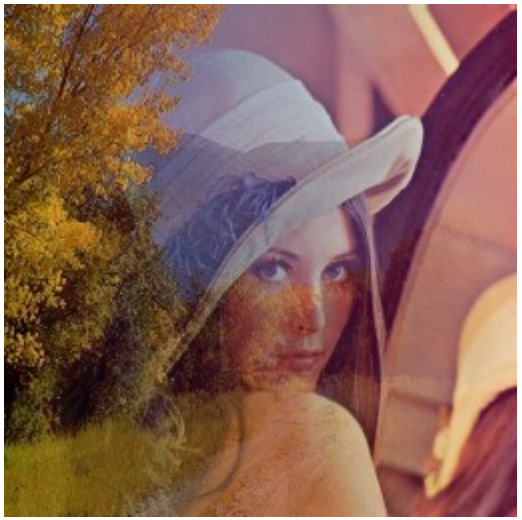
(V atop (1 xor 2)) over L

V atop (1 xor 2)

Ukázka – prolínání



fade(L, horiz)



fade(L, horiz) over V

Operace „plus”

Aditivní operátor **A plus B**

[R_A+R_B, G_A+G_B, B_A+B_B, $\alpha_A+\alpha_B$]

– pozor na přetečení!

Příklad 1: **prolínání dvou obrázků**

fade(A,t) plus fade(B,1 - t)

Příklad 2: **hořící strom**

(FFire plus (BFire out Tree)) over darken(Tree,0.8) over Background

Originál je z filmu Star Trek II (1982) – „Genesis Effect“
https://www.youtube.com/watch?v=Qe9qSLYK5q4

Literatura

J. Foley, A. van Dam, S. Feiner, J. Hughes: ***Computer Graphics, Principles and Practice***, 835-843

T. Porter, T. Duff (Lucasfilm): ***Compositing Digital Images*****,** Computer Graphics 18(3), 1984

– https://keithp.com/~keithp/porterduff/p253-porter.pdf

matfyz

Computer Graphics Charles University

Filtrace obrazu

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

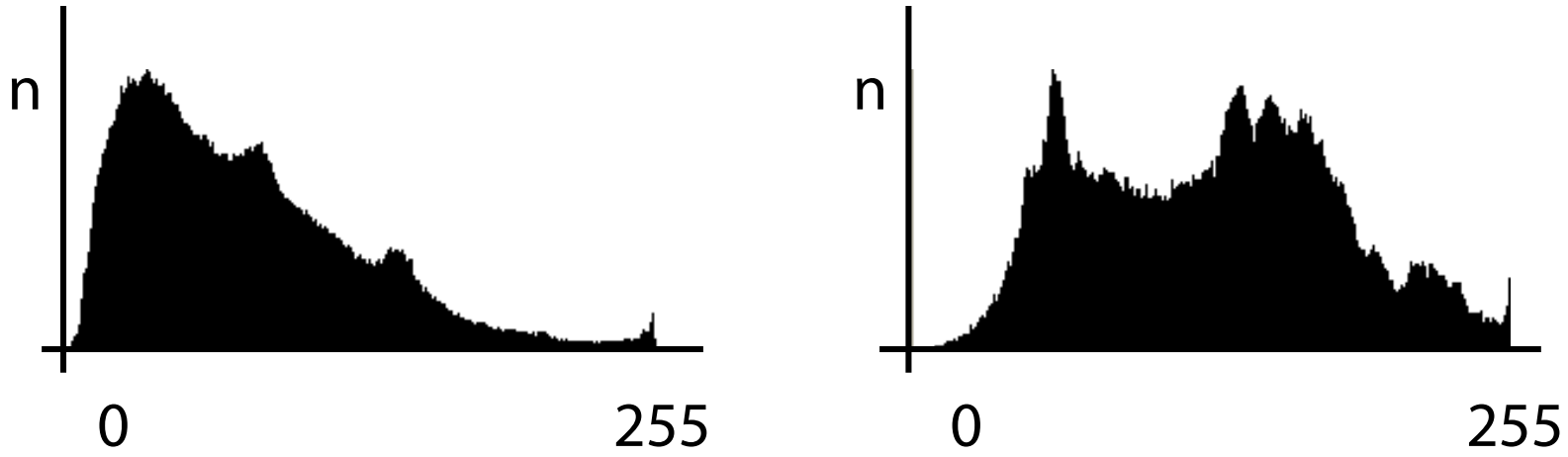
https://cgg.mff.cuni.cz/~pepca/

Histogram obrázku

Tabulka četností jednotlivých jasových (barevných) hodnot

- **spojitý případ** – hustota pravděpodobnosti

Přímé použití – fotografie



Jasové poměry obrázku

Histogram → první odhad **expozice obrazu**

Přeexponované nebo podexponované snímky

Nedostatečný nebo příliš velký **kontrast**

„**Dobrý histogram**“

- obraz má odstíny ve všech částech škály
- ~ detaily čitelné ve stínech i jasných partiích

Nedal by se špatný histogram „opravit“?

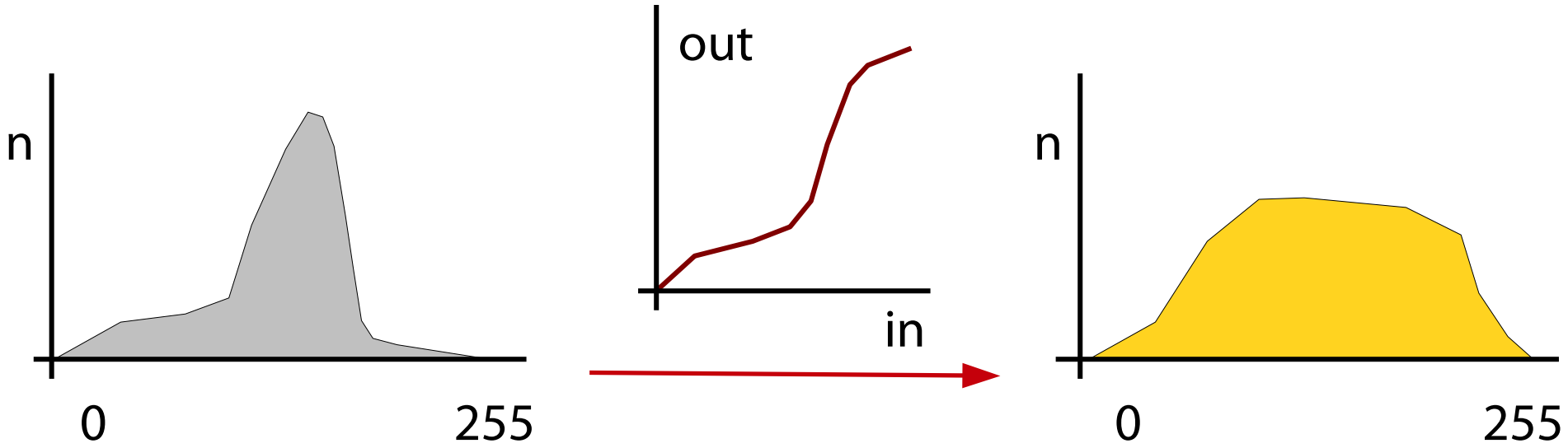
Jasová transformace

Převodní funkce („transfer function“) mezi jasy na vstupu a výstupu

– t: R → R (obyčejně [0, 1] → [0, 1])

Gamma-korekce

Zvětšování kontrastu



Ekvalizace histogramu

Umělá jasová transformace

- snaží se o vyrovnaný histogram

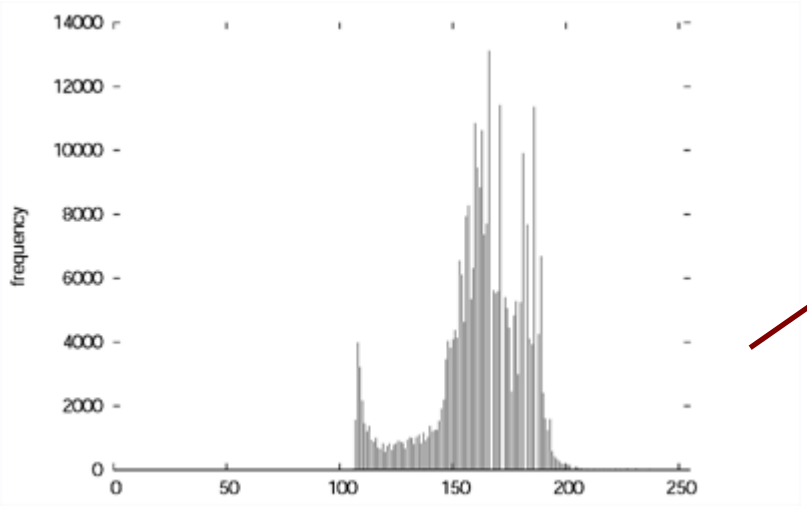
Manipuluje s celými jasovými „sloupečky“

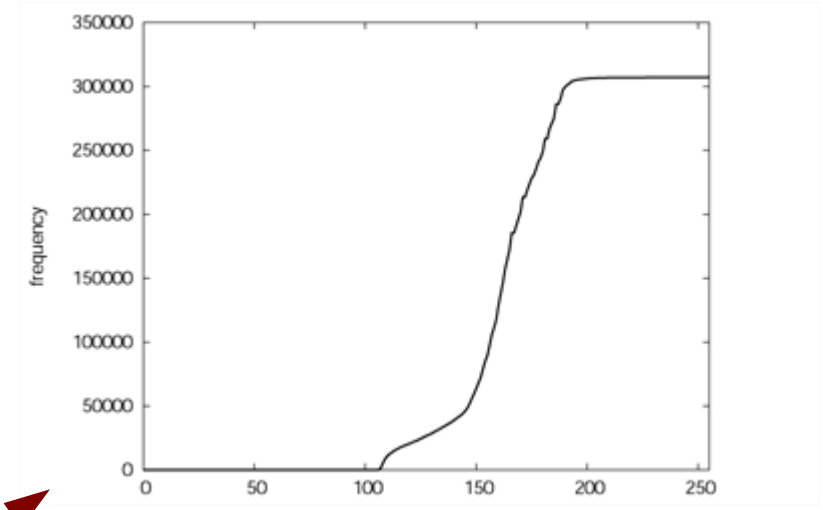
- případně rozděluje četnější odstíny stochasticky

Lokální ekvalizace histogramu

- analýzu dělá jen na (větším) okolí daného pixelu
- může zlepšit čitelnost na celé ploše obrázku
- nezachovává jednobarevné plochy!

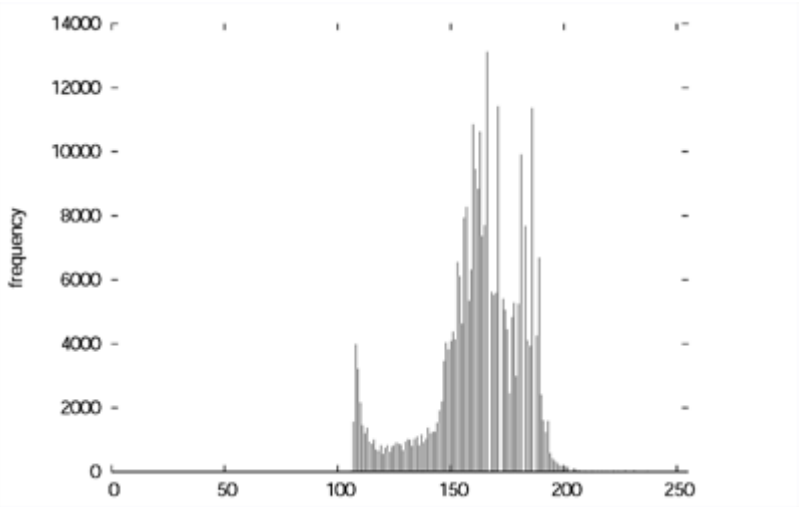
Příklad globální ekvalizace

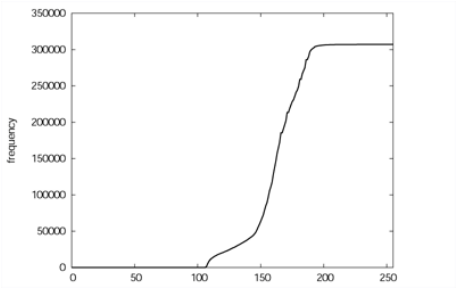


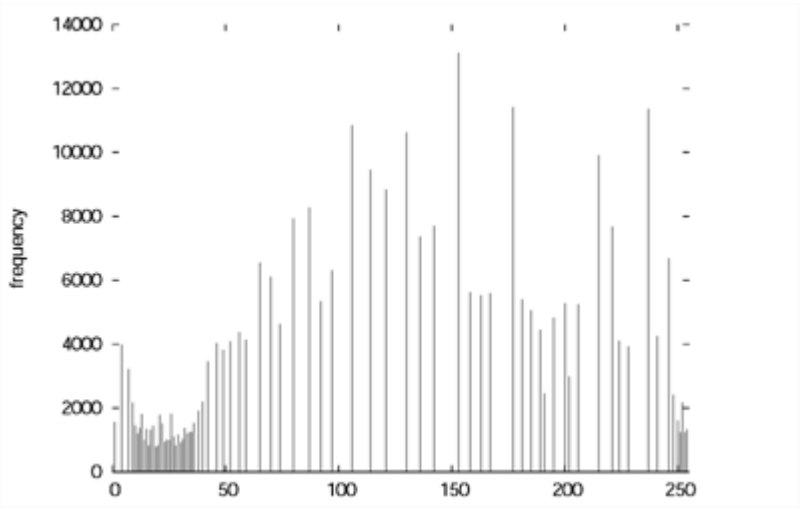


Kumulovaný histogram (jasová transformace)

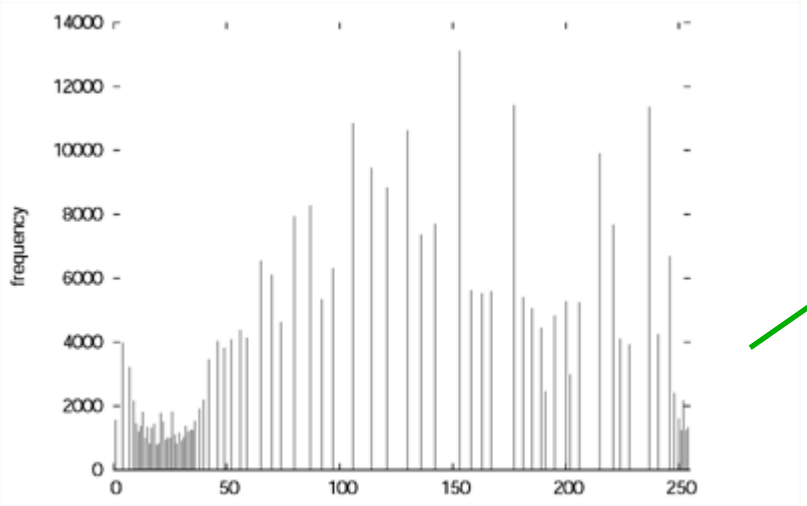
Jasová transformace

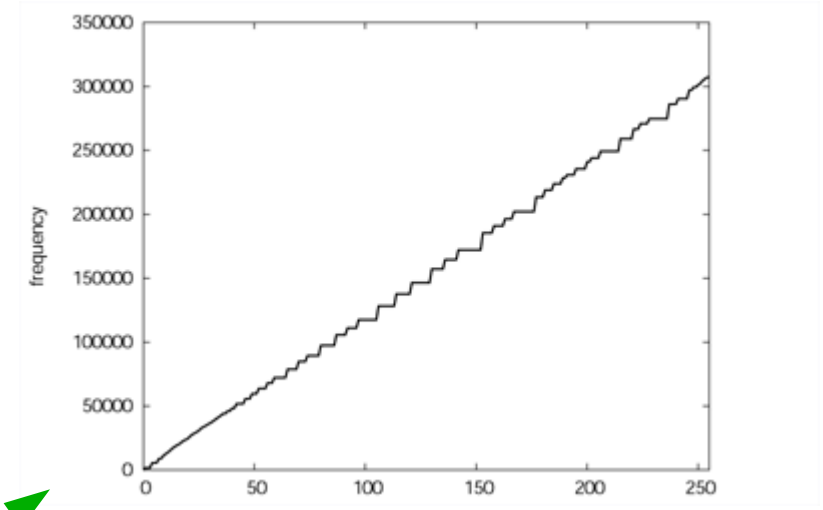






Výsledek po ekvalizaci





Kumulovaný histogram

Barevné operace

1. převod **RGB** → **HSV**

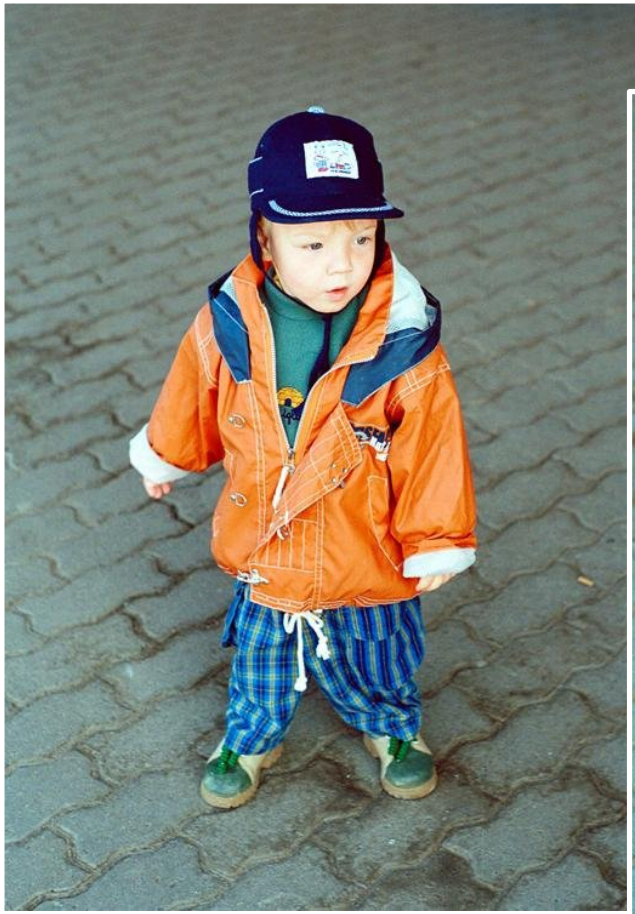
2a. manipulace se **sytostí S**

2b. manipulace s **odstínem H**

- změna barvy objektu
- selektivní přebarvování …

3. převod zpět **H'S'V'** → **R'G'B'**

HSV operace



HSV operace

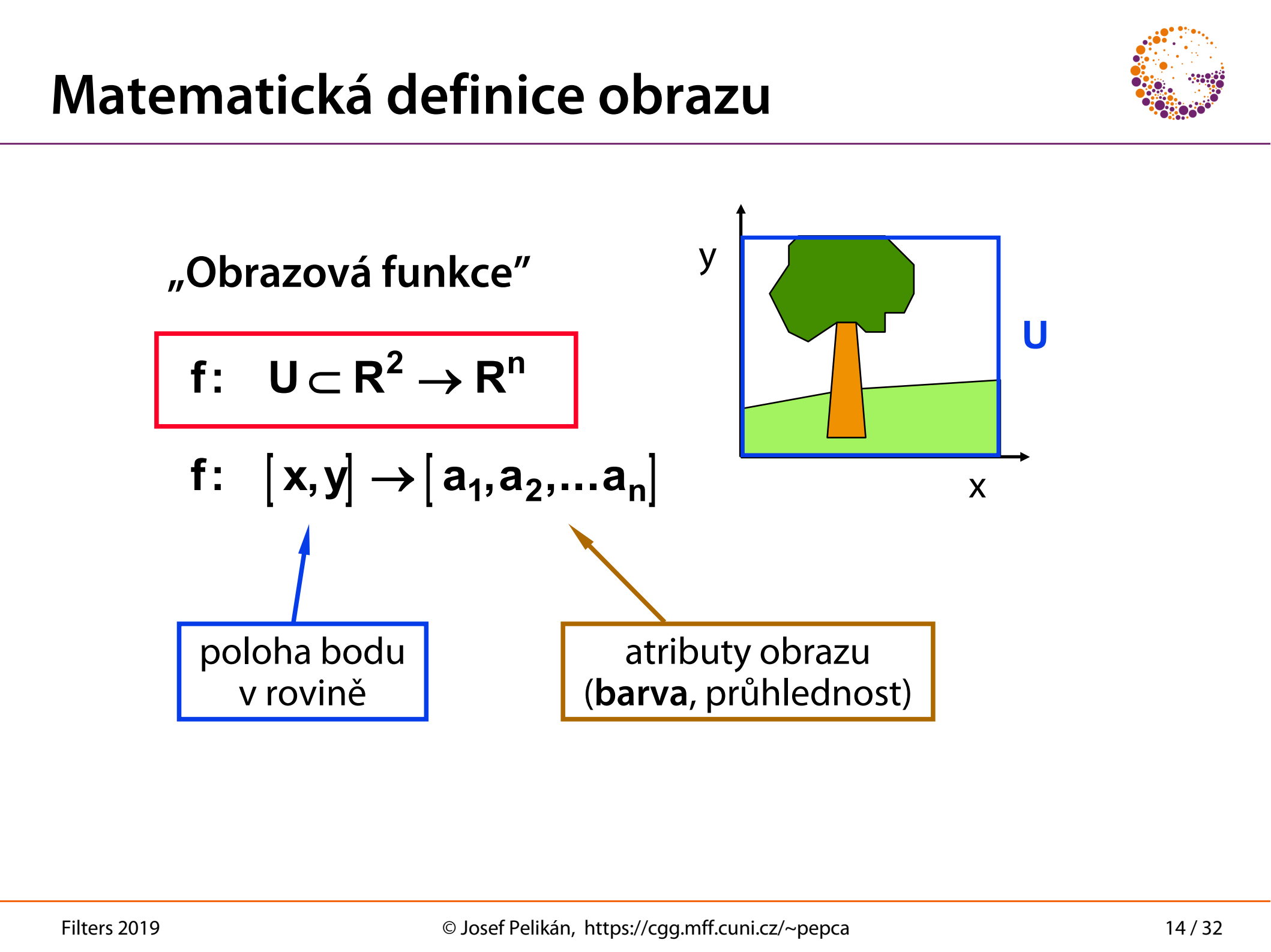
Příklady barevných operací

(algoritmus: Miroslav Hrivík)

Příklady barevných operací

(foto a algoritmus: David Marek)

Matematická definice obrazu



„Obrazová funkce"

$$f: \quad U \subset R^2 \rightarrow R^n$$

$$f: \quad [x, y] \rightarrow [a_1, a_2, \ldots a_n]$$

poloha bodu v rovině

atributy obrazu (**barva**, průhlednost)

Konvoluce

Spojitá varianta „váženého klouzavého průměru“

- váhová funkce (konvoluční jádro) **g**

Úzká souvislost s **Fourierovou transformací**

- spektrální prostor
- filtry typu „dolní propust“ apod.

$$(f * g)(x) = \int_{-\infty}^{\infty} f(t) \cdot g(x-t)\, dt$$

(1D varianta)

Diskrétní konvoluce

„Vážený klouzavý průměr“ na posloupnosti (tabulce)

– váhová posloupnost (tabulka) **g**

Souvislost s **diskrétní Fourierovou transformací**

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m] \cdot g[n-m]$$

(1D varianta)

Účinky konvoluce

Dolní propust (jen kladné hodnoty **g**)

- rozmazání obrazu
- potlačení šumu

Horní propust (kladné i záporné hodnoty, součet **0**)

- detekce hran v obraze
- po modifikaci: ostření obrazu

Složitější spektrální filtry

Další efekty („emboss“ …)

Rozmazání obrazu

Originál

Gauss

1	*2*	*1*
2	*4*	*2*
1	*2*	*1*

/ 16

Detekce hran („high-pass filter“)

Originál

Sobel (2 směry)

1	*2*	*1*
0	*0*	*0*
-1	*-2*	*-1*

1	*0*	*-1*
2	*0*	*-2*
1	*0*	*-1*

Ostření obrazu

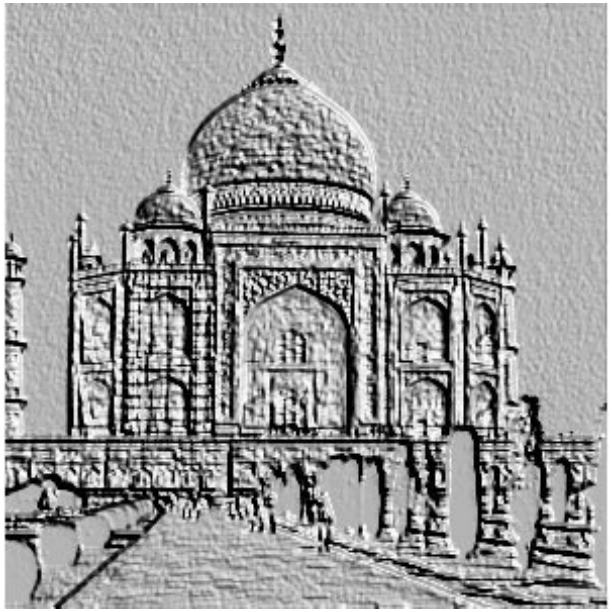
Laplacián

0	***-1***	***0***
-1	***4***	***-1***
0	***-1***	***0***

Zaostření

0	***-1***	***0***
-1	***5***	***-1***
0	***-1***	***0***

„Emboss“ efekt



Emboss

Originál

-1	***0***	***0***
0	***1***	***0***
0	***0***	***0***

Neuniformní rozmazání

Originál

Radiální rozmazání
(1D rozmazání)

Nelineární filtry („rank filtry“)

Okénková filtrace (podobně jako u konvoluce)

V okénku se hodnoty pixelů **seřadí podle velikosti**

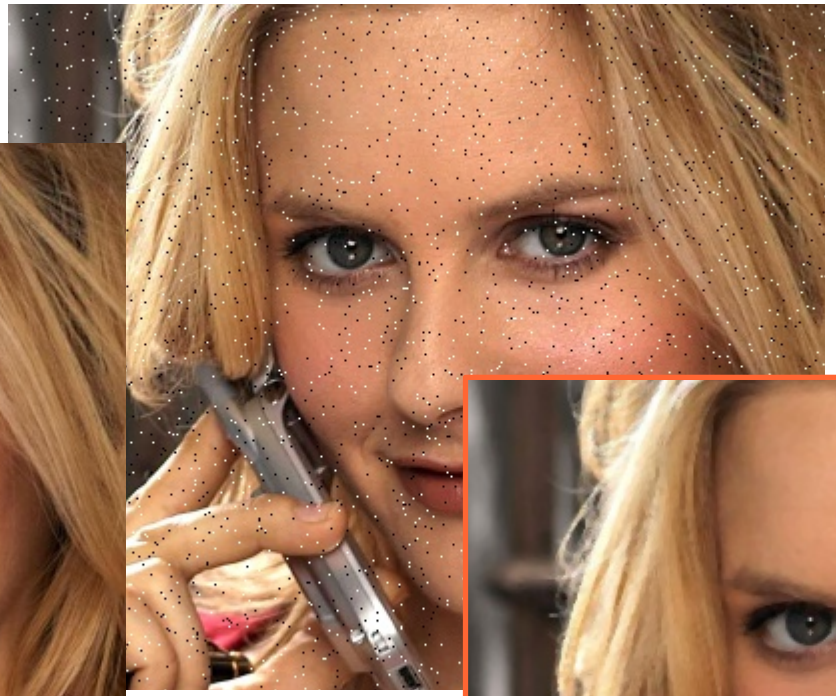
- **medián:** potlačení šumu, umělecké efekty …
- **minimum:** „eroze“
- **maximum:** „dilatace“

Různé tvary okna

- čtverec
- kruh
- křížek (zachová ostré rohy)

Medián na potlačení šumu

originál



pepř & sůl

medián 3×3

Dilatace a eroze

dilatace

eroze

Potlačování šumu

Pokročilejší metody se snaží o **zachování hran** obrazu

- nelze použít obyčejnou redukci vysokých frekvencí

Varianty **mediánového filtru**

Neizotropické filtrování

- rozmazávání se děje ve směru „vrstevnic“ obrazu (kolmo na gradient obrazové funkce)

Filtrace **rotující maskou**

- několik uvažovaných okolí daného pixelu
- průměruje („mediánuje“) se v okénku s minimálním rozptylem

Umělecké efekty

Napodobení **malířských/kreslířských technik**

Simulované tahy štětcem/perem/pastelem

Efekty typu „**mozaika**“, vitráž, …

NPR (nefotorealistické) efekty

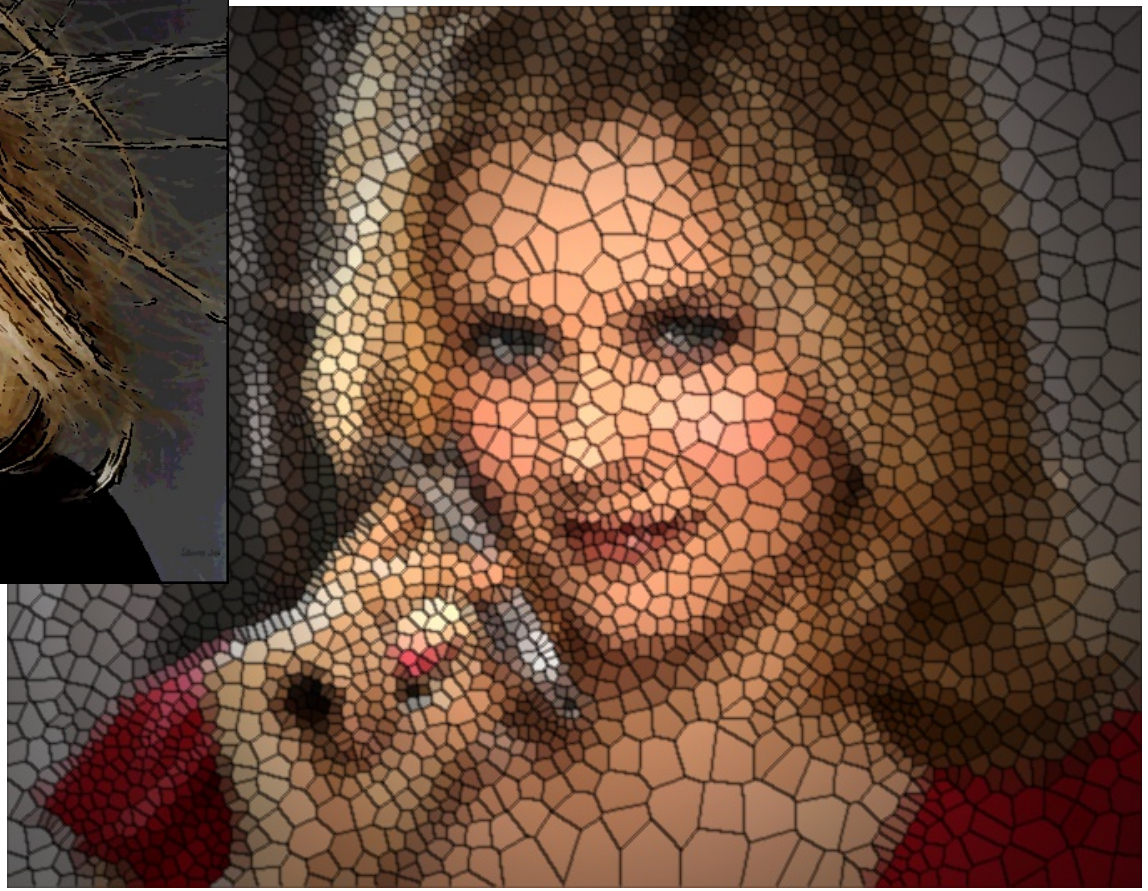
- zvýrazňování hran
- vyplňování vnitřních oblastí objektů
- narůstání oblastí (aplikace segmentačních metod)
- …

Příklad – umělecký efekt

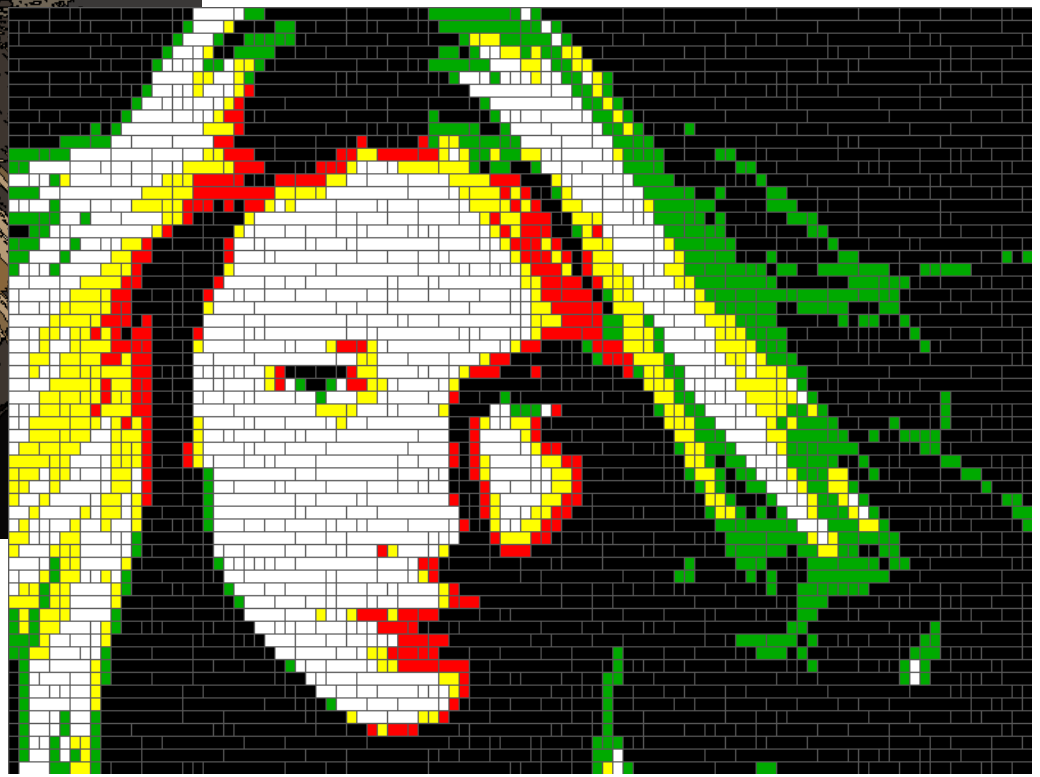
originál

kresba

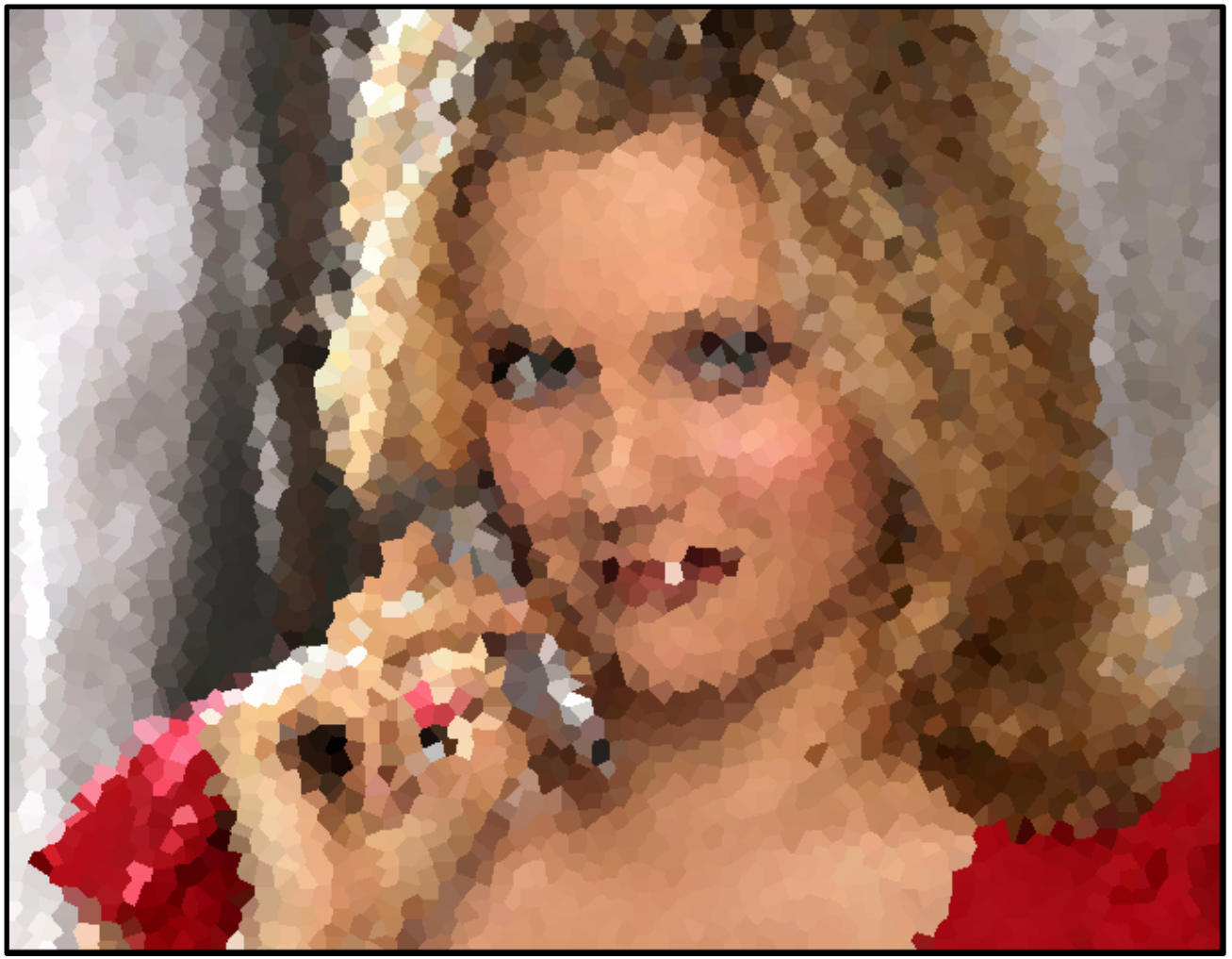
Příklad – NPR filtry



Příklad – NPR filtry



Příklad – mozaika



Literatura

Pratt W. K.: ***Digital Image Processing: PIKS Inside***, 3rd Edition, Wiley-Interscience, 2001

Gonzales R. C, Woods R. E.: ***Digital Image Processing***, 3rd Edition, Prentice Hall, 2007

matfyz

Computer Graphics Charles University

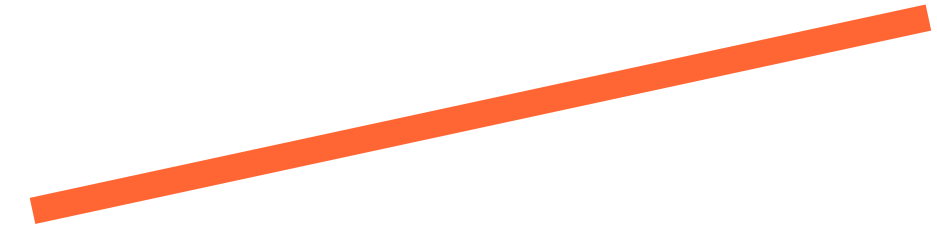
Kreslení čar

CGG MFF UK Praha

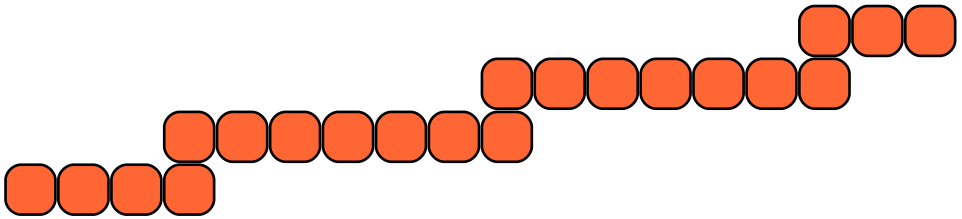
pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/

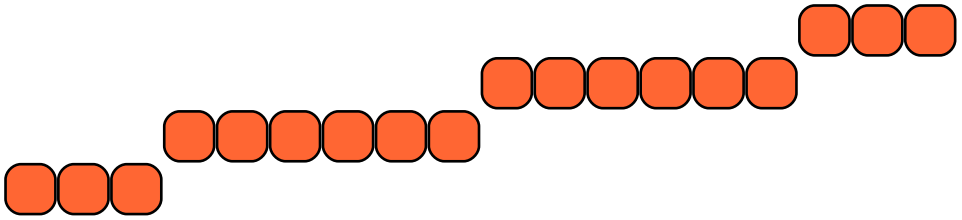
Kreslení úseček



vektorové zařízení

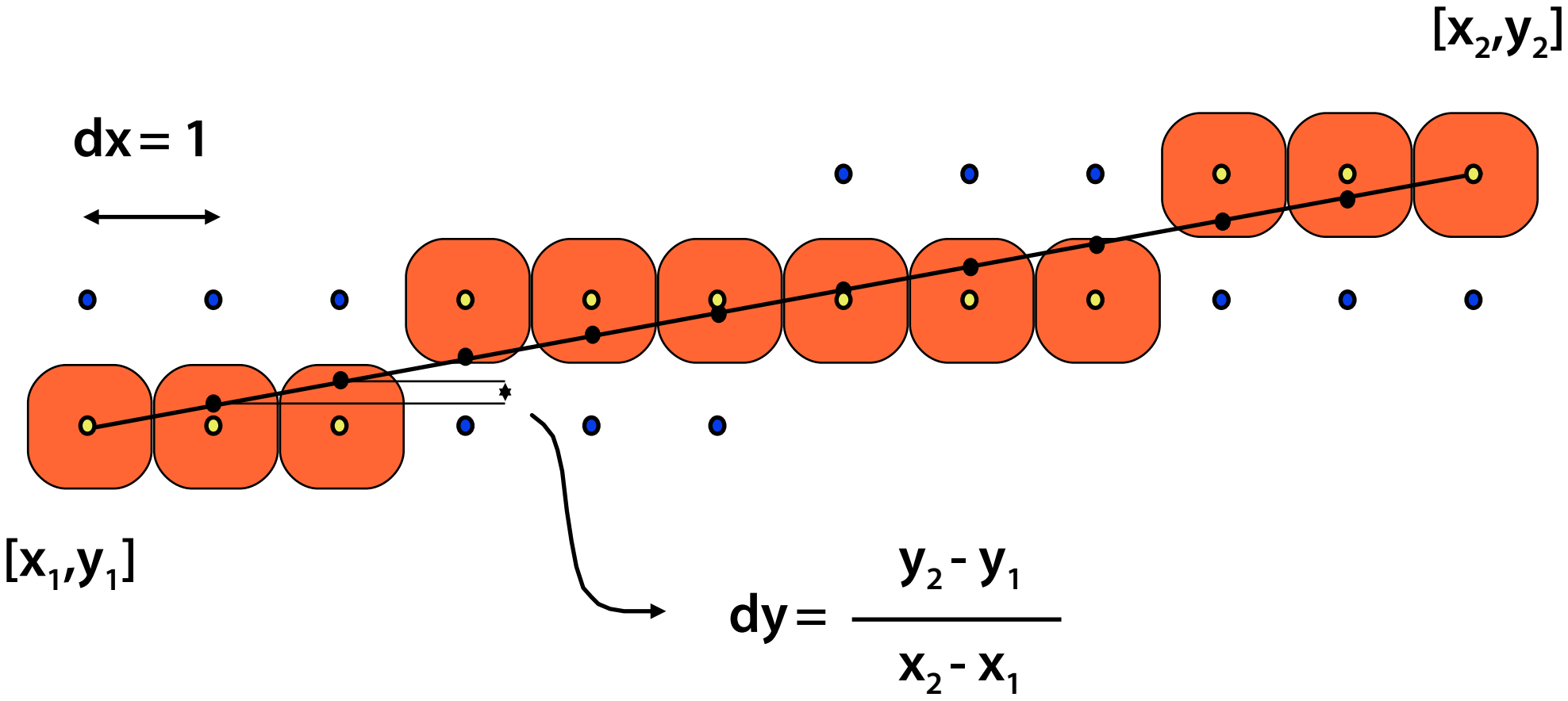


rastrové zařízení



OK

DDA algoritmus



DDA algoritmus

```
void LineDDA (int x1, int y1, int x2, int y2, Color color)
  // předpoklady: x1 < x2, |y2-y1| < |x2-x1|
{
  float y = y1;
  float dy = (y2 - y1) / (x2 - x1);
  PutPixel(x1, y1, color);
  while (x1 < x2)
  {
    x1++;
    y += dy;
    PutPixel(x1, round(y), color);
  }
}
```

DDA algoritmus

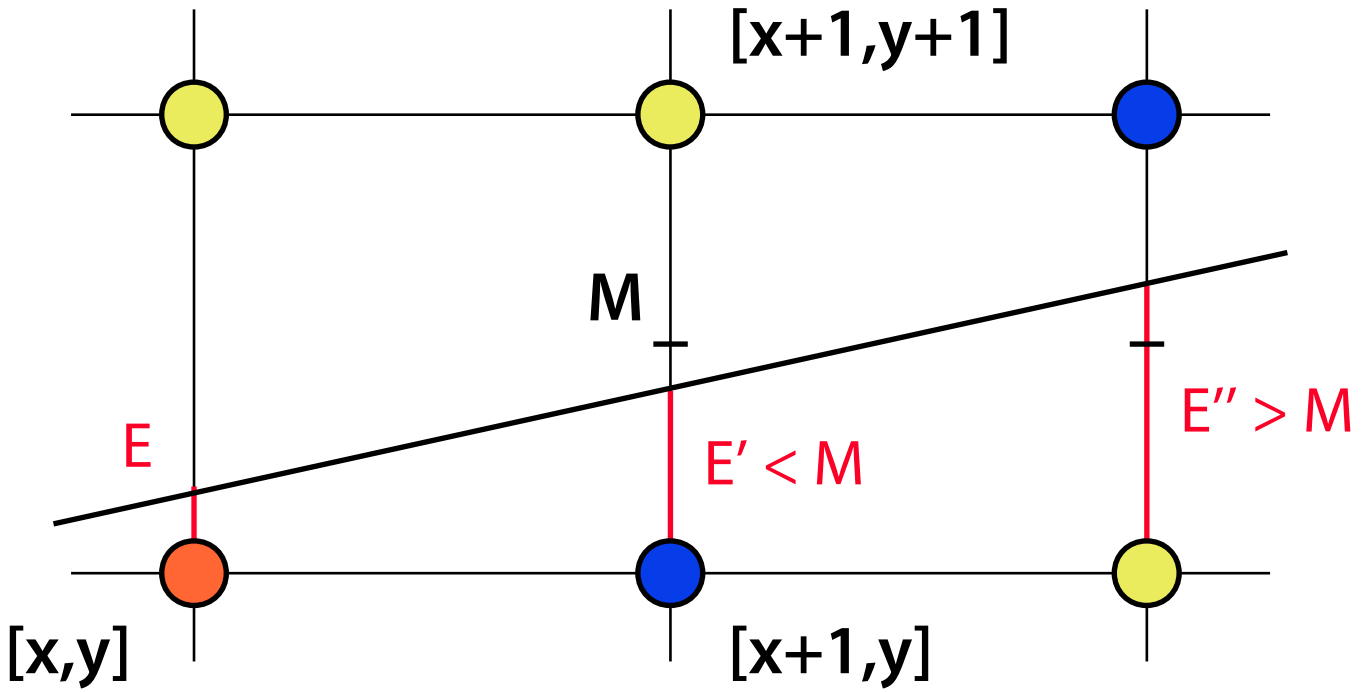
Výhody

- snadná implementace (HW)
- bez větvení kódu

Nevýhody

- nutno počítat s vyšší **přesností** (float, double, fixed point)
- jedno **dělení** a v cyklu **zaokrouhlování**

Bresenhamův algoritmus



$$dx = x_2 - x_1$$

$$dy = y_2 - y_1$$

$$E' = E + \frac{dy}{dx} \leq M = \frac{1}{2}$$

Celočíselné odvození

$$E' = E + \frac{dy}{dx} \leq \frac{1}{2} \quad / \cdot 2dx$$

$$2dx \cdot E' = 2dx \cdot E + 2dy \leq dx \quad / - dx$$

$$dx(2E' - 1) = dx(2E - 1) + 2dy \leq 0$$

$$D' = D + 2dy \leq 0$$

$$D_0 = 2dy - dx$$

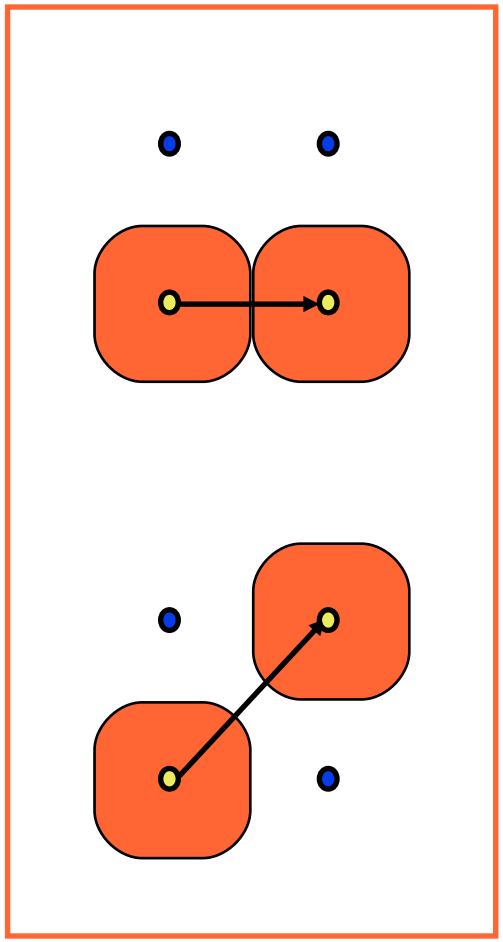
$$D \leq 0 \Rightarrow D' = D + 2dy, \qquad y' = y$$

$$D > 0 \Rightarrow D' = D + 2dy - 2dx, \qquad y' = y + 1$$

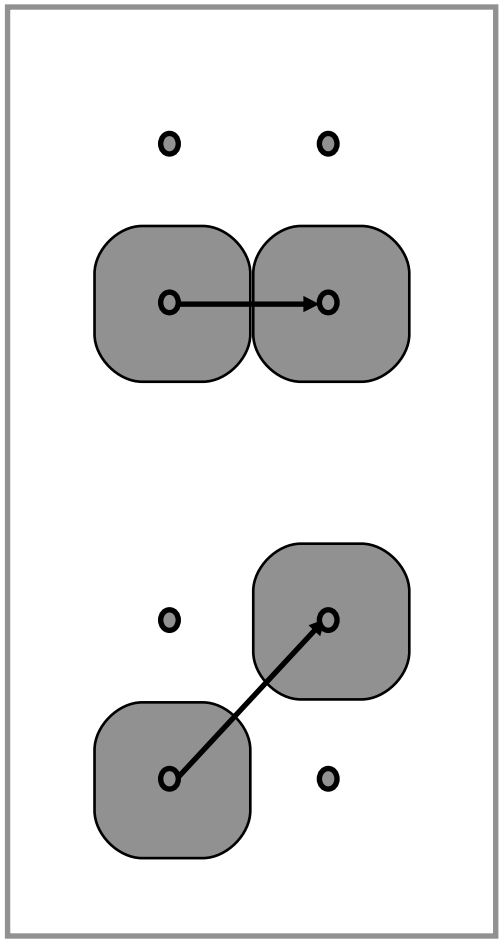
Bresenhamův algoritmus

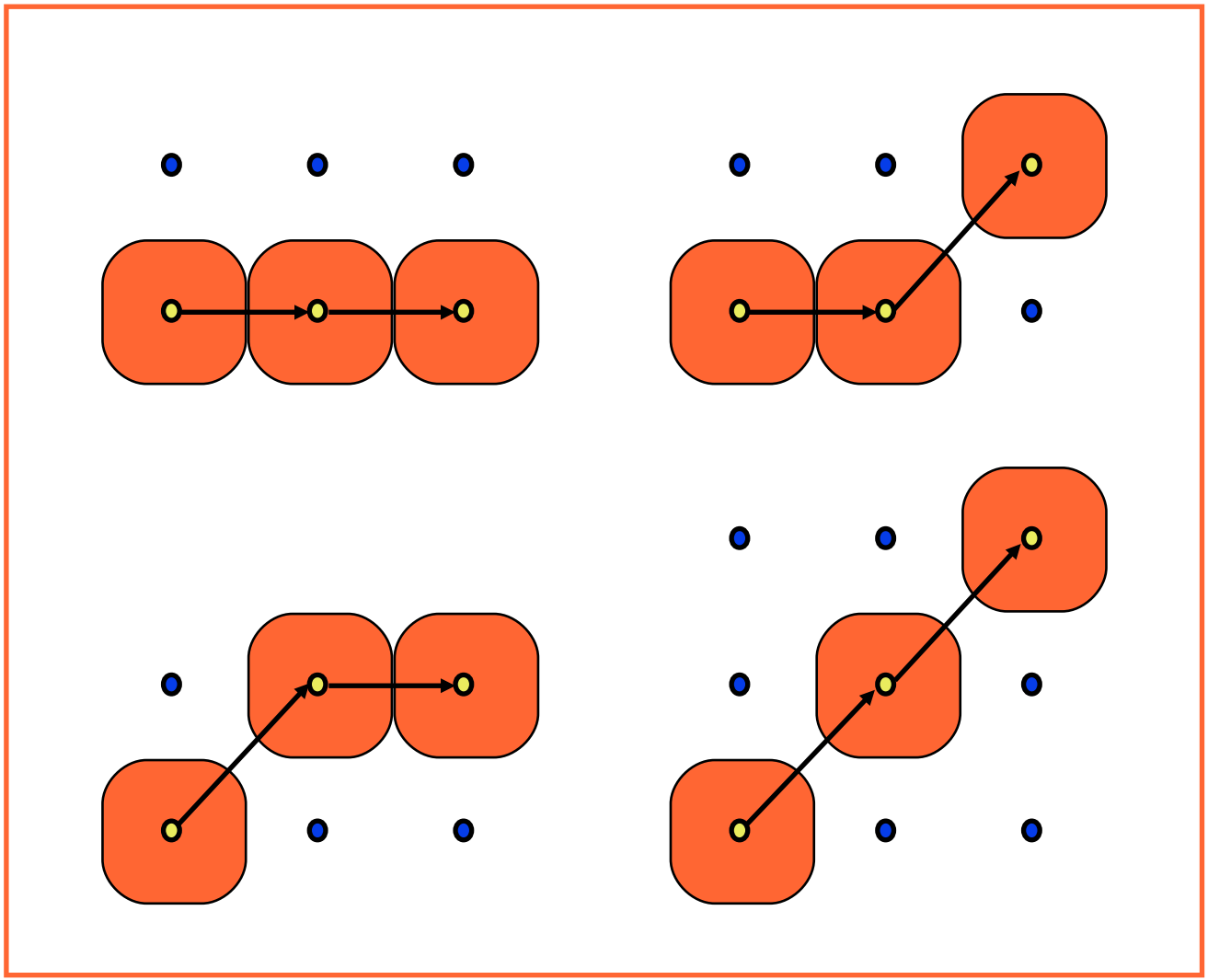
```
void LineBres (int x1, int y1, int x2, int y2, Color color)
   // předpoklady: x1 < x2, |y2-y1| < |x2-x1|
{
   int dx   = x2 - x1;
   int dy   = y2 - y1;
   int D    = 2 * dy - dx;
   int inc0 = 2 * dy;
   int inc1 = 2 * (dy - dx);
   PutPixel(x1, y1, color);
   while (x1 < x2)
   {
     if (D <= 0)
       D += inc0;
     else
     {
       D += inc1;
       y1++;
     }
     x1++;
     PutPixel(x1, y1, color);
   }
}
```

Jednokrokový algoritmus

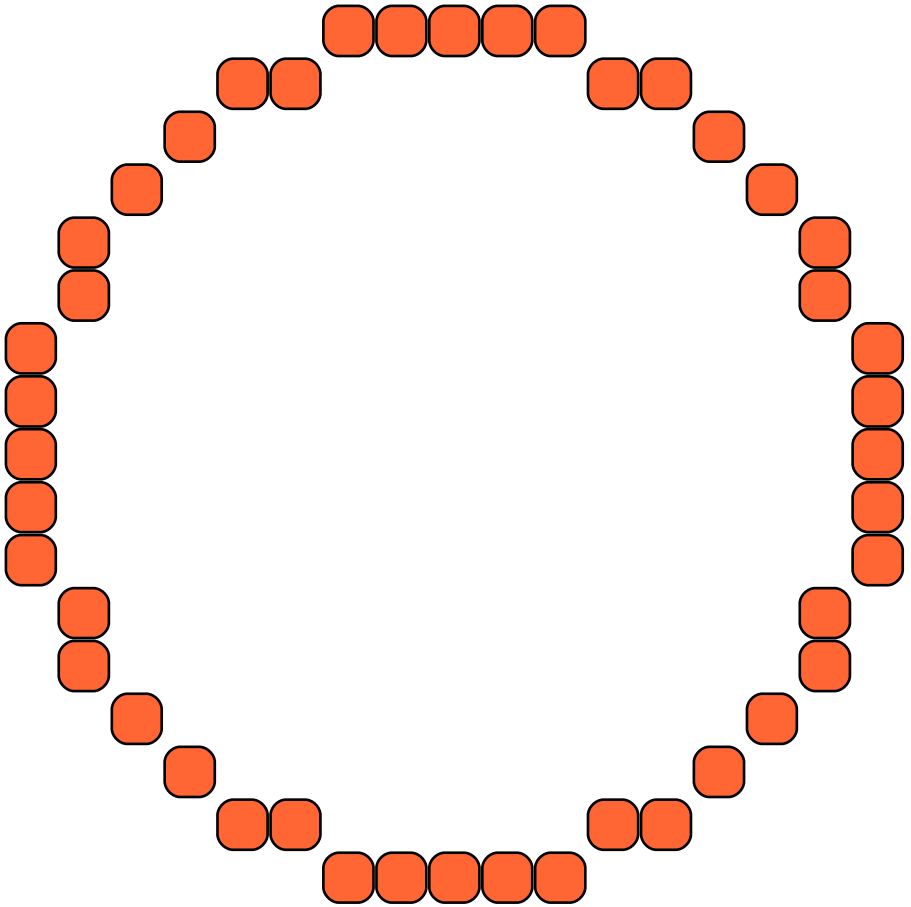


Vícekrokové algoritmy

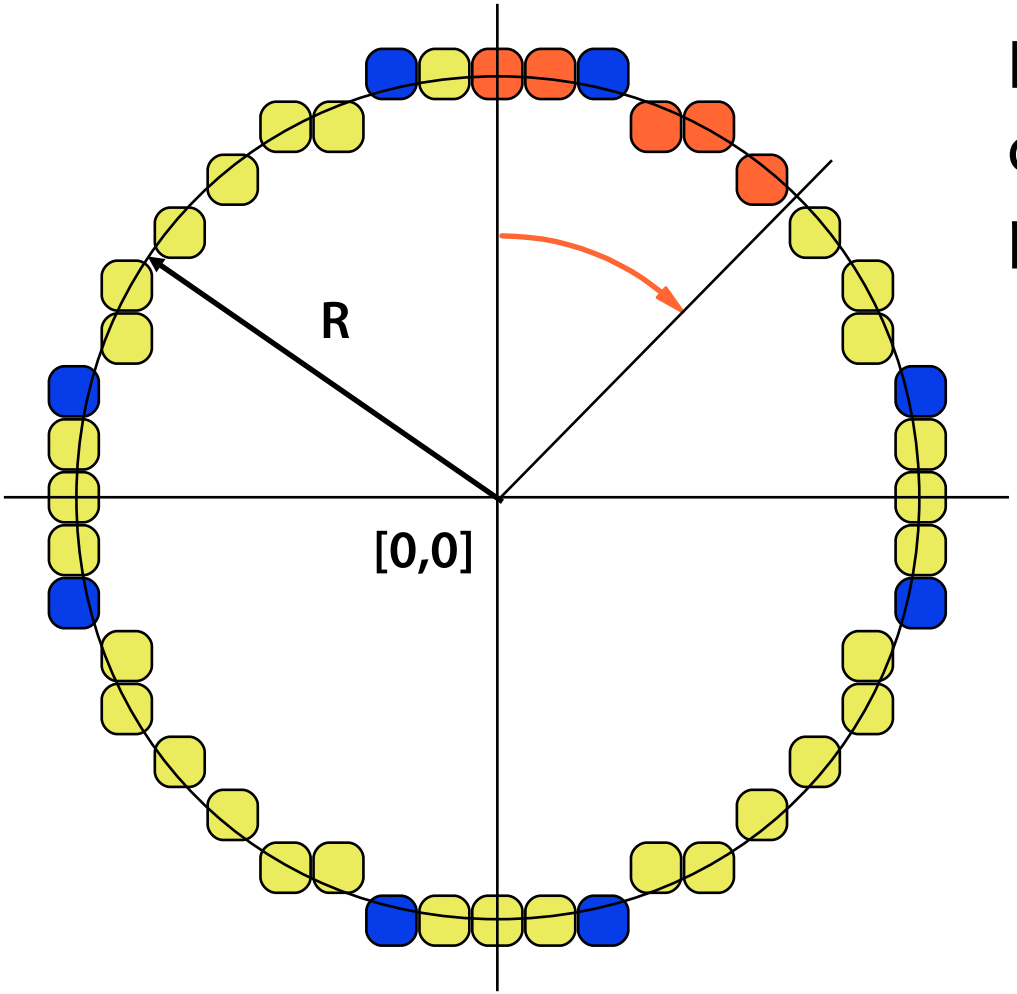




Kreslení kružnice

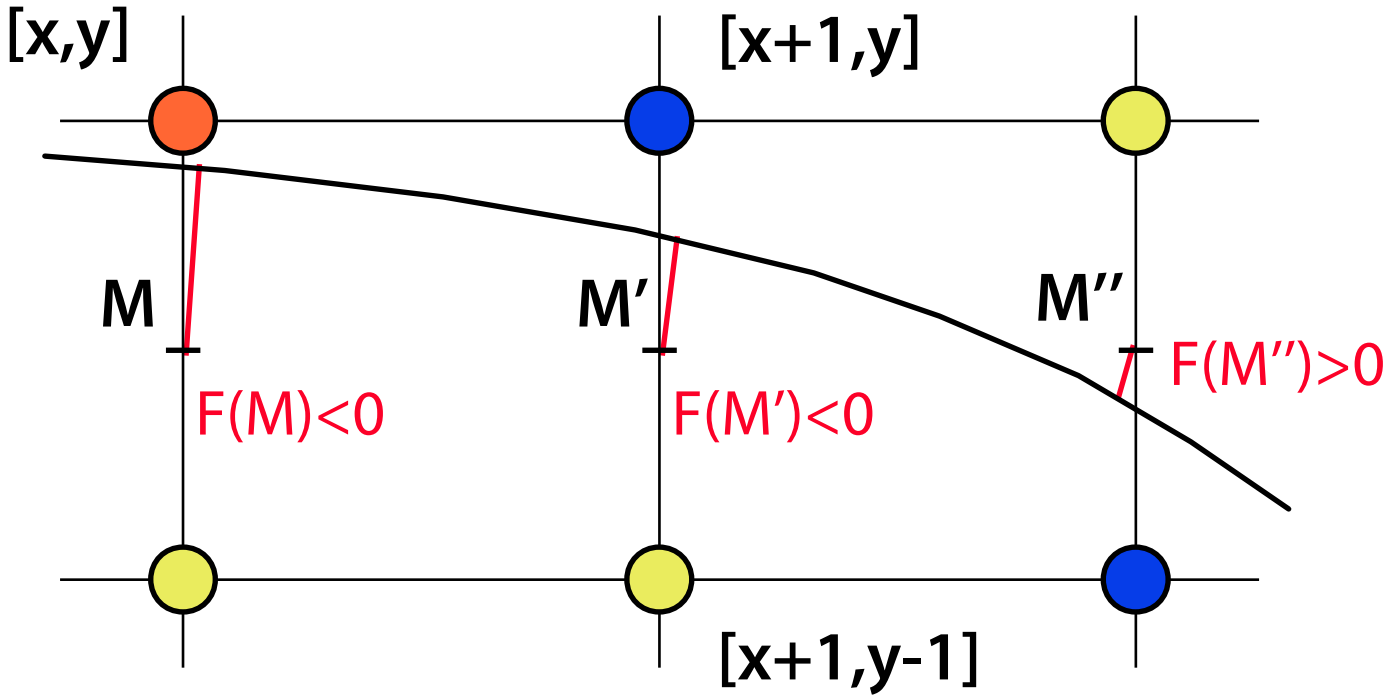


Kreslení kružnice



Kreslí se jen jedna **osmina** oblouku – zbytek se přenese pomocí symetrií

Bresenhamův algoritmus



$$F(M) = M_x^2 + M_y^2 - R^2$$

Inkrementální odvození

1) $F(M') = (x+1)^2 + (y-\frac{1}{2})^2 - R^2 \quad < \quad 0$

$F(M'') = (x+2)^2 + (y-\frac{1}{2})^2 - R^2 = F(M') + 2x + 3$

2) $F(M') \geq 0$

$F(M'') = (x+2)^2 + (y-\frac{3}{2})^2 - R^2 = F(M') + 2x - 2y + 5$

$D_0 = 1.25 - R \quad \{1 - R\}$

$D < 0 \Rightarrow D' = D + 2x + 3, \qquad y' = y$

$D \geq 0 \Rightarrow D' = D + 2x - 2y + 5, \qquad y' = y - 1$

Kreslení kružnice – pomocná funkce

```
void CirclePoints (int x0, int y0, int x, int y, Color color)
{
  PutPixel(x0 + x, y0 + y, color);
  PutPixel(x0 + y, y0 + x, color);
  PutPixel(x0 + x, y0 - y, color);
  PutPixel(x0 + y, y0 - x, color);
  PutPixel(x0 - x, y0 + y, color);
  PutPixel(x0 - y, y0 + x, color);
  PutPixel(x0 - x, y0 - y, color);
  PutPixel(x0 - y, y0 - x, color);
}
```

Kreslení kružnice

```
void CircleBres (int x0, int y0, int R, Color color)
{
  int x = 0;
  int y = R;
  int D = 1 - R;
   CirclePoints(x0, y0, 0, R, color);
   while (y > x)
   {
     if (D < 0)
        D += 2 * x + 3;
     else
     {
        D += 2 * (x - y) + 5;
        y--;
     }
     x++;
     CirclePoints(x0, y0, x, y, color);
   }
}
```

Literatura

J. Foley, A. van Dam, S. Feiner, J. Hughes: ***Computer Graphics, Principles and Practice***, 72-87

Jiří Žára a kol.: ***Počítačová grafika, principy a algoritmy***, 91-100, 106-112

matfyz

Computer Graphics Charles University

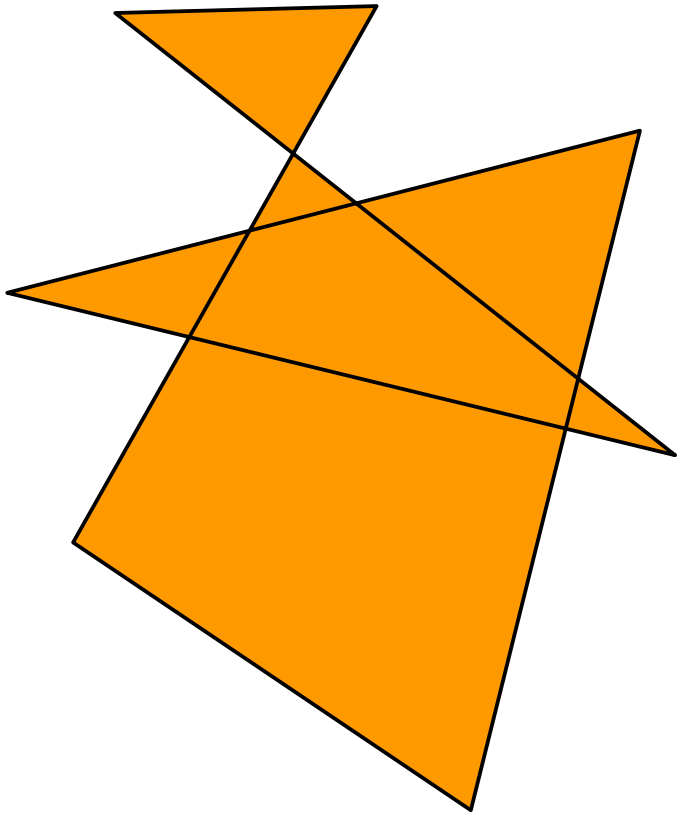
Vyplňování n-úhelníka

CGG MFF UK Praha

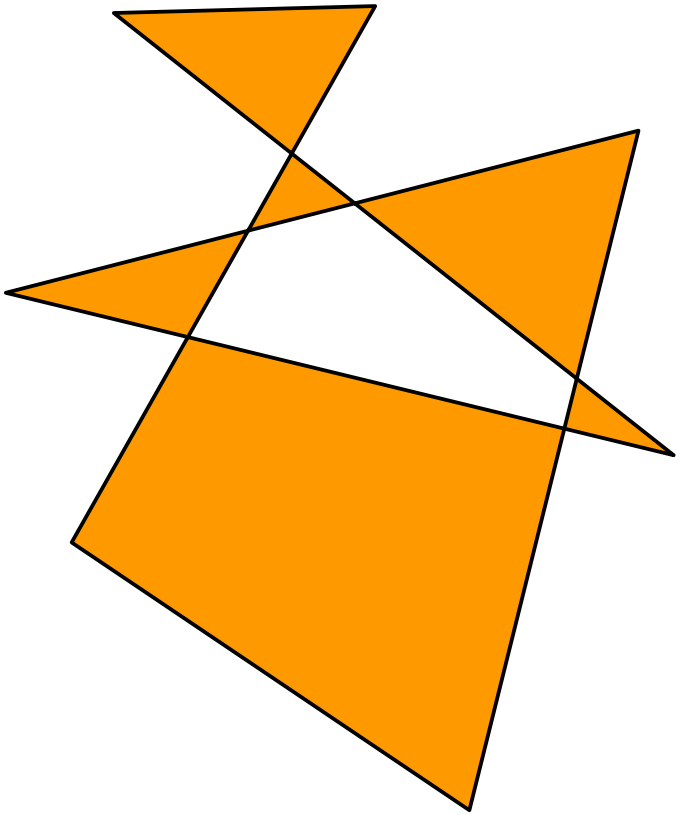
pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/

Pravidla vyplňování

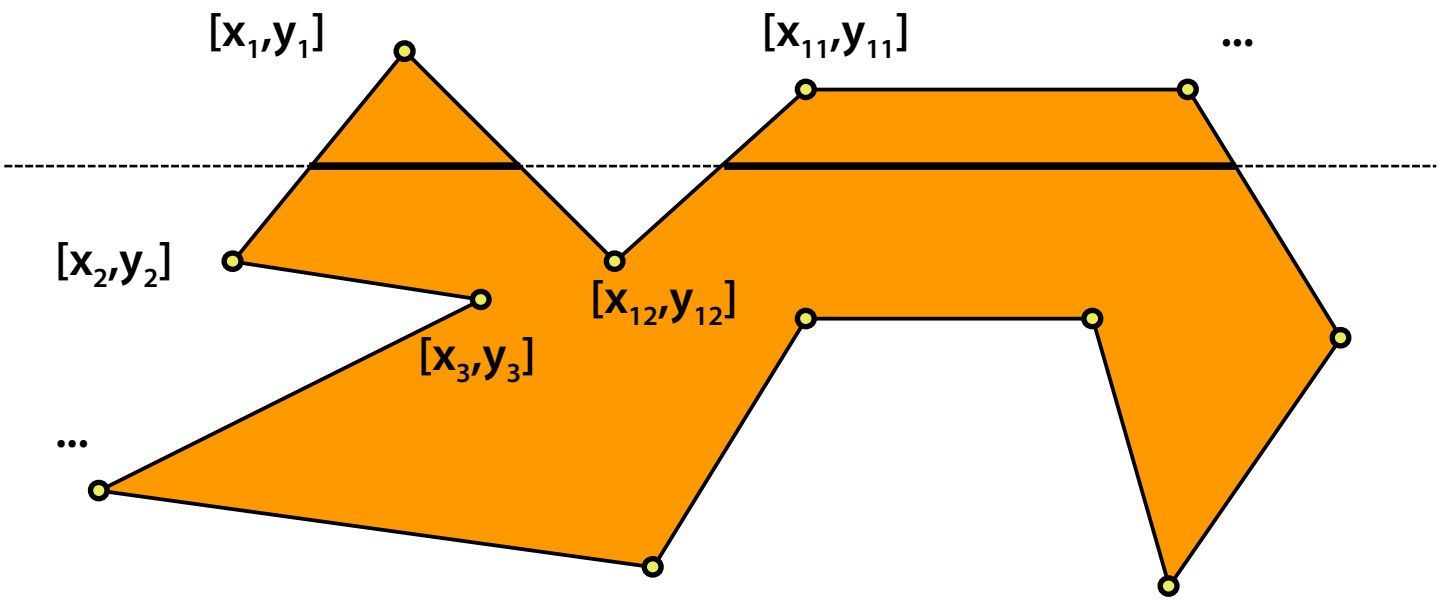


všechny vnitřní body



jen liché body
(„odd-even" rule)

Řádkový algoritmus



N-úhelník je zadán posloupností svých vrcholů

Může být **nekonvexní**

Možné zjednodušení – vyplňují se jen **liché body**

1. předzpracování

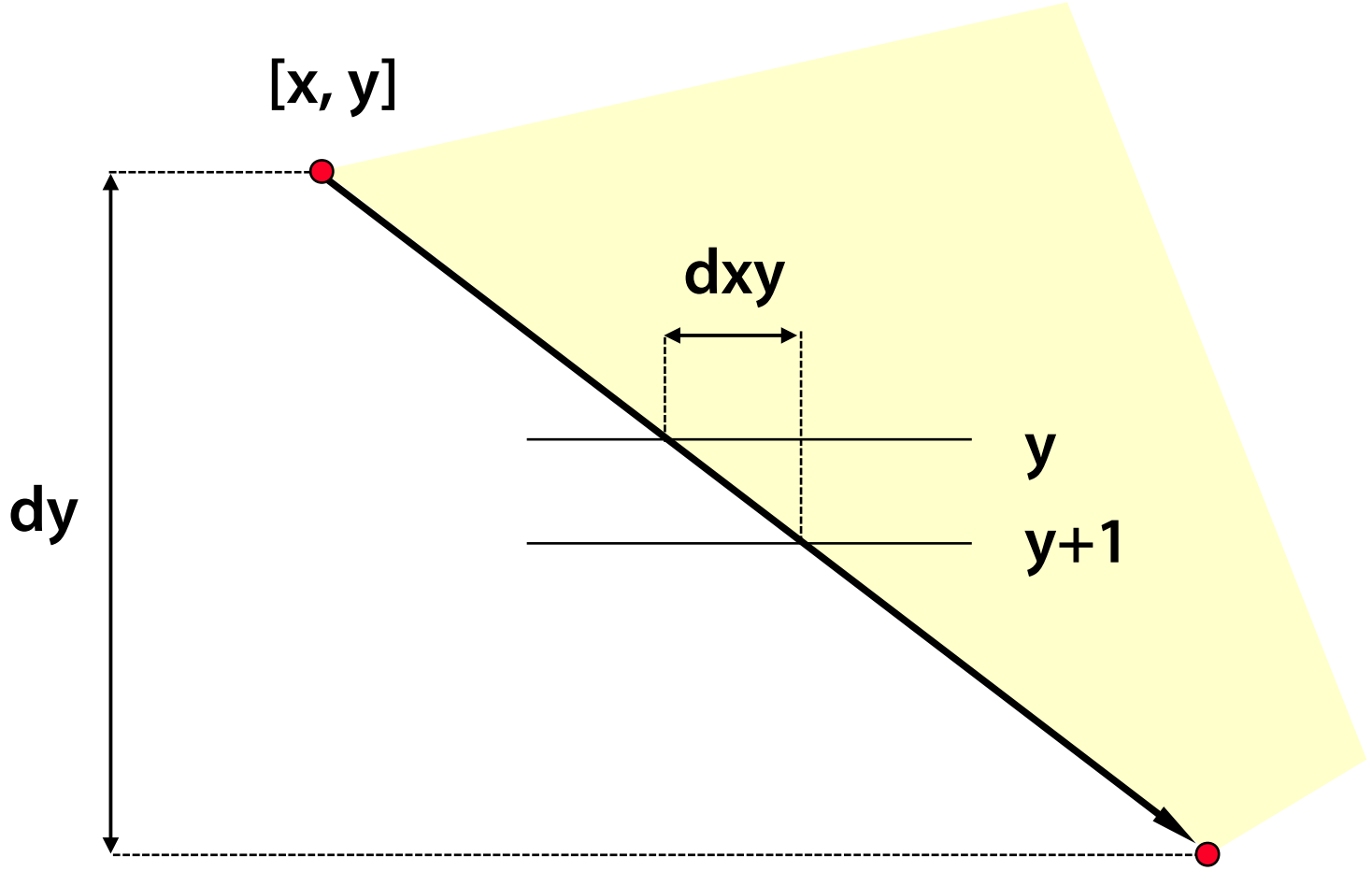
N-úhelník rozložíme na **jednotlivé hrany**

Vodorovné hrany odstraníme

Pro ostatní hrany vytvoříme **pracovní záznamy**

- hrany orientujeme směrem shora dolů

Pracovní záznam pro hranu



Pracovní záznam pro hranu

```
double x;
```
// souřadnice ***x*** horního koncového bodu,
// později souřadnice průsečíku s aktuální řádkou

```
int y;
```
// souřadnice ***y*** horního koncového bodu

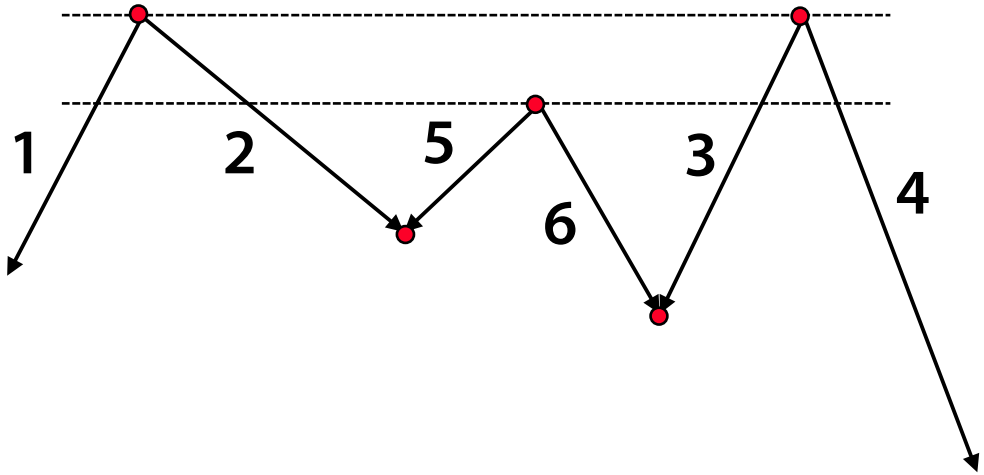
```
int dy;
```
// výška hrany v pixelech: $|y_2 - y|$

```
double dxy;
```
// změna ***x*** při posunutí na následující řádku (směrnice): $(x_2 - x) / dy$

2. inicializace seznamu S

Všechny předzpracované hrany setřídíme do **vstupního seznamu *S*** podle kritérií:

1. vzestupně podle ***y***
2. vzestupně podle ***x***
3. vzestupně podle ***dxy***

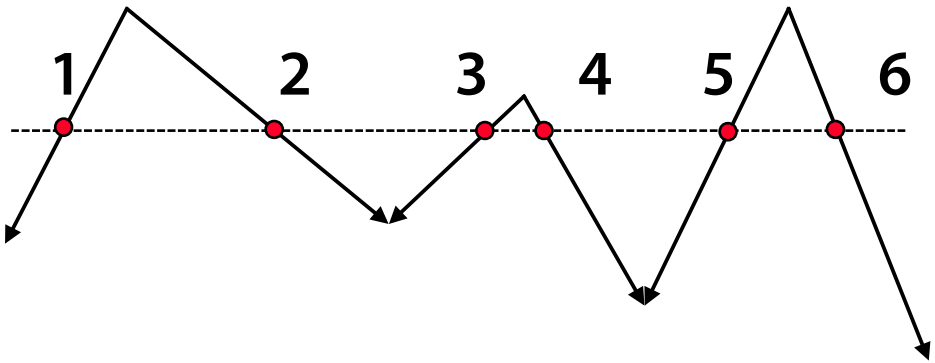


3. aktuální seznam A

Aktuální seznam *A* bude obsahovat všechny hrany, které protínají aktuální řádku. Seznam budeme udržovat setříděný:

❷ vzestupně podle ***x***

❸ vzestupně podle ***dxy***

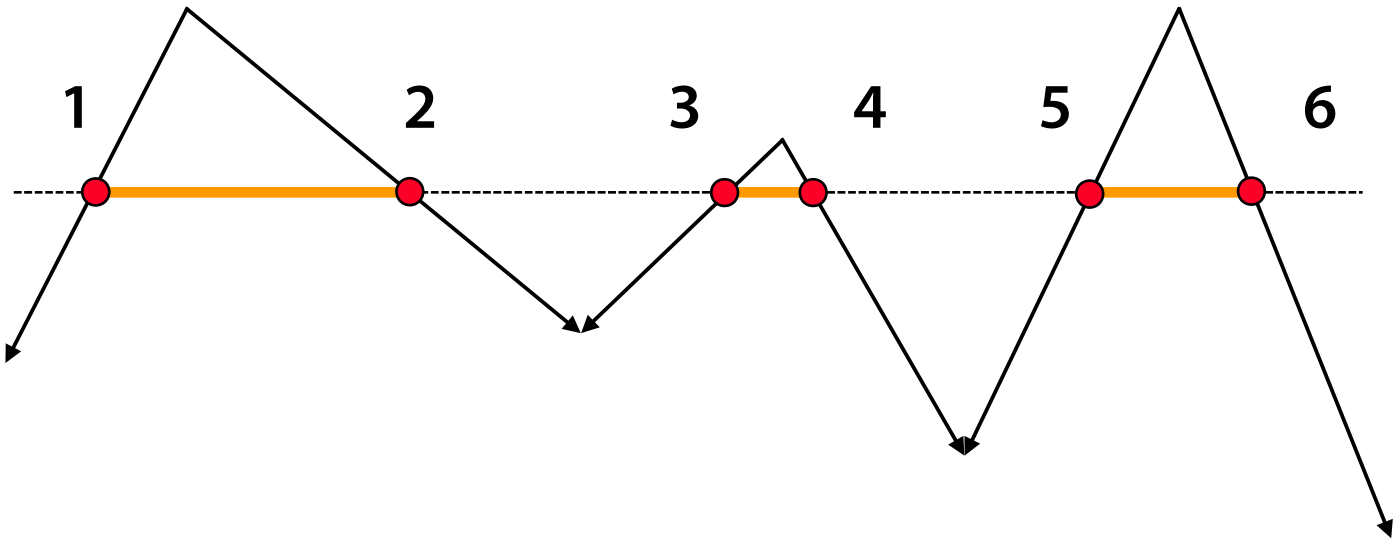


Při inicializaci zařadíme do ***A* počáteční úsek seznamu *S*** – hrany se shodným (tj. minimálním) ***y***

4. vykreslení aktuální řádky

Je třeba projít **aktuální seznam *A*** a vykreslit úseky odpovídající vnitřku n-úhelníka

- kreslím každý úsek mezi lichým a sudým záznamem
- při jiném pravidle vyplňování by byly podmínky složitější…



5. přechod na další řádku

Aktualizace seznamu *A*

```
dy--;
if (dy == 0)
{  /* odstraň hranu ze seznamu A ... */
}
else
  x += dxy;
```

Kontrola **setřídění *A***

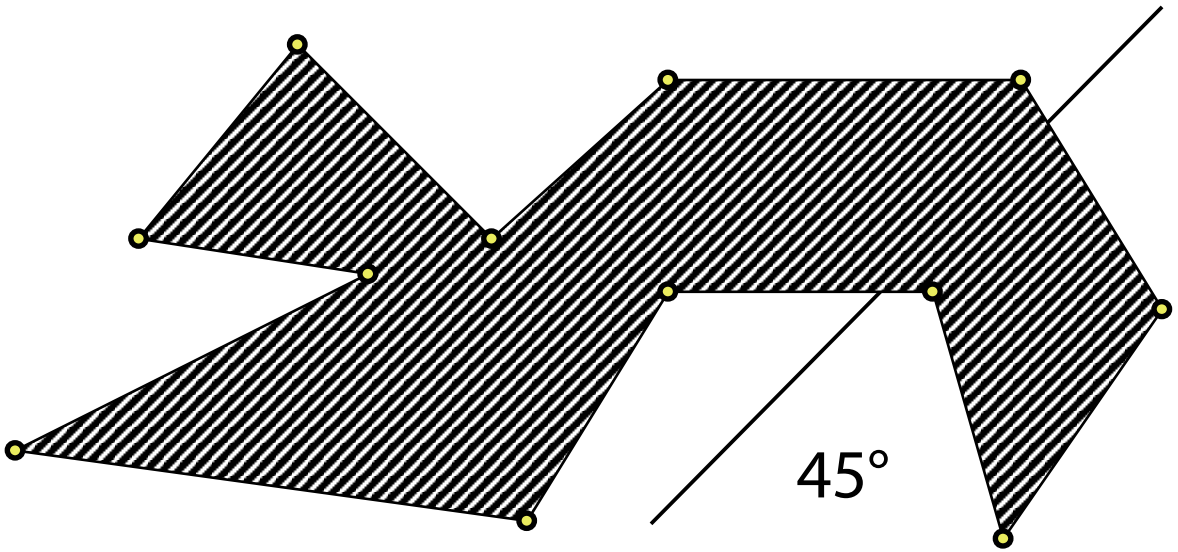
Zatřídění **nových hran** ze seznamu ***S*** do seznamu ***A*** (počáteční úsek ***S***)

6. podmínka ukončení cyklu

Jestliže je **seznam *A* neprázdný**, výpočet pokračuje krokem **4**

Jinak algoritmus **končí**

Šrafování

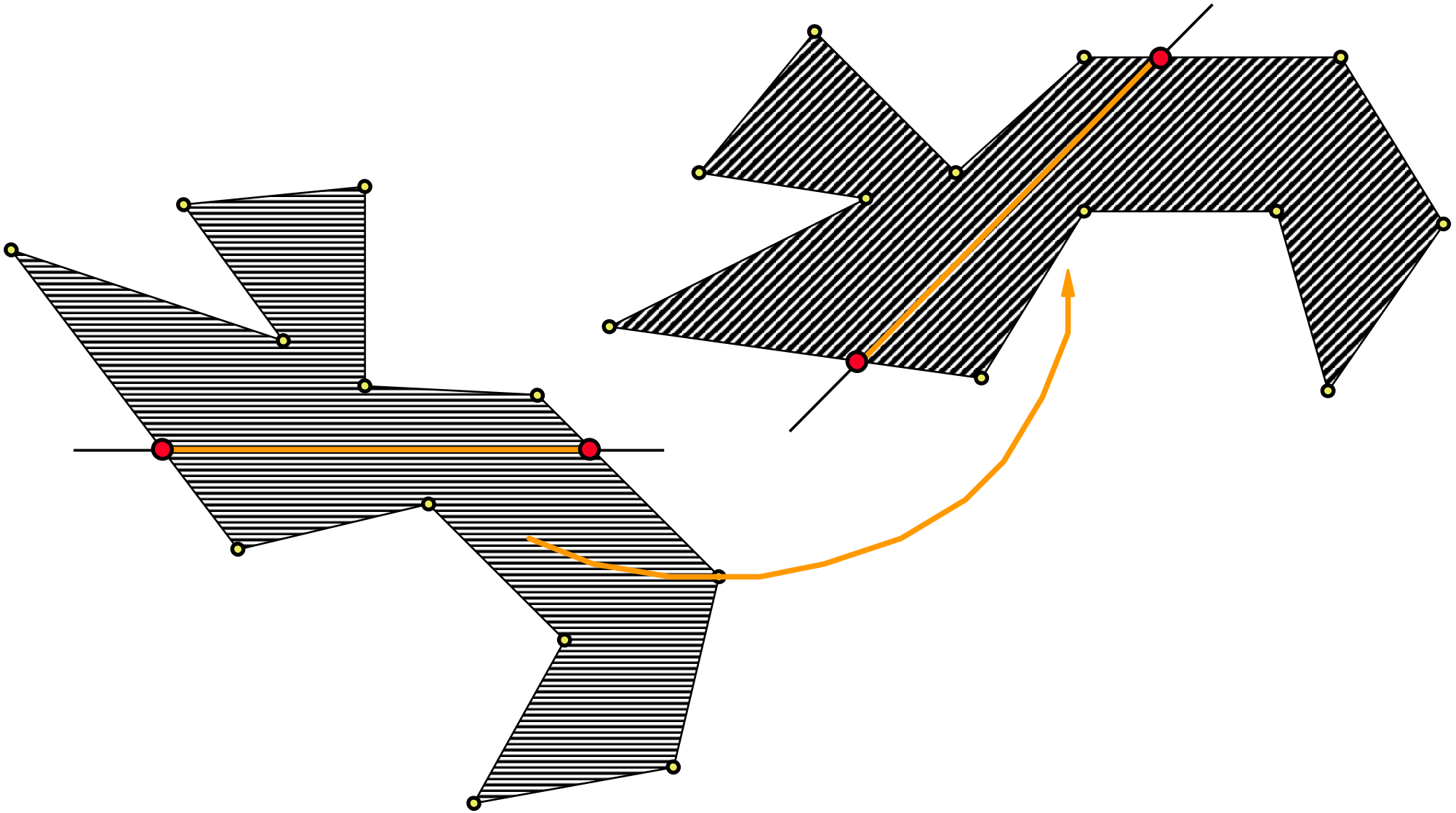


Otočím vrcholy n-úhelníka o opačný úhel

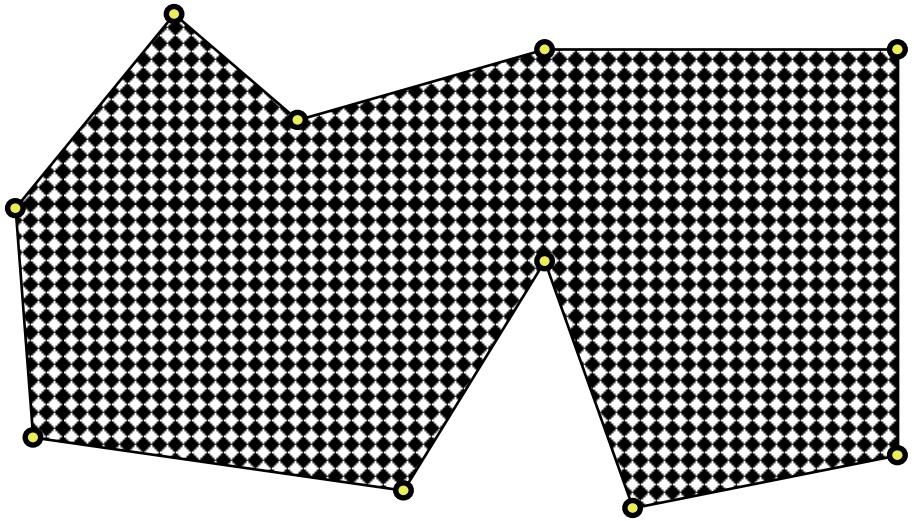
Kreslím každý **k-tý řádek**

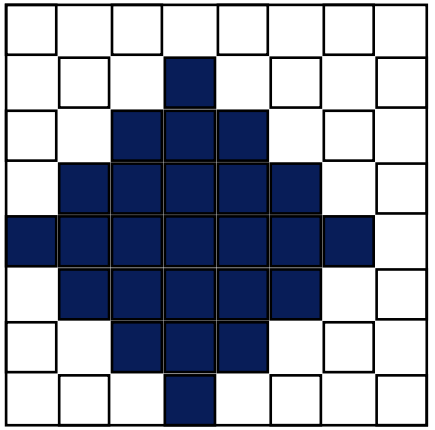
Před kreslením každou úsečku **otočím zpět**

Šrafování



Vyplňování vzorkem





matice M[8,8]

Vzorek je zadán **maticí pixelů** (např. 8×8)

Každý pixel se kreslí předpisem

– PutPixel(**x**, **y**, **M[y % 8, x % 8]**);

Literatura

J. Foley, A. van Dam, S. Feiner, J. Hughes: ***Computer Graphics, Principles and Practice***, 92-99

Jiří Žára a kol.: ***Počítačová grafika, principy a algoritmy***, 129-138

matfyz

Computer Graphics Charles University

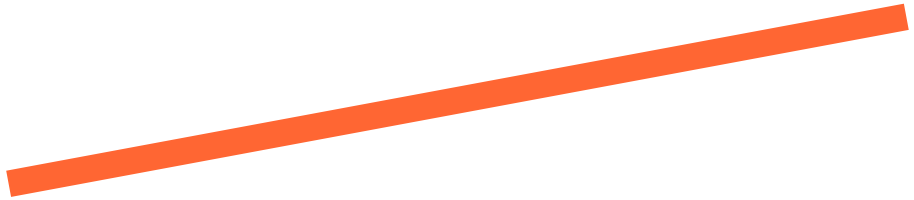
Vyhlazování – „anti-aliasing“

CGG MFF UK Praha

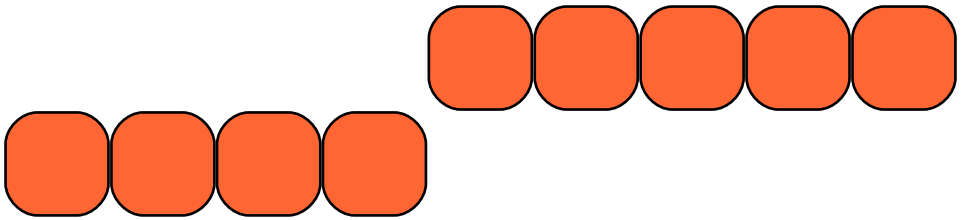
pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/

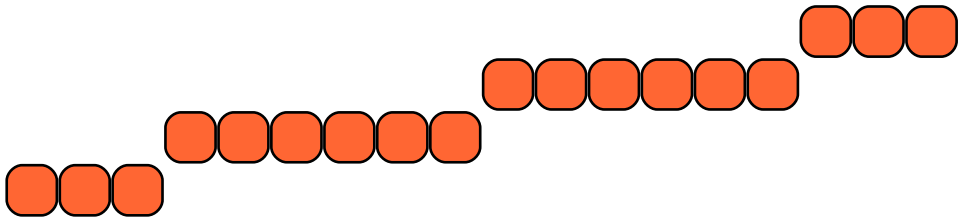
Úsečky na rastrovém zařízení



ideální úsečka

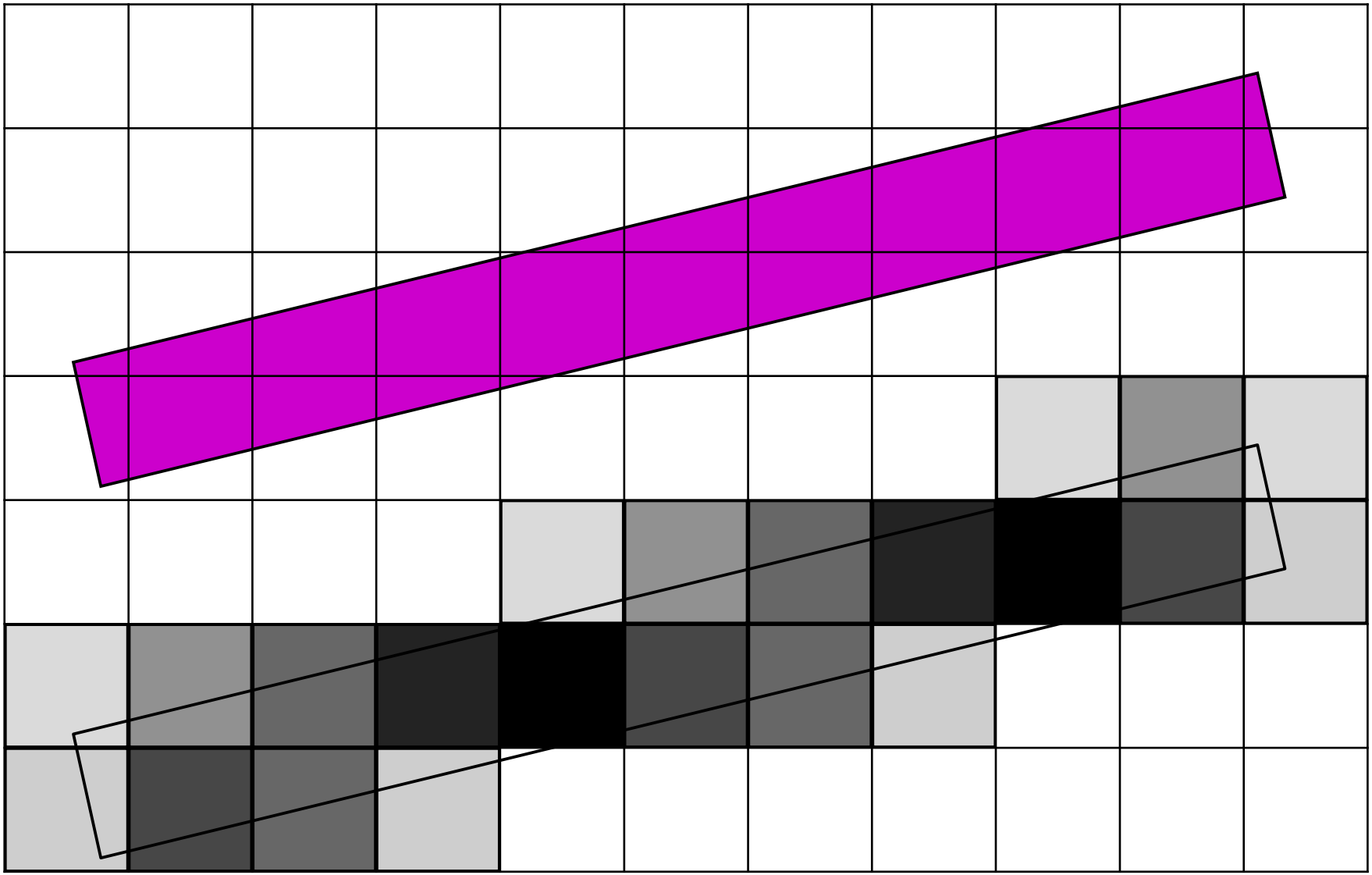


rastrová kresba

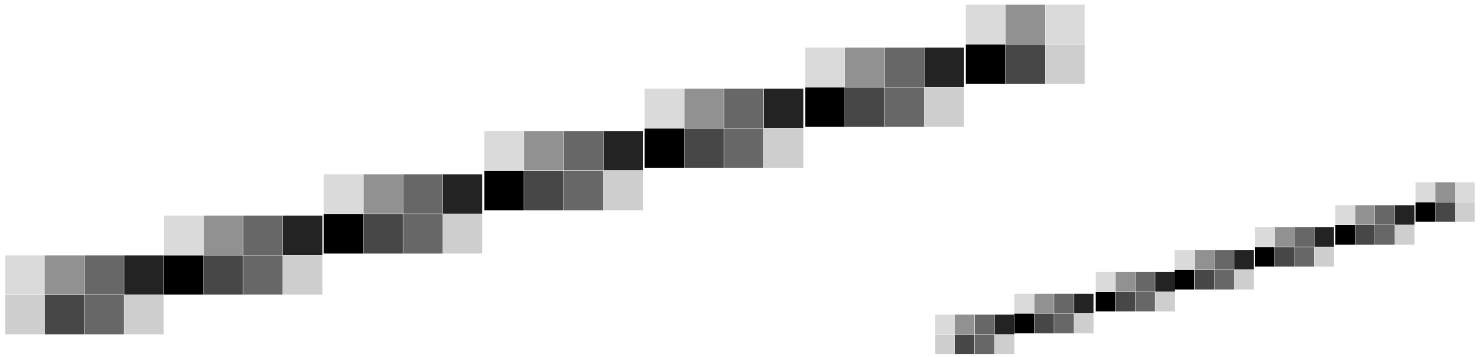


dvakrát větší rozlišení

Pokrytí plochy pixelu



Pokrytí plochy pixelu



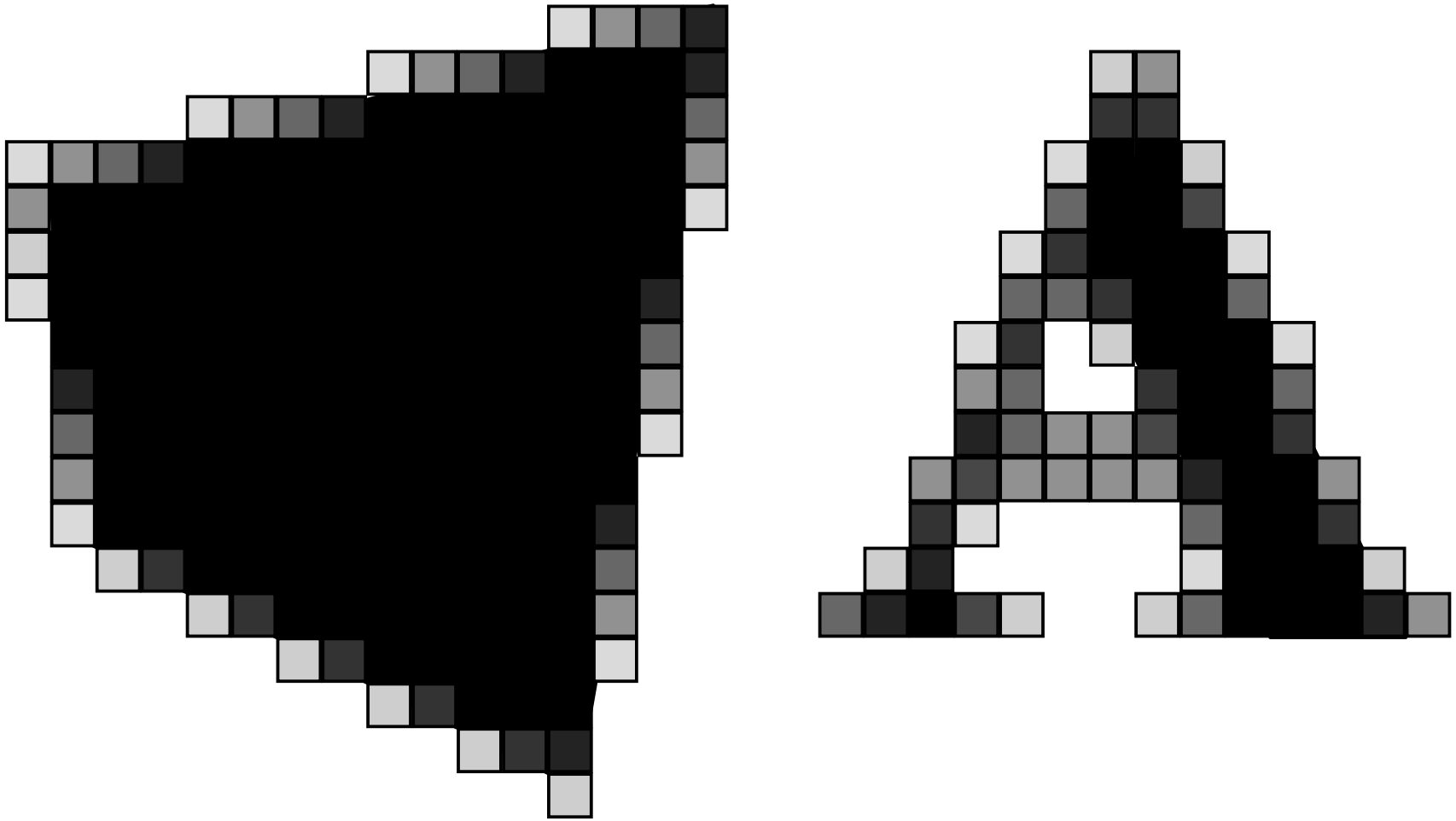
Ke kreslení se použije **více odstínů** dané barvy

- zvětšení prostorového rozlišení na úkor barevného

Pixely i kreslené objekty jsou **plošné útvary**

Každý pixel se rozsvítí intenzitou úměrnou **ploše jeho pokryté části**

N-úhelníky a text



Kreslení s vyhlazováním

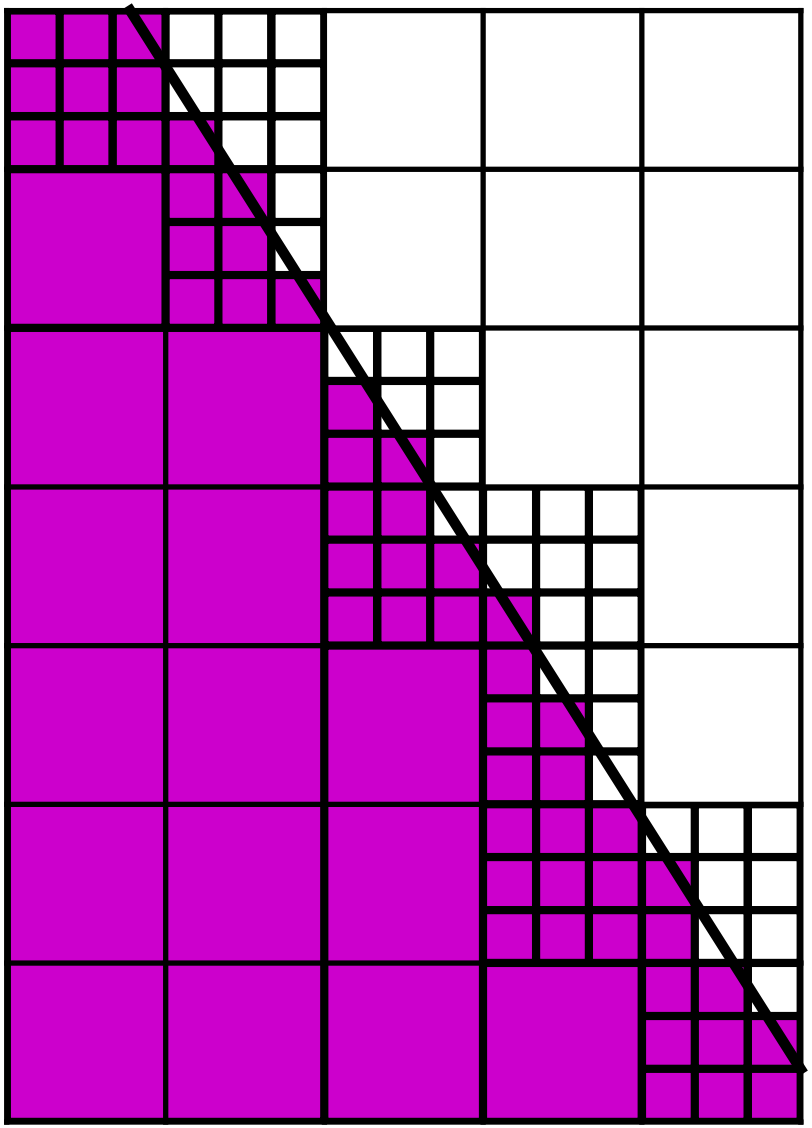
Úsečka: kreslí se vždy oba pixely, mezi kterými úsečka prochází

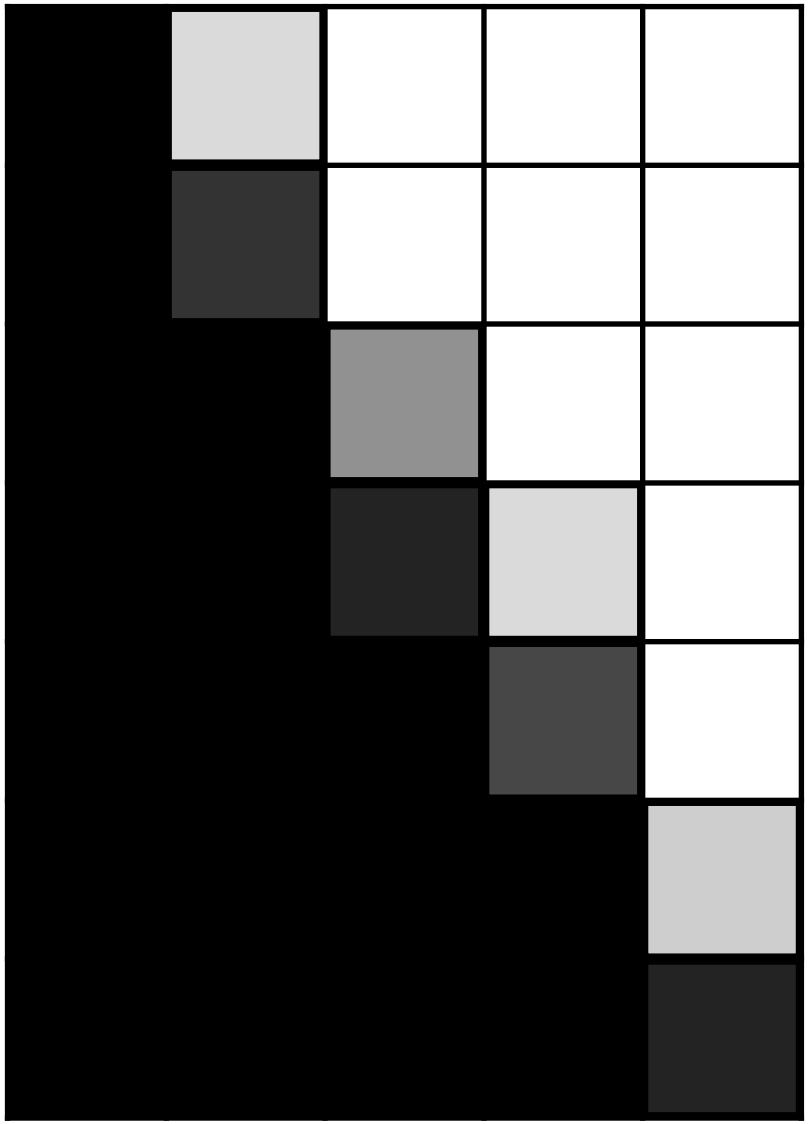
- intenzita se určí podle vzdálenosti středu pixelu od úsečky (desetinná část **y** v DDA, člen **D** v Bresenhamově algoritmu)

N-úhelník: kreslí se všechny pixely, do jejichž plochy n-úhelník zasahuje

- intenzita okrajových pixelů se spočítá opět podle vzdálenosti (desetinná část **y**, **D**)

Převzorkování – „super-sampling”





Vícenásobné vzorkování, super-sampling

Objekt se nakreslí do bufferu **ve větším rozlišení** (při zvětšení např. 2× až 4×)

- každý pixel se rozloží na „subpixely”

Barevný odstín skutečně kresleného pixelu se určí jako aritmetický průměr odstínů jeho subpixelů

- někdy se používá **vážený průměr** (subpixely ležící ve středu pixelu mají větší váhu)

Literatura

J. Foley, A. van Dam, S. Feiner, J. Hughes: ***Computer Graphics, Principles and Practice***, 132-140

Jiří Žára a kol.: ***Počítačová grafika, principy a algoritmy***, 100-101, 147-151

matfyz

Computer Graphics Charles University

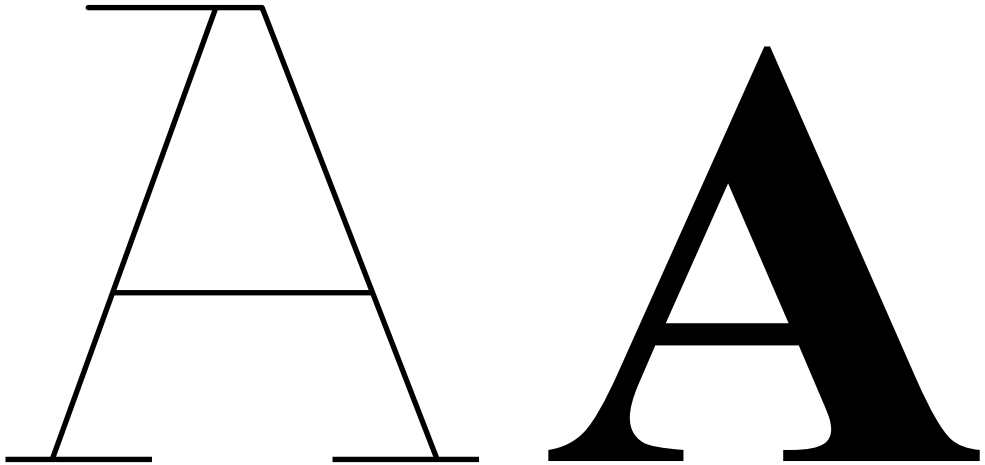
Kreslení písma

CGG MFF UK Praha

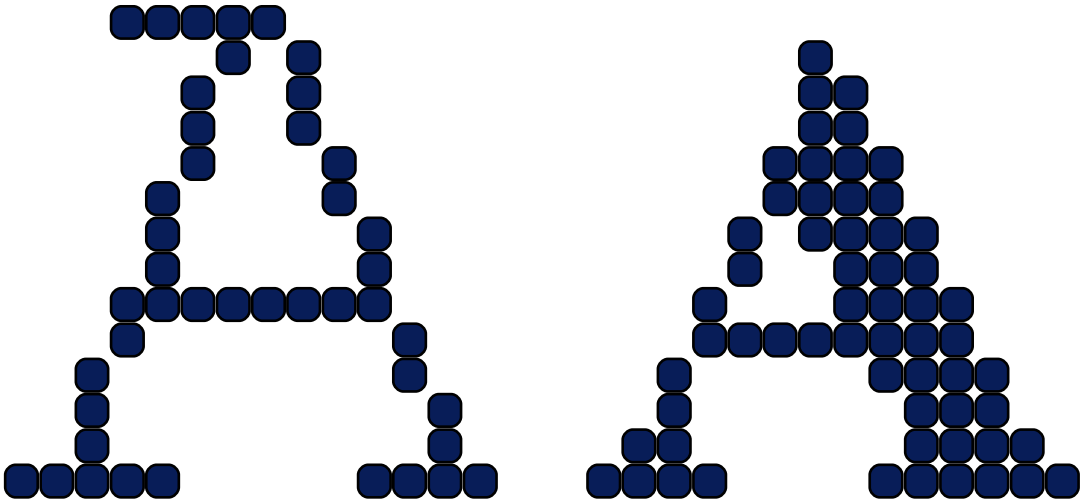
pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/

Definice písma



vektorové písmo
(čárové a vyplněné)



rastrové písmo

Definice písma

Vektorové písmo

- obrysy písmen jsou zadány pomocí úseček, oblouků kružnic a elips nebo spline křivek
- před kreselním se musí převést do **rastrové podoby**
- lze jej snadno **škálovat** (neprofesionálně) a **otáčet**

Rastrové písmo

- písmena jsou zadána **bitovou maticí** („bitmapou”) pro každou velikost písma
- snadno se kreslí (HW „BitBlt” operace)

Font cache

Převod **vektorového písma** do rastrové podoby je časově náročný

- jednotlivá písmena se v textu mnohokrát opakují
- rastrová podoba písmen se ukládá do „font cache”

Prvek „font cache”

- druh písma (font), velikost (v pt), orientace, kód písmene, velikost rastrového obrazu (v pixelech)
- **„bitmapa“** nebo odkaz do společného bitového pole

Použití „font cache”

Je-li potřeba nakreslit konkrétní písmeno **X**, podívám se nejprve do cache

- pro rychlé vyhledávání mohu použít **hašování**
- jestliže jsem písmeno našel, nakreslím ho pomocí „BitBlt”

Neúspěch při hledání ve „font cache”

- písmeno musím převést do rastrové podoby, přidám ho do „cache”
- z „cache” odstraňuji **nejdéle nepoužívané položky**

„Distance field“ reprezentace (C. Green)

Tvar písmene je předpočítán v podobě „**distance field**“

- vzdálenost od okraje objektu je zakódována odstínem v textuře

Fonty se dají dobře **škálovat** a kvalitně vykreslovat

- akcelerováno na GPU
- zvětšování (bilineární interpolace)
- v shaderu – porovnání s prahovou hodnotou
- možnost přidat obrysové efekty…

Literatura

J. Foley, A. van Dam, S. Feiner, J. Hughes: Computer Graphics, Principles and Practice, 127-131, 976-979

Jiří Žára a kol.: Počítačová grafika, principy a algoritmy, 119-126

Chris Green: *Improved alpha-tested magnification for vector textures and special effects*, SIGGRAPH 2007

matfyz

Computer Graphics Charles University

Kódování rastrových obrázků

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/

Použití

Úsporné uložení rastrového obrázku

- proti běžným textovým algoritmům lze využít dvojrozměrné povahy dat

Efektivnější operace s jednoduchými obrázky a **bitovými maskami**

- množinové operace s bitovými maskami
- superpozice obrázků

RLE kódování („Run-Length Encoding”)

Využívá se **koherence** ve vodorovném směru

- sousední pixely mají často stejnu hodnotu
- nejvýhodnější u málo barevných obrázků

Speciální příznak pro uložení **„běhu”**

`ESC {počet} {pixel}` (PCX)

Dva typy paketů – **„kopírovací”** a **„opakovací”**

`COPY {počet} {data ...}` (Targa, BMP…)

`FILL {počet} {pixel}`

Kvadrantový strom („quadtree”)

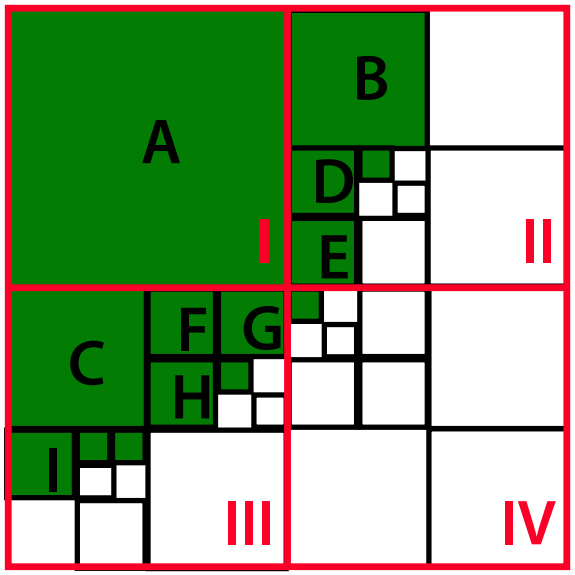
Využívá se **koherence** ve vodorovném i svislém směru

- úsporně se kódují větší souvislé plochy jedné barvy
- **adaptivní princip**
  - » postupné dělení „zajímavých” (=členitých) oblastí

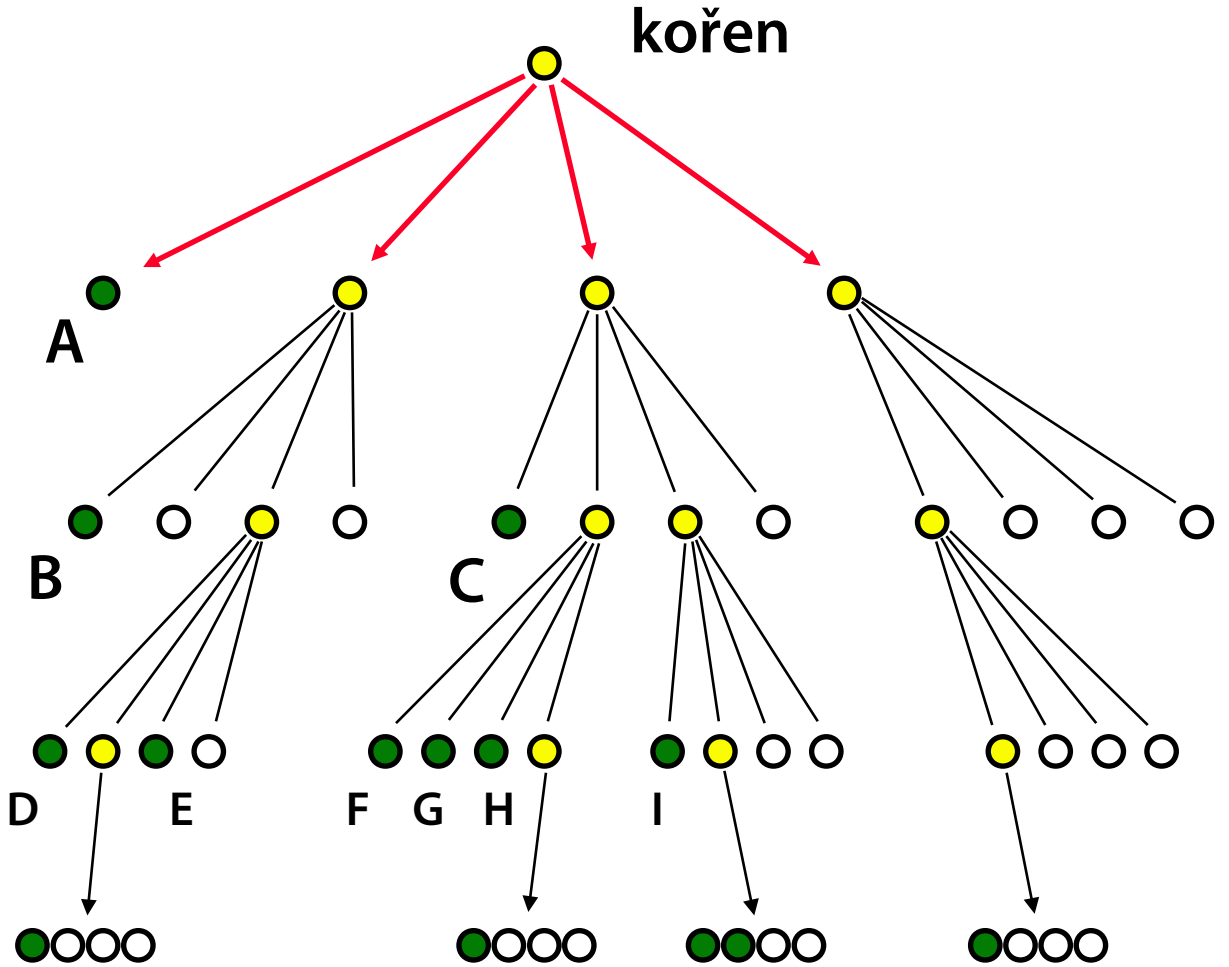
Aplikace kvadrantového stromu

- kódování obrazu
- úsporné uložení **bitové masky** (množinové operace)
- pomocná datová struktura pro **rychlé vyhledávání**

Kvadrantový strom („quadtree”)



16 × 16
(256 bytů)



12 záznamů (96 bytů)

Kódování kvadrantového stromu

Podle definice (metoda „shora-dolů”)

- daný čtverec se zkontroluje ⇒ je-li vícebarevný, rozdělí se na čtyři části, atd. (rekurze, „pre-order”)
- hodnoty některých pixelů se čtou **několikanásobně**

Metoda **„zdola-nahoru”**

- začíná se od **čtverečků 2×2**, jednobarevné oblasti se spojují do větších uzlů grafu, atd. … a nakonec se vytvoří **kořen stromu** (rekurze, „post-order”)
- každý pixel se čte pouze **jedenkrát**

Množinové operace

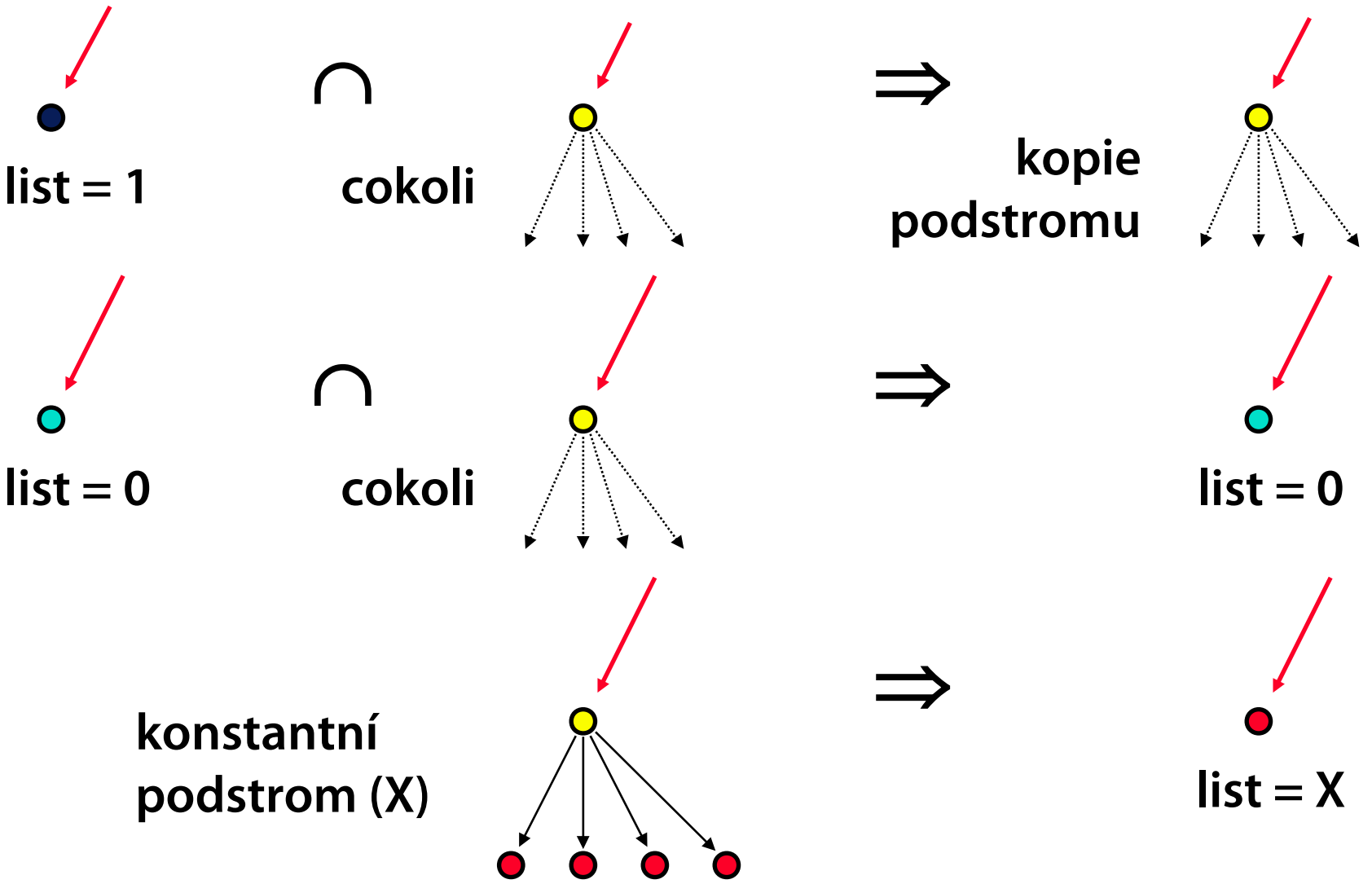
Kvadrantové stromy reprezentují **jednobitovou informaci** (množinu, masku)

- množinové operace (sjednocení, průnik, rozdíl, ..)
- předpokládá se shodný definiční obor operandů

Prochází se paralelně všechny **vstupní stromy** a současně se konstruuje **výsledný strom**

- všechny vstupní uzly jsou vnitřní $\Rightarrow$ „rozděl a panuj”
- jeden vstupní uzel je listem $\Rightarrow$ podle typu množinové operace se zpracují ostatní vstupní podstromy

Příklad – pravidla pro operaci „průnik”



Implementační poznámky

Kódování **obecné oblasti**

- zakóduje se nejmenší čtverec rozměru $\mathbf{2^n \times 2^n}$, který danou oblast obsahuje
- pixely ležící **mimo oblast** se zakódují speciální hodnotou („outside")
- jiná varianta: vnějším pixelům se přiřadí hodnota **okrajových pixelů** („don't care") – největší úspora

Úsporné **hybridní kódování** obrázku

- je-li podstrom větší než bitmapa, ukládám **bitmapu**

Implementační poznámky II

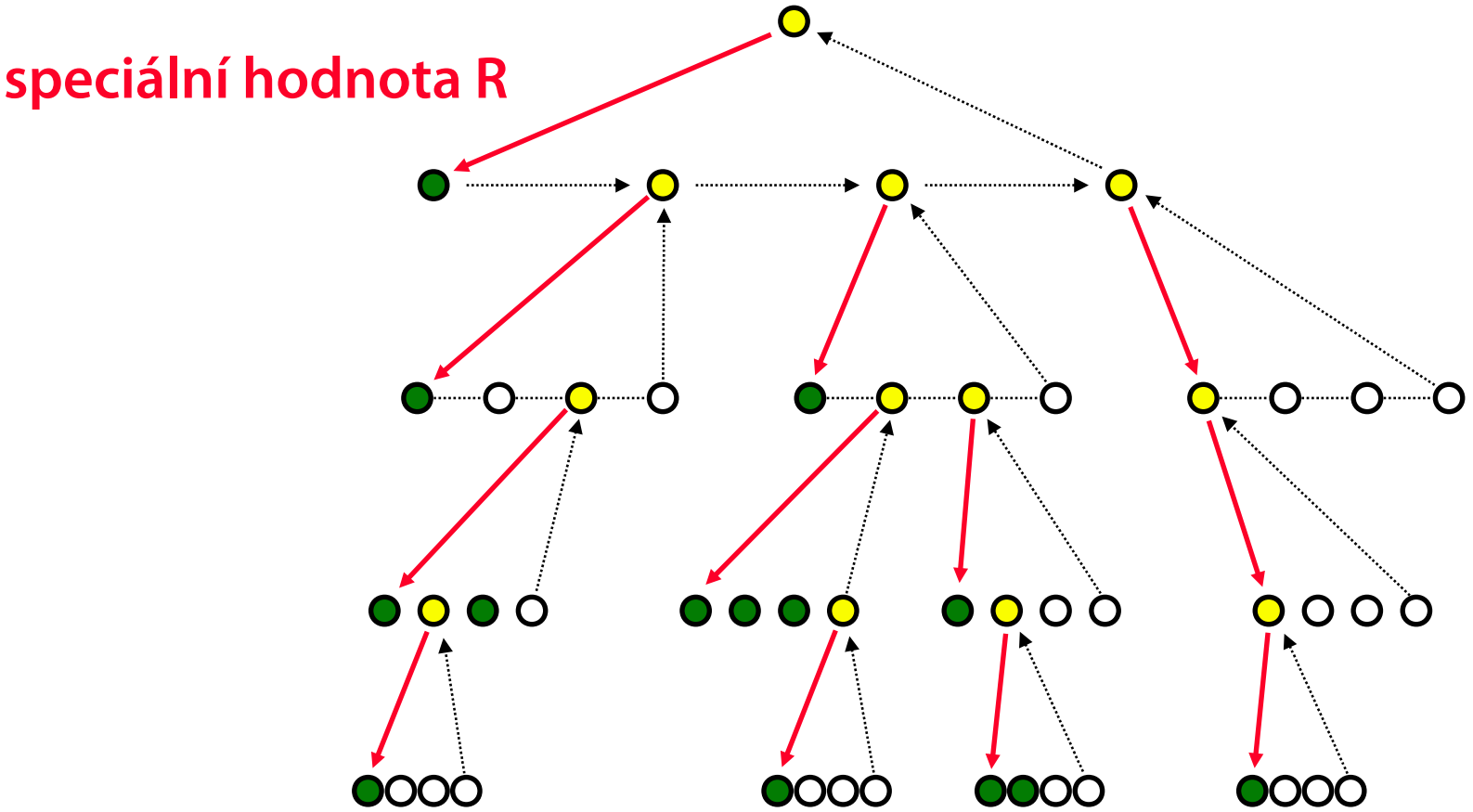
Společné větve kvadrantového stromu

- **opakuje-li** se ve stromu nějaká větev (podstrom) několikrát, uloží se pouze jednou a pak se na ně může odkazovat i odjinud
- ze stromu se stává hierarchický **graf** (ADG – acyklický orientovaný graf)
- společná větev může být použita v různých úrovních

Lineární uložení kvadrantového stromu (serializace)

- **průchod stromem** zleva-doprava („pre-order”)

Lineární uložení



R1R10R1R1000100R1R111R1000R1R1100000
RRR1000000000 … 49 položek

Řádkový seznam („X-transition list”)

Rastrová reprezentace **množiny** (jednobitové masky) **v rovině**

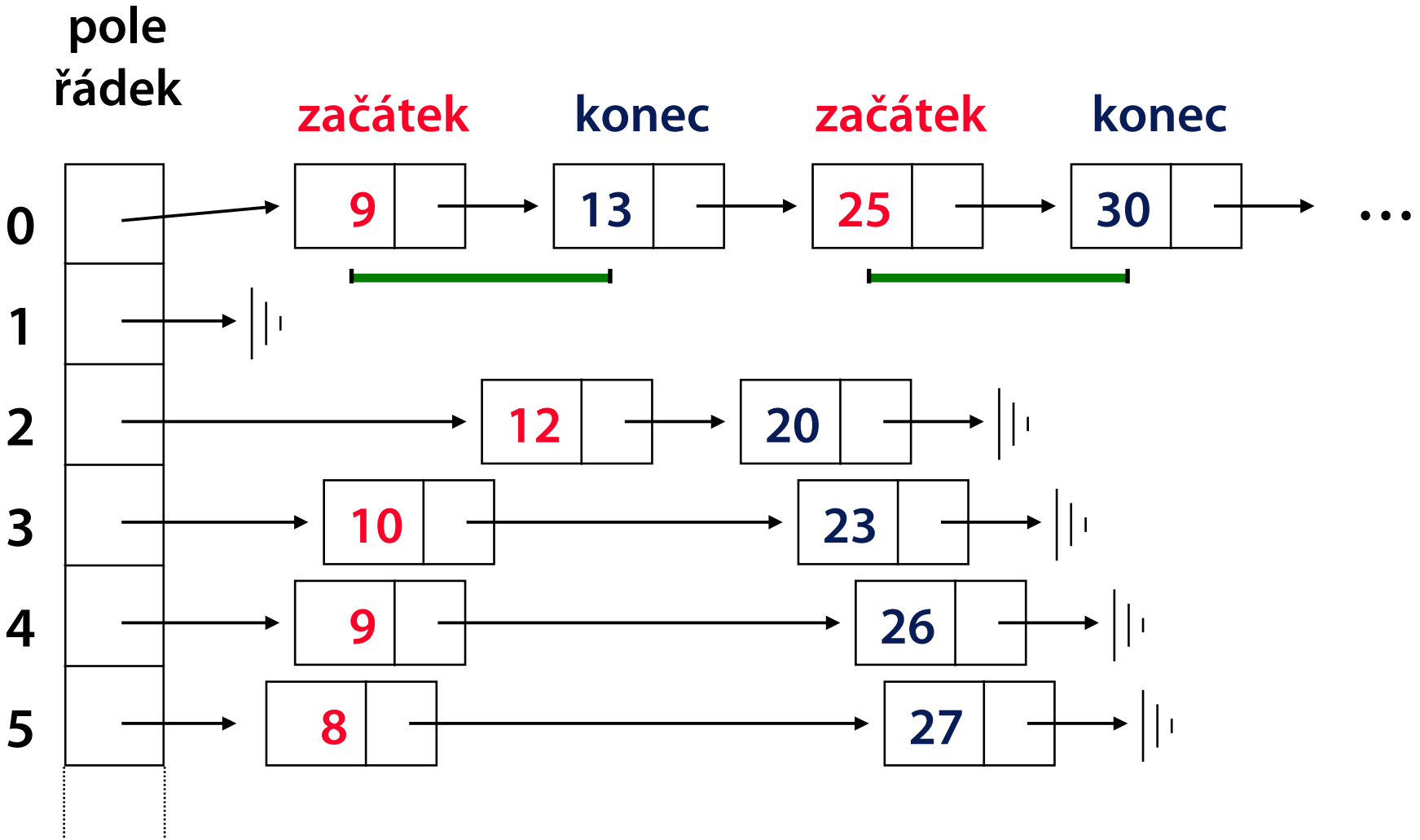
- efektivní implementace **množinových operací** (slévání uspořádaných seznamů)
- lze použít při **vyplňovacích algoritmech**

Výhodná pro **oblasti s jednoduchým okrajem**

Pro každou řádku se ukládá uspořádaný **seznam pixelů**, kterými prochází hranice oblasti

- v těchto pixelech se mění **0** → **1** nebo **1** → **0**

Řádkový seznam změn



Množinové operace

Doplněk

- v každé řádce se přidá/odstraní prvek **[0]**

Binární operace – slévají se seznamy změn příslušných vstupních řádek

- nonekvivalence (**XOR**) je nejsnazší – jen se slévá (a odstraňují duplicitní záznamy)
- u jiných operací se zařazují na výstup jen některé záznamy (podle stavové pomocné proměnné)

Množinová operace na jedné řádce

```
void MergeLists (HList L1, HList L2, out HList Result)
{
  Result.Init();
  bool state = false;              // výstupní stav
  bool in1 = false, in2 = false;   // vstupní stavy
  while (!L1.Empty && !L2.Empty)
  {
    int x = min(L1.First, L2.First);
    if (x == L1.First)             // odebírám z L1
    {
      in1 = !in1; L1.Get();
    }
    if (x == L2.First)             // odebírám z L2
    {
      in2 = !in2; L2.Get();
    }
    if (BooleanOperation(in1, in2) != state)
    {
      Result.Put(x);
      state = !state;
    }
  }
  // ...dočtení zbytku neprázdného vstupního seznamu
}
```

Literatura

J. Foley, A. van Dam, S. Feiner, J. Hughes: Computer Graphics, Principles and Practice, 844-846, 552-555, 992-996

matfyz

Computer Graphics Charles University

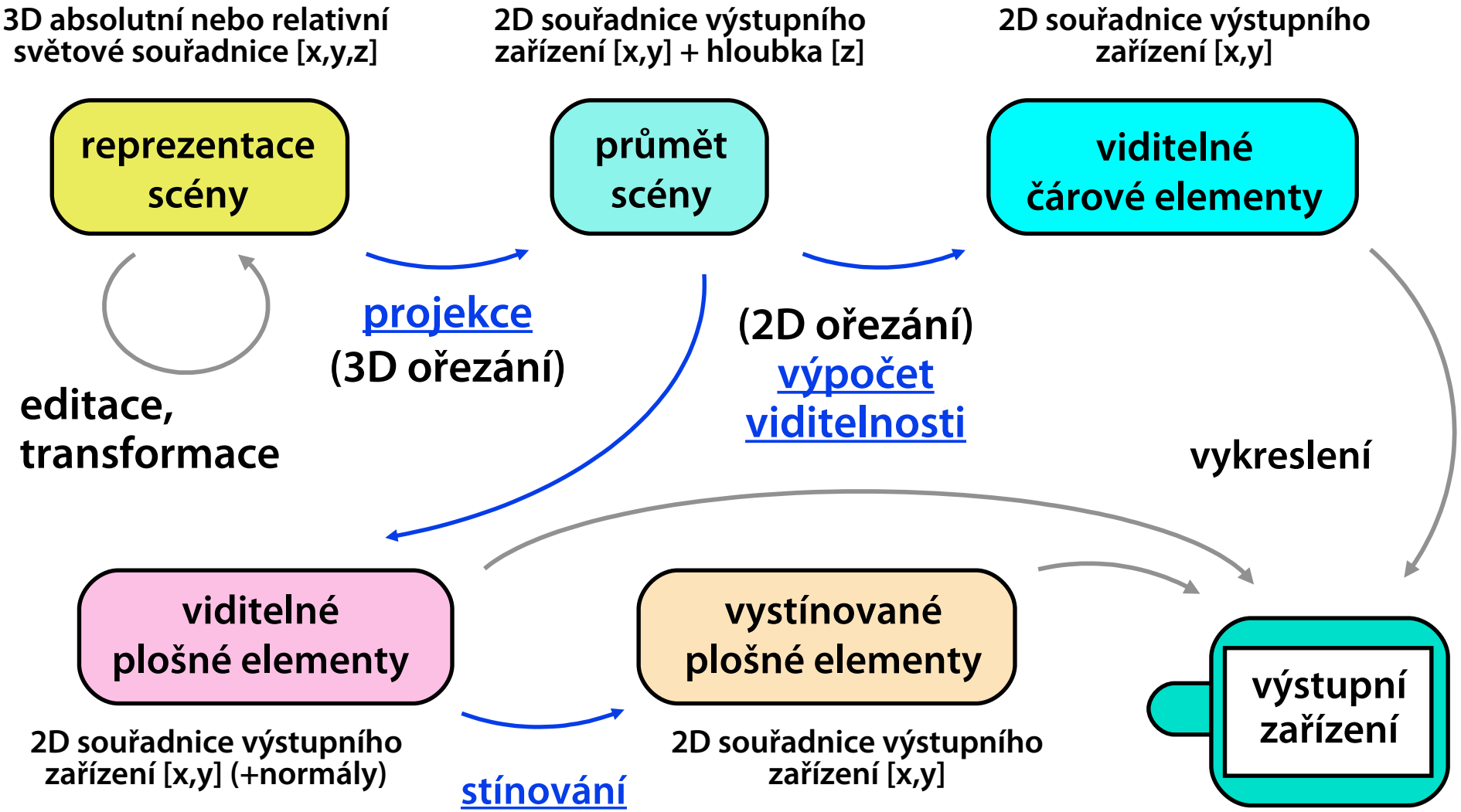
Úvod do 3D grafiky

CGG MFF UK Praha

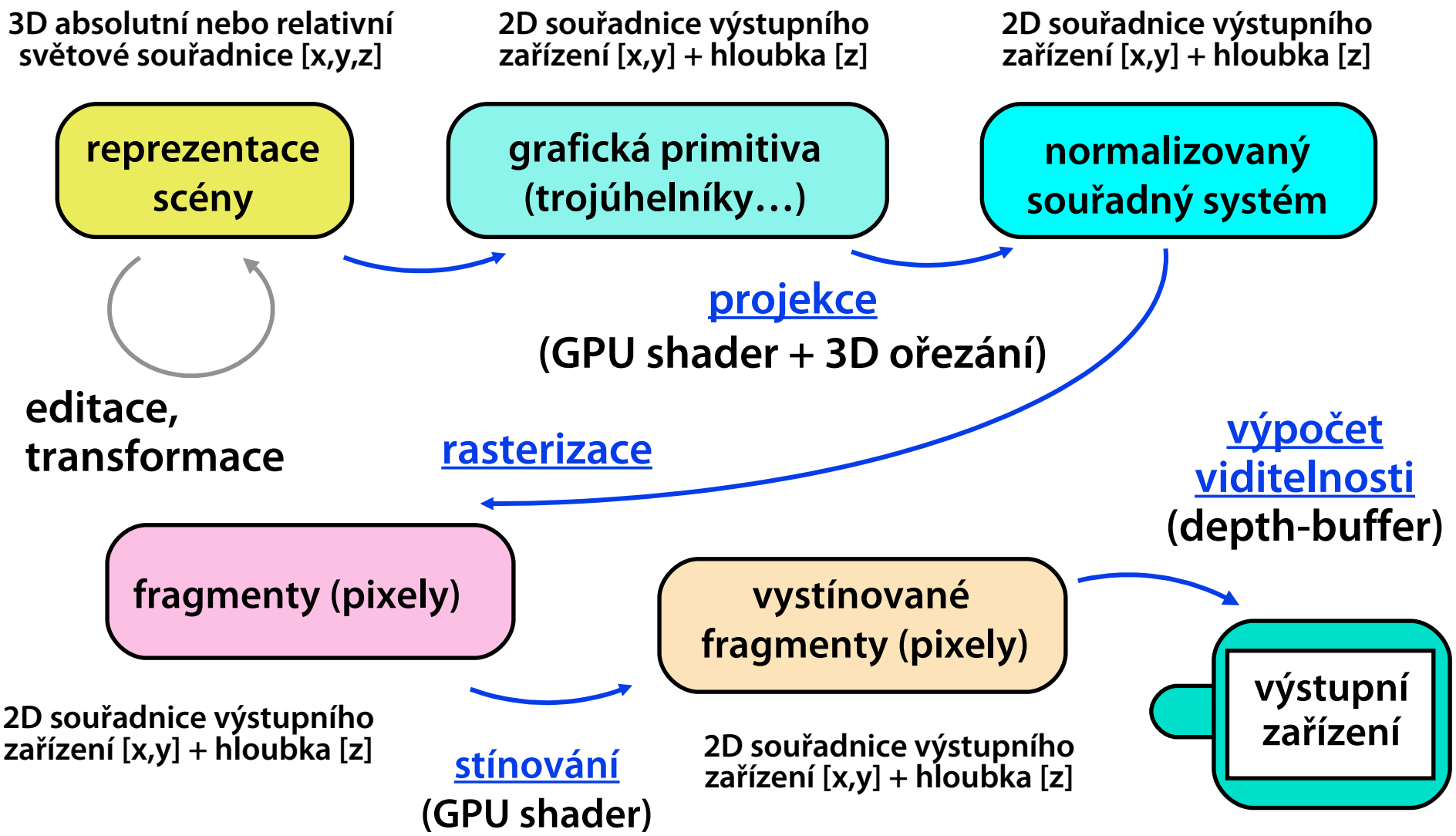
pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/

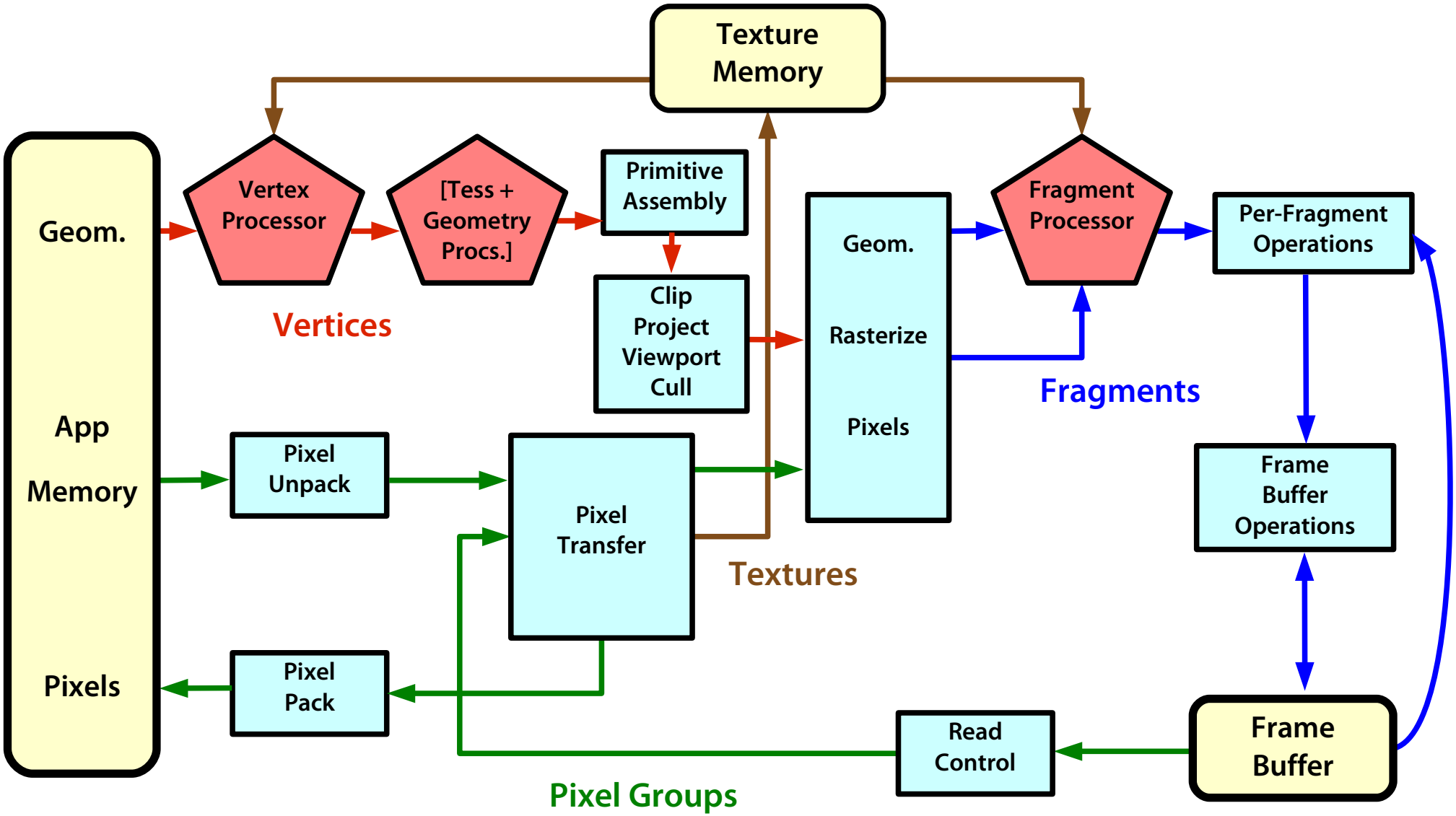
3D grafický systém – historie



3D grafický systém – GPU



GPU (OpenGL)



Fáze zpracování scény

Editace, transformace (práce s 3D daty)

- funkce modelovacího programu (CAD, animační systém…)
- u čistě zobrazovacích systémů chybí (simulační programy, hry…)

Projekce (příp. i s 3D ořezáním)

- transformace prostorových souřadnic do roviny (se zachováním „hloubky Z" pro výpočet viditelnosti)
- různé úhly pohledu, perspektiva

Fáze zpracování scény II

2D ořezání

- odstranění objektů ležících mimo kreslený výřez

Výpočet viditelnosti

- odstranění zakrytých objektů nebo jejich částí
- **čárová kresba** (kreslí se jen hrany)
- **plošná kresba** (vybarvují se vnitřky ploch)

Stínování

- zlepšení prostorového vjemu napodobením **osvětlení scény** (někdy i vržené stíny)

matfyz

Computer Graphics Charles University

Lineární transformace

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/

Požadavky

Běžně používané transformace

- posunutí, otočení, zvětšení/zmenšení, zkosení…
- rovnoběžná i perspektivní projekce

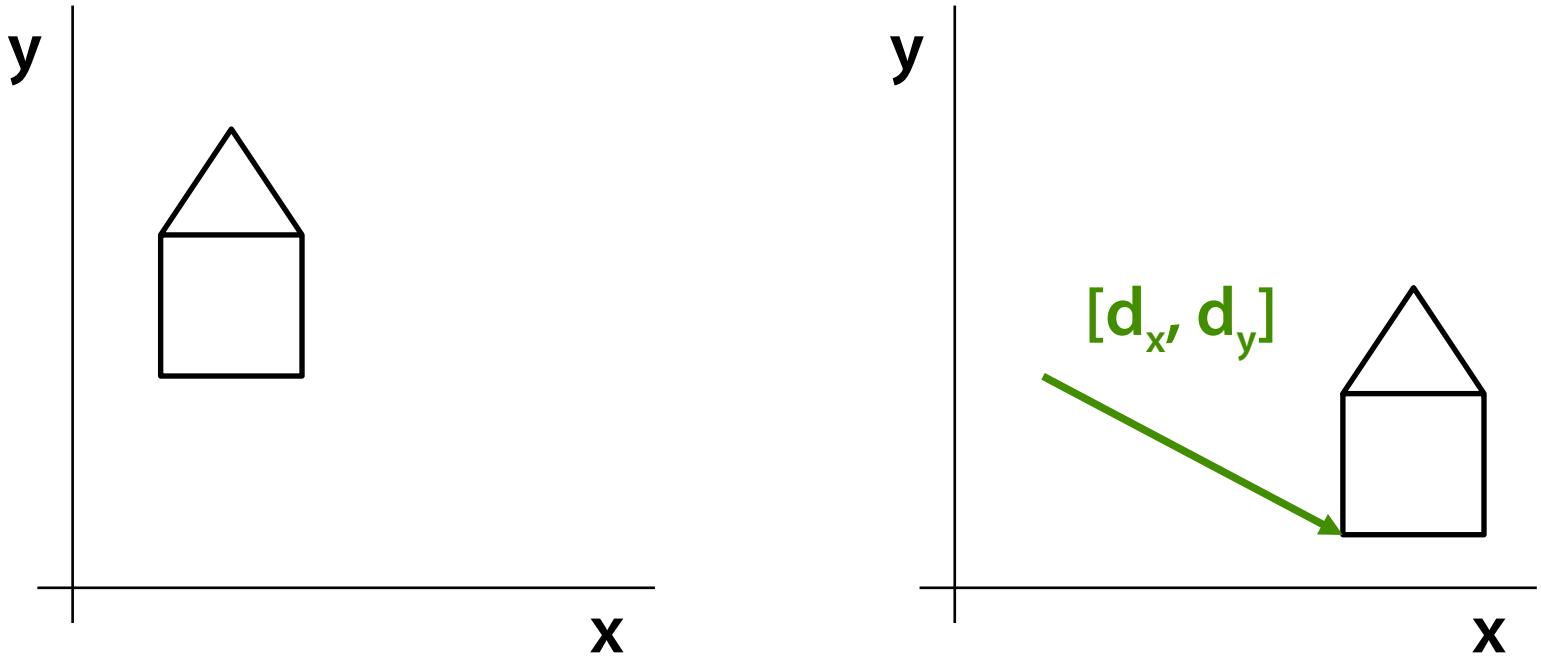
Snadná a **efektivní implementace**

- výpočty se provádějí masově (běžně i 10^8 transformací najednou)

Zvláštní úkony

- zřetězení jednoduchých transformací, výpočet inverzní transformace…

Posunutí v rovině



$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} + \begin{bmatrix} d_x & d_y \end{bmatrix}$$

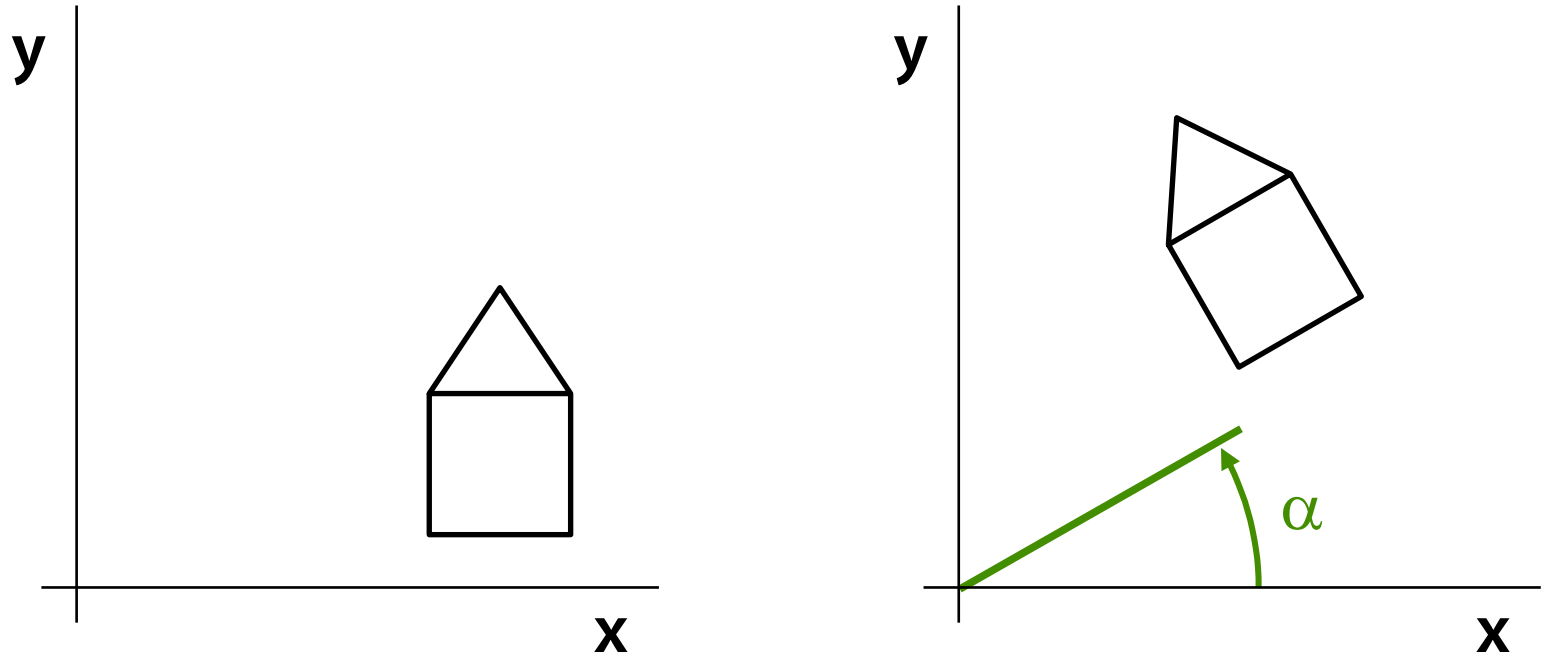
Maticové transformace

Násobení vektoru souřadnic **maticí zprava**

- à la **DirectX** (**OpenGL** to má obráceně)
- kartézské souřadnice bodu **[x, y]** tvoří **řádkový vektor**
- **transformační matice** je čtvercová (v rovině má rozměr 2×2)

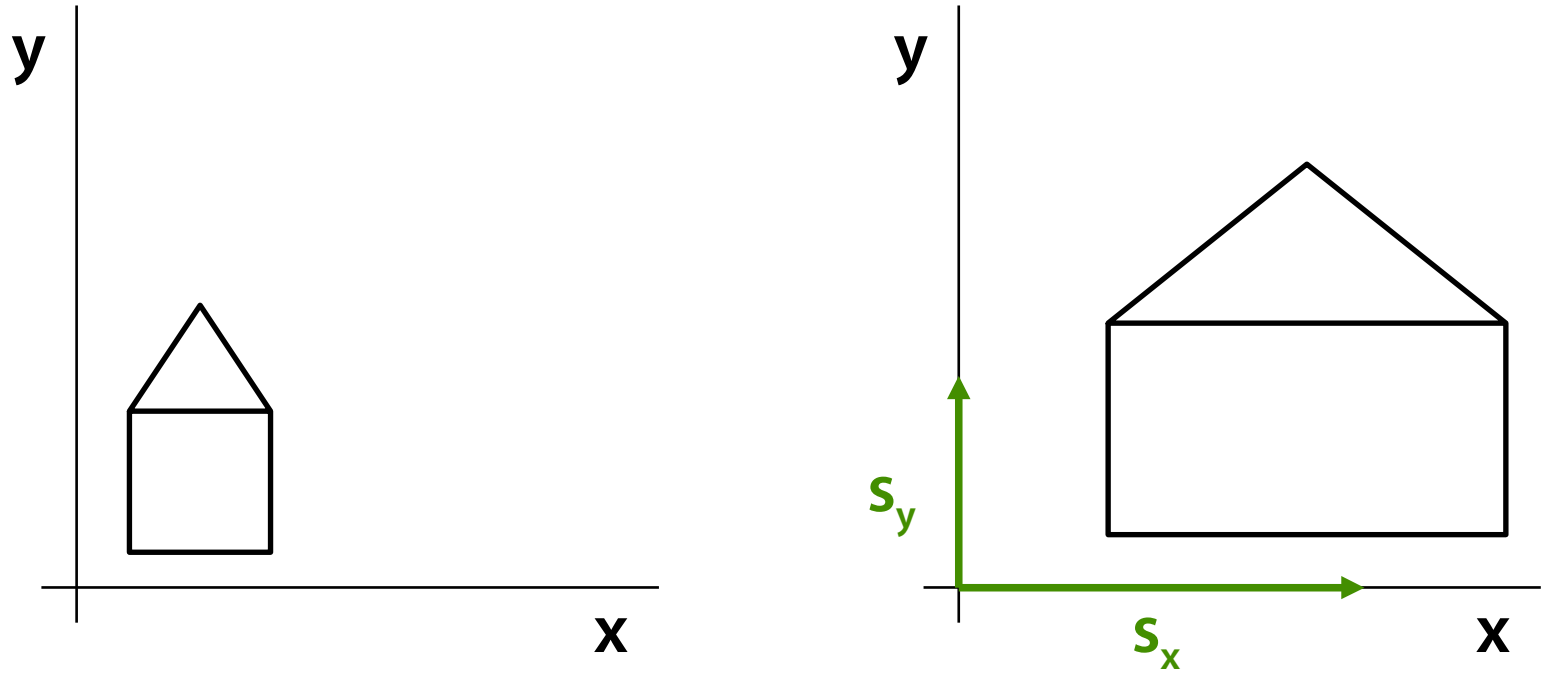
$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \cdot \begin{bmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{bmatrix}$$

Otočení v rovině kolem počátku



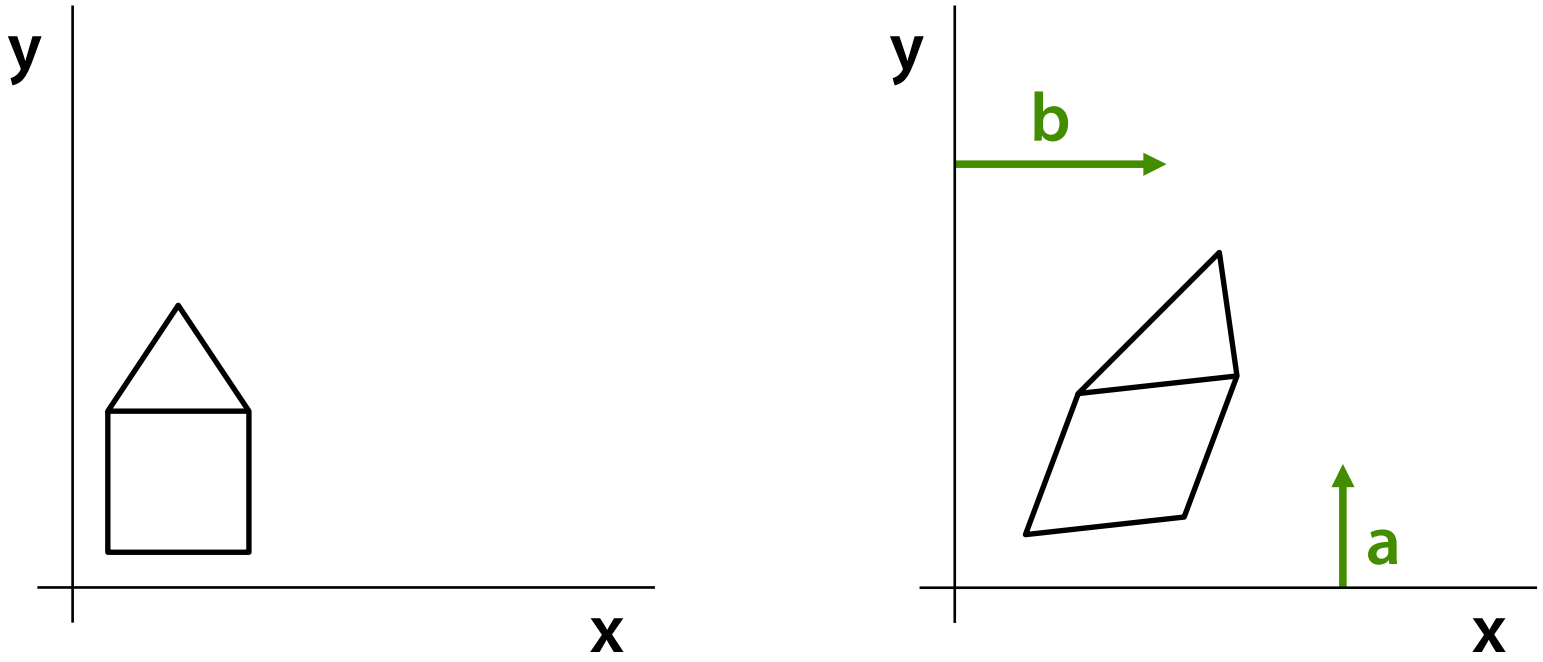
$$\mathbf{R}(\alpha) = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix}$$

Zmenšení / zvětšení v rovině



$$S(s_x, s_y) = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

Zkosení v rovině



$$Sh(a,b) = \begin{bmatrix} 1 & a \\ b & 1 \end{bmatrix}$$

Homogenní souřadnice

Jednotná reprezentace **afinních transformací**

- transformace zachovávající rovnoběžnost
- **posunutí** nelze v kartézských souřadnicích reprezentovat maticově

Nejpoužívanější **neafinní transformace**

- **perspektivní transformace** (projekce)

Reprezentace složených transformací

- násobení matic (asociativita)

Algebraická motivace

Přímka v rovině má souřadnice **[a, b, c]** (mnohoznačné)

$$a \cdot x + b \cdot y + c = 0$$

Bod v rovině má souřadnice **[x, y]** (jednoznačné)

Úloha 1: hledání **přímky** **[a, b, c]** procházející dvěma danými body $[x_1, y_1]$ a $[x_2, y_2]$

$$a \cdot x_1 + b \cdot y_1 + c = 0$$
$$a \cdot x_2 + b \cdot y_2 + c = 0$$

soustava (1)

Algebraická motivace II

Úloha 2: hledání **bodu** [**x, y**], ve kterém se protnou dvě dané přímky [$\mathbf{a_1, b_1, c_1}$] a [$\mathbf{a_2, b_2, c_2}$]

$$\mathbf{a_1 \cdot x + b_1 \cdot y + c_1 = 0}$$
$$\mathbf{a_2 \cdot x + b_2 \cdot y + c_2 = 0}$$

soustava (**2**)

Soustava (1) má vždy (nekonečně mnoho) řešení

Soustava (2) má řešení jen pokud není $\mathbf{a_1 \cdot b_2 = a_2 \cdot b_1}$

Algebraická motivace

Po rozšíření roviny o **nevlastní body** a zavedení **homogenních souřadnic** [x, y, w] budou obě předchozí úlohy symetrické a soustava (2') bude vždy řešitelná

$$a \cdot x_1 + b \cdot y_1 + c \cdot w_1 = 0$$
$$a \cdot x_2 + b \cdot y_2 + c \cdot w_2 = 0$$

soustava (1')

$$a_1 \cdot x + b_1 \cdot y + c_1 \cdot w = 0$$
$$a_2 \cdot x + b_2 \cdot y + c_2 \cdot w = 0$$

soustava (2')

Převody souřadnic

Kartézské na homogenní

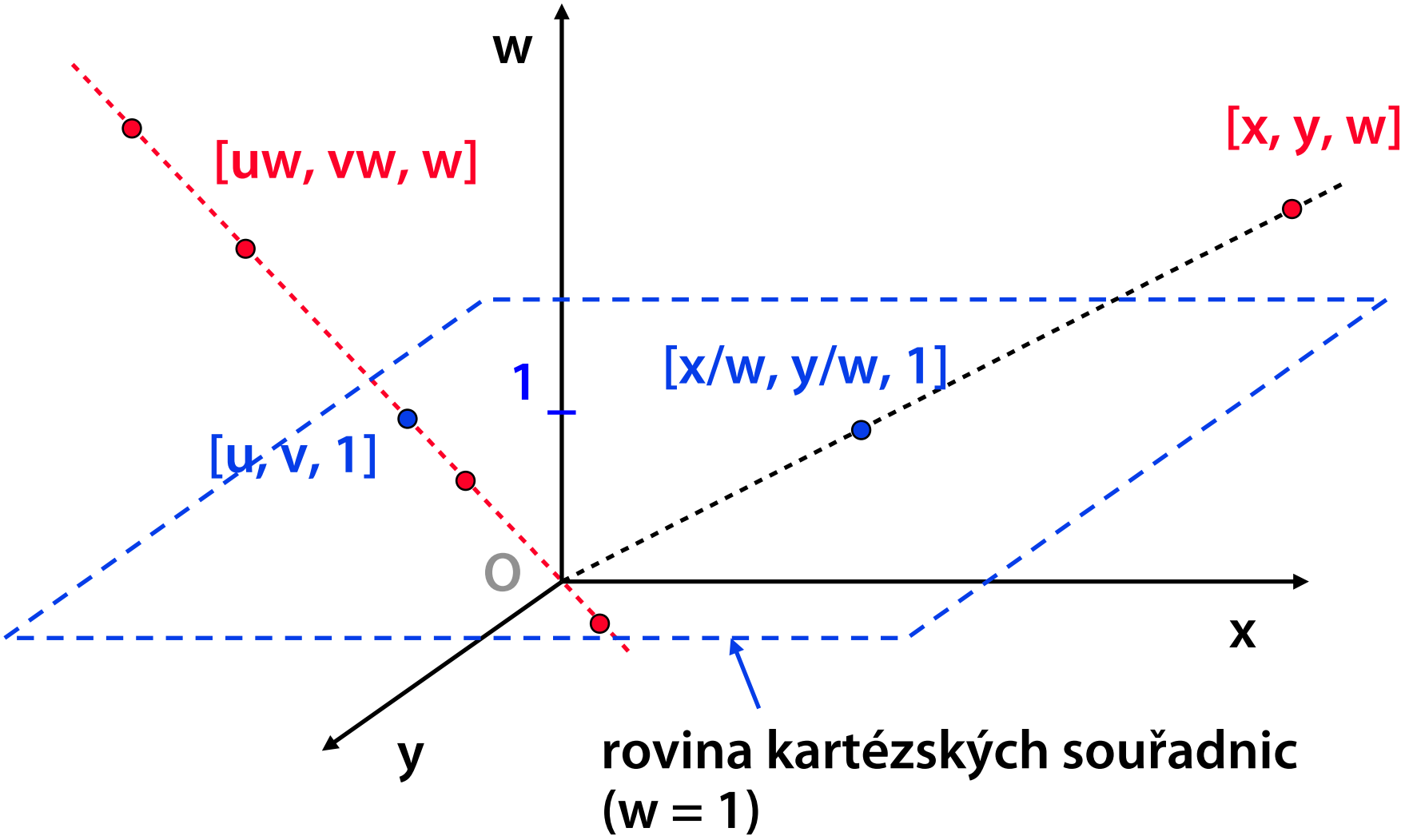
$$[x \quad y] \rightarrow [x \quad y \quad 1]$$

Homogenní na kartézské (jen vlastní body)

$$[x \quad y \quad w] \rightarrow \left[\frac{x}{w} \quad \frac{y}{w}\right]$$

$$w \neq 0$$

Projektivní prostor



Homogenní transformační matice

Posunutí
(„translation")

$$T(t_x, t_y) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

Otočení („rotation")
kolem počátku

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

Zmenšení / zvětšení
(„scale")

$$S(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Homogenní transformační matice II

Zkosení
(„shear")

$$\mathbf{Sh}(a,b) = \begin{bmatrix} 1 & a & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Složené transformace

$$\left(\left(\left([x, y, w] \cdot T_1\right) \cdot T_2\right) \cdot T_3\right) = [x, y, w] \cdot \left(T_1 \cdot T_2 \cdot T_3\right)$$

Otočení o úhel α kolem bodu [x, y]

$$R(x, y, \alpha) = T(-x,-y) \cdot R(\alpha) \cdot T(x, y)$$

Transformace v průmětně

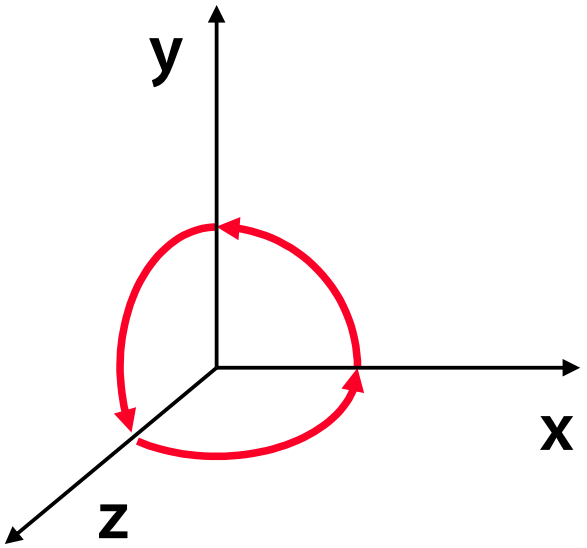
souřadné systémy na výstupu

Převod reálných souřadnic do **souřadnic zobrazovaného okna**

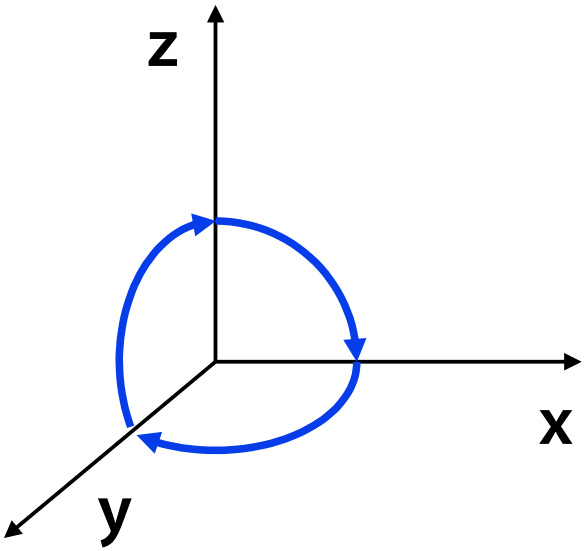
$$X_{int} = \text{round}(D_x + S_x * X_f)$$

$$Y_{int} = \text{round}(D_y + S_y * Y_f)$$

Prostorové souřadnice



levotočivý systém
(„right-handed")



pravotočivý systém
(„left-handed")

OpenGL (y), Vulkan (-y?), 3ds Max (z), Blender (z), Wavefront=OBJ, HoloLens1 (y)…

Glide, DirectX (y), RenderMan (y), Unity (y), UE 4 (z), C4D (y), POV-ray (y), NDS (clipping)…

Jedinou jistotou je **osa x mířící doprava**

Homogenní souřadnice

$$\begin{bmatrix} x & y & z \end{bmatrix} \rightarrow \begin{bmatrix} x & y & z & 1 \end{bmatrix}$$

$$\begin{bmatrix} x & y & z & w \end{bmatrix} \rightarrow \begin{bmatrix} \frac{x}{w} & \frac{y}{w} & \frac{z}{w} \end{bmatrix} \quad (w \neq 0)$$

Maticová transformace

$$\begin{bmatrix} x' & y' & z' & w' \end{bmatrix} = \begin{bmatrix} x & y & z & w \end{bmatrix} \cdot \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ t_{41} & t_{42} & t_{43} & t_{44} \end{bmatrix}$$

Homogenní transformační matice

Posunutí

$$T(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$$

Zkosení

$$Sh(a, b, c, d, e, f) = \begin{bmatrix} 1 & a & b & 0 \\ c & 1 & d & 0 \\ e & f & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Homogenní transformační matice II

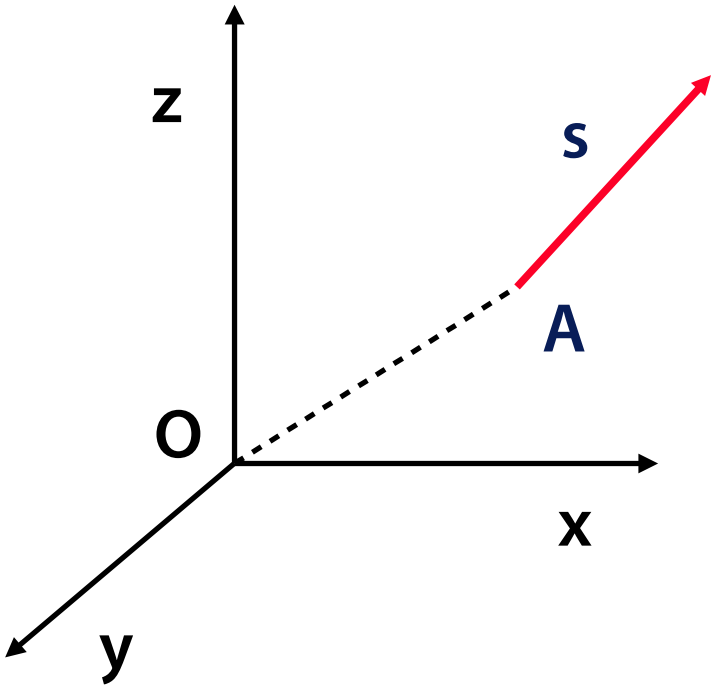
Otočení
kolem osy **y**

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

Otočení
kolem osy **z**

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

Přenos polopřímky do osy z



Polopřímka je zadána bodem **A** a směrovým vektorem **s**

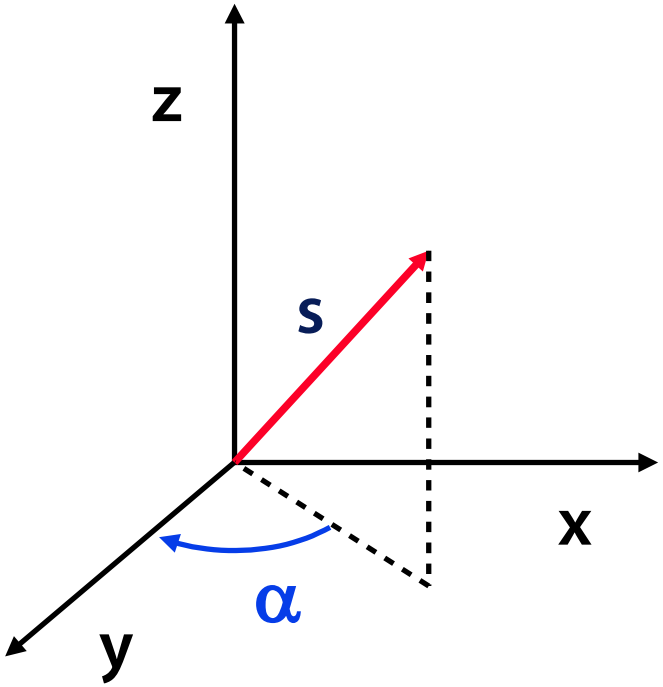
$$\mathbf{M} = \mathbf{T}(-\mathbf{A})$$

$$\mathbf{M}^{-1} = \mathbf{T}(\mathbf{A})$$

1. krok

přenesení bodu **A** do počátku

Přenos polopřímky do osy z



$$\mathbf{M} = \mathbf{T}(-\mathbf{A}) \cdot \mathbf{R}_z(\alpha)$$

$$\mathbf{M}^{-1} = \mathbf{R}_z(-\alpha) \cdot \mathbf{T}(\mathbf{A})$$

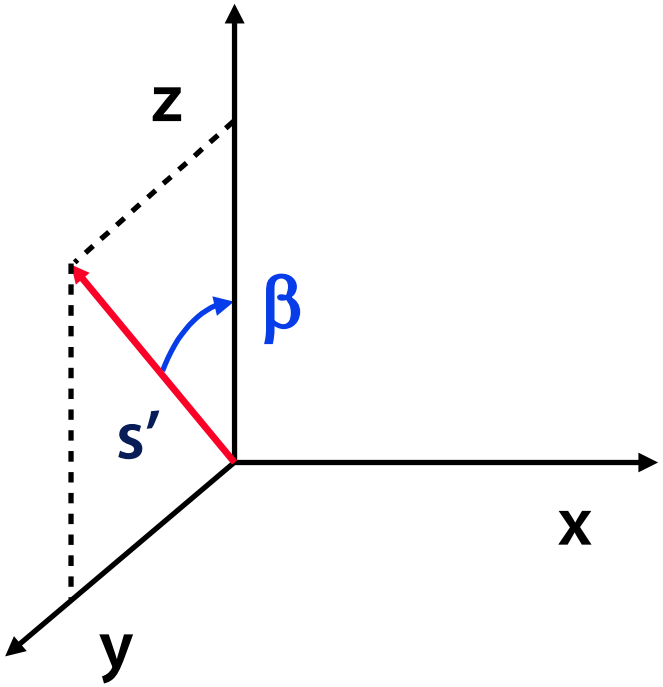
$$\cos\alpha = \frac{s_y}{\sqrt{s_x^2 + s_y^2}}$$

$$\sin\alpha = \frac{s_x}{\sqrt{s_x^2 + s_y^2}}$$

2. krok

otočení polopřímky do roviny **yz** (okolo osy **z**)

Přenos polopřímky do osy z



$$\mathbf{M} = \mathbf{T}(-\mathbf{A}) \cdot \mathbf{R}_z(\alpha) \cdot \mathbf{R}_x(\beta)$$

$$\mathbf{M}^{-1} = \mathbf{R}_x(-\beta) \cdot \mathbf{R}_z(-\alpha) \cdot \mathbf{T}(\mathbf{A})$$

$$\cos\beta = \frac{s_z}{\sqrt{s_x^2 + s_y^2 + s_z^2}}$$

$$|\sin\beta| = \frac{\sqrt{s_x^2 + s_y^2}}{\sqrt{s_x^2 + s_y^2 + s_z^2}}$$

3. krok

otočení polopřímky do osy **z** (okolo osy **x**)

Aplikace transformace M

Shrnutí transformace M

$$\mathbf{M}(\mathbf{A}, \mathbf{s}) = \mathbf{T}(-\mathbf{A}) \cdot \mathbf{R}_z(\alpha) \cdot \mathbf{R}_x(\beta)$$

$$\mathbf{M}(\mathbf{A}, \mathbf{s})^{-1} = \mathbf{R}_x(-\beta) \cdot \mathbf{R}_z(-\alpha) \cdot \mathbf{T}(\mathbf{A})$$

Otočení kolem dané osy

$$\mathbf{R}(\mathbf{A}, \mathbf{s}, \theta) = \mathbf{M}(\mathbf{A}, \mathbf{s}) \cdot \mathbf{R}_z(\theta) \cdot \mathbf{M}(\mathbf{A}, \mathbf{s})^{-1}$$

Zrcadlové převrácení podle dané roviny

$$\mathbf{Mirror}(\mathbf{A}, \mathbf{n}) = \mathbf{M}(\mathbf{A}, \mathbf{n}) \cdot \mathbf{S}(1,1,-1) \cdot \mathbf{M}(\mathbf{A}, \mathbf{n})^{-1}$$

Výpočet inverzní transformace

1. inverze matice $\mathbf{M}^{-1}$

2. po krocích

$$\mathbf{M} = \mathbf{A} \cdot \mathbf{B} \cdot \mathbf{C}$$

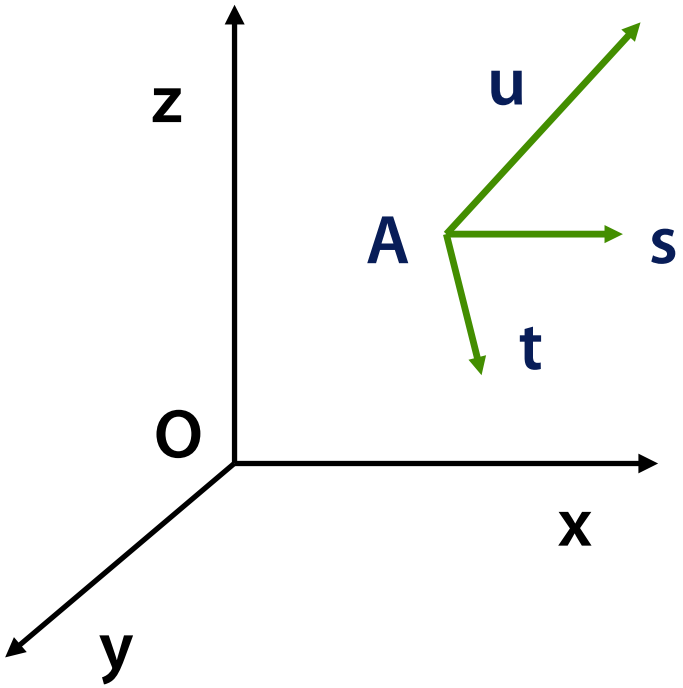
$$\mathbf{M}^{-1} = \mathbf{C}^{-1} \cdot \mathbf{B}^{-1} \cdot \mathbf{A}^{-1}$$

3. transpozice (ortonormální matice)

$$\mathbf{R}^{-1} = \mathbf{R}^{T}$$ **pro ortonormální matici R**

(ortonormální jsou všechny **rotační matice**)

Převod mezi souřadnými systémy



Souřadný systém je zadán svým počátkem **A** a trojicí vektorů **s, t, u**

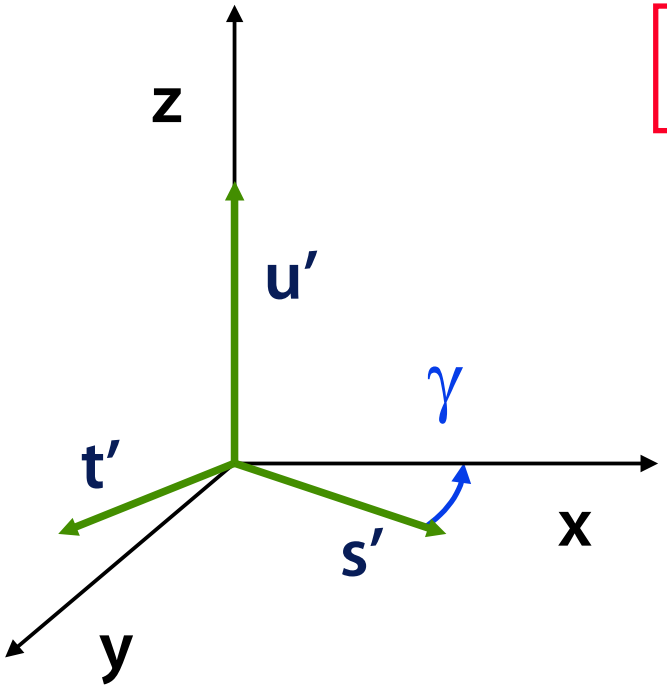
$$\mathbf{Cs} = \mathbf{M(A, u)}$$

$$\mathbf{Cs}^{-1} = \mathbf{M(A, u)}^{-1}$$

1. krok

přenesení polopřímky **(A, u)** do osy **z**

Převod mezi souřadnými systémy



$$Cs(A,s,t,u) = M(A,u)\cdot R_z(\gamma)$$

$$Cs(A,s,t,u)^{-1} = R_z(-\gamma)\cdot M(A,u)^{-1}$$

$$\cos\gamma = \frac{|s\cdot M(A,u)|_x}{|s\cdot M(A,u)|}$$

$$\sin\gamma = \frac{|s\cdot M(A,u)|_y}{|s\cdot M(A,u)|}$$

2. krok

ztotožnění os **s′**→**x** a **t′**→**y** (otočením kolem **z**=**u′**)

Literatura

J. Foley, A. van Dam, S. Feiner, J. Hughes: ***Computer Graphics, Principles and Practice***, 201-227

Jiří Žára a kol.: ***Počítačová grafika, principy a algoritmy***, 73-84

matfyz

Computer Graphics Charles University

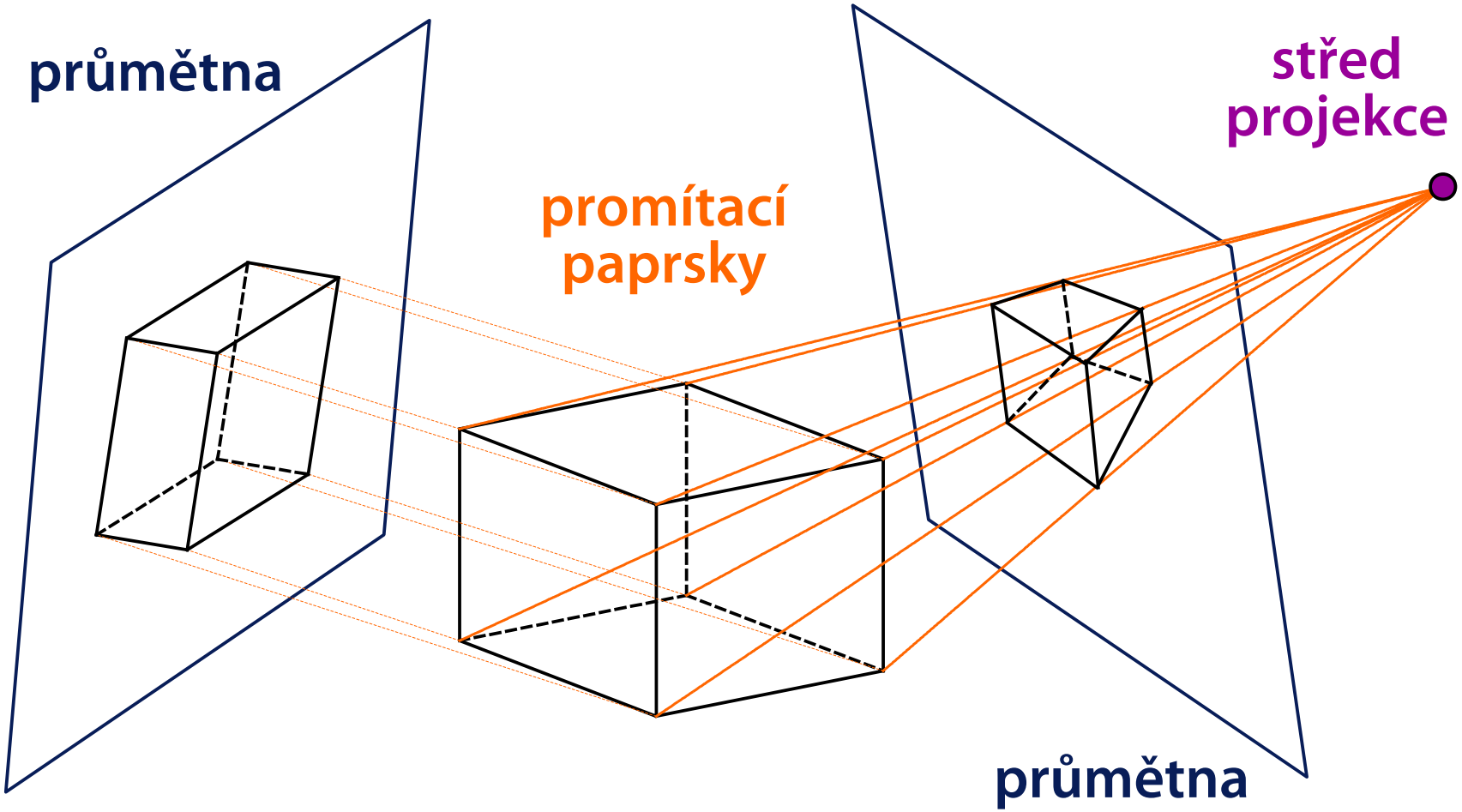
Zobrazovací projekce

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/

Základní pojmy



Klasifikace lineárních projekcí

Rovnoběžné projekce

- promítací paprsky jsou navzájem rovnoběžné

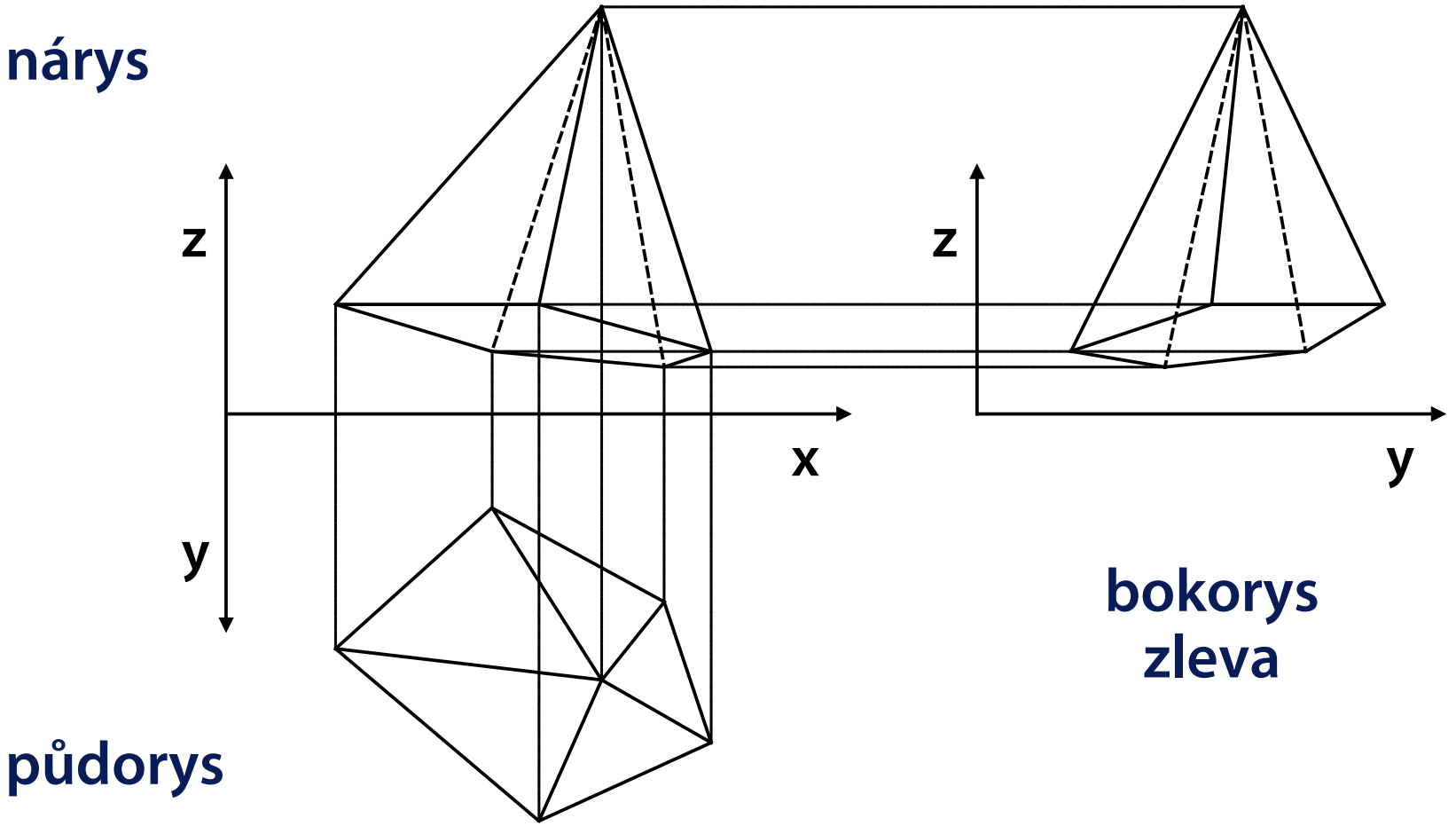
Kolmé projekce

- promítací paprsky jsou kolmé na průmětnu
- Mongeova projekce, půdorys, nárys, bokorys
- axonometrie (obecná kolmá projekce)

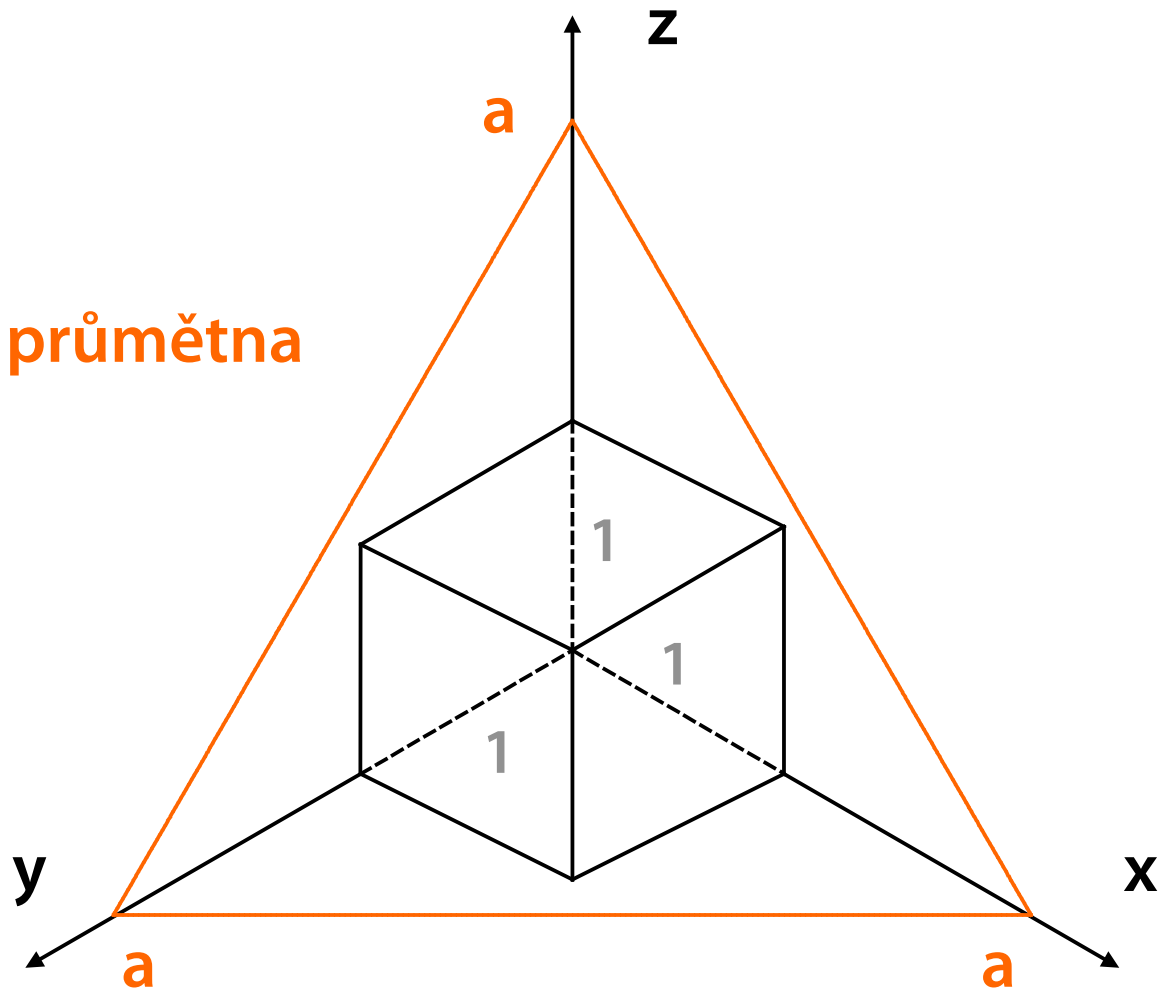
Kosoúhlé projekce

- kabinetní projekce (zkrácení měřítka osy **z** na **1/2**)
- kavalírní projekce (stejné měřítko na všech osách)

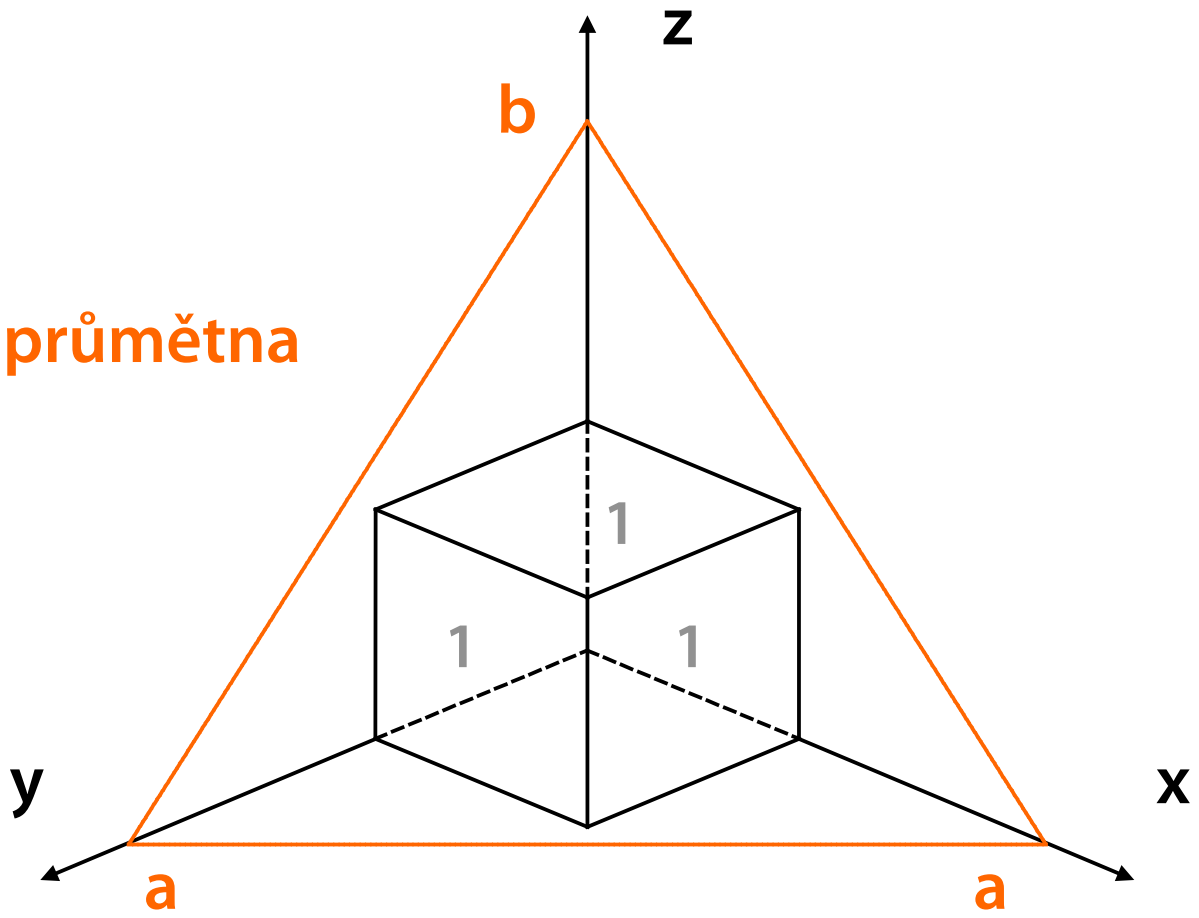
Mongeova projekce



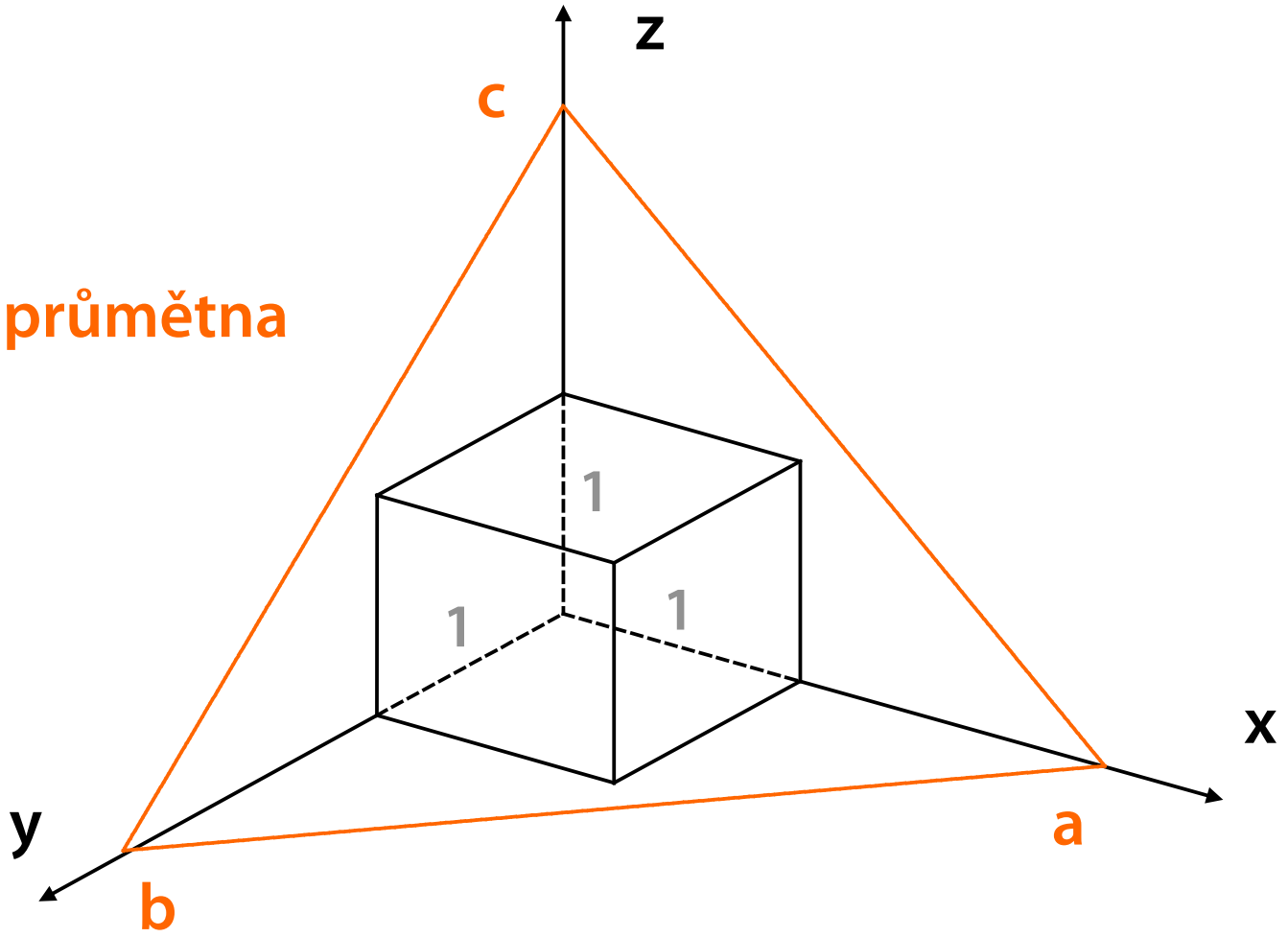
Axonometrie – isometrie



Axonometrie – dimetrie

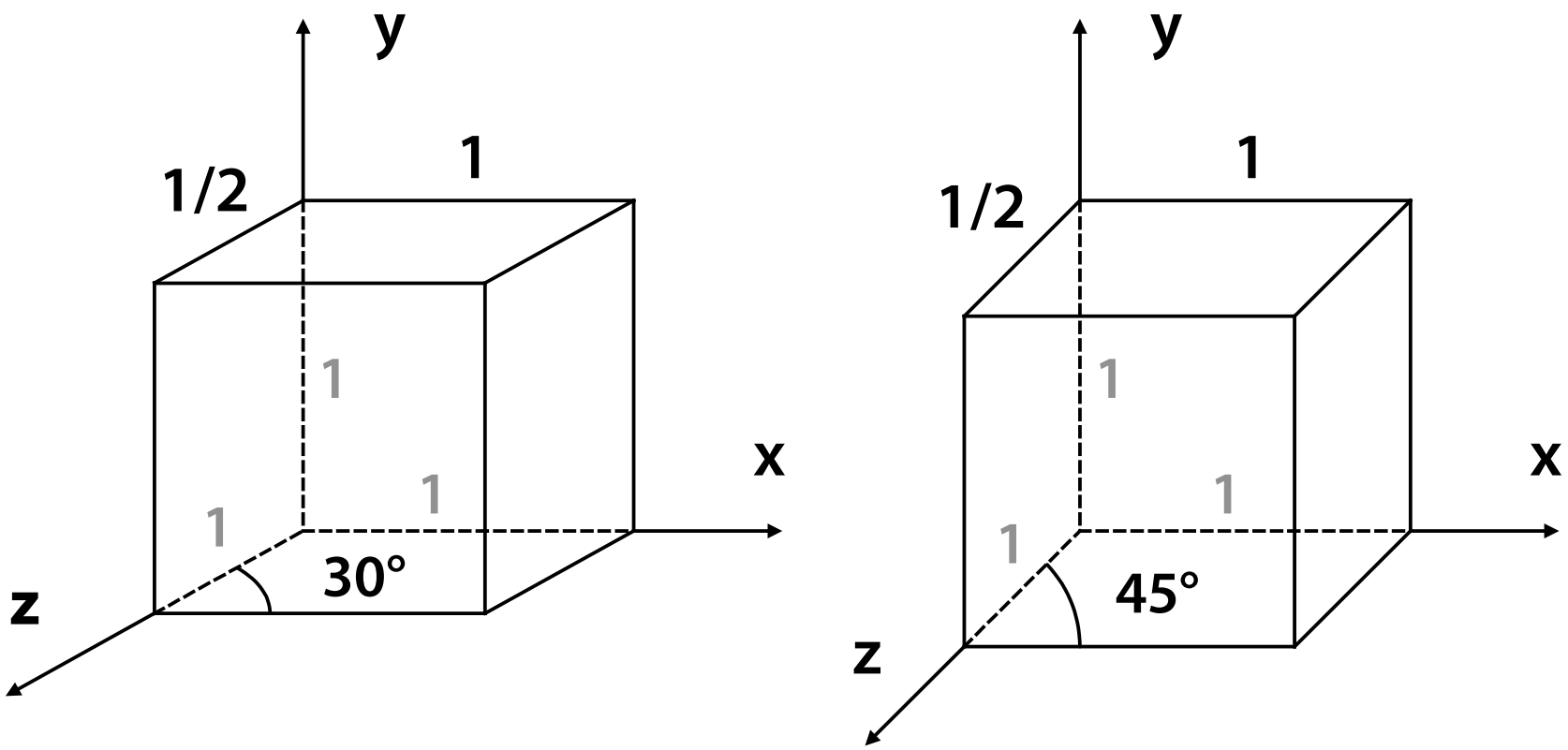


Axonometrie – trimetrie



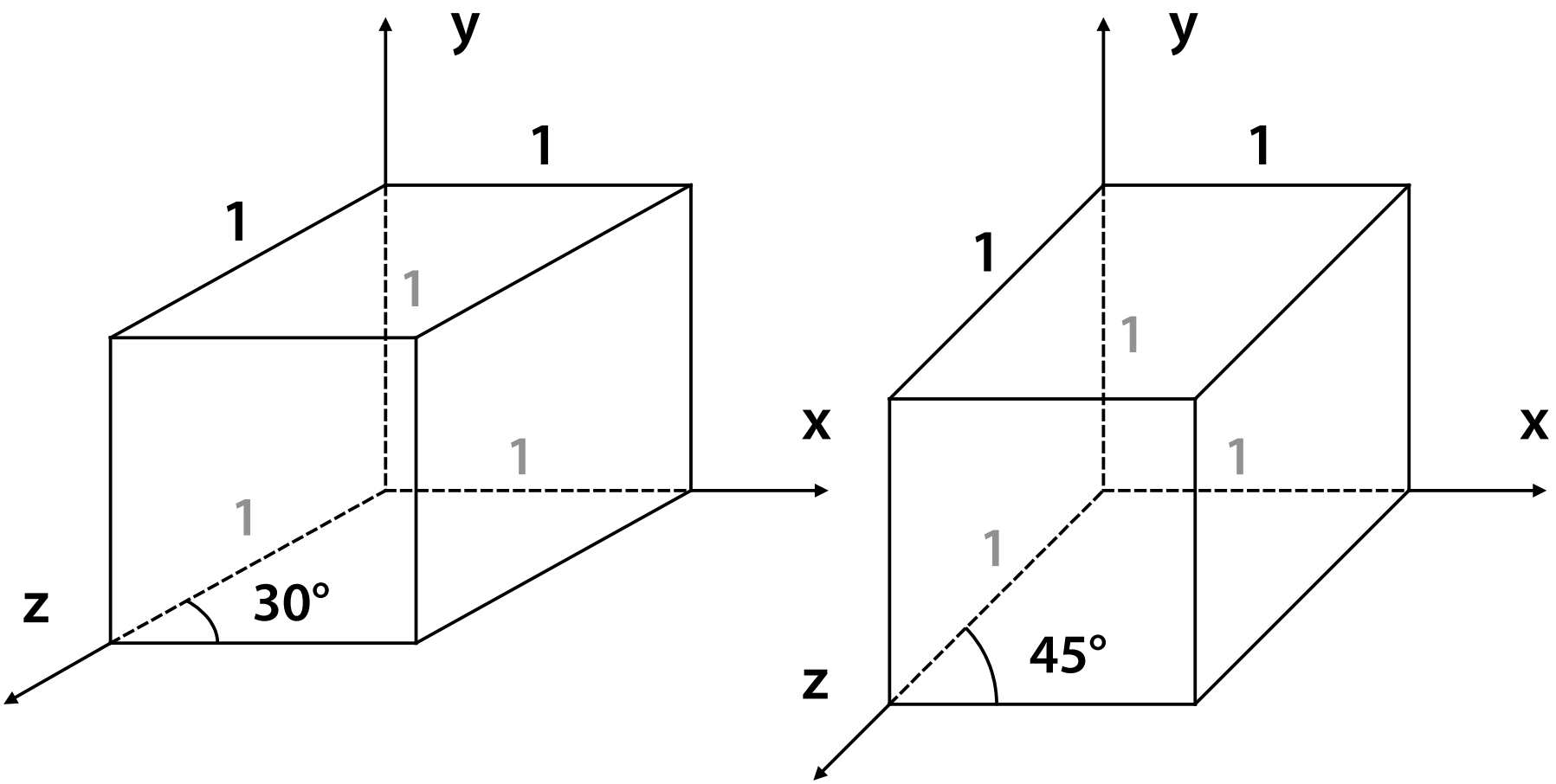
Kabinetní projekce

průmětna = xy



Kavalírní projekce

průmětna = xy



Klasifikace lineárních projekcí

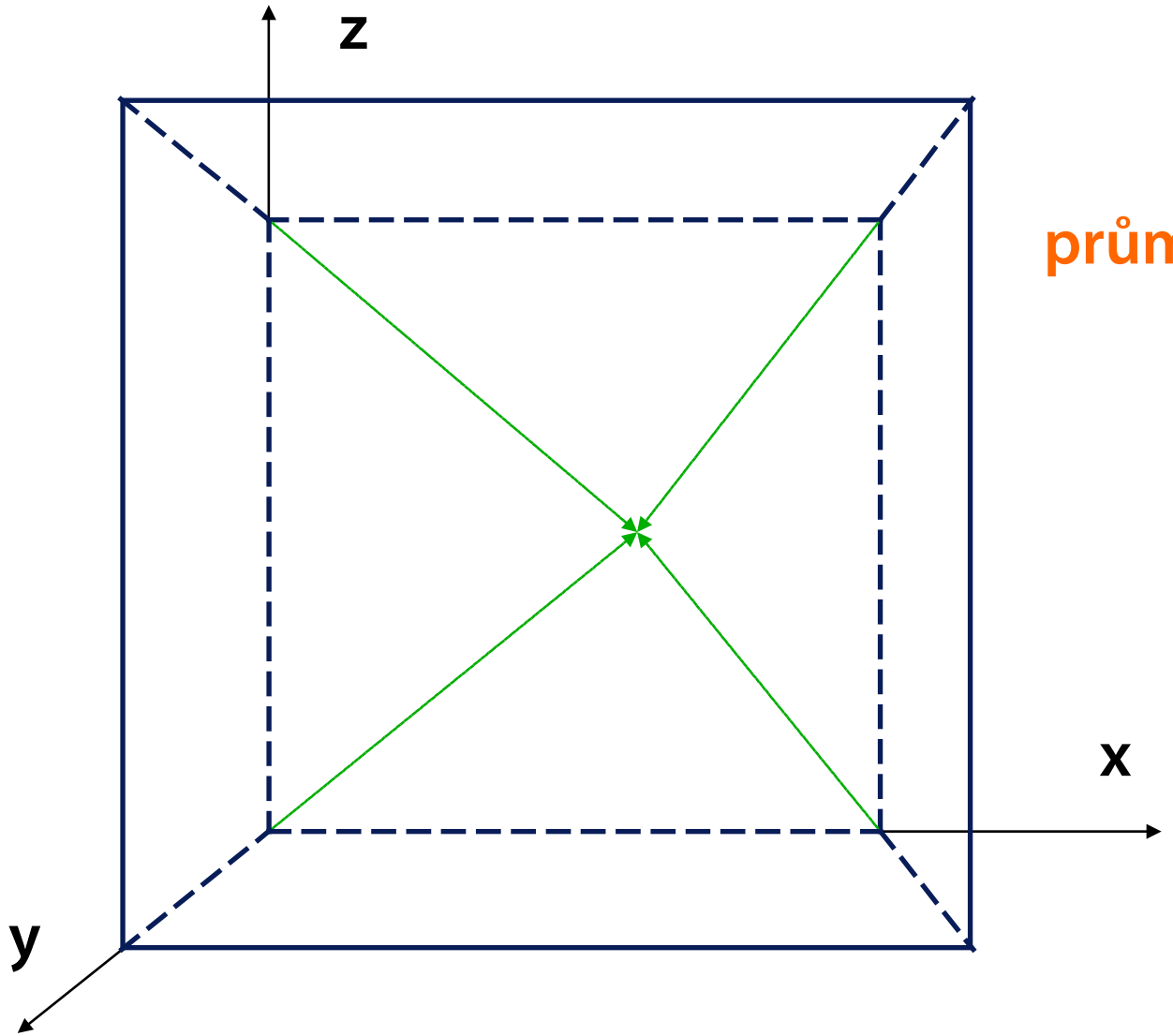
Středové projekce („perspective projections“)

- promítací paprsky tvoří svazek procházející jedním bodem, **středem projekce**
- nezachovává se rovnoběžnost (úběžníky)

Jednobodová perspektiva

- průmětna je rovnoběžná se dvěma souřadnými osami
- rovnoběžky se třetí osou se protínají v jednom hlavním úběžníku

Jednobodová perspektiva



průmětna || xz

Další středové projekce

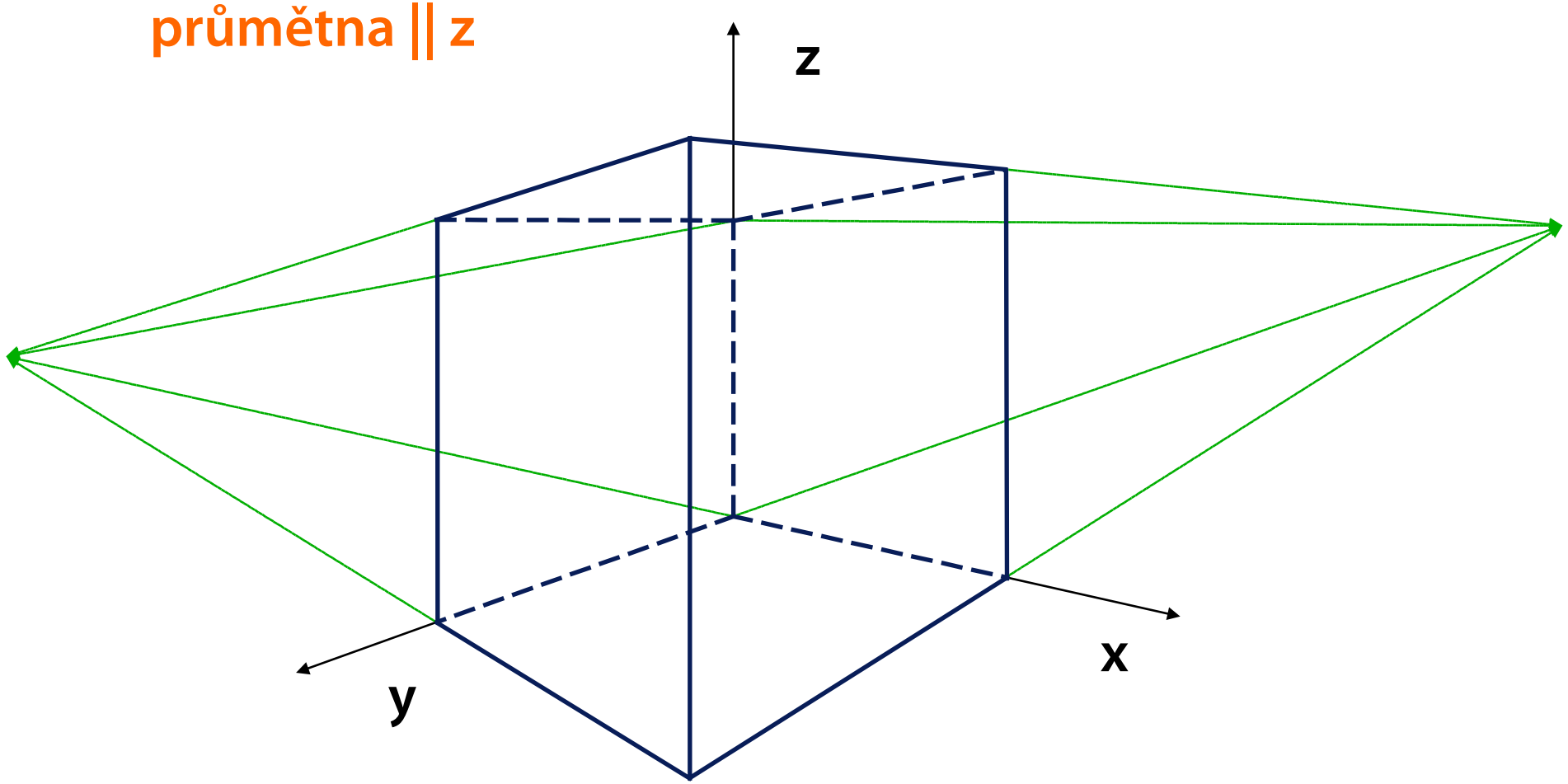
Dvoubodová perspektiva

- průmětna je rovnoběžná s jednou souřadnou osou
- rovnoběžky s ostatními osami se protínají ve dvou hlavních úběžnících

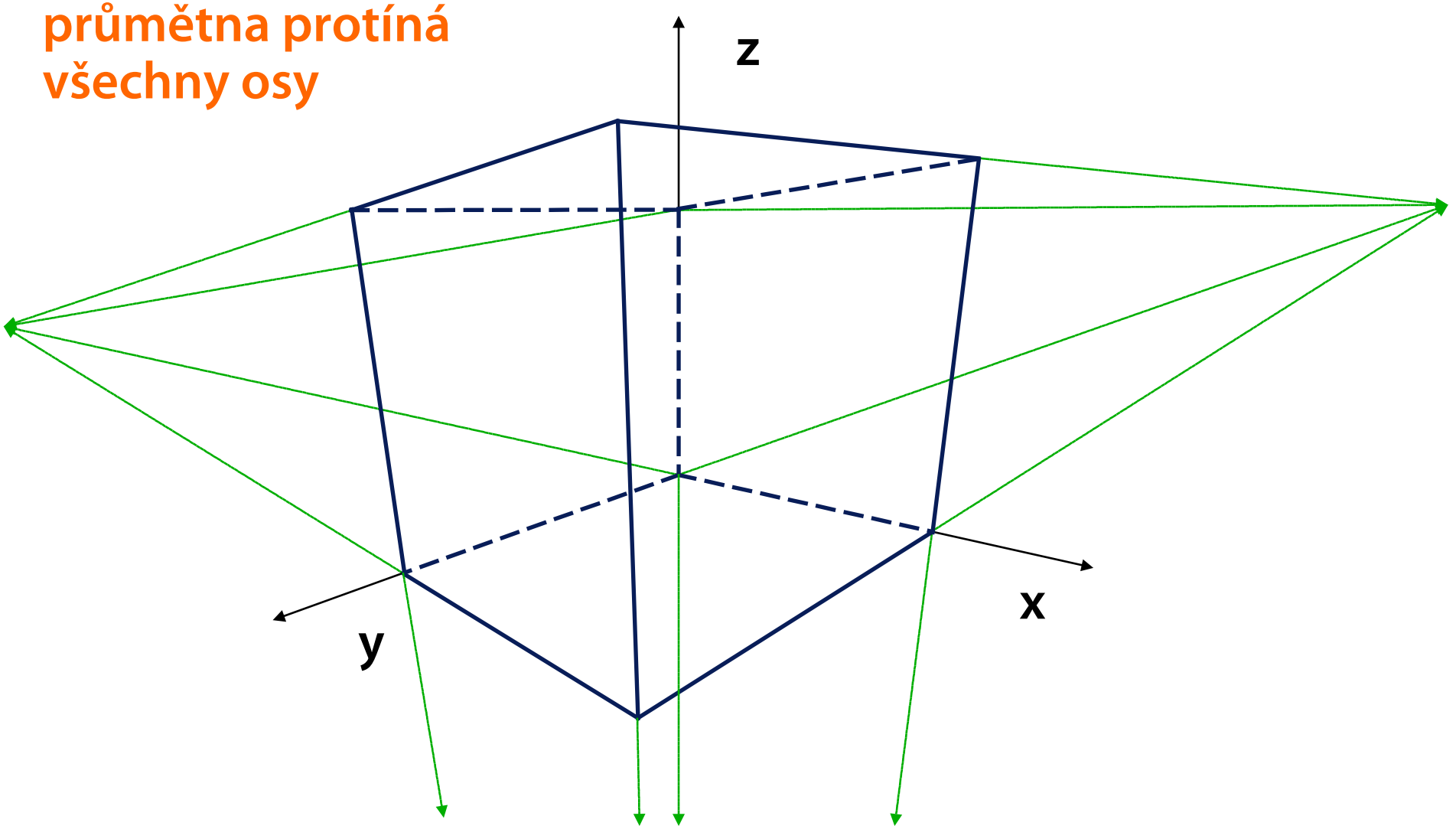
Tříbodová perspektiva

- průmětna má zcela obecnou polohu
- rovnoběžky se souřadnými osami se protínají ve třech hlavních úběžnících

Dvoubodová perspektiva



Tříbodová perspektiva



Implementace kolmé projekce

[**x, y**] budou souřadnice bodu v průmětu, **z** jeho hloubka (vzdálenost od pozorovatele)

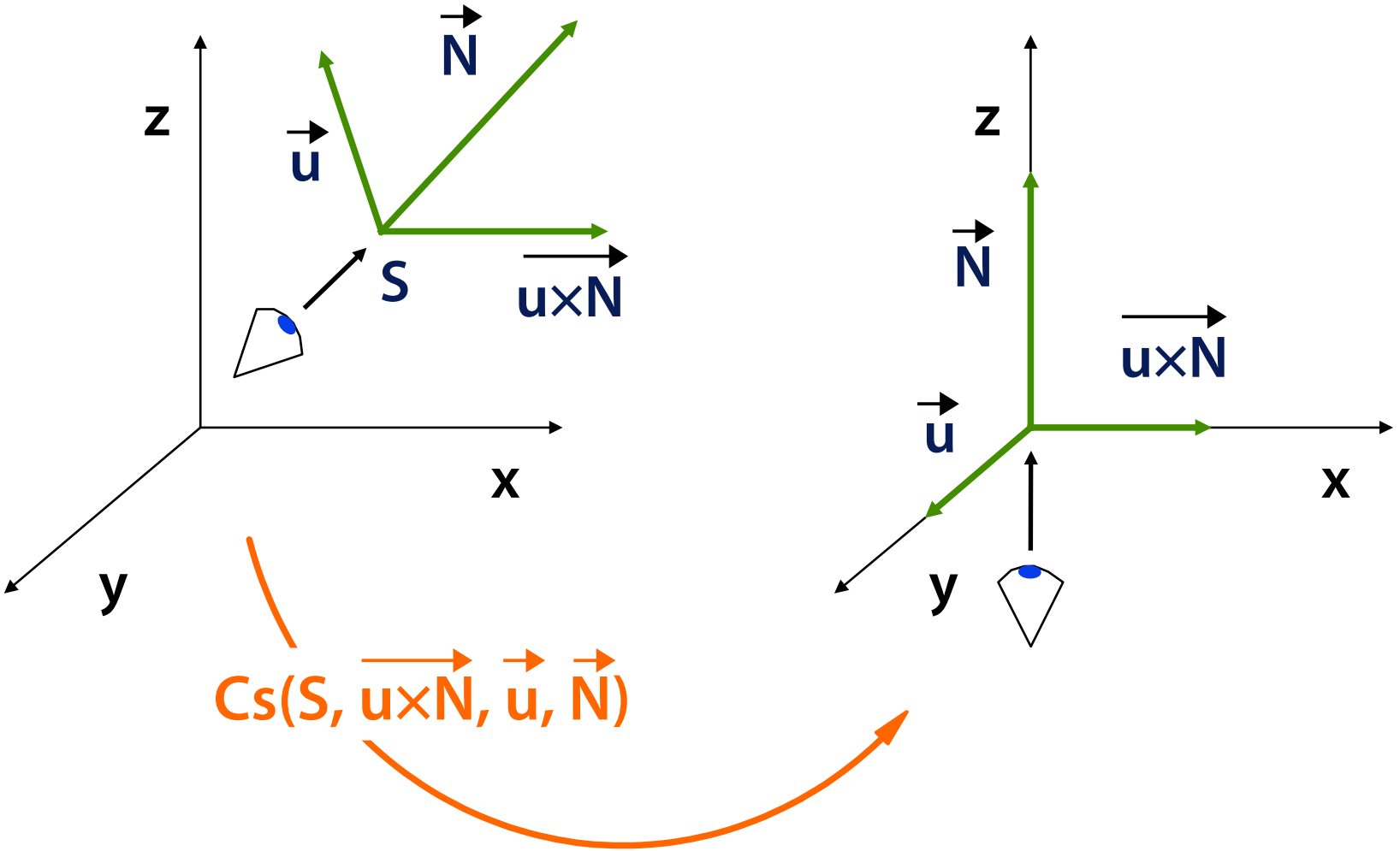
Základní pohledy (půdorys, nárys, bokorys)

- pouze permutace složek **x**, **y** a **z** (s příp. změnou znaménka)

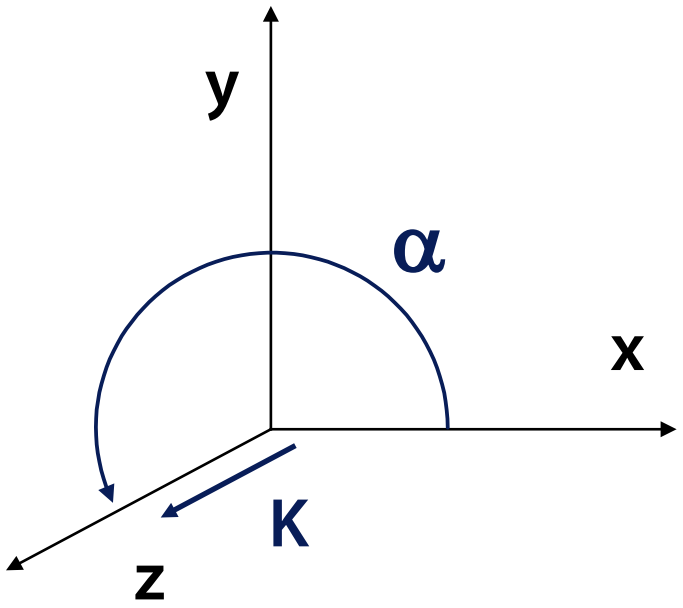
Obecná kolmá projekce (axonometrie)

- **směr pohledu** (normálový vektor průmětny): $\vec{\mathbf{N}}$
- **svislý vektor**: $\vec{\mathbf{u}}$
- převedení do základního pohledu: $\mathbf{Cs(S, \overrightarrow{u \times N}, \vec{u}, \vec{N})}$

Obecná kolmá projekce



Implementace kosoúhlé projekce



průmětna: xy
koeficient zkrácení: K
úhel průmětu osy z: α

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ K\cdot\cos\alpha & K\cdot\sin\alpha & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Implementace středové projekce

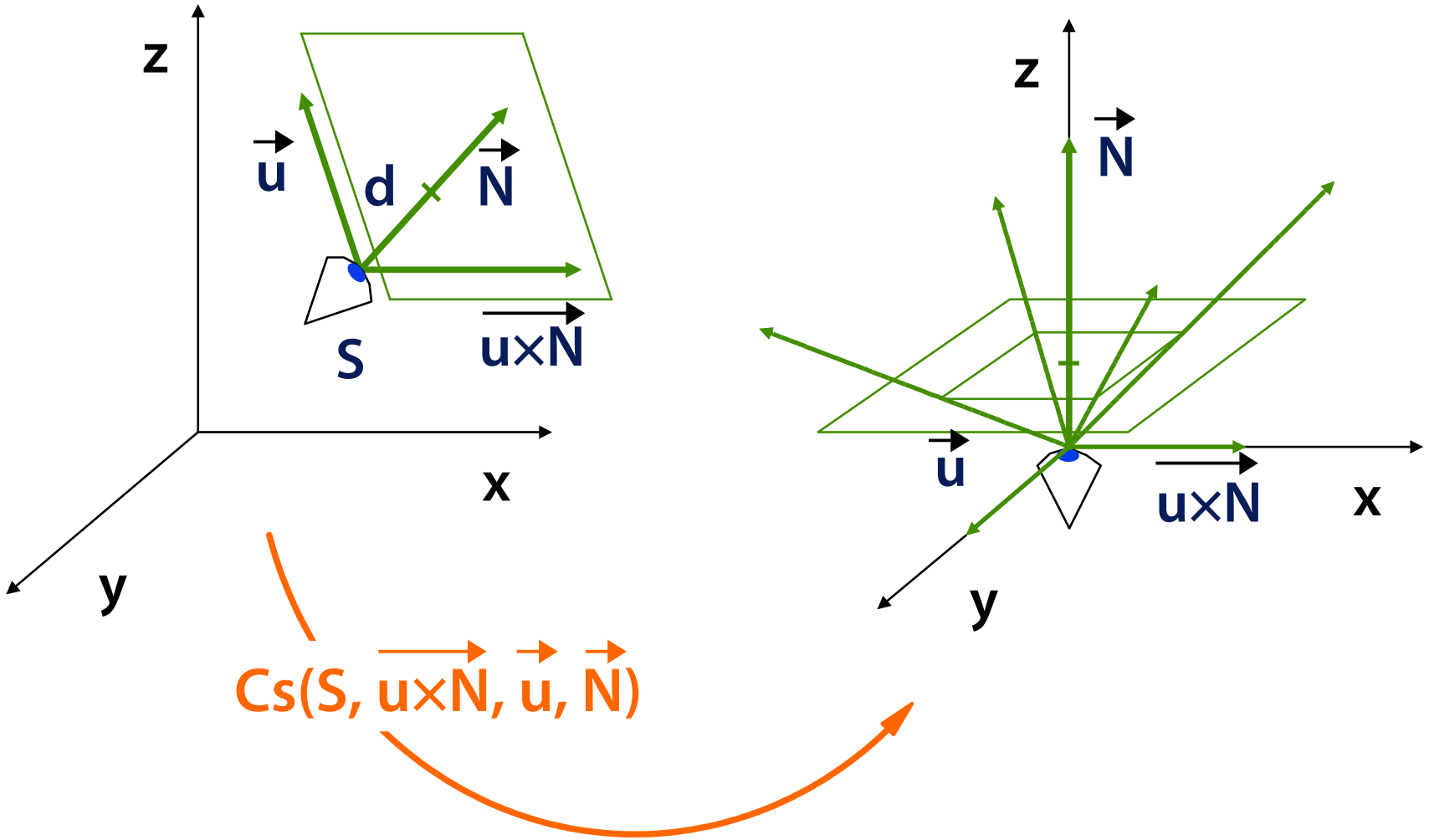
Obecná středová projekce

- střed projekce: **S**
- směr pohledu (normálový vektor průmětny): $\vec{\mathbf{N}}$
- vzdálenost průmětny od středu projekce: **d**
- svislý vektor: $\vec{\mathbf{u}}$

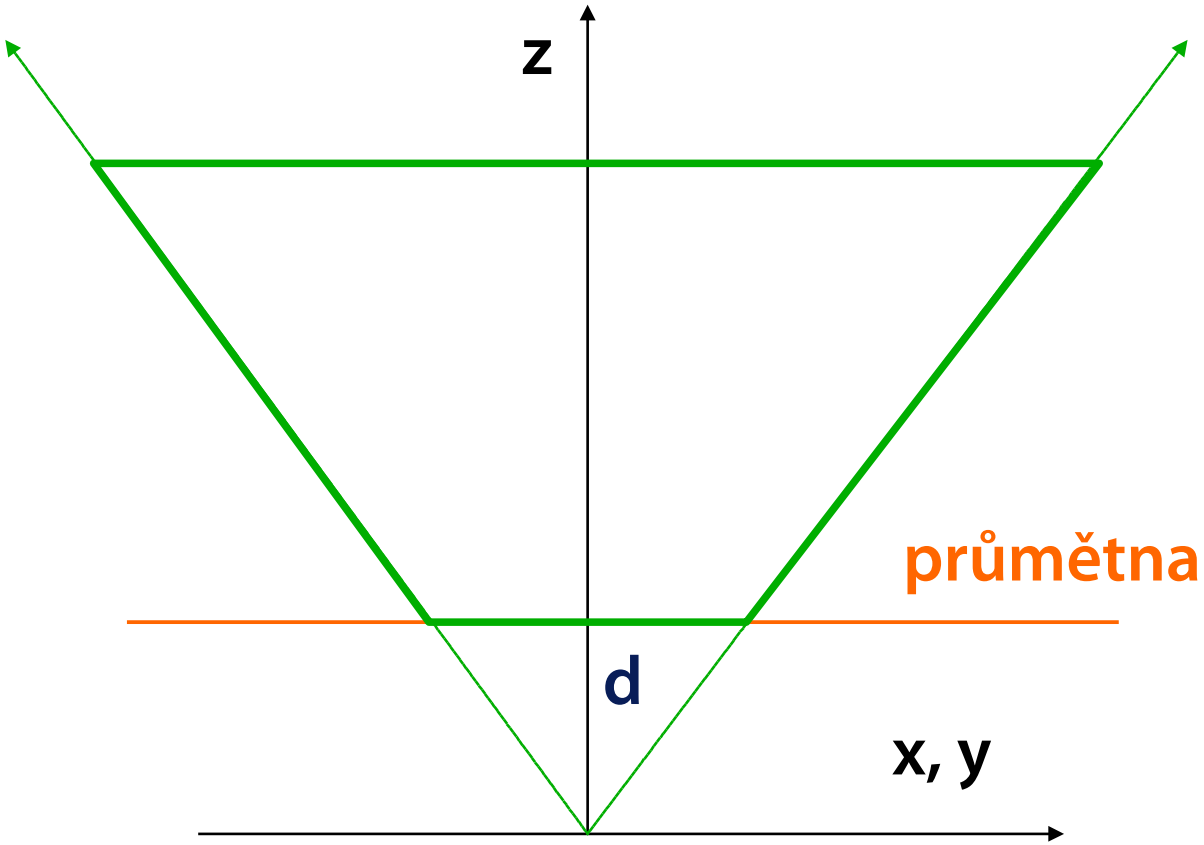
Promítací transformace

- převedení do **základní polohy** (střed projekce do počátku, směr pohledu do osy **z**): $\mathbf{Cs(S, \overrightarrow{u \times N}, \vec{u}, \vec{N})}$
- **perspektivní projekce**: např. [**x · d/z, y · d/z, z**]

Převedení do základní polohy



Perspektivní transformace



Nezachovává linearitu útvarů!

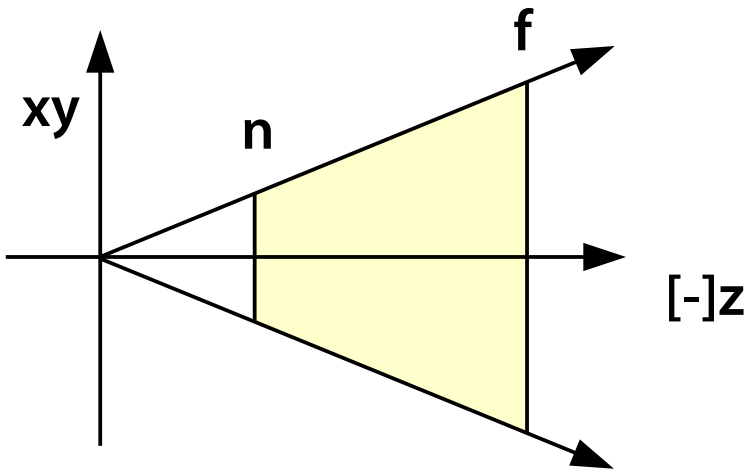
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/d \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Perspektivní projekce na GPU

Vzdálený bod **f** může být i v nekonečnu

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ -\frac{r+l}{r-l} & -\frac{t+b}{t-b} & \frac{f+n}{f-n} & 1 \\ 0 & 0 & -\frac{2fn}{f-n} & 0 \end{bmatrix}$$

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ -\frac{r+l}{r-l} & -\frac{t+b}{t-b} & 1 & 1 \\ 0 & 0 & -2n & 0 \end{bmatrix}$$



Transformace lineárních útvarů

Perspektivní transformace úsečky Per

– je zřejmé, že **neplatí** rovnost

$$\mathbf{Per(A + t \cdot [B - A]) = Per(A) + t \cdot [Per(B) - Per(A)]}$$

Použití **diferenčních algoritmů (DDA)** při výpočtu viditelnosti

– mějme bod **C(u)** na úsečce **Per(A)Per(B)**

$$\mathbf{C(u)_{x,y} = Per(A)_{x,y} + u \cdot [Per(B)_{x,y} - Per(A)_{x,y}]}$$

– potřebujeme, aby i pro hloubku **z** platilo

$$\mathbf{C(u)_{z} = Per(A)_{z} + u \cdot [Per(B)_{z} - Per(A)_{z}]}$$

Interpolation in the screen space

Concerns **clip-space** and next spaces

- clip space: [x, y, z, w]
- NDS: [x/w, y/w, z/w, w] ("w" could be preserved)
- window space (fragments): $[x_i, y_i, z_i, w]$

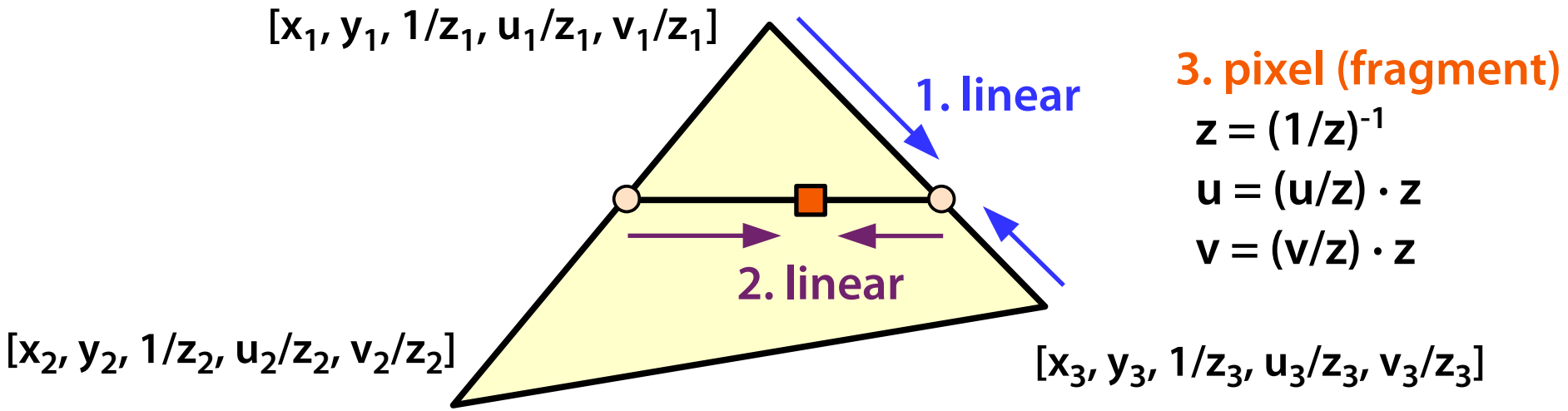
Projective perspective transformation maps depth **z** to NDS **non-linearly!**

– **nonuniform accuracy** of z-buffer (distant parts could be less accurate: minimization of **f/n** ratio!)

\+ **interpolation of depth 1/z** can be done in the screen space **linearly**

„W-buffer" instead of „z-buffer" (not frequently used)

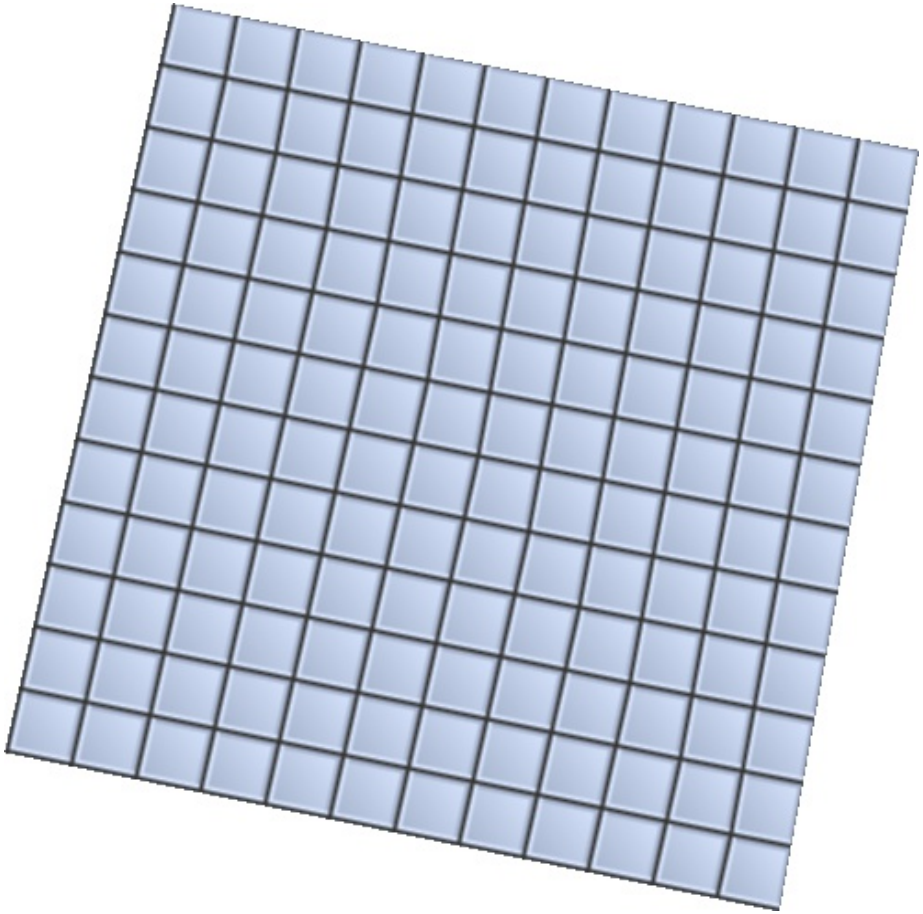
Perspective-correct interpolation



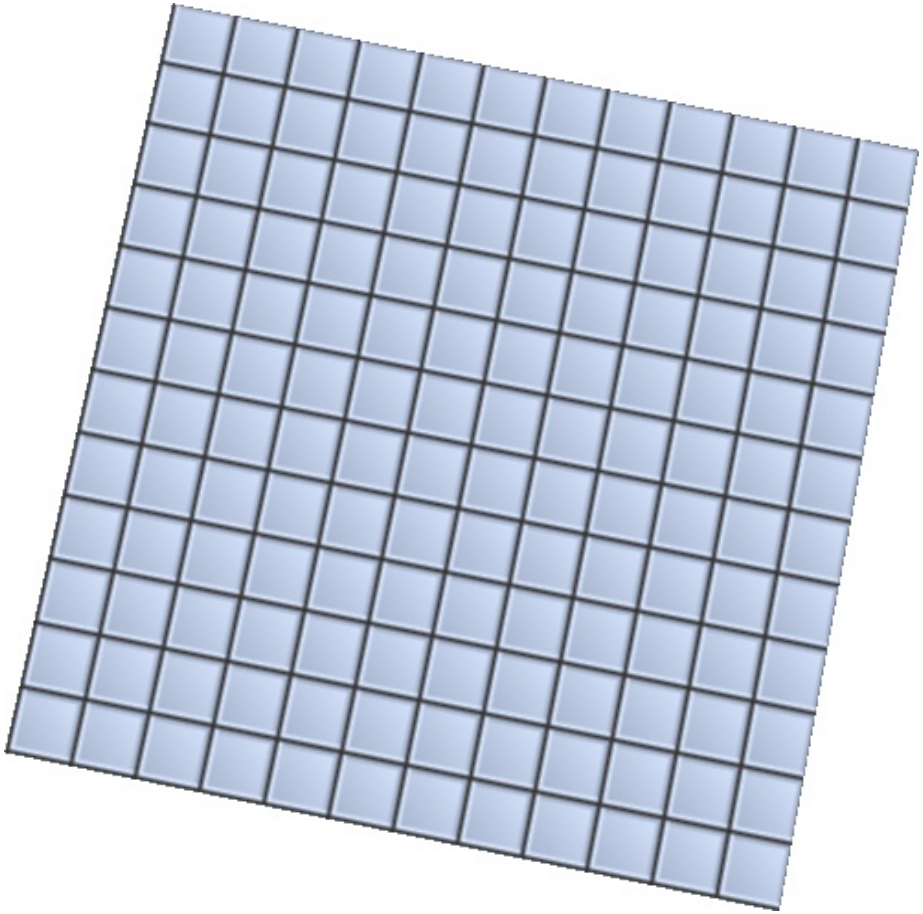
Linear interpolation + division (hyperbolic interpolation)

- **reciprocal depth "1/z"** can be interpolated linearly
- for additional attributes (**texture coordinates**) perspective-correct adjustment must be used
- quantities **"1/z"**, **"u/z"**, **"v/z"**… are linearly interpolated, **[u, v]** is then computed by division

Interpolation example I

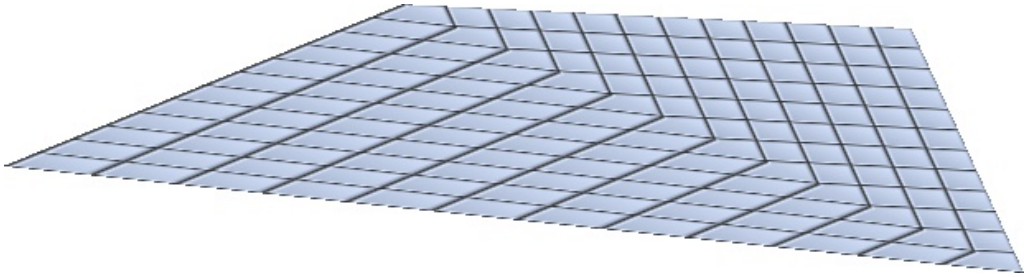


Affine

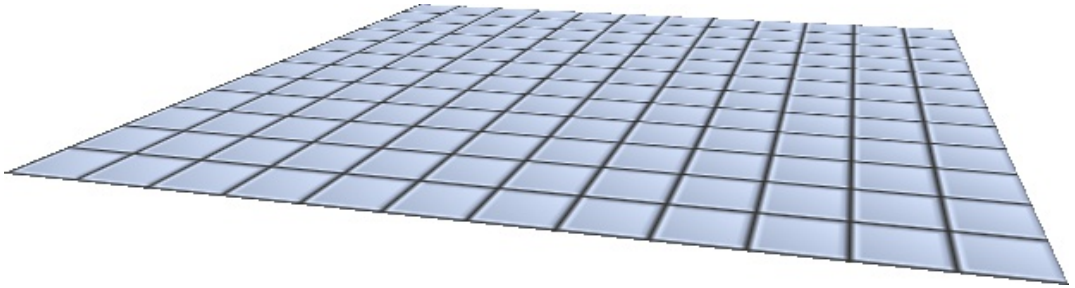


Correct

Interpolation example II

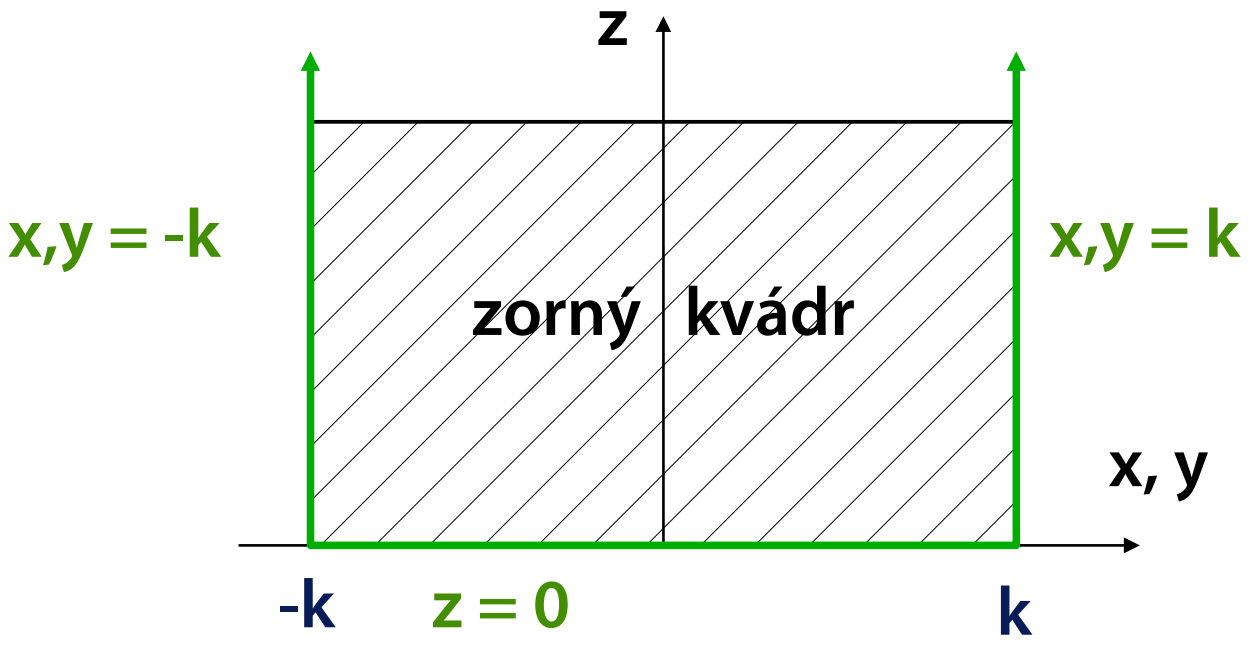


Affine



Correct

4D ořezávání



Hraniční nadroviny

$x = -kw,\ x = kw,\ y = -kw,\ y = kw,\ z = 0$

pro $w > 0$: $-kw < x < kw,\quad -kw < y < kw,\quad 0 < z$

Literatura

J. Foley, A. van Dam, S. Feiner, J. Hughes: ***Computer Graphics, Principles and Practice***, 229-283

Jiří Žára a kol.: ***Počítačová grafika, principy a algoritmy***, 277-291

Kok-Lim Low: ***Perspective-Correct Interpolation*** (report, proof), University of North Carolina at Chapel Hill, 2002

matfyz

Computer Graphics Charles University

Reprezentace 3D scén

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/

Metody reprezentace 3D scén

Objemové reprezentace

- přímé informace o vnitřních objemech těles
- snadný test **„bod×těleso”** (leží daný bod uvnitř tělesa?)
- **zobrazování** může být obtížnější
- používají se též jako **pomocné datové struktury** pro rychlé vyhledávání

Povrchové reprezentace

- přímé informace o povrchu těles (hrany, stěny)
- obtížnější test **„bod×těleso”** (tělesa vůbec nemusí mít vnitřní objem)
- poměrně snadné **zobrazování**

Objemové reprezentace

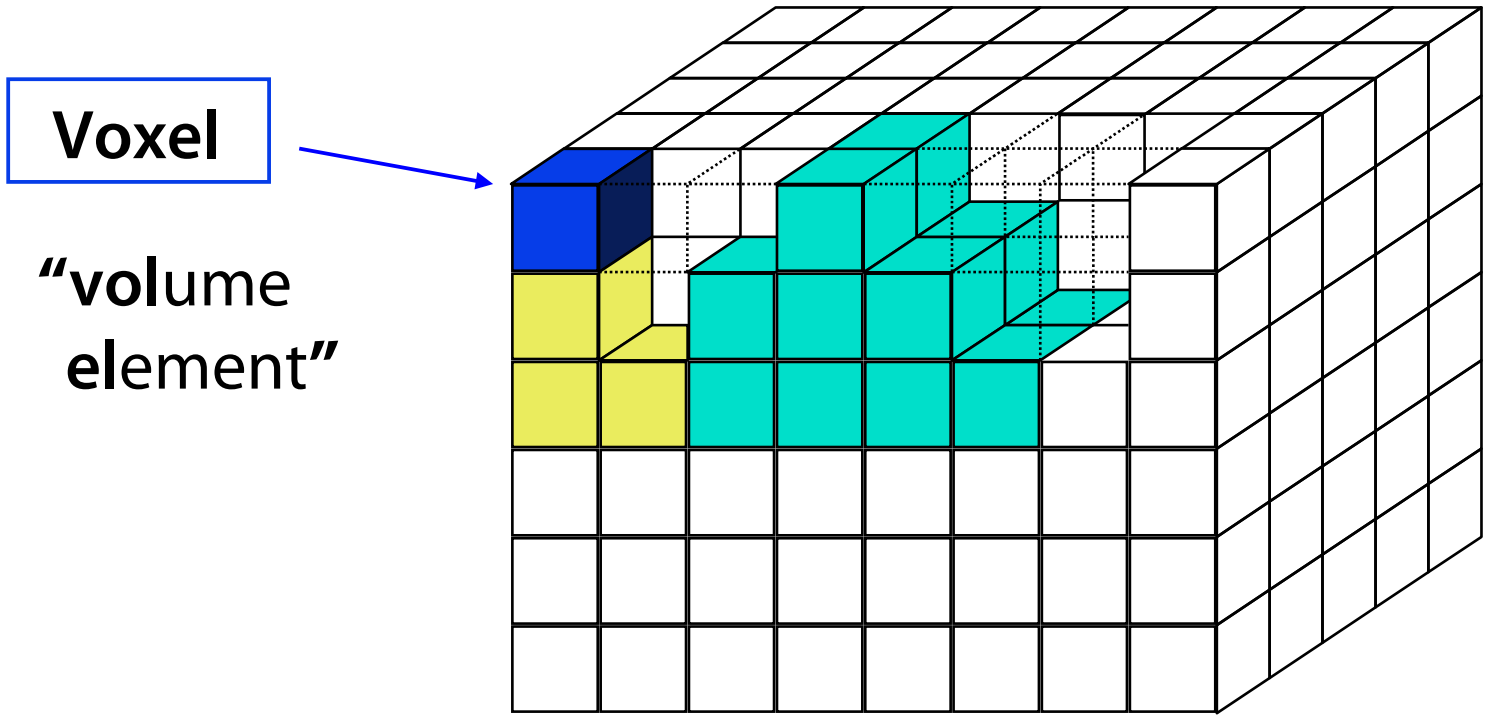
Výčtové reprezentace

- přímé vyčíslení obsazeného prostoru (diskrétní reprezentace – omezená přesnost)
- používají se hlavně jako pomocné datové struktury pro rychlé vyhledávání
- **buněčný model**, **oktantový strom**

CSG (Constructive Solid Geometry)

- velice silná a přesná metoda (elementární tělesa, geometrické transformace, **množinové operace**)
- obtížnější **zobrazování** (vrhání paprsku)

Buněčný model



Pole k × l × m voxelů

Jednobitová varianta: 0 – nic, 1 – těleso

Vícebitová varianta: 0 – nic, n > 0 – těleso číslo **n**

Zobrazování buněčného modelu

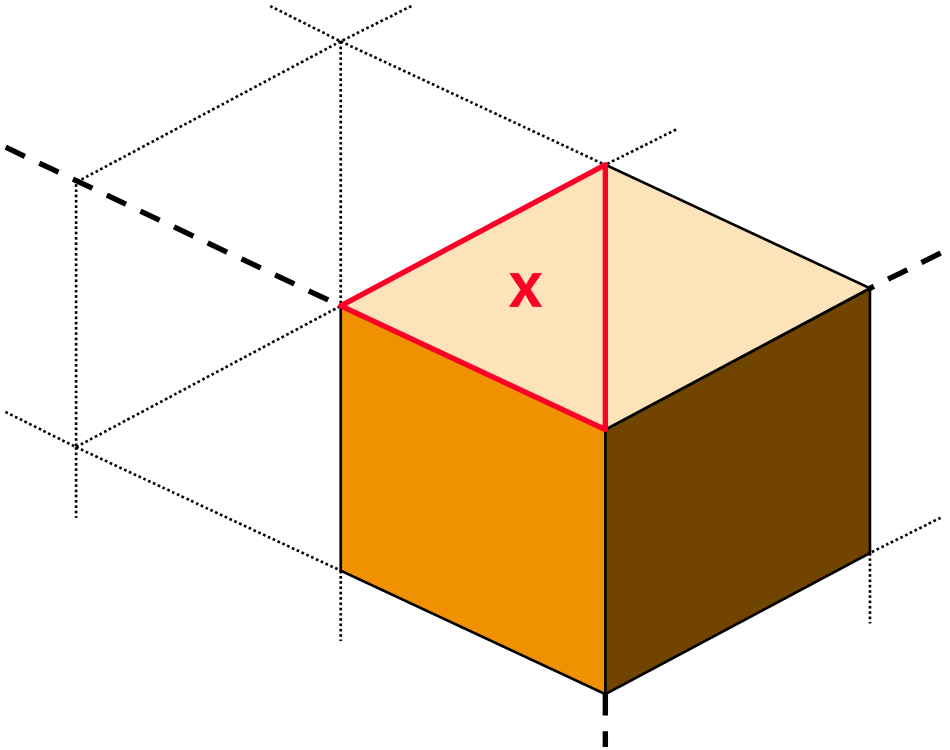
Kreslení odzadu-dopředu

- pouze přivrácené stěny voxelů
- pouze stěny na povrchu těles (stěny mezi **0** a **>0**)
- **vícenásobné překreslování**

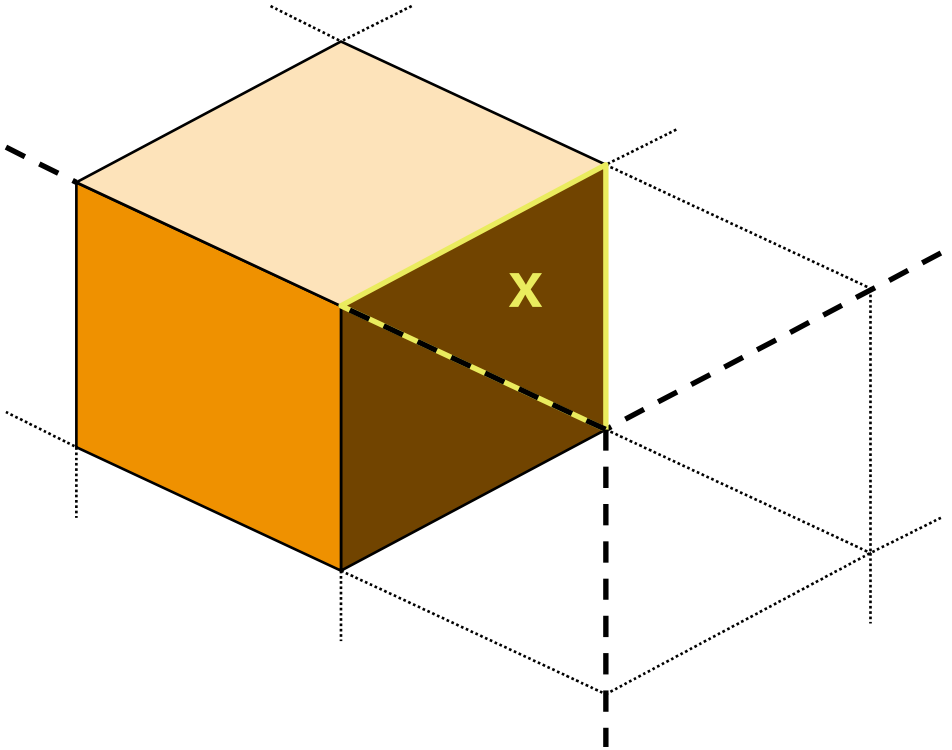
Speciální promítání

- velmi efektivní algoritmus bez zbytečného překreslování
- **„Ant-attack"** na ZX-Spectru (128×128×8 voxelů)

Speciální promítání

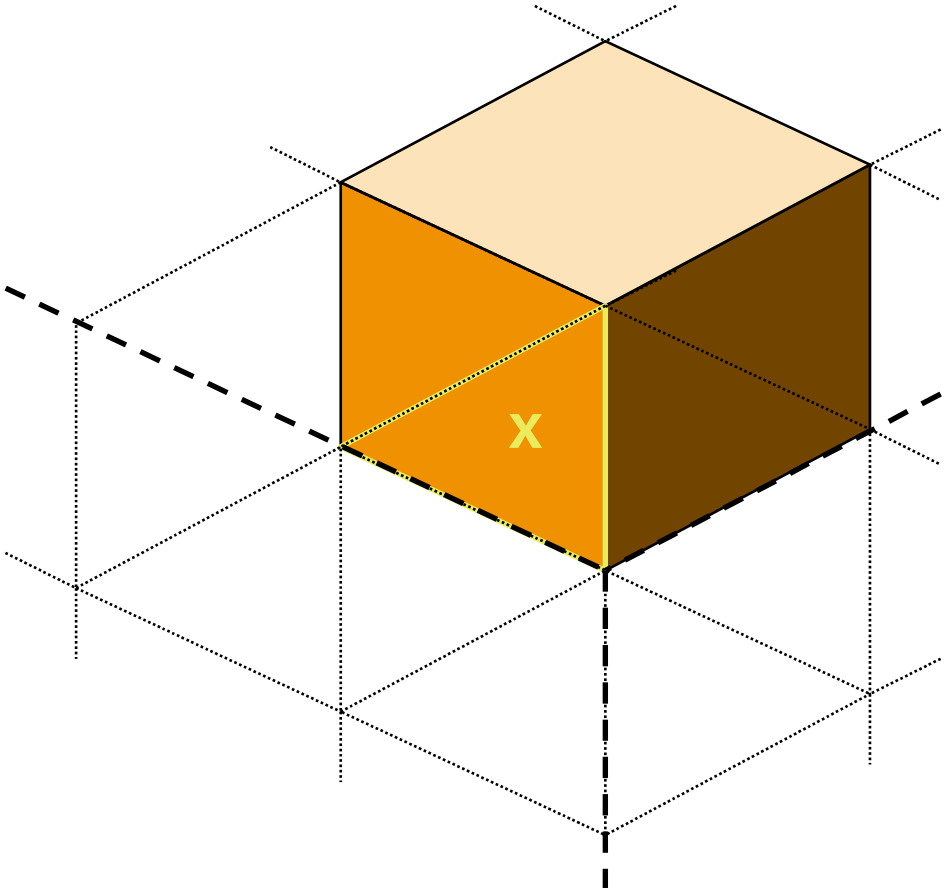


1. horní stěna
[0,0,0]

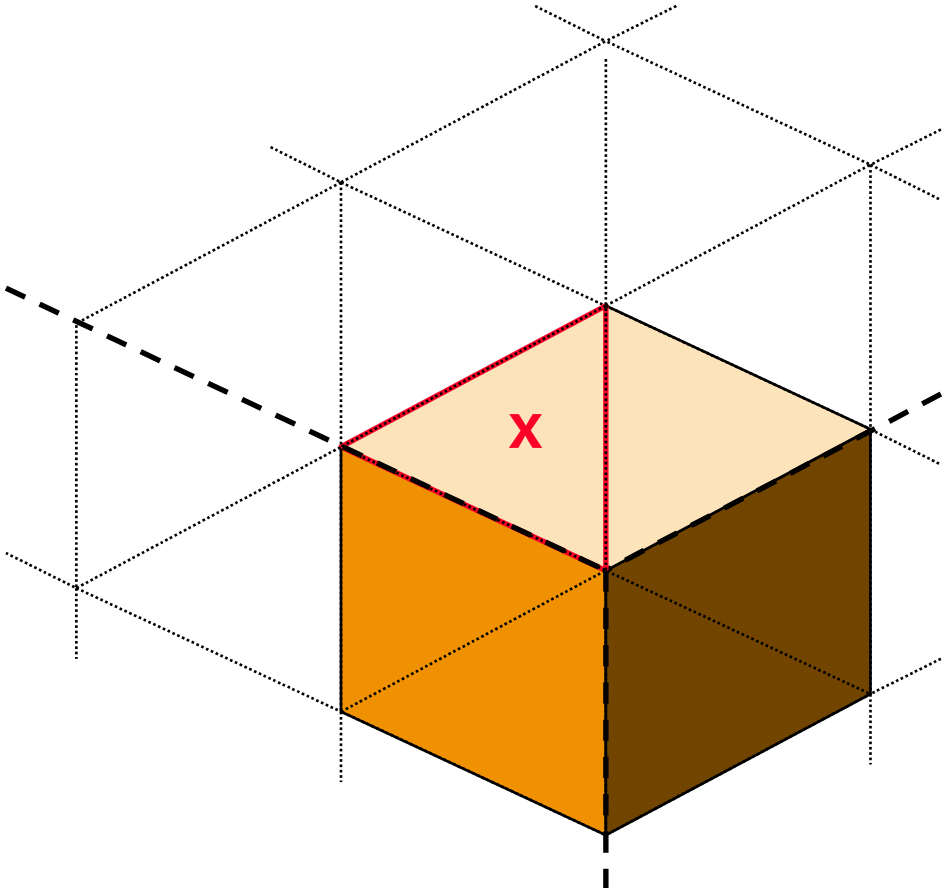


2. pravá stěna
[0,1,0]

Speciální promítání II

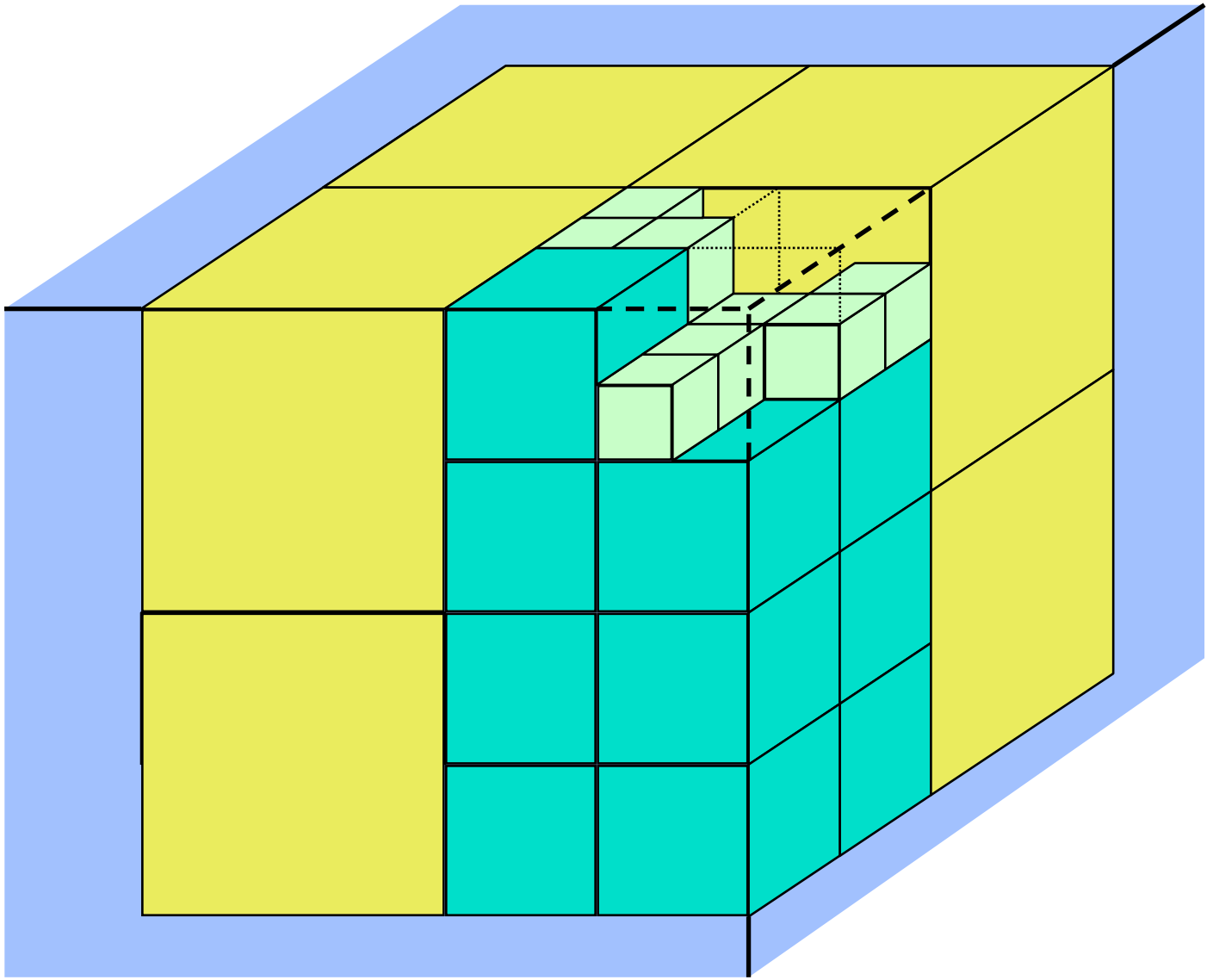


3. levá stěna
[1,1,0]



4. horní stěna
[1,1,1]

Oktantový strom („Octree”)



Oktantový strom

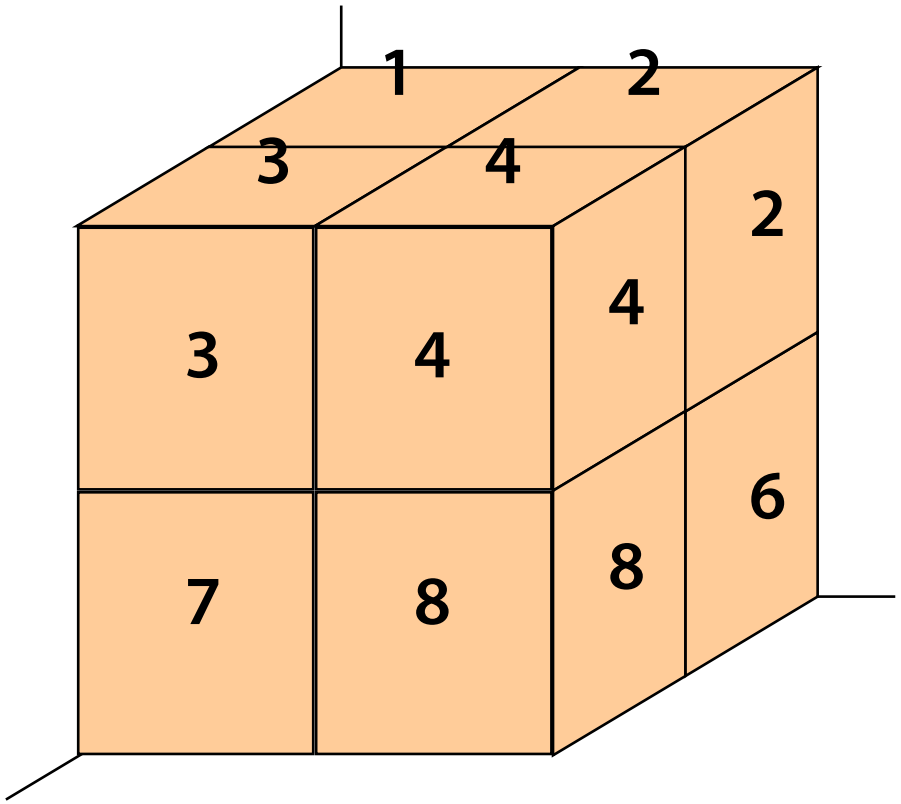
3D analogie kvadrantového stromu

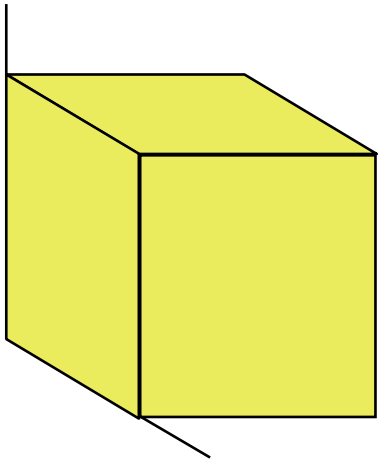
- je-li vnitřek krychle nehomogenní, rozdělí se na osm částí (dělí se až do úrovně voxelu)
- úspora paměti proti buněčnému modelu

Kreslení odzadu-dopředu

- pouze přivrácené stěny krychlí
- pouze stěny na povrchu těles (stěny mezi **0** a **>0**)
- několikanásobné překreslování některých pixelů

Kreslení odzadu-dopředu





Pořadí

5-6-1-2-7-8-3-4

Pořadí

6-5-2-1-8-7-4-3

CSG (Constructive Solid Geometry)

Elementární geometrická tělesa

- snadno definovatelná a vyčíslitelná
- kvádr, poloprostor, hranol, koule, válec, kužel…

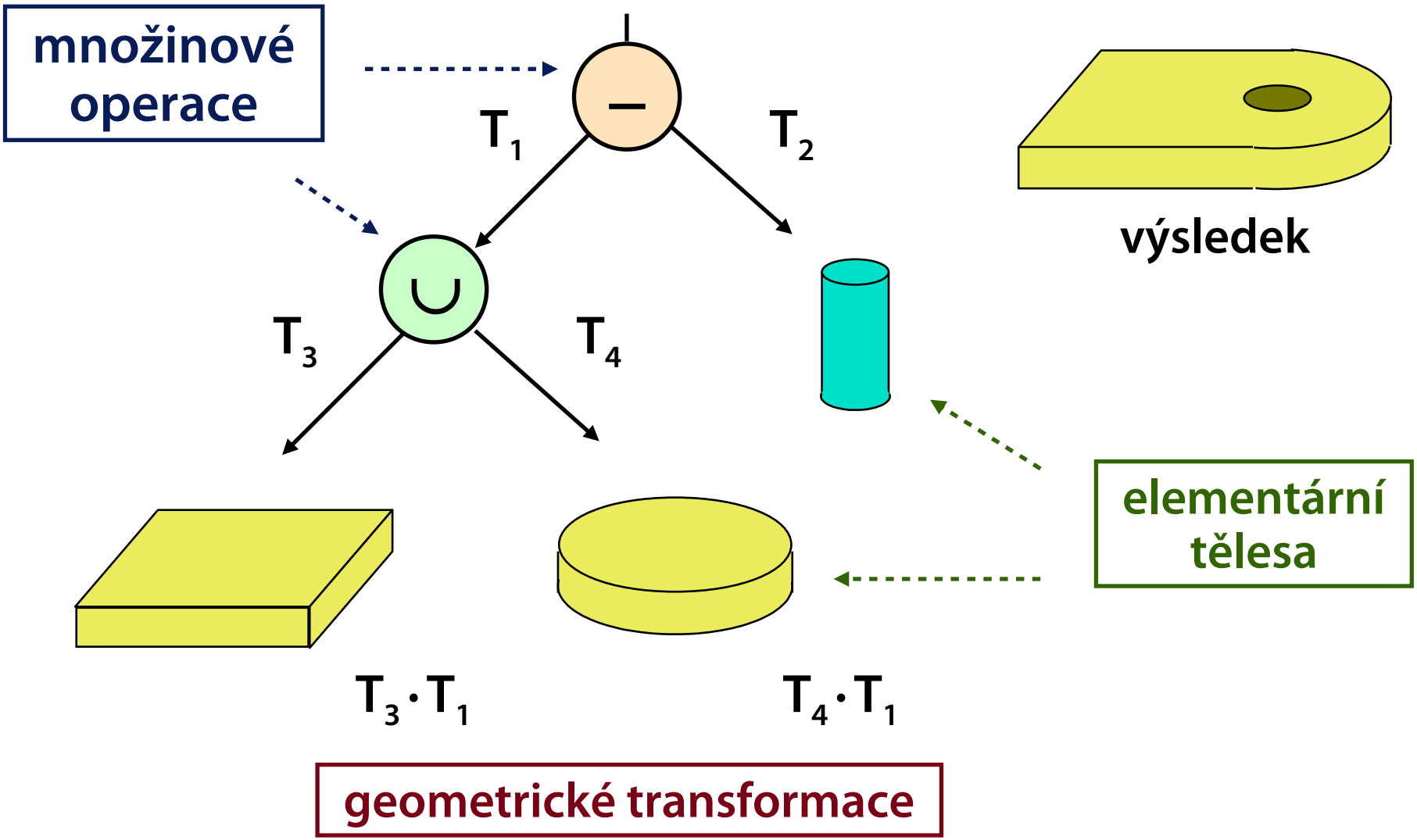
Množinové operace

- kompozice složitějších těles z elementárních
- sjednocení, průnik, rozdíl…

Geometrické transformace

- modifikace elementárních i složitějších těles
- homogenní maticové transformace

CSG strom


množinové operace
T1
T2
T3
T4
výsledek
elementární tělesa
T3 · T1
T4 · T1
geometrické transformace

Transformace v CSG stromu

Význam (sémantika) transformace $\mathbf{T}_i$

- $\mathbf{T}_i$ mohou být uloženy v každé hraně CSG stromu
- převod souřadnic ze soustavy podtělesa (podstromu, elementárního tělesa) do soustavy nadtělesa
- „podtěleso transformuji pomocí $\mathbf{T}_i$ před tím, než ho přidám do nadtělesa"

Snadná transformace libovolného podstromu

- změním pouze jednu matici

Inverzní transformace $\mathbf{T}_i^{-1}$

- pro vyčíslovací algoritmy (test „bod×CSG", zobrazení)

Transformace v CSG stromu

Uložení transformací jen v listech

- **kumulované součiny** (např. $\mathbf{T_3 \cdot T_2 \cdot T_1}$ nebo inverzní $\mathbf{T_1^{-1} \cdot T_2^{-1} \cdot T_3^{-1}}$)
- urychlení vyčíslovacích algoritmů (pro editaci je výhodnější distribuované uložení transformací)

Úsporné uložení elementárních těles

- tělesa jsou uložena v **normovaném tvaru**, všechny změny se provádí geometrickými transformacemi
  - » krychle (jednotková, vrchol v počátku)
  - » koule (jednotková, střed v počátku)
  - » válec (vodorovná podstava – jednotkový kruh, svislá osa, výška 1)
  - » …

Test „bod × CSG strom”

Leží daný bod A uvnitř tělesa?

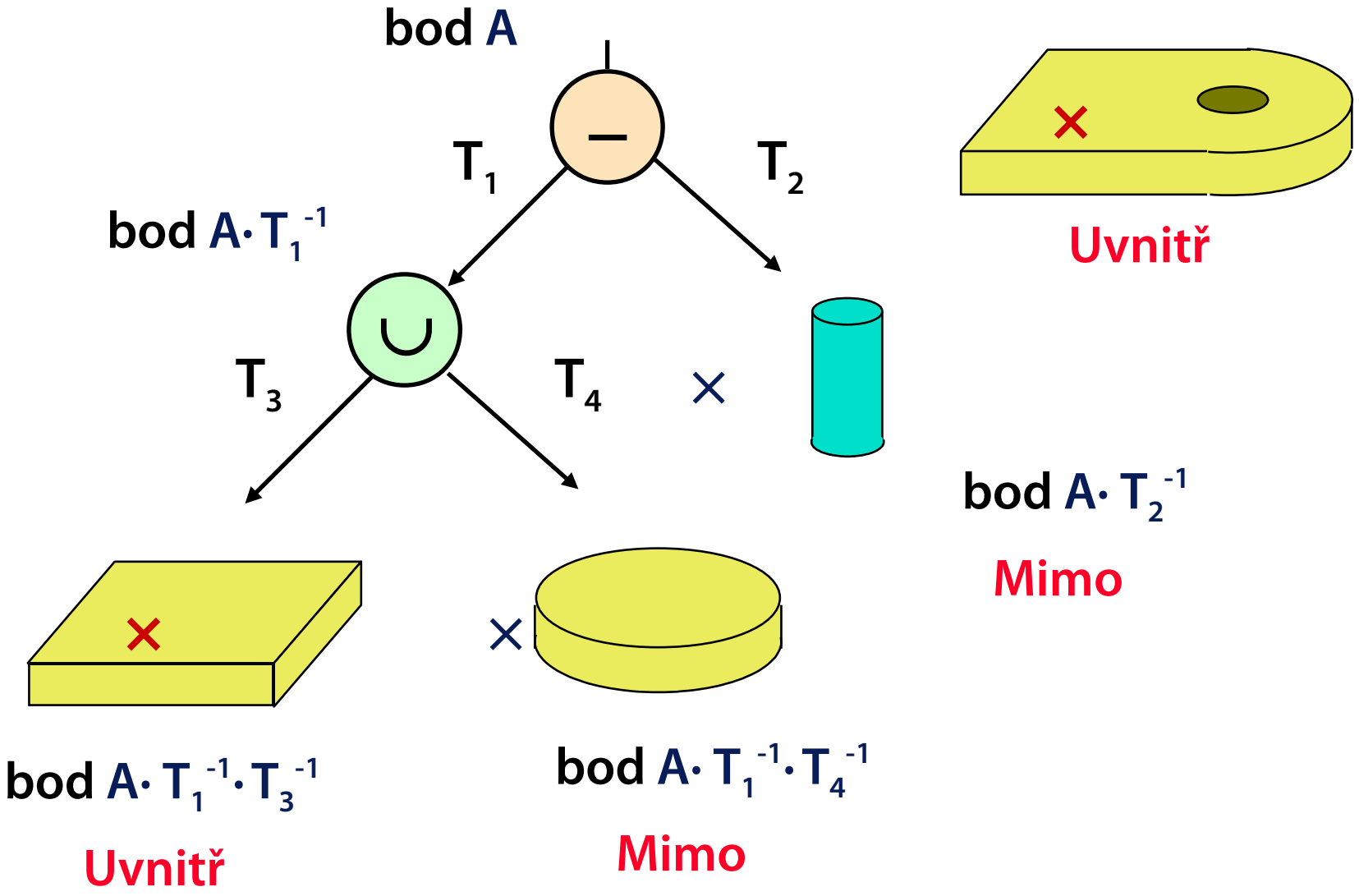
- někdy chceme zjistit i podtělesa obsahující bod **A**

Testy **„bod × elementární těleso”** jsou snadné!
(především pro normované tvary těles)

Průchod CSG stromem

- souřadnice bodu **A** se převádějí do souřadných soustav elementárních těles (inverzní transformace)
- místo množinových operací se provádějí jejich **boolovské ekvivalenty** ($\vee$ místo $\cup$, $\wedge$ místo $\cap$ …)

Test „bod × CSG strom”



Zobrazování CSG reprezentace

Převedení do povrchové reprezentace

- pro každý druh **elementárního tělesa**: rutina převádějící těleso na **mnohostěn**
- **množinové operace nad mnohostěny** (omezená přesnost – výsledek nemusí být správně ani v topologickém smyslu)

Vrhání paprsku („Ray-casting")

- přesné zobrazování v **rastrovém prostředí** (pixelová přesnost)
- výpočetně náročnější metoda

Povrchové reprezentace

„Drátový model”

- pseudo-povrchová reprezentace
- pouze **vrcholy** a **hrany** těles (nelze použít pro výpočet viditelnosti)

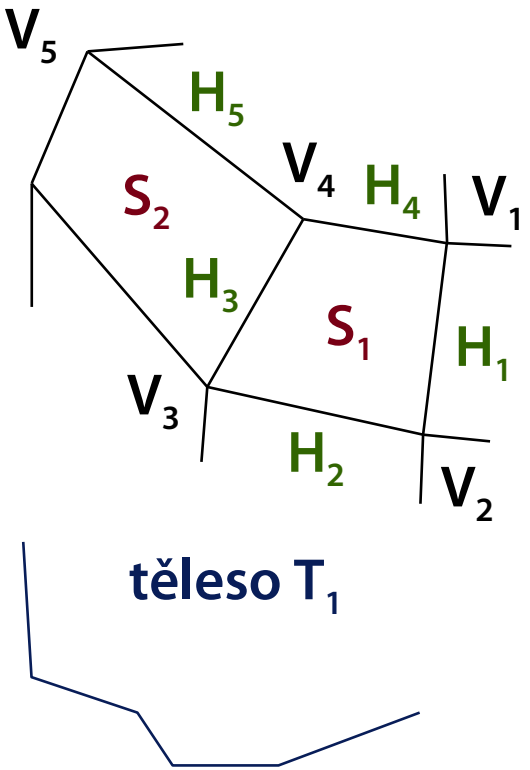
VHS(T) reprezentace

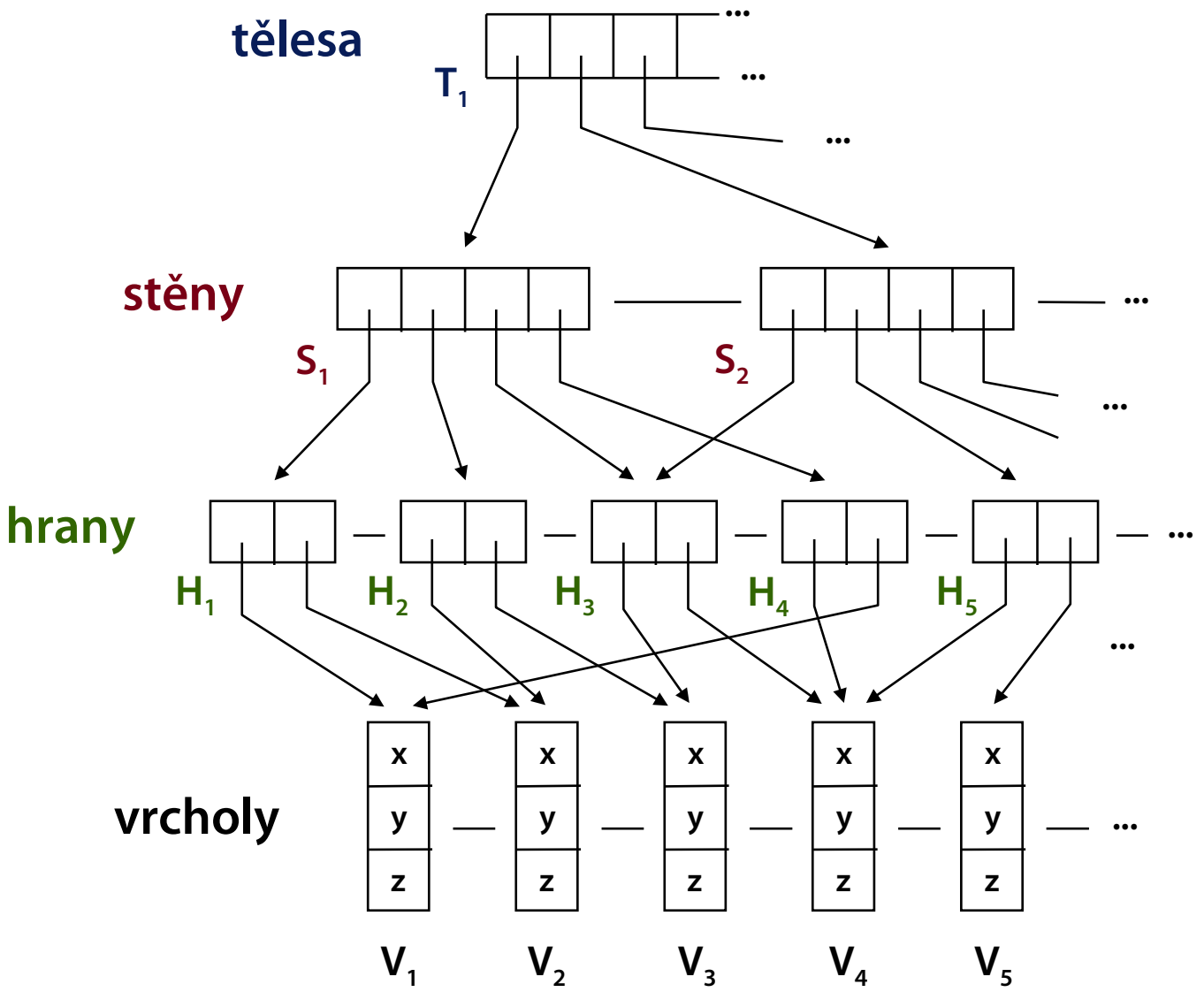
- kompletní topologická informace: seznamy **vrcholů**, **hran**, **stěn** (a **těles**)

„Okřídlená hrana” („winged-edge”)

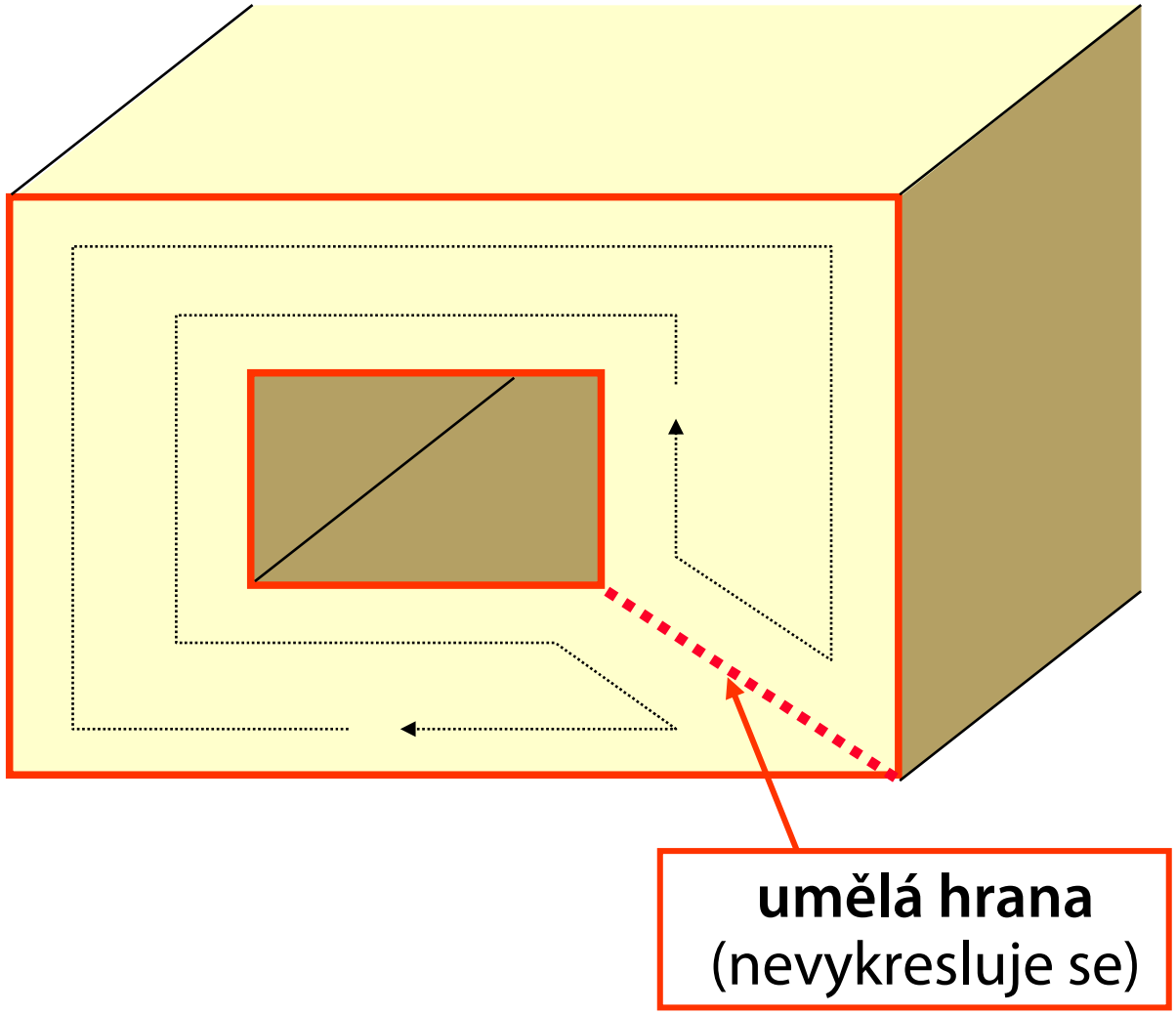
- redundantní informace pro **rychlé vyhledávání** sousedních objektů (hrany incidentní s vrcholem…)

Povrchová reprezentace VHST



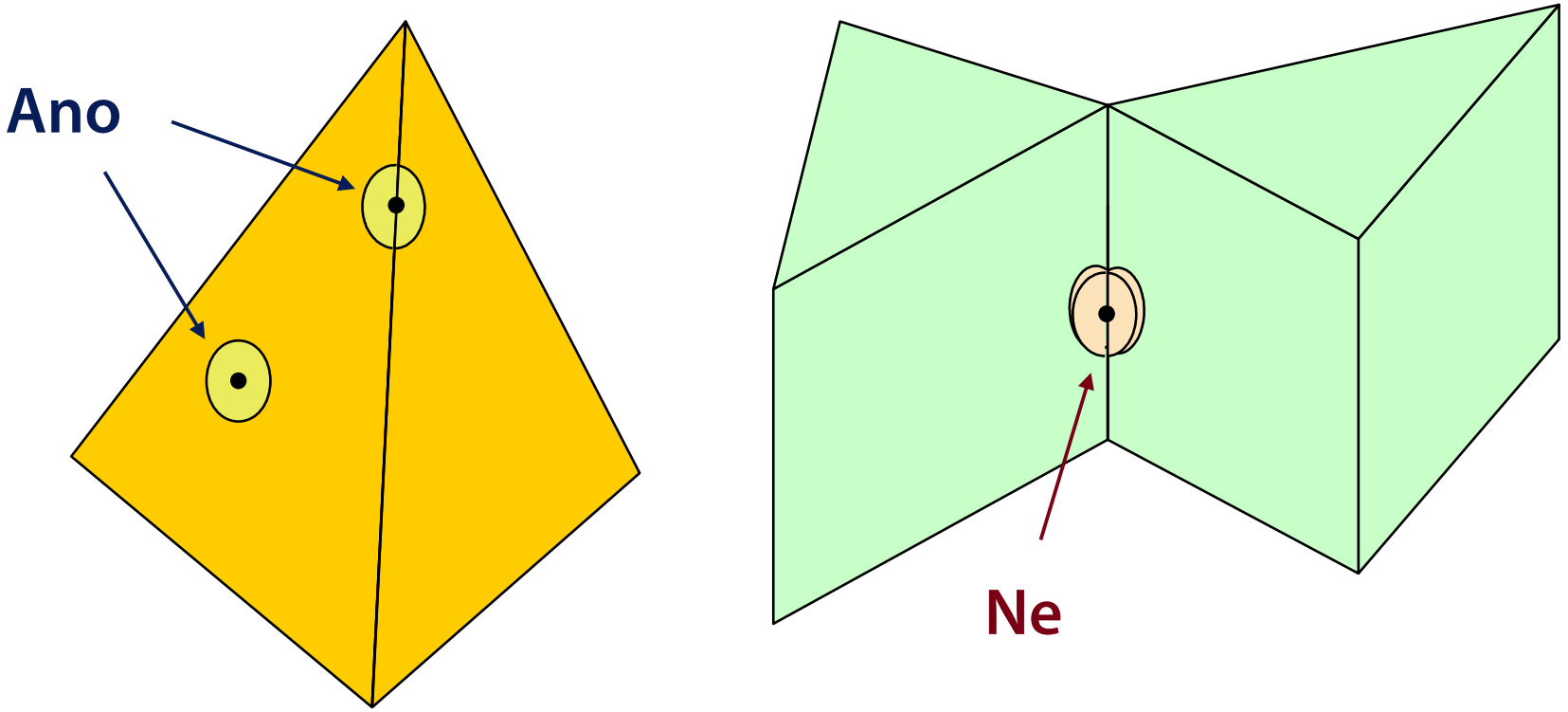


„Děravá" stěna

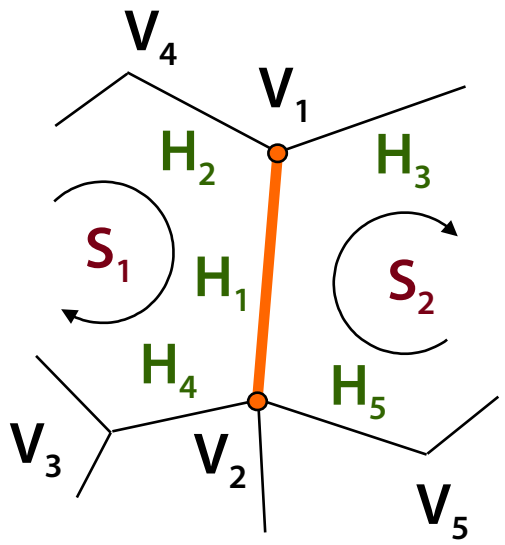


„2-manifold” (manifold, varieta dim=2)

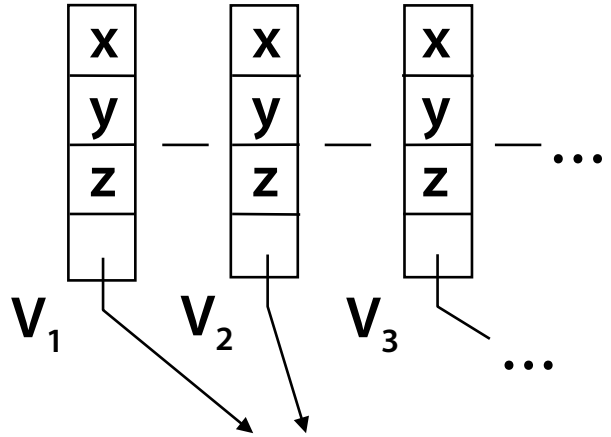
Def: Pro každý povrchový bod existuje okolí, které je topologicky ekvivalentní s rovinou



Okřídlená hrana („winged-edge”)



těleso T_1

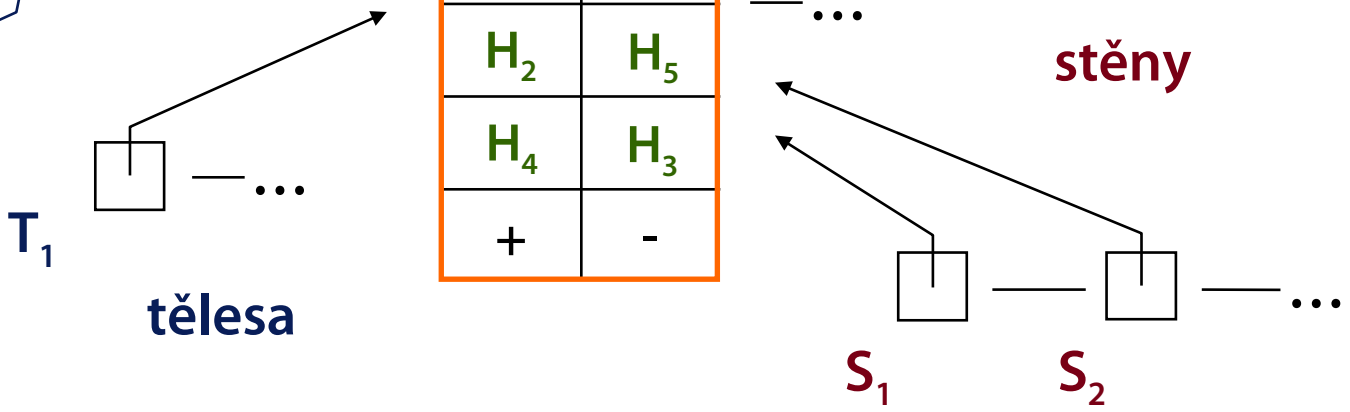


hrany

H_1

V_1	V_2
S_1	S_2
H_2	H_5
H_4	H_3
+	-

stěny



Další informace v databázi

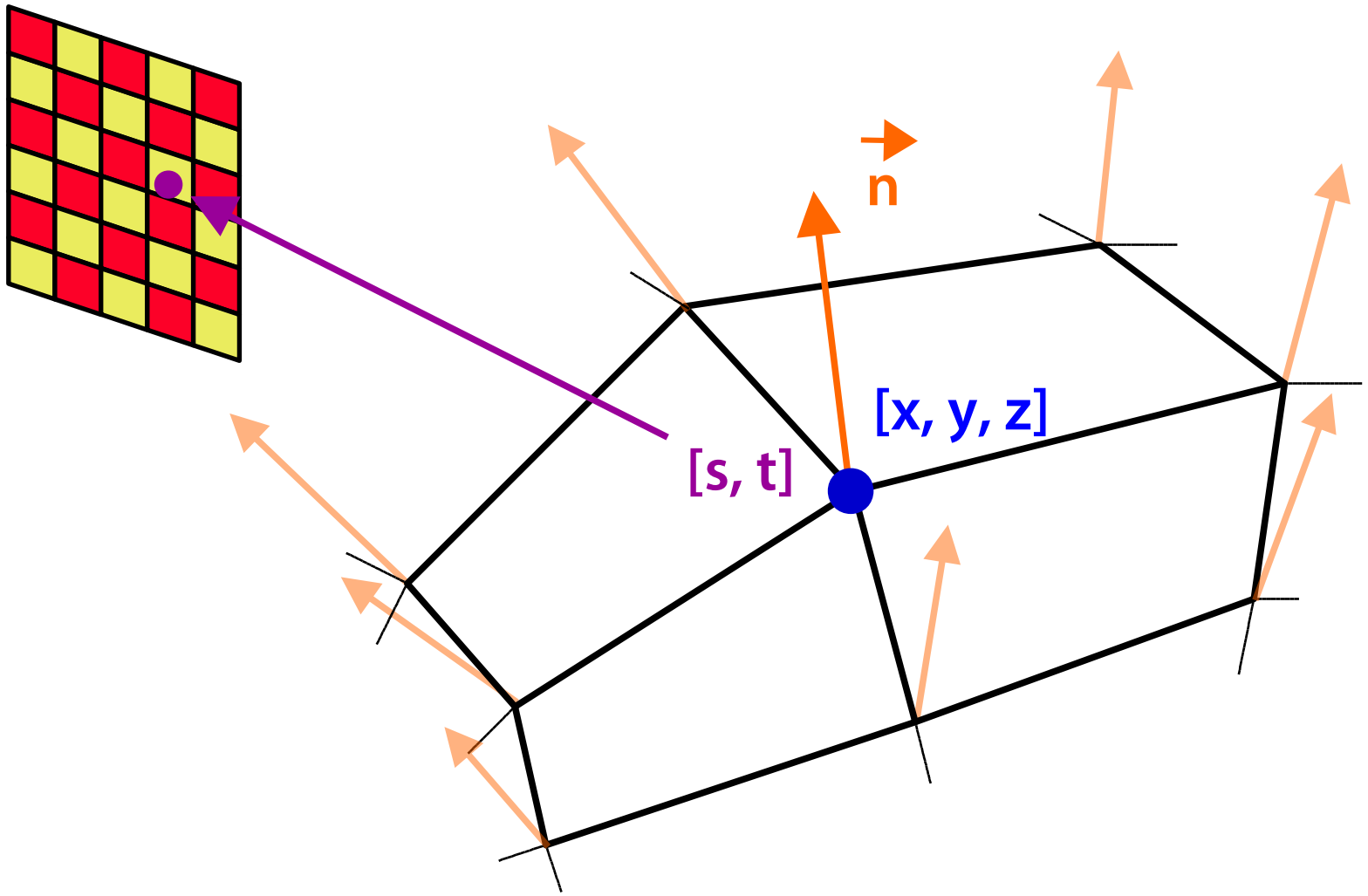
Vrchol (vertex, V)
- barva, texturové souřadnice
- normálový vektor (stínování hladké plochy)

Hrana (edge, E)
- příznak umělé hrany (pro reprezentaci děravých stěn)

Stěna (face, F)
- barva, materiál, textura
- normálový vektor (stínování, přivrácená/odvrácená)

Těleso (solid, S)
- barva, materiál, textura

Data ve vrcholu



Data ve vrcholu

Souřadnice

- [**x, y, z**] nebo [**x, y, z, w**]

Normálový vektor

- pro stínování (později), může být interpolován

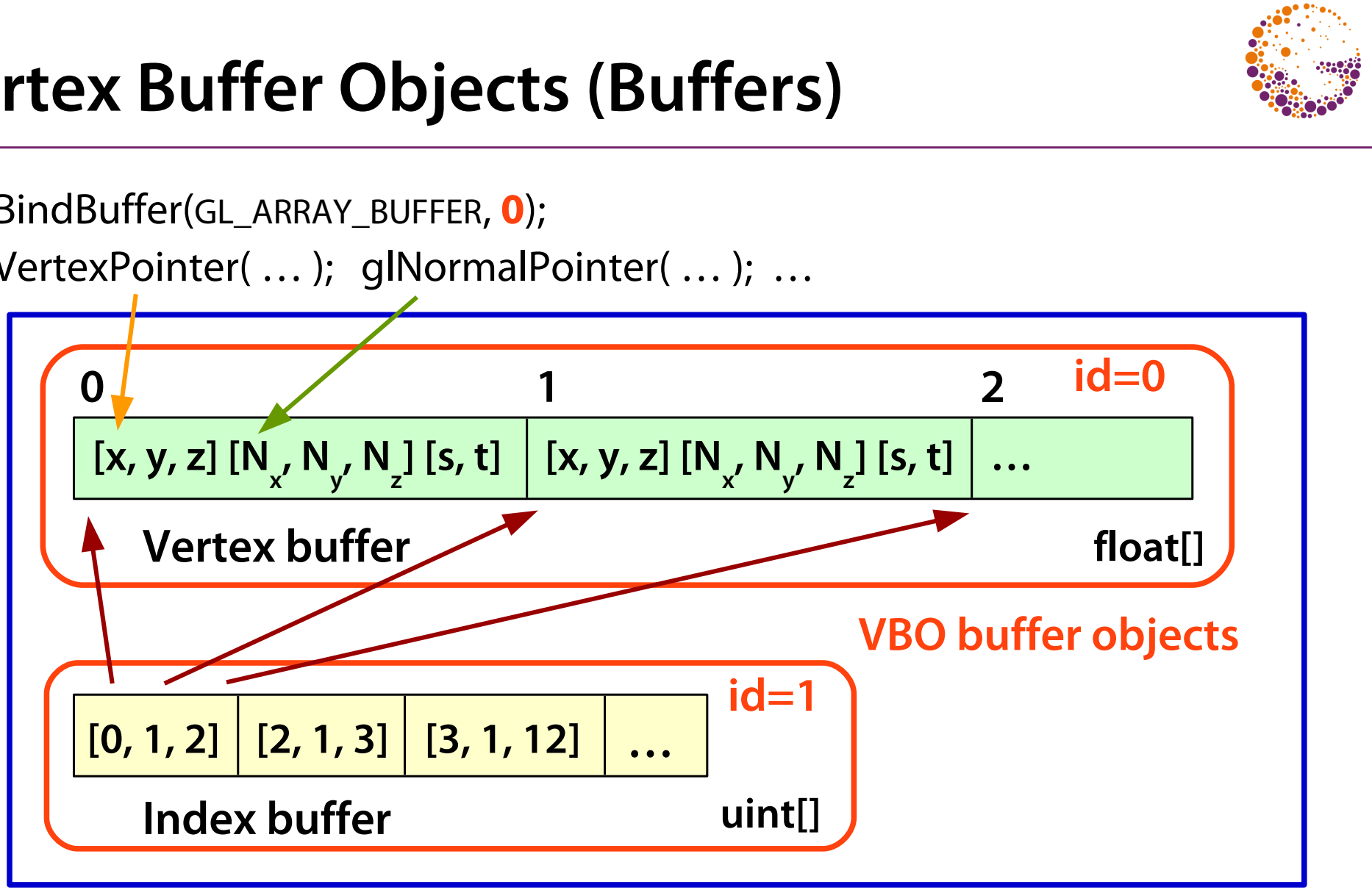
Barva

- explicitní barva povrchu, může být interpolována

Texturové souřadnice

- nejčastěji 2D souřadnice – [**s, t**] nebo [**u, v**]
  - » normalizovaný prostor textury $[\mathbf{0, 1}]^2$
- automaticky se při kreslení interpolují do pixelů

Vertex Buffer Objects (Buffers)



„Corner Table“ (trojúhelníky)

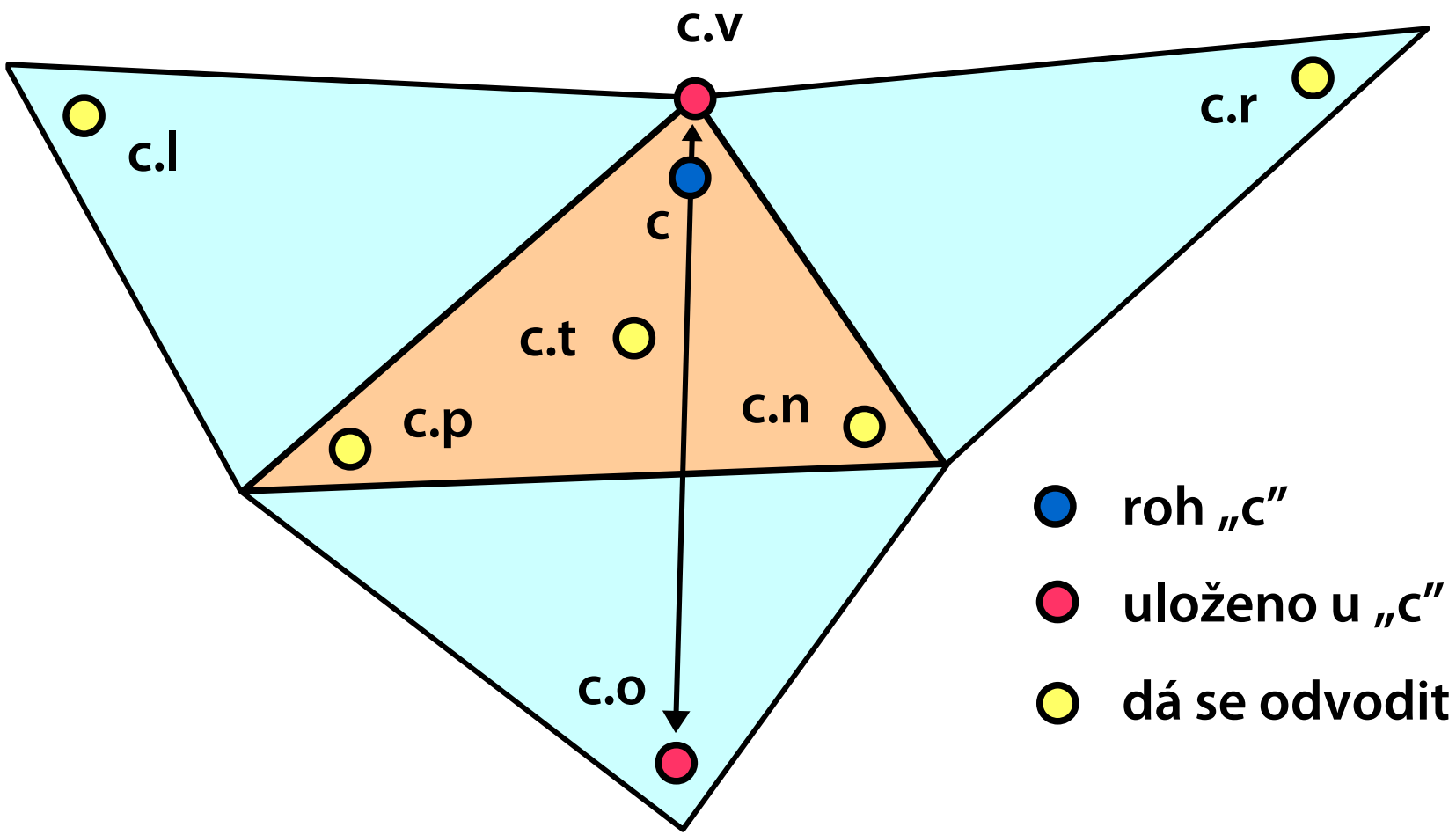
Tabulka vrcholů G[v]

- souřadnice, normála, barva, texturové souřadnice…

Tabulka rohů V[c]

- jeden vnitřní roh trojúhelníka
- index vrcholu (**„c.v“**)
- rohy jsou uloženy za sebou v 1D poli, CW orientace stěn
- protější roh protějšího trojúhelníka (**„c.o“**)
- implicitní údaje, navigace
  - » číslo trojúhelníka t = c div 3
  - » ostatní rohy c.n = (c mod 3 == 2) ? c-2 : c+1, c.p = c.n.n
  - » další sousední trojúhelníky c.l = c.n.o, c.r = c.p.o

„Corner Table“



„Corner Table“ – kódování

Tabulka **„o“** se nemusí přenášet

– lze ji jednoznačně rekonstruovat z „v“

Hodnoty indexů „v“ se mohou bitově ořezat

– index vrcholu obsahuje **$31 - \log_2 v$** úvodních nul

Ořezání souřadnic

– obalový kvádr celého objektu, uvnitř se používají relativní souřadnice (10 až 16 bitů na složku)

– predikce polohy vrcholů – při inkrementálním průchodu daty (např. „Edgebreaker“) – další úspory

„Edgebreaker“ (Rossignac 1999)

Úsporné kódování tri-mesh pomocí inkrementálního průchodu sítí

- **pozice vrcholů** se komprimují za pomoci predikce (až 7 bpv)
- **topologie sítě** se ukládá velice úsporně na základě průchodu (1.0 až 1.8 bpv)

„CLERS“ kód

- **pět možností**, jak pokračovat z aktuálního trojúhelníka
- velký potenciál entropické komprese (některé kroky/ posloupnosti jsou mnohem pravděpodobnější)

Eulerovy zákony

Pro **jednoduchý polyedr** (bez děr) platí vzorec

$$V - H + S = 2$$

V – počet vrcholů, **H** – počet hran, **S** – počet stěn

Zobecněný vzorec (dovoluje díry)

$$V - H + S - D = 2\,(T - G)$$

D – počet děr ve stěnách, **T** – počet těles, **G** – počet děr procházejících celým tělesem (Genus)

Eulerovy operátory

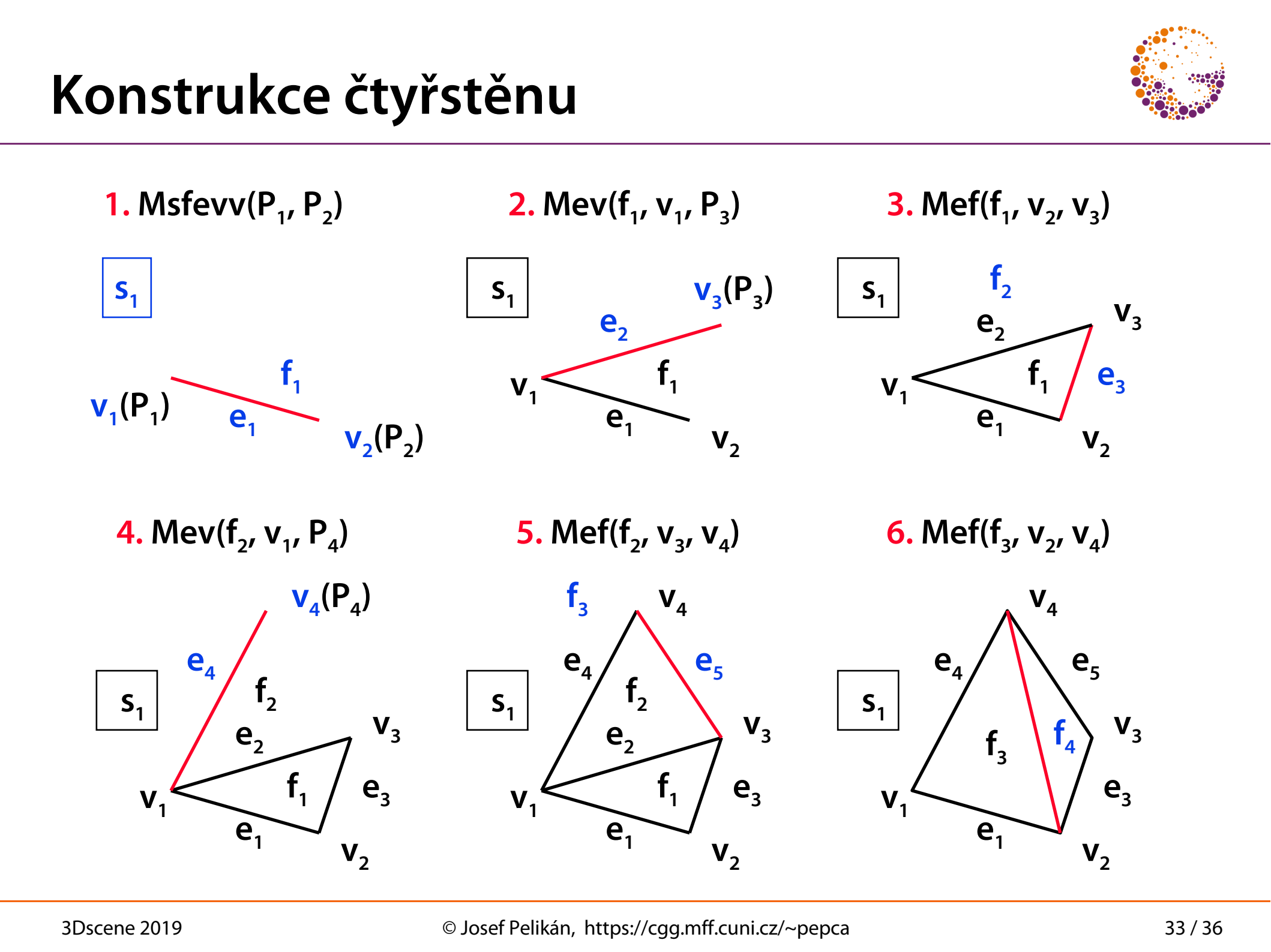
Konstrukce 2-manifoldu po krocích

- v každém kroku je zajištěna platnost Eulerových vzorců (těleso je topologicky korektní)
- ke každému operátoru existuje inverzní (snadná implementace příkazu „Undo")

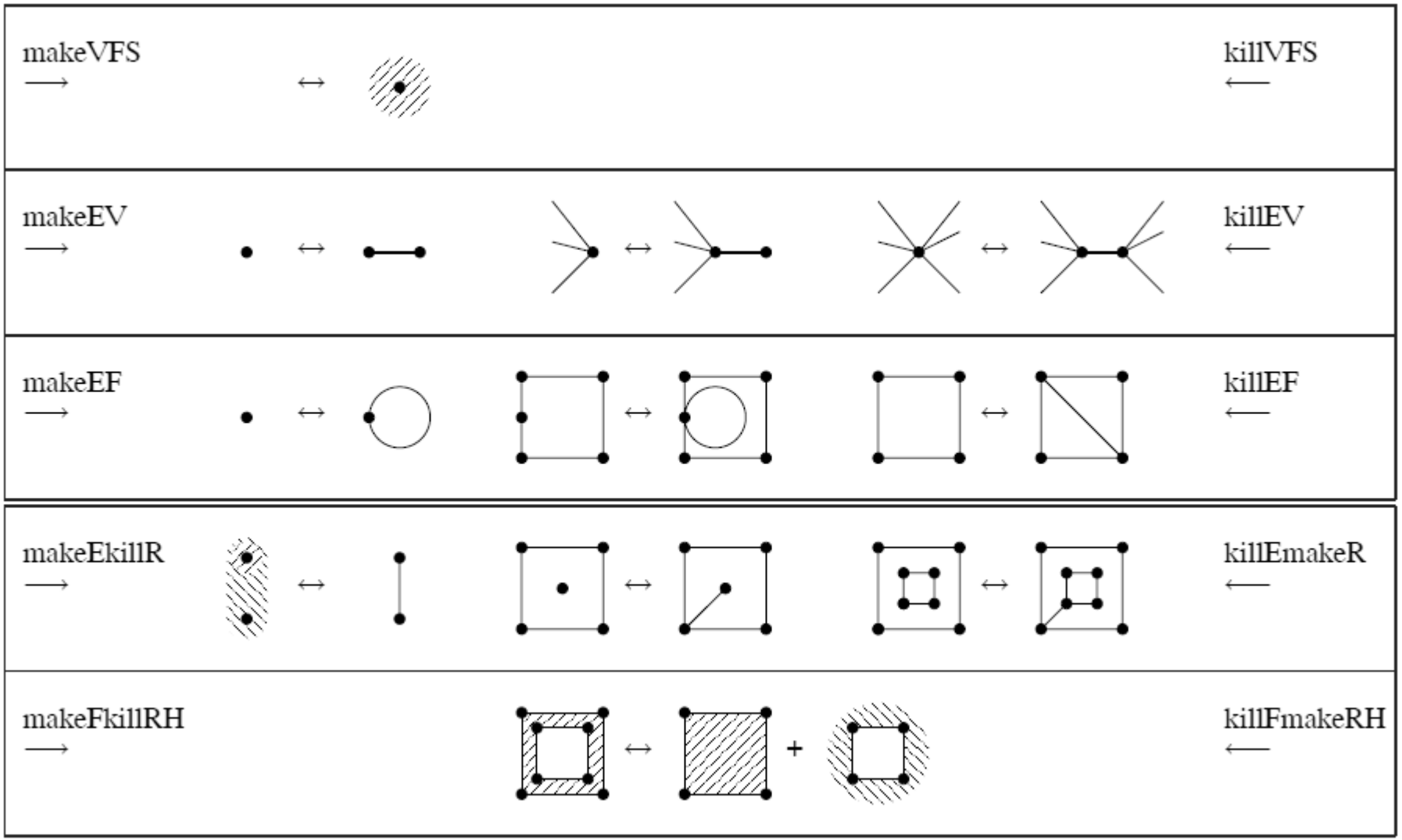
Příklady **Eulerových operátorů**

- **Msfevv(P_1, P_2)** „make solid, face, edge, vertex, vertex"
- **Mev(f_1, v_1, P_2)** „make edge, vertex"
- **Mef(f_1, v_1, v_2)** „make edge, face"
- **Kef(e)** „kill edge, face"

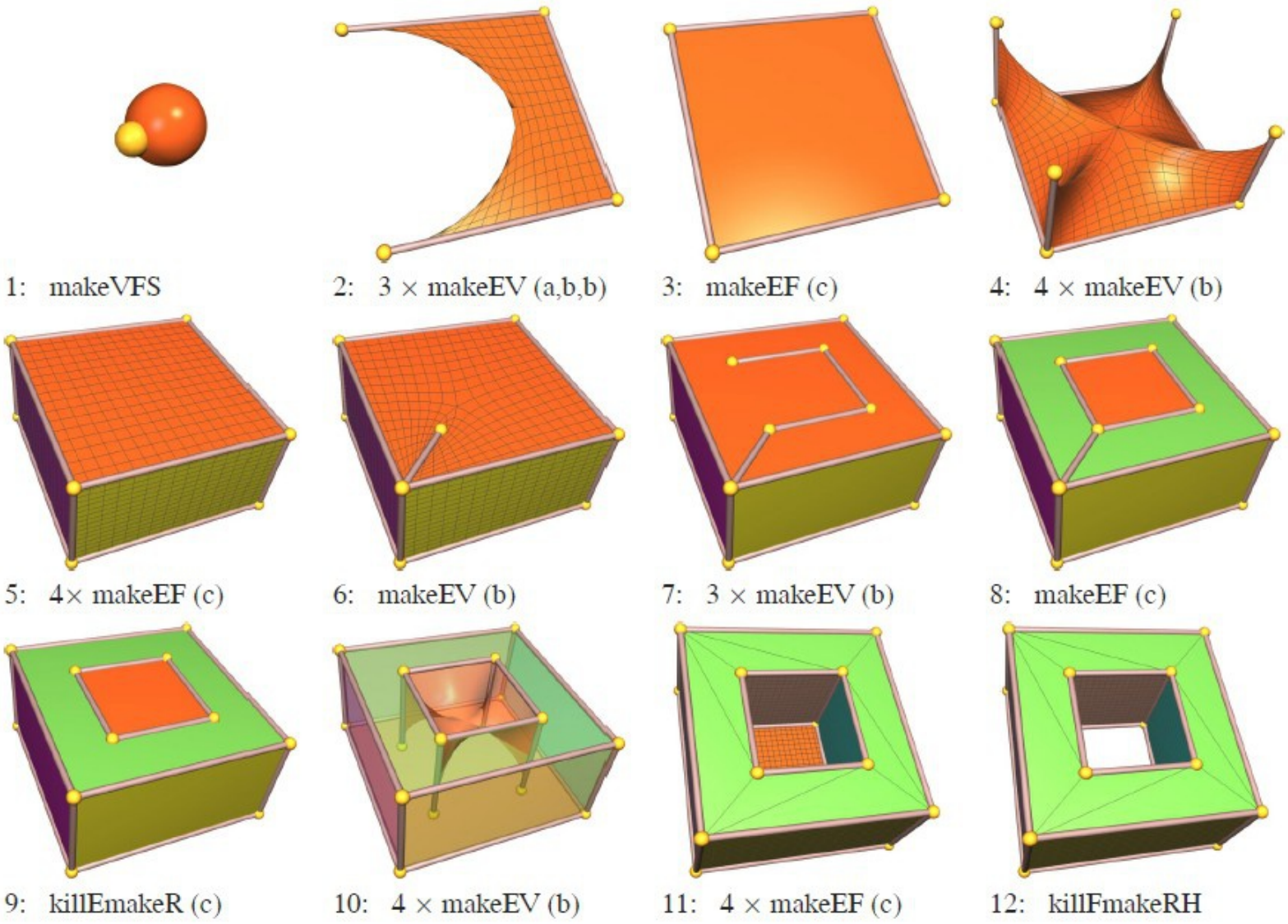
Konstrukce čtyřstěnu



Rozšířená sada operací



Krychle s otvorem



Literatura

J. Foley, A. van Dam, S. Feiner, J. Hughes: ***Computer Graphics, Principles and Practice***, 534-562, 712-714

Jiří Žára a kol.: Počítačová grafika, principy a algoritmy, 234-238

J. Rossignac, A. Safanova, A. Szymczak: ***3D compression made simple: Edgebreaker on a Corner Table***, 2001

H. Lopes, J. Rossignac, A. Safonova, A. Szymczak and G. Tavares: ***Edgebreaker: A Simple Compression Algorithm for Surfaces with Handles***, C&G Intl. J, vol.27(4), 553-567, 2003

Sven Havemann: Generative Mesh Modeling, PhD thesis, 2005, TU Braunschweig, pp. 59-79

matfyz

Computer Graphics Charles University

Hierarchický model

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/

Hierarchie v 3D modelování

Kompozice „zdola-nahoru”

- složitější objekty se sestavují z jednodušších
- při modelování se často několikanásobně opakují některé části objektů (stavební prvky, součástky)

Databáze 3D objektů

- (nejen) ve strojírenství a stavebnictví se často používají standardní – normalizované – prvky

Parametrická konstrukce

- jednotlivé instance objektu se mohou mírně lišit
- parametry/atributy, mechanismus dědičnosti…

Hierarchické modelování

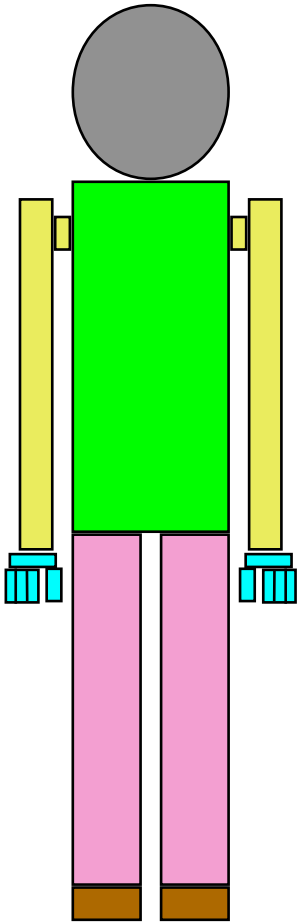
Scéna se skládá z objektů

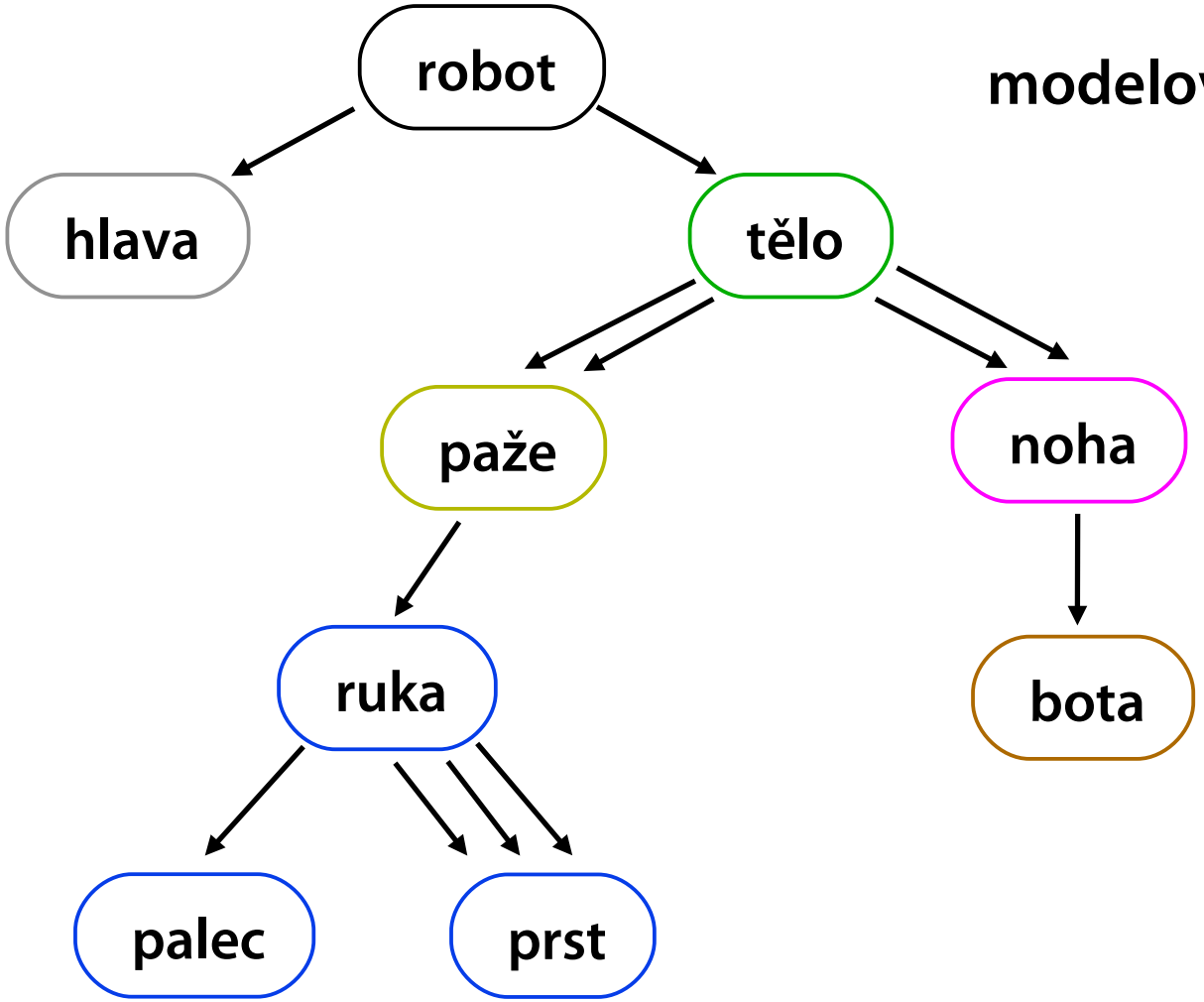
- objekty se skládají z komponent
  - » komponenty se skládají ze součástek
    - součástky se skládají z…

Hierarchické modelování je přirozené a efektivní

- v databázi mohou být uloženy celé knihovny dílů, ze kterých si konstuktér/umělec vybírá
- další vlastnosti hierarchické metodiky
  - » **atributy** uzlů hierarchie (dědičnost, parametrizovatelnost)
  - » **relativní transformační matice** (poloha potomka je definována pouze vzhledem k rodičovskému uzlu)

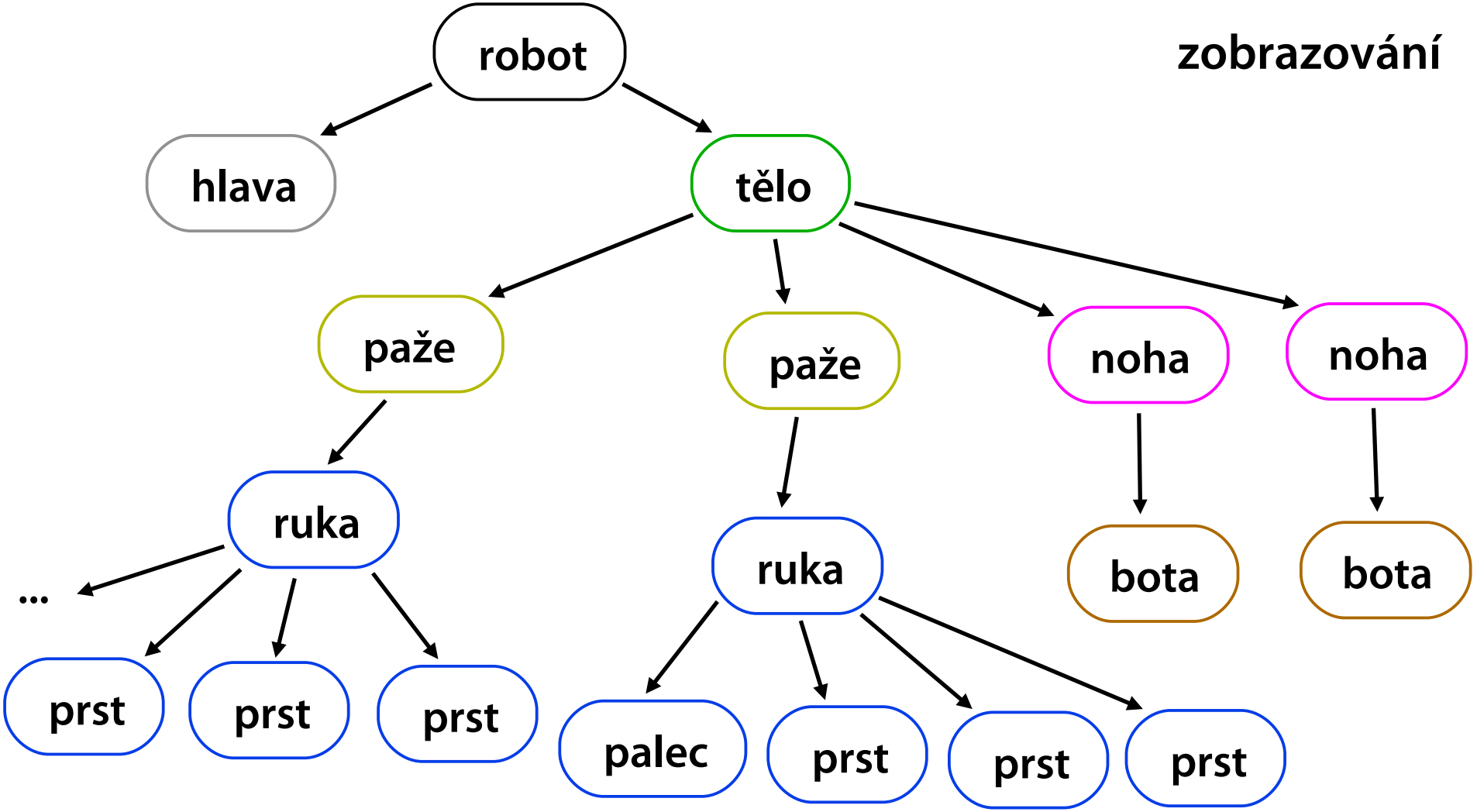
Hierarchický model robota





modelování

Strom modelu robota



Uložení objektu v databázi

Globální (implicitní) **atributy a parametry**

- vzhled (barva, materiál), přesnost aproximace křivek…

Vlastní 3D prvky

- tělesa, stěny, plochy, VB, IB… (podle typu modelu)
- souřadná soustava spojená s objektem
- lokální hodnoty atributů a parametrů

Odkazy na použité podobjekty

- transformační matice (relativní transformace)
- modifikace parametrů a atributů podobjektu

Reprezentace modelu v paměti

Převedení acyklického grafu do formy **stromu**

- uzel **= instance objektu**
- geometrická data se nesdílejí

Souřadnice **vrcholů těles**, **řídících uzlů** ploch…
podléhají geometrickým transformacím a projekci

~ relativní souřadnice uvnitř objektu – 3D

~ absolutní (světové) souřadnice ve scéně – 3D

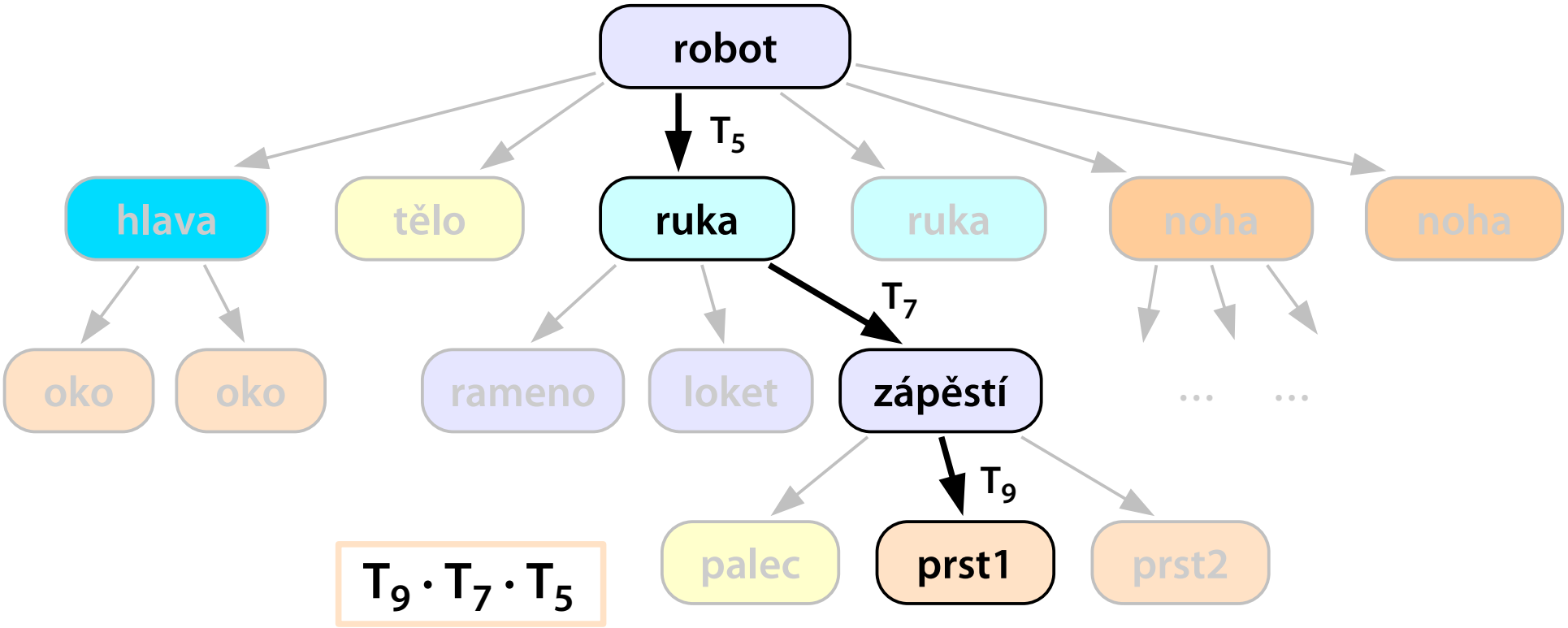
~ promítnuté souřadnice – 2D nebo 3D (z = hloubka)

~ souřadnice výstupního zařízení – 2D (celočíselné)

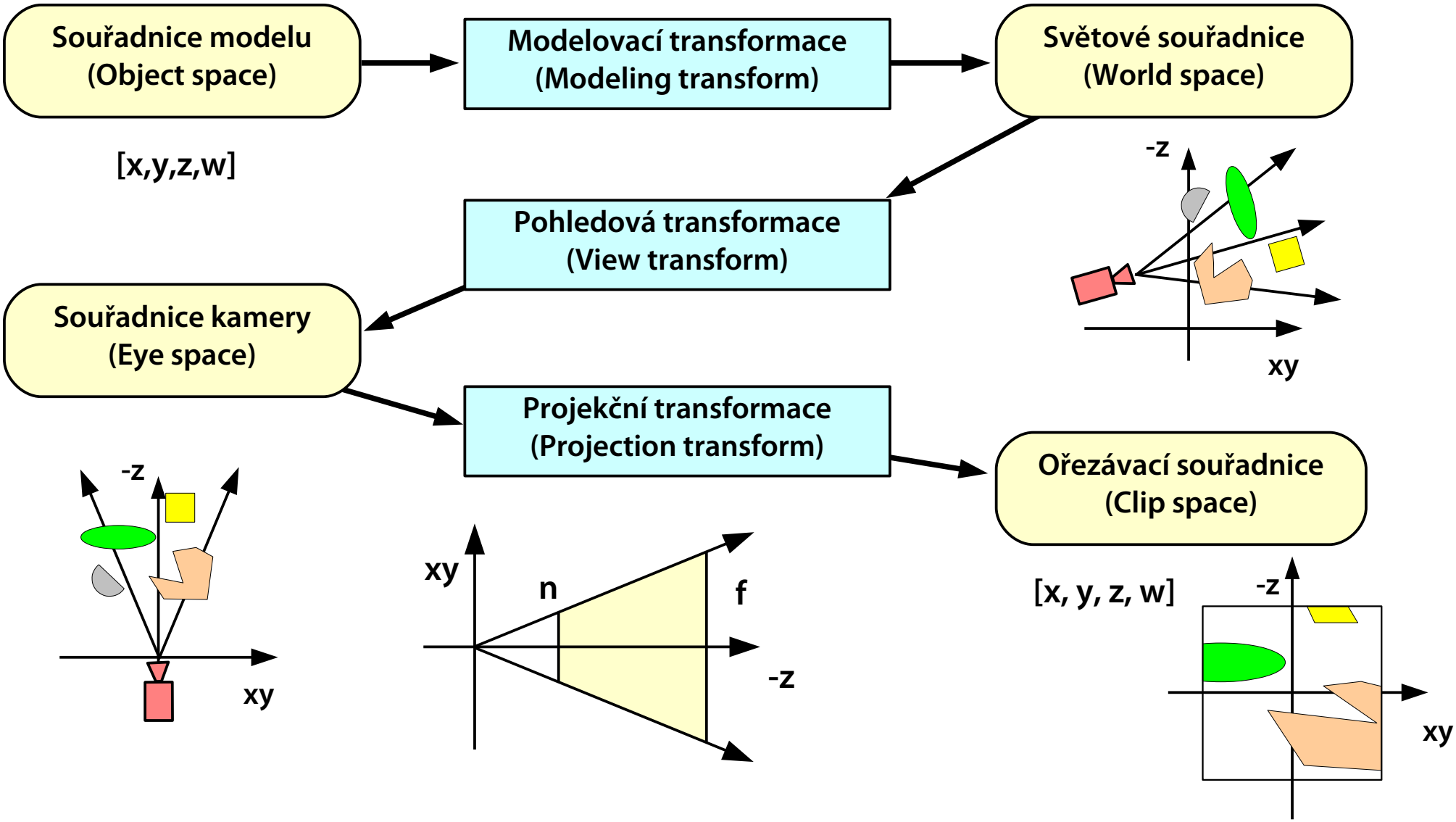
Relativní transformace

Transformace **listu scény** (síť trojúhelníků) do světových souřadnic se skládá z posloupnosti transformací

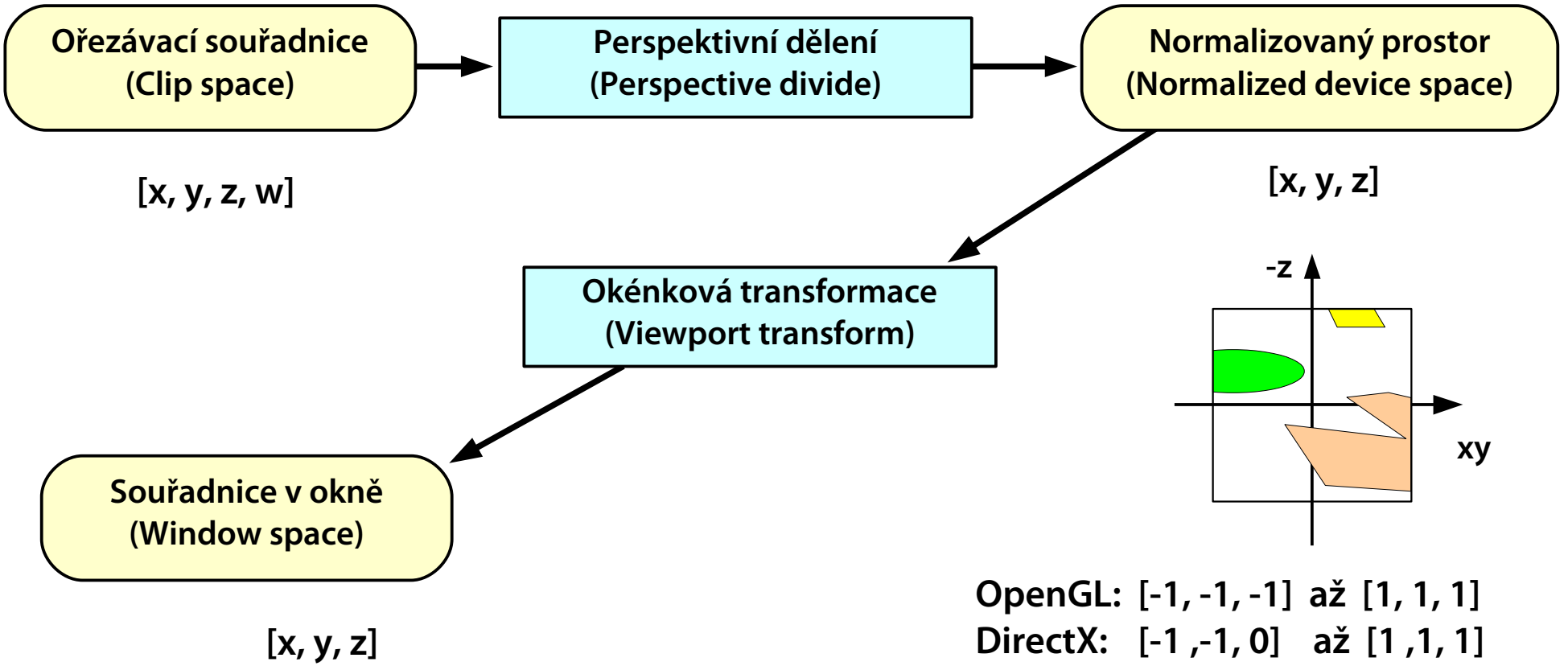
– součiny matic i modelové transformace může počítat **GPU**



Souřadné soustavy



Souřadné soustavy II



[x, y] skutečná velikost v pixelech na obrazovce (fragmenty)
z hloubka kompatibilní s z-bufferem

Souřadné soustavy III

Souřadnice modelu

- databáze objektů, ze kterých se skládá scéna
- 3D modelovací programy (3DS Max, Maya, Blender, Rhino…)

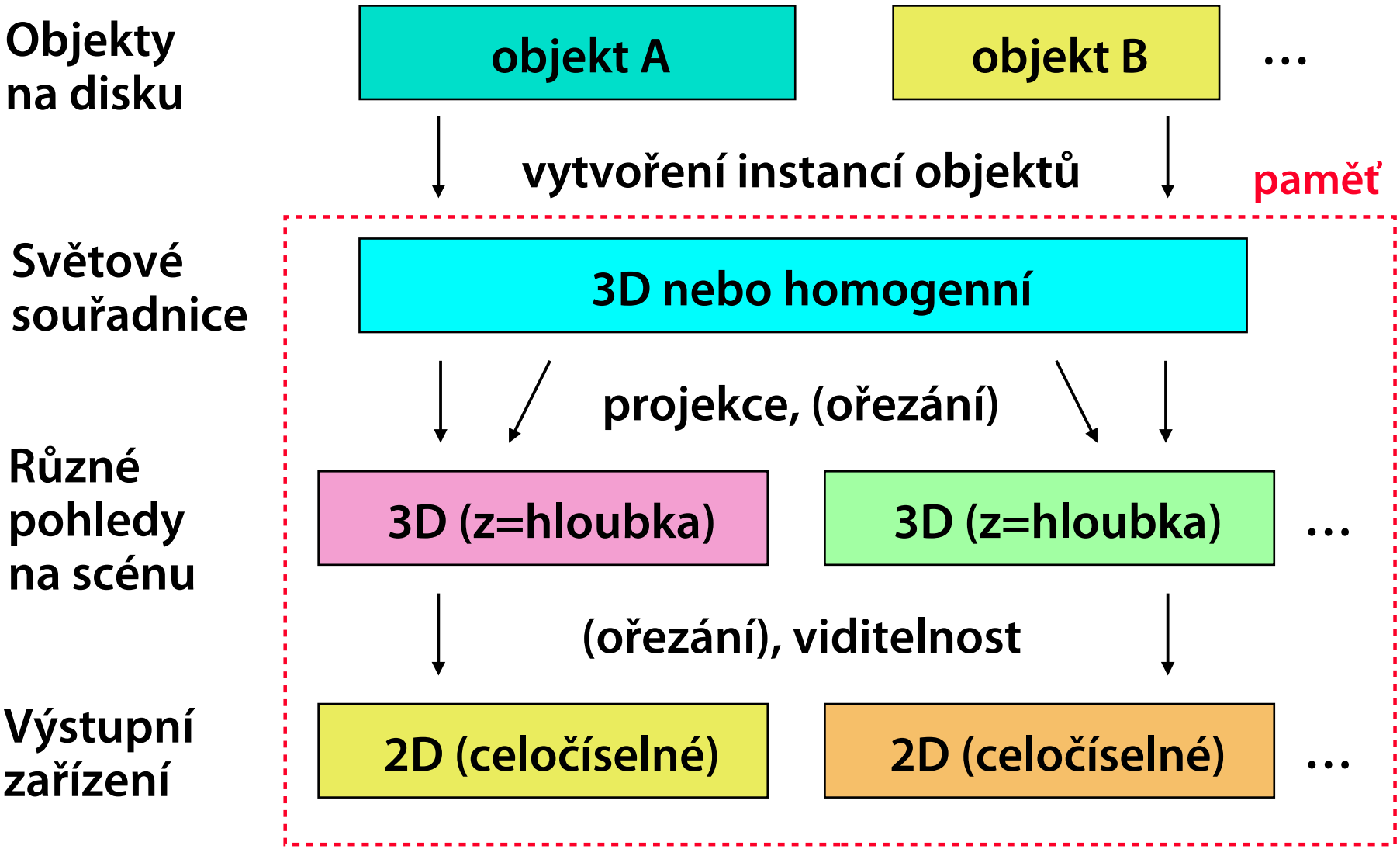
Světové souřadnice

- absolutní souřadnice virtuálního 3D světa
- vzájemná poloha jednotlivých **instancí objektů**

Souřadnice kamery

- 3D svět se transformuje do relativních souřadnic kamery
- střed projekce: **počátek**, směr pohledu: **-z** (nebo **z**)

SW rendering – pole souřadnic



Hierarchické 3D formáty

PHIGS(+) (ANSI, ISO)

- „Programmer's Hierarchical Interactive Graphics System"

OpenInventor, **Performer** (oba SGI)

- objektové nadstavby nad OpenGL

VRML („Virtual Reality Modeling Language")

- WebSpace (World-Wide Web)

OpenSG, **X3D** (Web3D), **FataMorgana** (.FMO)…

Vstupní formáty zobrazovacích programů

- PoV Ray, RayShade, Radiance…

Graf scény („scene graph“)

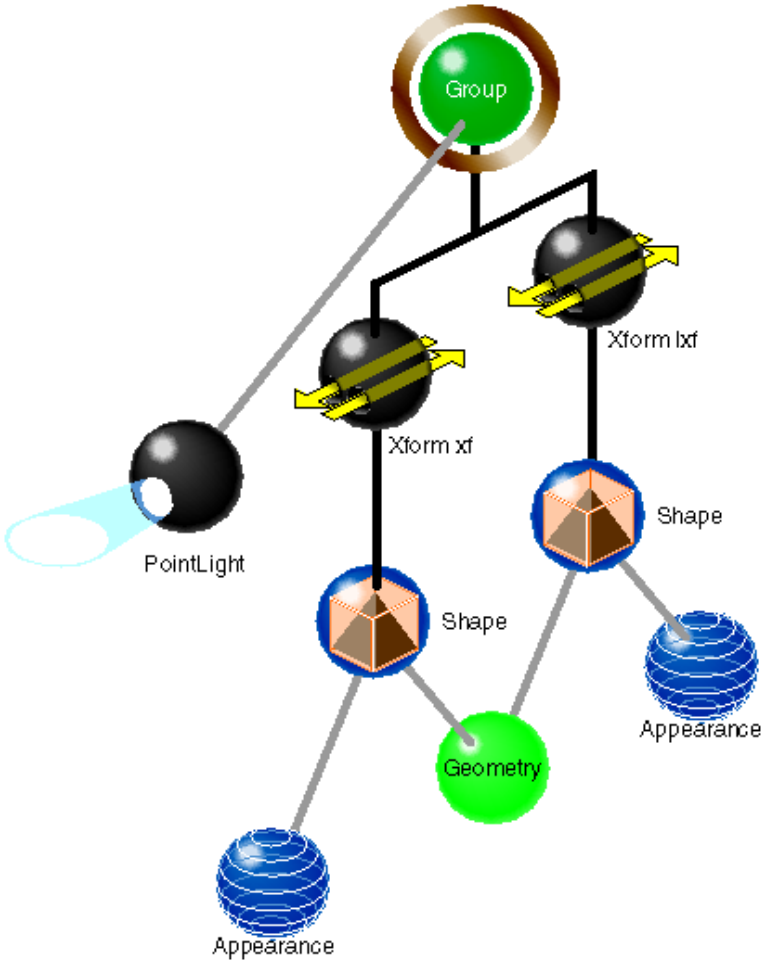
Scéna je reprezentována stromem (či DAG)

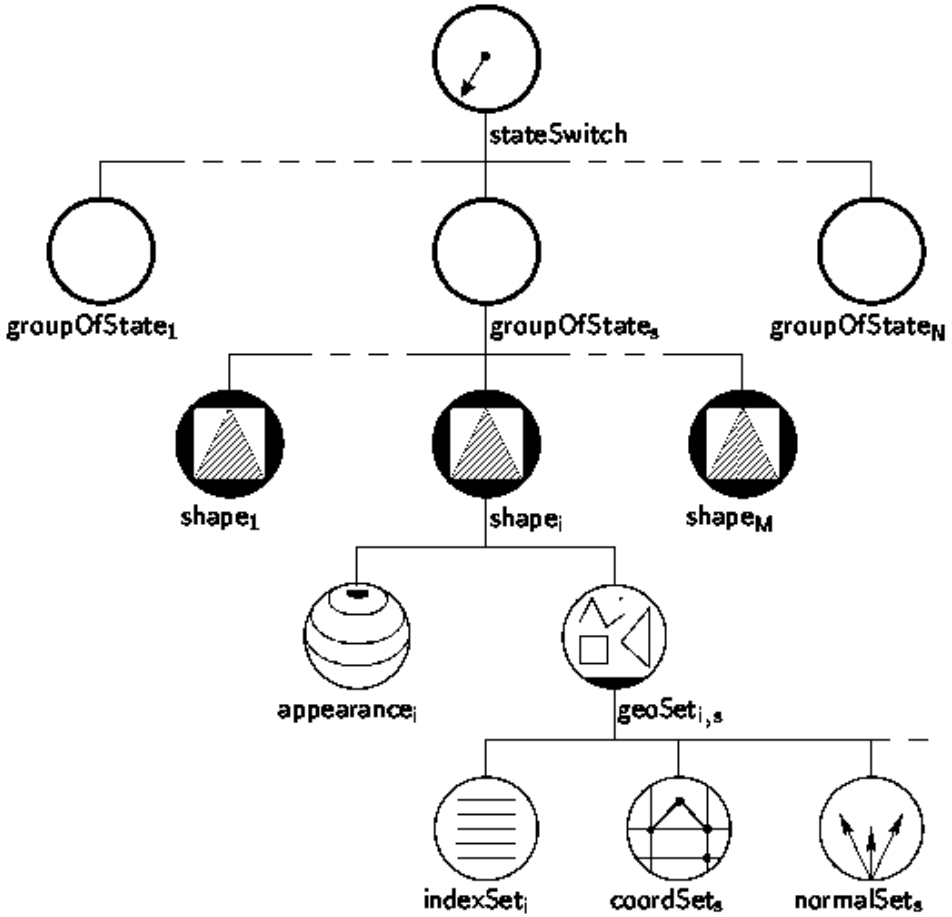
- *vnitřní uzly* – transformace, změny atributů, „skupiny”, přepínače… mohou být závislé na čase
- *listy* – geometrie (vrcholy, normály), světla, materiály…
- DAG – některé listy nebo i celé podgrafy mohou být sdíleny (např. společná geometrie, sady atributů)

Výsledek je definován **průchodem grafem** („in-order“)

- *vnitřní uzly* modifikují parametry, kontext, souřadnou soustavu
- *listy* přispívají k vlastnímu výsledku (primitiva scény)

Graf scény





Images © SGI

Literatura

J. Foley, A. van Dam, S. Feiner, J. Hughes: ***Computer Graphics, Principles and Practice***, 285-346

Web3D konsorcium: https://www.web3d.org/

matfyz

Computer Graphics Charles University

Mathematics for 3D graphics

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/

Content

Homogeneous coordinates, matrix transformations
- coordinate-system conversions

Coordinate systems, projections, frustum

Orientations
- Euler angles, quaternions
- orientation interpolation

Smooth interpolations and approximations
- spline functions, natural spline, B-spline
- Hermite-type interpolations
- KB spline, Catmull-Rom…

Geometric transformations in 3D

Cartesian 3D coordinate vector [x, y, z]

Multiplying by a 3×3 matrix

– **row** vector multiplied **from the right** (DirectX)

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

– **column** vector multiplied **from the left** (OpenGL)

Transform matrices 3×3 have serious drawback – **cannot do translations!**

Homogeneous coordinates

Homogeneous coordinate vector [x, y, z, w]

Transformation: multiplying by a 4×4 matrix

$$[x, y, z, w] \cdot \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = [x', y', z', w']$$

Homogeneous matrix is able to do **translations** and **perspective projections**

Coordinate conversions

From **homogeneous coordinates** [x, y, z, w] into Cartesian coordinates: by division (w≠0) **[x/w, y/w, z/w]**

Coordinate vector **[x, y, z, 0]** does not correspond to any real point in space

- can be interpreted as a **directional vector** (point in infinity)

From **Cartesian coordinates** to homogeneous: trivial extension **[x, y, z]** … **[x, y, z, 1]**

Elementary transformations

Affine transformation

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ t_1 & t_2 & t_3 & 1 \end{bmatrix}$$

Upper left submatrix **[a_{11} to a_{33}]** defines scaling, orientation and shear

Vector **[t_1, t_2, t_3]** defines translation

- translation is performed as the last step

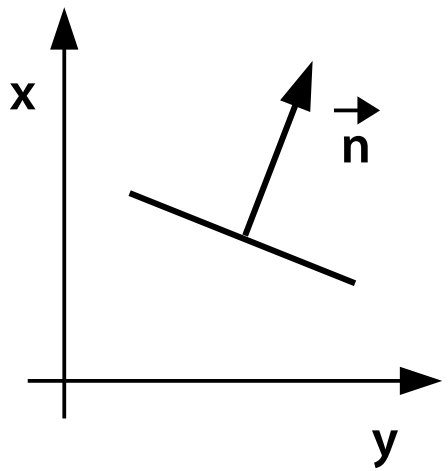
Normal vector transformation

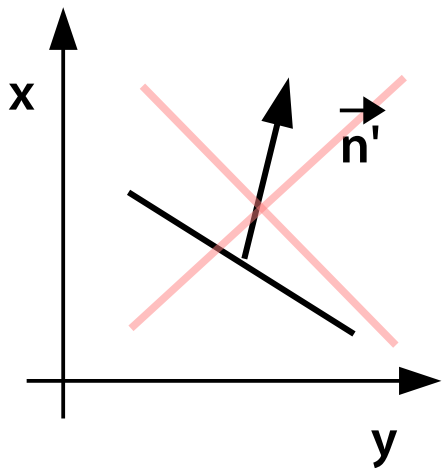
Normal vectors must not be transformed by regular matrices (like point positions are)

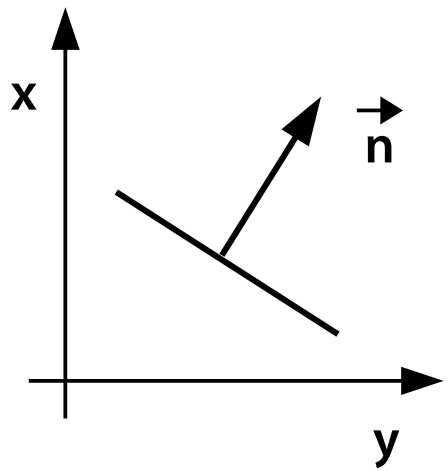
– exception: M is rotational (orthonormal)

Normal-vector transformation matrix **N**:

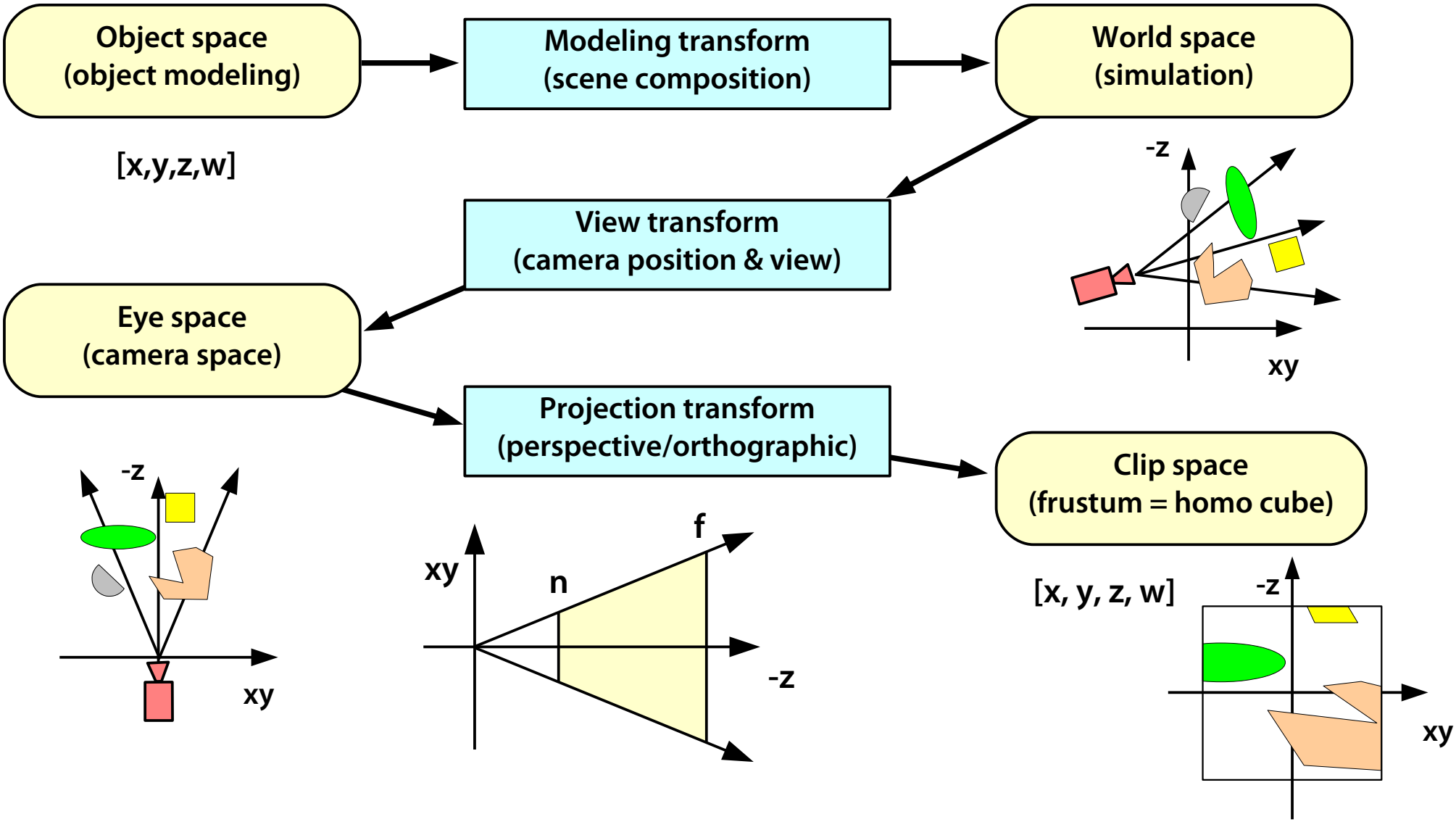
$$N = (M^{-1})^T$$



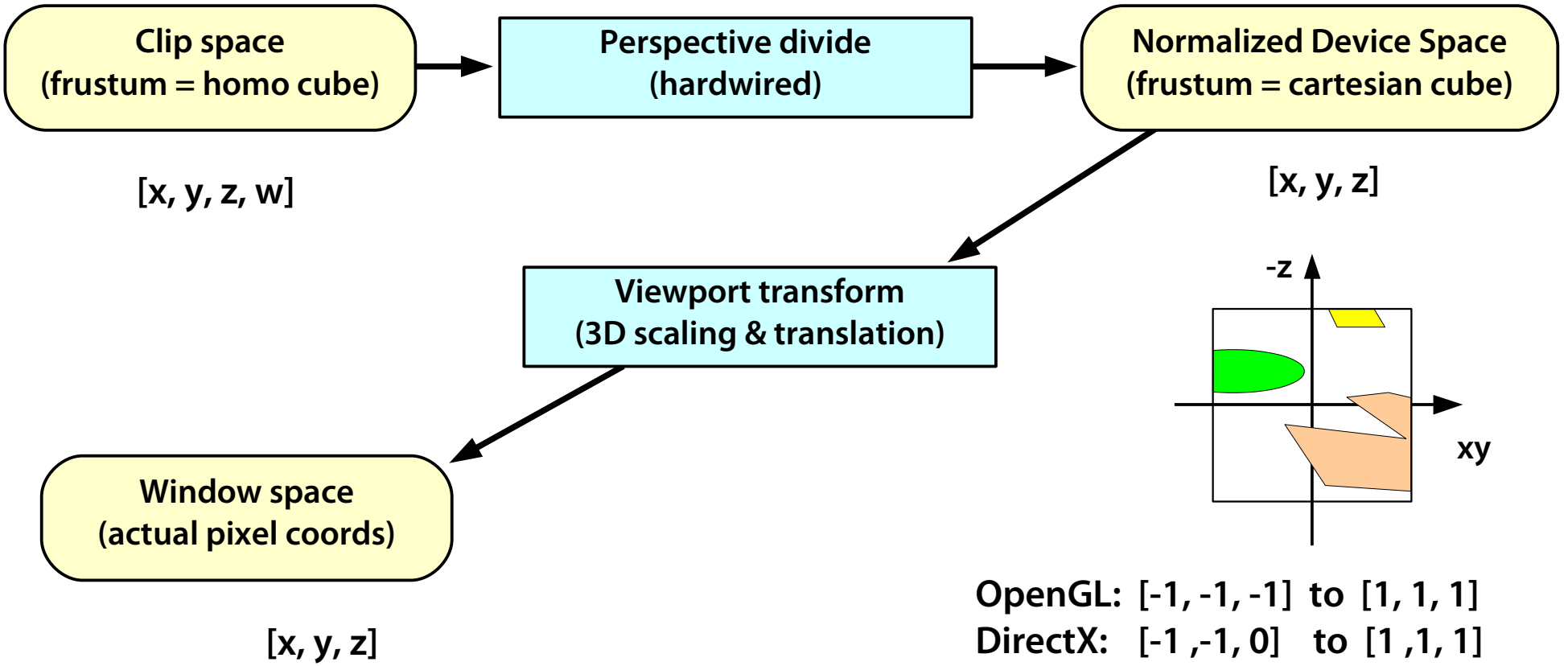




Coordinate systems in OpenGL



Coordinate systems in OpenGL



[x, y] actual screen coordinates (fragments)
z depth value compatible with actual depth-buffer

Coordinate systems in OpenGL

Object space

- modeling of individual objects, modularity
- 3D modeling software (3DS Max, Blender, Rhino…)

World space

- absolute (real) coordinates in simulated virtual world
- object instantiation, collision detection, AI planning…

Camera space

- the whole virtual world transforms into coordinates relative to a camera
- center of projection: **origin**, view direction: **-z** (or **z**)

Coordinate systems & transformations

Transformation "model → camera"

- altogether – "model-view" matrix
- world coordinates are not directly used in rendering pipeline

Projection transformation

- defines visible volume = **frustum** [**l**, **r**, **b**, **t**, **n**, **f**]
- front & back clip distances: **n**, **f**
- result: homogeneous coordinate (before clipping)

"Clip space"

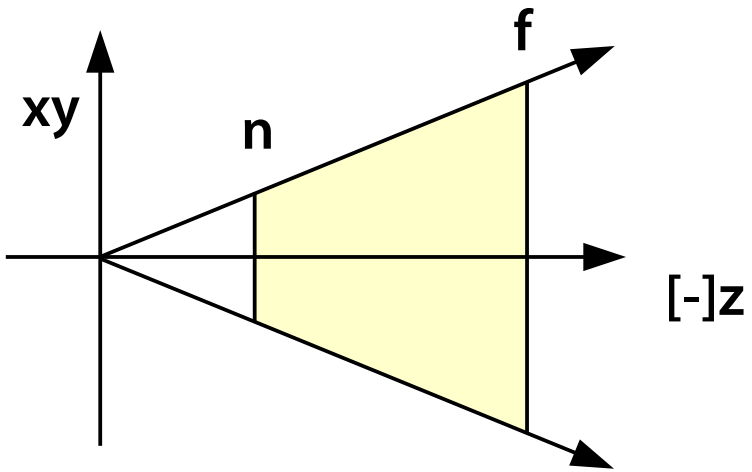
- **mandatory output coordinate** of vertex shader!

Projection transform (perspective)

Far point **f** can be **in infinity**

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ -\frac{r+l}{r-l} & -\frac{t+b}{t-b} & \frac{f+n}{f-n} & 1 \\ 0 & 0 & -\frac{2fn}{f-n} & 0 \end{bmatrix}$$

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ -\frac{r+l}{r-l} & -\frac{t+b}{t-b} & 1 & 1 \\ 0 & 0 & -2n & 0 \end{bmatrix}$$



Coordinate systems & transforms

Perspective division

- just converts **homogeneous** coordinates into **cartesian**

Normalized coordinates (“NDS”)

- standard-sized cube/cuboid
- OpenGL: [-1, -1, -1] to [1, 1, 1]
- DirectX: [-1, -1 ,0] to [1, 1, 1]

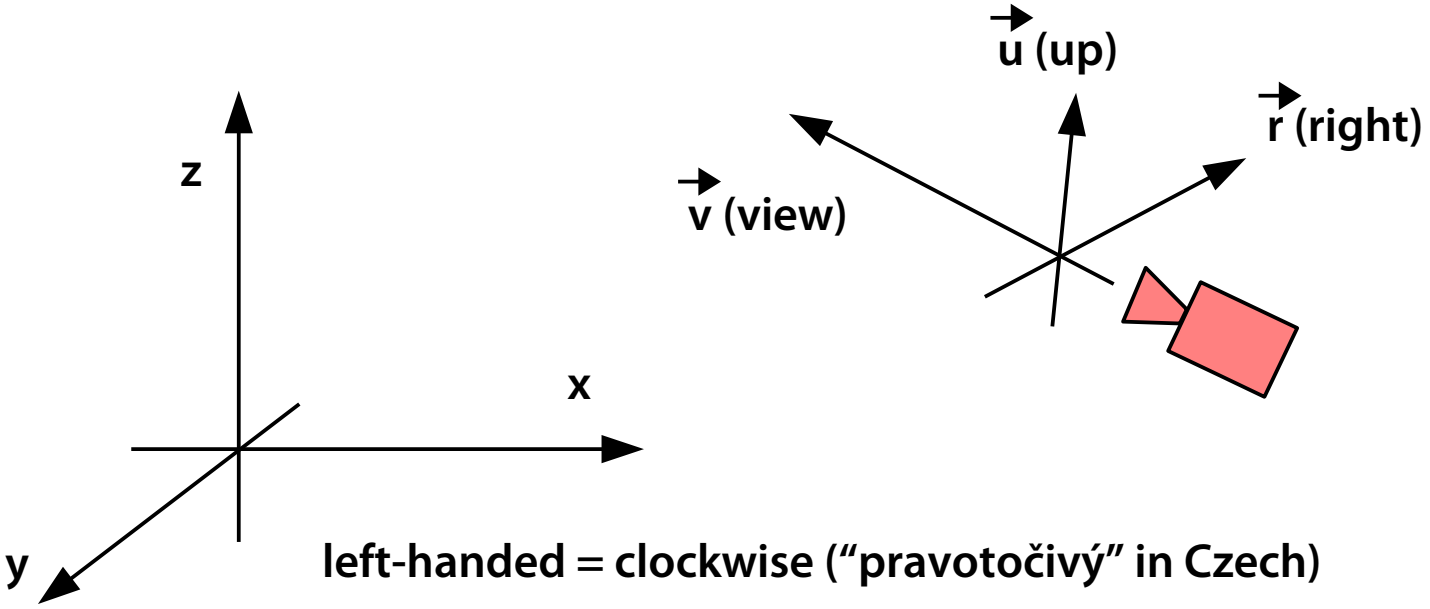
Window coordinates (“window space”)

- result of **linear adjustment** to window size in pixels
- used in **rasterizer** and all **fragment processing**

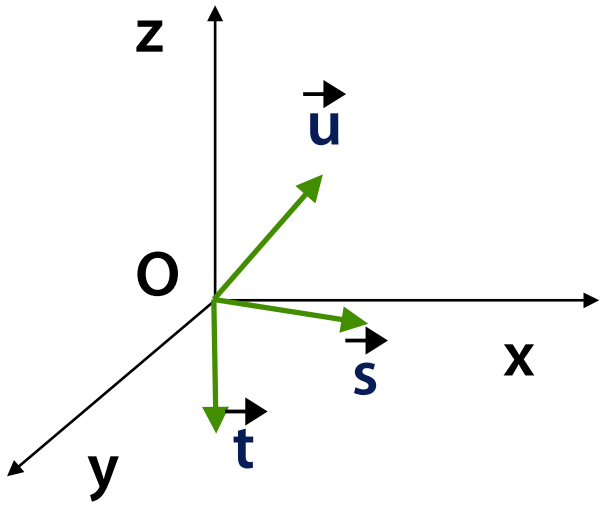
Rigid body transformation

Preserves **shapes**, alters **orientation & position**

- **translation** and **rotation**
- **conversion between coordinate systems** (e.g. between world-space and camera-space)



Conversion between two orientations



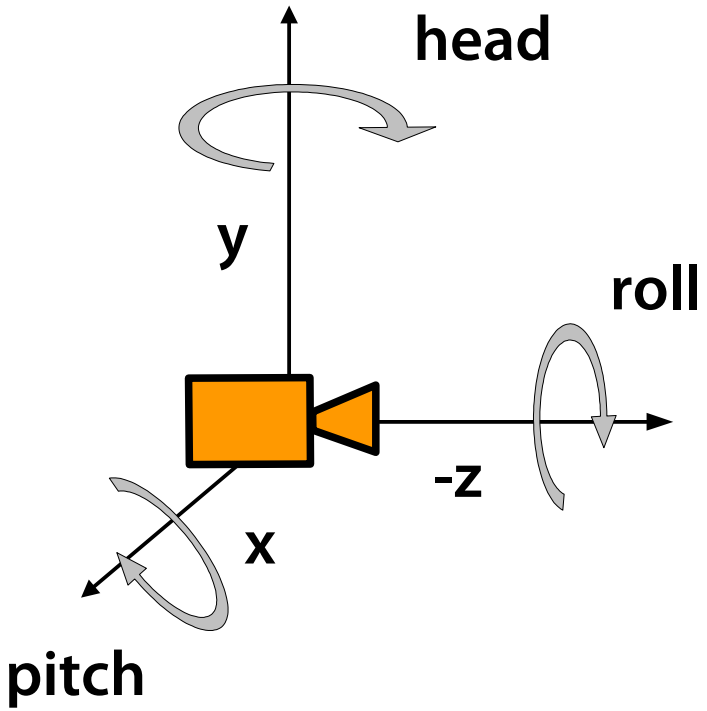
Coordinate system has an origin **O** and is defined by three unit vectors **[s, t, u]**

$$[1,0,0]\cdot M_{stu\to xyz} = s$$
$$[0,1,0]\cdot M_{stu\to xyz} = t$$
$$[0,0,1]\cdot M_{stu\to xyz} = u$$

$$M_{stu\to xyz} = \begin{bmatrix} s_x & s_y & s_z \\ t_x & t_y & t_z \\ u_x & u_y & u_z \end{bmatrix}$$

$$M_{xyz\to stu} = M^T_{stu\to xyz}$$

Euler transformation



Arbitrary rotation decomposed into **three components**

– Leonard Euler (1707-1783)

$$E(h, p, r) = R_y(h) \cdot R_x(p) \cdot R_z(r)$$

h (**head**, yaw): plan view direction

p (**pitch**): forward/backward pitching

r (**roll**): rolling around the view vector

Euler transformation II

Result matrix of rotation

$$E=\begin{pmatrix} c(r)c(h)-s(r)s(p)s(h) & s(r)c(h)+c(r)s(p)s(h) & -c(p)s(h) \\ -s(r)c(p) & c(r)c(p) & s(p) \\ c(r)s(h)+s(r)s(p)c(h) & s(r)s(h)-c(r)s(p)c(h) & c(p)c(h) \end{pmatrix}$$

s(x) … sin(x), c(x) … cos(x)

Backward matrix → angles computation h, p, r

- p … e_{23}
- r … e_{21}/e_{22}
- h … e_{13}/e_{33}

Rotations: different conventions

Main convention

- 1. rotation around **z** by φ
- 2. rotation around **x'** by θ
- 3. rotation around **z''** by ψ

X-convention

- 1. rotation around **z**
- 2. rotation around original **x**
- 3. rotation around original **z**

More systems (24): aeronautics, gyroscopes, physics…

Quaternions

Sir William Rowan **Hamilton**, 16 Oct 1843 (Dublin)

- $i^2 = j^2 = k^2 = ijk = -1$
- usage in graphics since 1985 (Shoemake)
- **generalization of complex numbers** in 4D space

$\mathbf{q} = (\mathbf{v}, w) = i\,x + j\,y + k\,z + w = \mathbf{v} + w$ sometimes $(w, \mathbf{v})$!

Imaginary part $\mathbf{v} = (x, y, z) = i\,x + j\,y + k\,z$

$i^2 = j^2 = k^2 = -1, \quad jk = -kj = i, \quad ki = -ik = j, \quad ij = -ji = k$

Quaternions: operations I

Addition

- $(\mathbf{v}_1, w_1) + (\mathbf{v}_2, w_2) = (\mathbf{v}_1 + \mathbf{v}_2, w_1 + w_2)$

Multiplication

- $\mathbf{q}\,\mathbf{r} = (\mathbf{v}_q \times \mathbf{v}_r + w_r \mathbf{v}_q + w_q \mathbf{v}_r,\ w_q w_r - \mathbf{v}_q \cdot \mathbf{v}_r)$

$$
\begin{aligned}
&i(q_y r_z - q_z r_y + r_w q_x + q_w r_x),\\
&j(q_z r_x - q_x r_z + r_w q_y + q_w r_y),\\
&k(q_x r_y - q_y r_x + r_w q_z + q_w r_z),\\
&q_w r_w - q_x r_x - q_y r_y - q_z r_z
\end{aligned}
$$

Quaternions: operations II

Conjugation

- $(\mathbf{v}, w)^* = (-\mathbf{v}, w)$

Norm (squared absolute value)

- $\|\mathbf{q}\|^2 = n(\mathbf{q}) = \mathbf{q}\,\mathbf{q}^* = x^2 + y^2 + z^2 + w^2$

Unit

- $\mathbf{i} = (\mathbf{0}, 1)$

Reciprocal

- $\mathbf{q}^{-1} = \mathbf{q}^* / n(\mathbf{q})$

Multiplication by a scalar

- $s\,\mathbf{q} = (0, s)\,(\mathbf{v}, w) = (s\,\mathbf{v}, s\,w)$

Unit quaternions

Every unit quaternion ($x^2 + y^2 + z^2 + w^2 = 1$) can be expressed as

- $\mathbf{q} = (\mathbf{u}_q \sin\phi, \cos\phi)$
- for some **unit 3D vector** $\mathbf{u}_q$

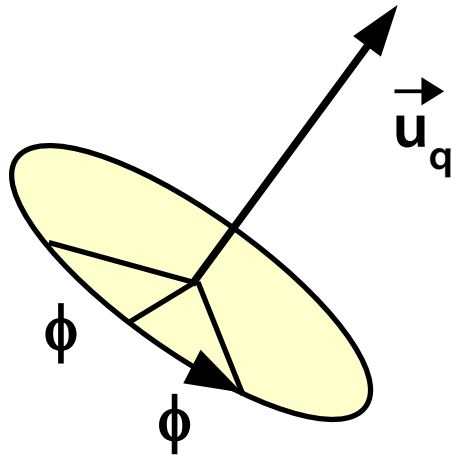
It represents a **rotation (orientation)** in 3D

- **ambiguity**: both **q** and **-q** represent the same rotation! $(\phi + \pi)$
- **identity** (zero rotation): (**0**, 1)

Power, exponential, logarithm

- $\mathbf{q} = \mathbf{u}_q \sin\phi + \cos\phi = \exp(\phi\,\mathbf{u}_q), \quad \log\mathbf{q} = \phi\,\mathbf{u}_q$
- $\mathbf{q}^t = (\mathbf{u}_q \sin\phi + \cos\phi)^t = \exp(t\phi\,\mathbf{u}_q) = \mathbf{u}_q \sin t\phi + \cos t\phi$

Rotation using a quaternion



Unit quaternion

- $\mathbf{q} = (\mathbf{u}_q \sin\phi, \cos\phi)$
- $\mathbf{u}_q$ ... axis of rotation, ϕ ... angle

Vector (point) in 3D: $\mathbf{p} = [p_x, p_y, p_z, 0]$

Rotation of vector (point) $\mathbf{p}$ around $\mathbf{u}_q$ by angle 2ϕ

$$\mathbf{p'} = \mathbf{q}\,\mathbf{p}\,\mathbf{q}^{-1} = \mathbf{q}\,\mathbf{p}\,\mathbf{q}^*$$

Quaternion ↔ matrix conversions

Quaternion **q** converted to a matrix

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

Reverse conversion is based on equations

$$\begin{aligned} m_{23} - m_{32} &= 4wx \\ m_{31} - m_{13} &= 4wy \\ m_{12} - m_{21} &= 4wz \\ tr\, M + 1 &= 4w^2 \end{aligned} \qquad \textbf{(\$)}$$

Matrix → quaternion II

1. "**matrix_trace+1**" has large enough absolute value

$$w=\frac{1}{2}\sqrt{tr\,M+1} \qquad x=\frac{m_{23}-m_{32}}{4w}$$

$$y=\frac{m_{31}-m_{13}}{4w} \qquad z=\frac{m_{12}-m_{21}}{4w}$$

2. … otherwise compute a component with largest absolute value first and then apply **$**

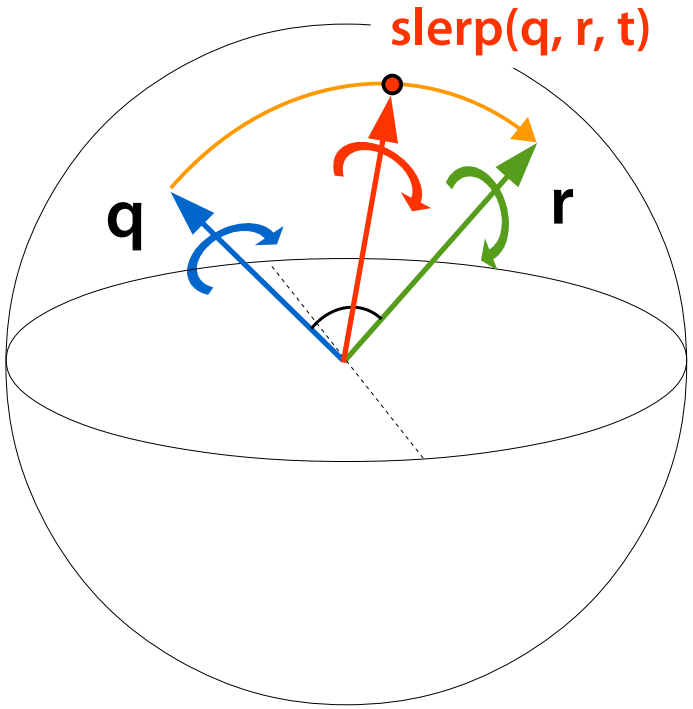
$$4x^2=1+m_{11}-m_{22}-m_{33}$$
$$4y^2=1-m_{11}+m_{22}-m_{33}$$
$$4z^2=1-m_{11}-m_{22}+m_{33}$$

Spherical linear interpolation (slerp)

Two quaternions **q** and **r** (**q** · **r** ≥ 0, else take **-q**)

Real parameter **0 ≤ t ≤ 1**

Interpolated quaternion **slerp(q, r, t) = q (q* r)^t**



$$slerp(q, r, t) = \frac{\sin(\phi(1-t))}{\sin\phi}\cdot q + \frac{\sin(\phi t)}{\sin\phi}\cdot r$$

$$\cos\phi = q_x r_x + q_y r_y + q_z r_z + q_w r_w$$

The shortest spherical arc
between **q** and **r**
(quaternion splines will be explained later)

Rotation between two vectors

Two vectors **s** and **t**

1. normalization of **s**, **t**
2. unit rotation axis $\mathbf{u} = (\mathbf{s} \times \mathbf{t}) / ||\mathbf{s} \times \mathbf{t}||$
3. angle between **s** and **t** $\mathbf{e} = \mathbf{s} \cdot \mathbf{t} = \cos 2\phi$

   $||\mathbf{s} \times \mathbf{t}|| = \sin 2\phi$
4. final quaternion $\mathbf{q} = (\mathbf{u} \cdot \sin \phi, \cos \phi)$

$$q=(q_v, q_w)=\left(\frac{1}{\sqrt{2(1+e)}}(s\times t), \frac{\sqrt{2(1+e)}}{2} \right)$$

Slerp of rotational matrices (theory)

Two rotational matrices **Q** and **R**

Real parameter $\mathbf{0 \le t \le 1}$

Interpolated matrix $\mathbf{slerp(Q, R, t) = Q\,(Q^T R)^t}$

Technical problem – how to do power operation on matrices?

Need to compute axis and angle $Q^T R$
(not very efficient)

See "RotationIssues.pdf" for details (D. Eberly)

Rotation representation – summary

Rotational matrix

\+ HW support, efficient point/vector transformation

– memory (float[9]), other operations are not so efficient

Rotational axis and angle

\+ memory (float[4] or float[6]), similar to quaternion

– inefficient composition and interpolation

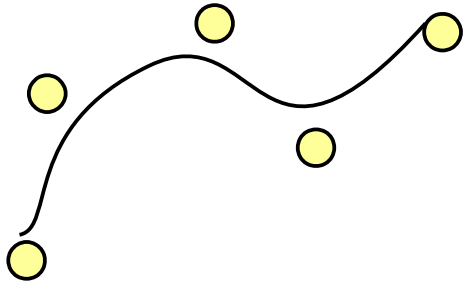
Quaternion

\+ memory (float[4]), composition, interpolation

– inefficient point/vector transformation

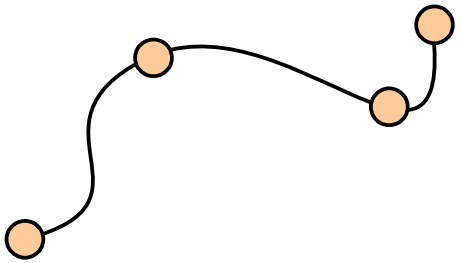
See “RotationIssues.pdf” for details (D. Eberly)

Approximation and interpolation



Approximation (e.g. B-spline)

- needs not to pass through control points



Interpolation (e.g. Catmull-Rom)

- curve passes through control points

Curve continuity

- $\mathbf{G^n}$ – geometric continuity of the $\mathbf{n^{th}}$ order ($\mathbf{G^0}$ – simple continuity, $\mathbf{G^1}$ – tangent, $\mathbf{G^2}$ – curvature…)
- $\mathbf{C^n}$ – analytical continuity of the $\mathbf{n^{th}}$ order, $\mathbf{n^{th}}$ derivative continuity ($\mathbf{C^1}$ – speed, $\mathbf{C^2}$ – acceleration), superior to geometric continuity

History

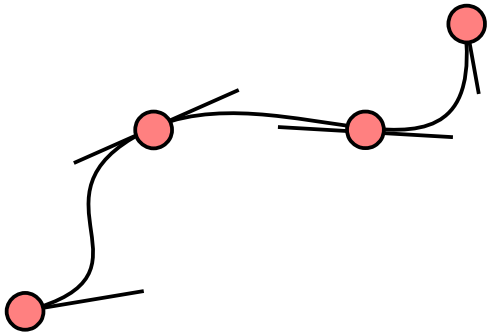
Curves in modeling industry

- Paul de Faget **de Casteljau**, Citroën (1959)
- Pierre **Bèzier** (Renault 1933-1975, UNISURF)
  - » late start, but his results were more popular
- application of spline function theory – mostly in USA (James **Ferguson**, 1964, Boeing, C^2 spline curves)

Spline function theory

- B-spline: Isaac Jacob **Schoenberg**, (ballistics, Aberdeen, MD, 1946)
- theory: Carl **de Boor** (also worked for General Motors)
- Gordon, Riesenfeld **united** Bèzier and B-spline curves (1972)

“Free-form“ curves I



Defined by a sequence of **control points**

- “control polygon”
- approximation or interpolation
- boundary conditions can be different

Controllability

- sometimes **tangent vectors** added in control points (Hermit)
- interpolation $\Rightarrow$ closer control

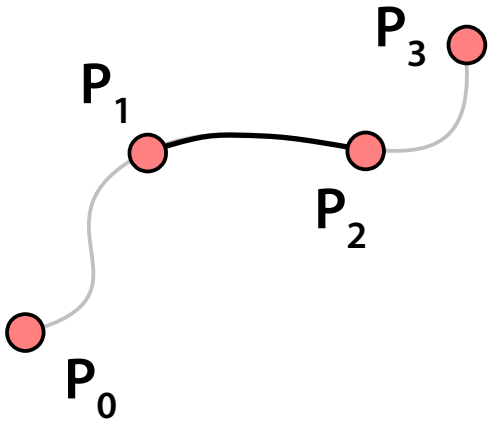
Locality

- change of single control point (one tangent vector) induces change in a **restricted neighborhood** only

“Free-form“ curves II

Parametric expression (**0 ≤ t ≤ 1**)

$$P(t) = \sum_{i=0}^{N-1} w_i(t) P_i$$



Convex hull property

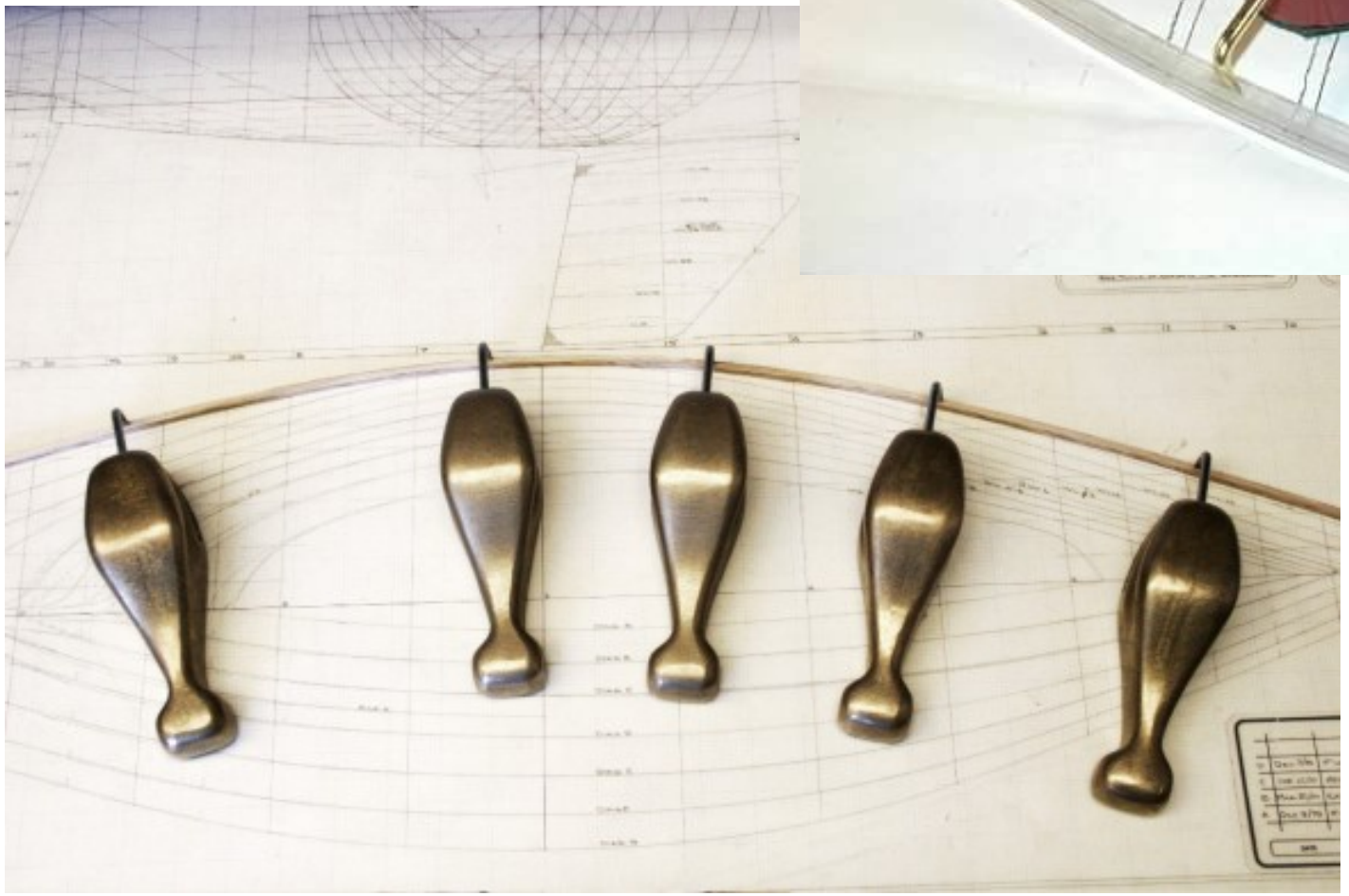
- curve lies in convex hull of its control polygon

Cauchy condition for blending functions

- sufficient for convex hull property
- ensures **affine transformation invariancy**

$$\sum_{i=0}^{N-1} w_i(t) = 1$$

Splines



© Jay Greer

© Edson International

Spline functions

Named after elastic ruler used in ship design (pinned in several points by “ducks”)

Definition: **spline function of degree n**

- piece-wise **polynomial** (of degree **n**)
- **maximum-smoothness connection**:

$\mathbf{C^{n-1}}$ – continuity of **n-1**$^{\mathbf{th}}$ derivative (polynomial of degree **n**)

- **global parametrization u**, $\mathbf{u_0 \le u \le u_N}$ $[u_0, u_1, \dots u_N]$
- individual parts are often uniformly parametrized – **uniform spline** $\mathbf{t_i = (u - u_i) / (u_{i+1} - u_i)}$, $\mathbf{0 \le t_i \le 1}$

Polynomial curve

Matrix notation

$$P(t) = \boldsymbol{TC} = [\, t^n, t^{n-1}, \ldots t, 1 \,] \cdot \begin{bmatrix} x_n & y_n & z_n \\ x_{n-1} & y_{n-1} & z_{n-1} \\ .. & .. & .. \\ x_1 & y_1 & z_1 \\ x_0 & y_0 & z_0 \end{bmatrix}$$

Basis matrix **M** and vector of geometric conditions **G**

$$\boldsymbol{C} = \boldsymbol{MG} = [\, m_{ij} \,]_{i=n,\, j=1}^{0,\, k} \cdot \begin{bmatrix} G_1 \\ .. \\ G_k \end{bmatrix} \qquad P(t) = \boldsymbol{TMG}$$

Matrix notation of a curve

P(t) = T C = T M G

- separation of a parameter vector (**T**) from polynomial basis (**M**) and geometric control conditions/points (**G**)
- differentiation (tangent, curvature) restricted to **T**
- control polynomial **TM** times "geometry" **G**

Cubic: n = 3, k = 4

$$Q(t) = [t^3, t^2, t, 1] \cdot \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \cdot \begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix}$$

Hermite cubic curve

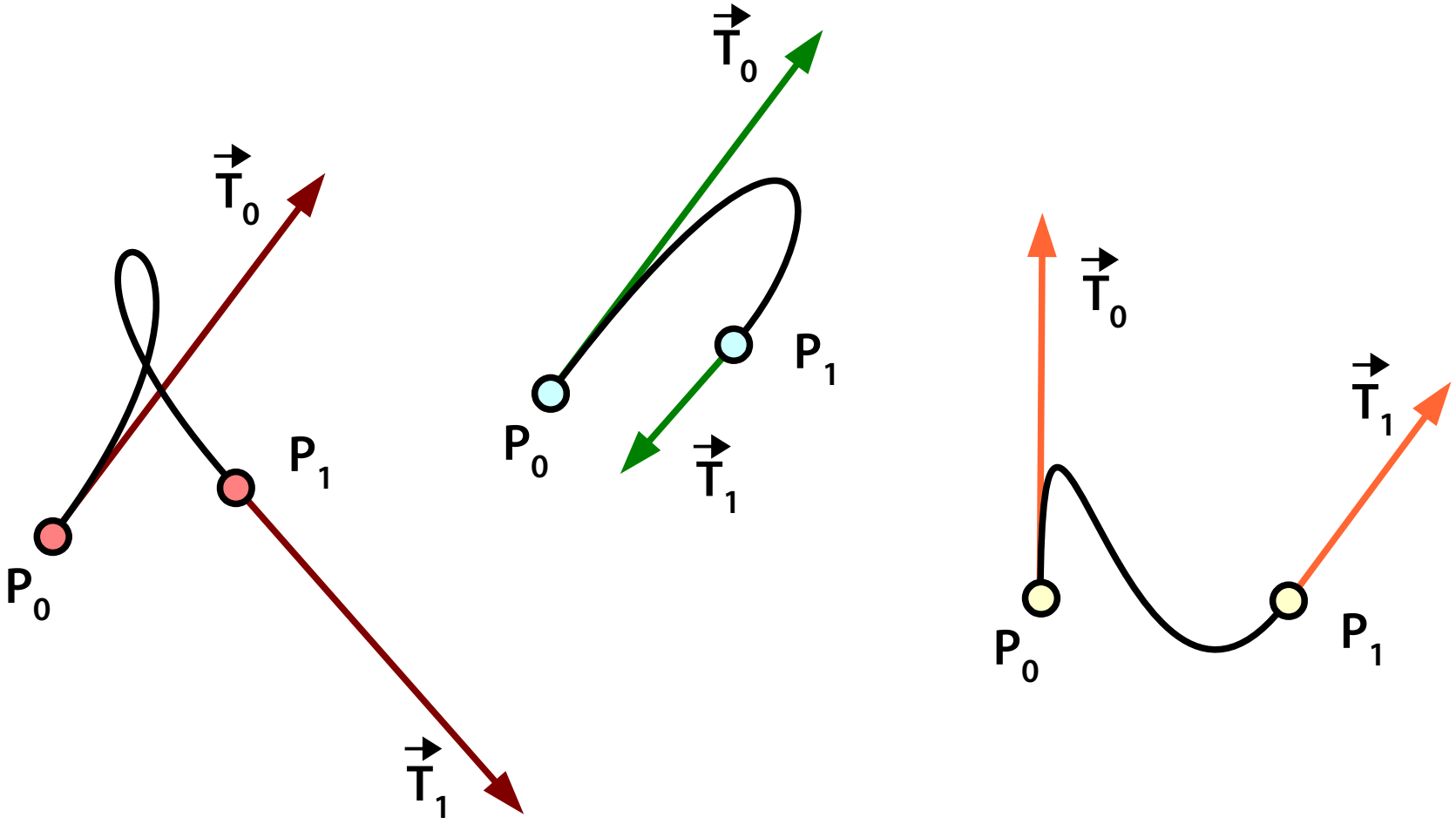
Ferguson curve (cubic)

Geometry: **endpoints** and **tangent vectors**

- beginning ($\mathbf{P}_0$) and end ($\mathbf{P}_1$) of a curve
- tangents in beginning ($\mathbf{T}_0$) and ending ($\mathbf{T}_1$) points

$$F(t) = [t^3, t^2, t, 1] \cdot \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_0 \\ P_1 \\ T_0 \\ T_1 \end{bmatrix}$$

Hermite cubic – examples



More curves

Interpolating cubics derived from Hermite

- general: **Kochanek-Bartels** (KB-spline, TCB cubic)
- special: **cardinal** spline, **Catmull-Rom** spline
- **Akima** interpolation ("Akima spline", not C^2)
- **D-spline** cubic

Another popular curves

- **Bèzier** curves
- **B-spline** curve, **Coons** spline (approximation)
- **natural** spline (interpolation)

Kochanek-Bartels cubic (KB-spline, TCB)

Derived from **Hermite** cubic (3DS Max, Lightwave)

- **tangent vectors** are derived from **control points**
- three additional scalar parameters (**zero** by default)
  - "**tension**" **t**: sharpness of a curve passing control point (absolute value of a tangent vector)
  - "**continuity**" **c**: in control points
  - "**bias**" **b**: tangent direction in control point

Left and **right** tangent ($\mathbf{T_0}$ and $\mathbf{T_1}$ in local sense):

$$L_i = \frac{(1-t)(1-c)(1+b)}{2} \cdot (P_i - P_{i-1}) + \frac{(1-t)(1+c)(1-b)}{2} \cdot (P_{i+1} - P_i)$$

$$R_i = \frac{(1-t)(1+c)(1+b)}{2} \cdot (P_i - P_{i-1}) + \frac{(1-t)(1-c)(1-b)}{2} \cdot (P_{i+1} - P_i)$$

Cardinal spline, Catmull-Rom spline

Special cases of KB-spline

cardinal spline

- parameter **a** only (in fact relates to "**t**", **c = b = 0**)

$$T_i = a \cdot (P_{i+1} - P_{i-1}) \qquad 0 \leqslant a \leqslant 1$$

Catmull-Rom spline

$$T_i = \frac{1}{2} \cdot (P_{i+1} - P_{i-1})$$

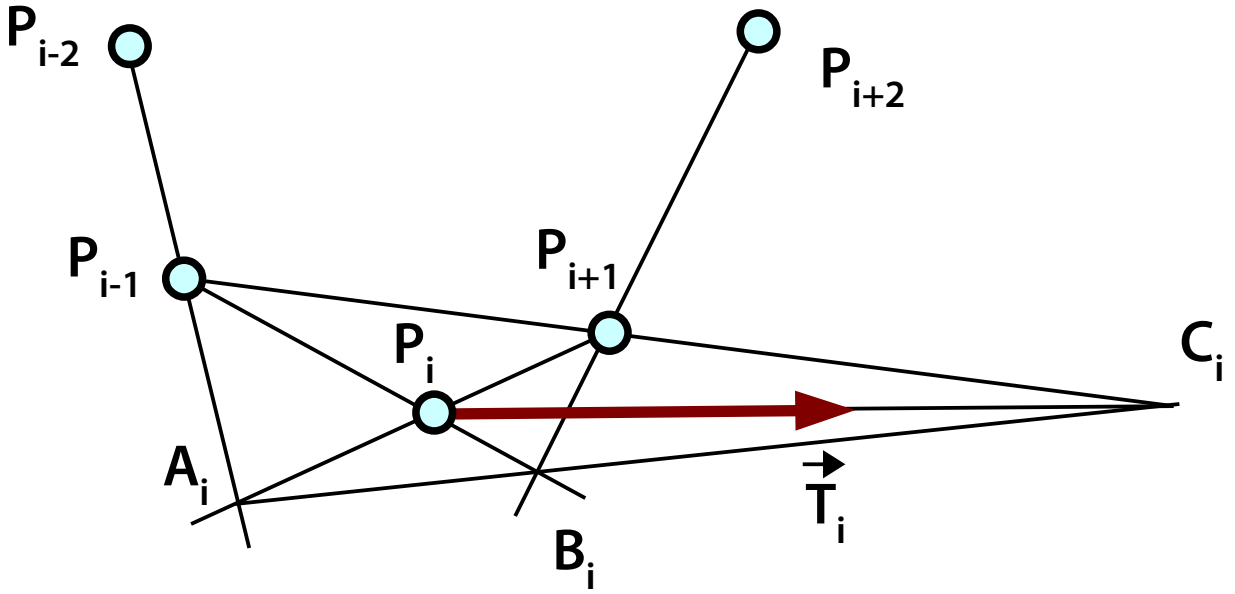
- a = t = 1/2

$$\boldsymbol{MG} = \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}$$

Akima interpolation

Alternative definition of **tangent vectors** for Hermite cubic:

- **non-C^2 !**

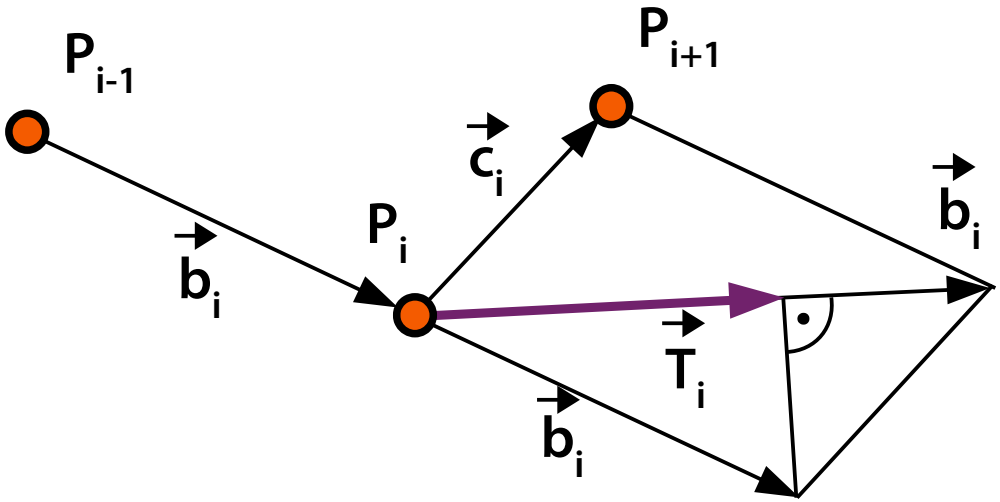


$$|\vec{T}_i| = |\overrightarrow{P_{i+1} - P_{i-1}}|$$

D-spline cubic

One more variant of Hermite cubic

– tangent vector computed by the **"D-interpolation"**



$$G = \begin{bmatrix} P_i \\ P_{i+1} \\ T_i \\ T_{i+1} \end{bmatrix}$$

Bèzier curves I

Polynomial curve of degree N

- **N+1** control points
  - » **boundary control points** define **endpoints** of a curve
  - » boundary control-point pairs define **tangent vectors**
- parametric expression using **Bernstein polynomials**
- easy $\mathbf{G^1}$ or $\mathbf{C^1}$ connection
- spline-join is also possible, but much more complicated

Bernstein polynomials:

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i} \quad 0 \leqslant i \leqslant n, \quad 0 \leqslant t \leqslant 1$$

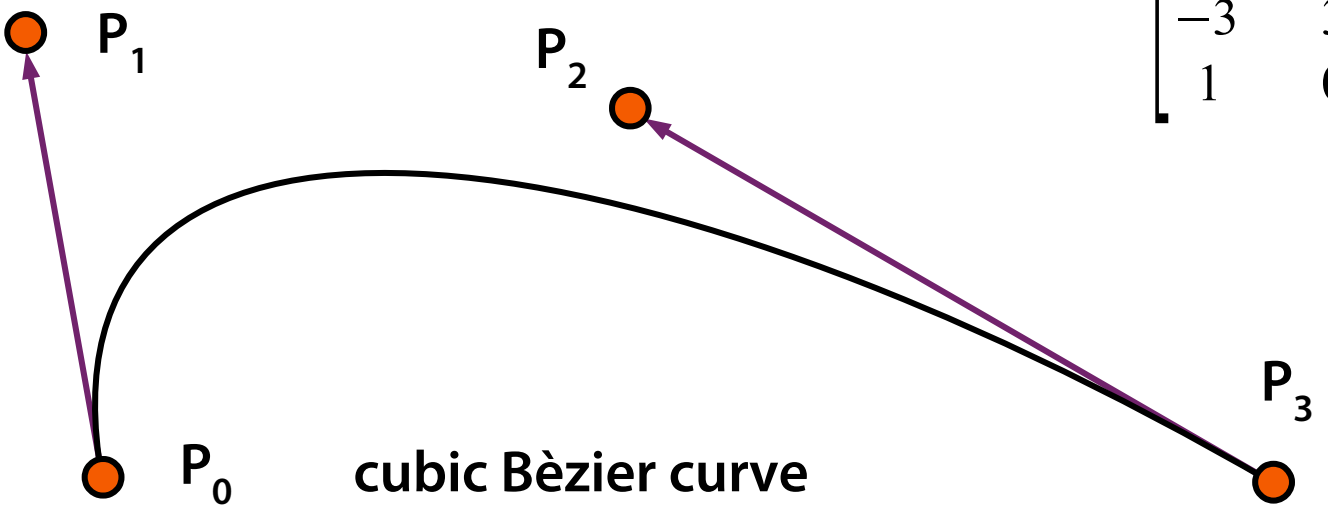
Bèzier curves II

Cauchy condition

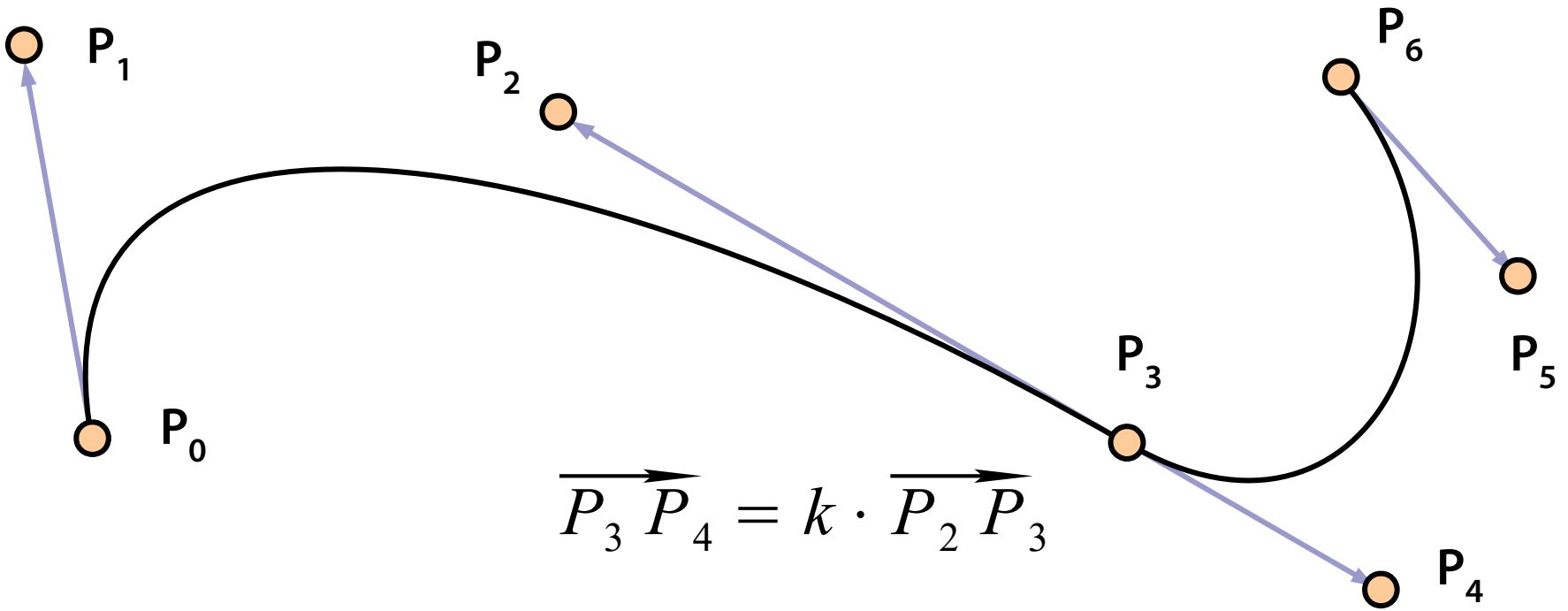
$\Rightarrow$ convex combination of control points

$$\sum_{i=0}^{n} B_i^n(t) = 1 \qquad for \quad 0 \leqslant t \leqslant 1$$

$$\boldsymbol{MG} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

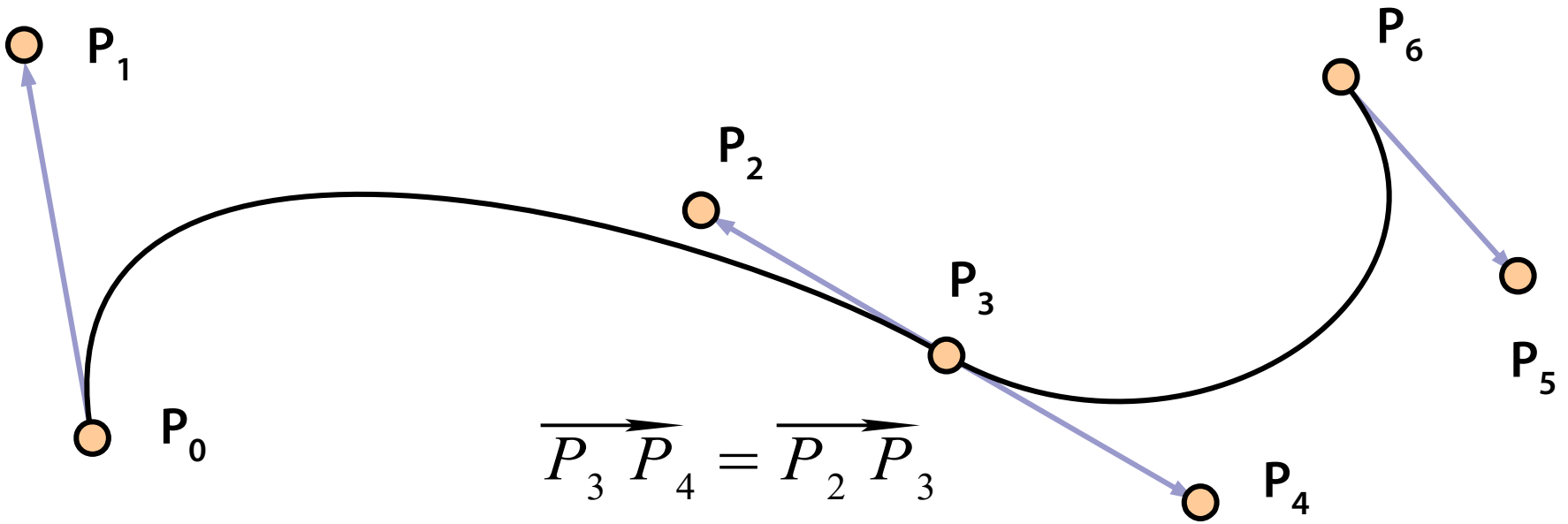


Joining Bèzier curves I



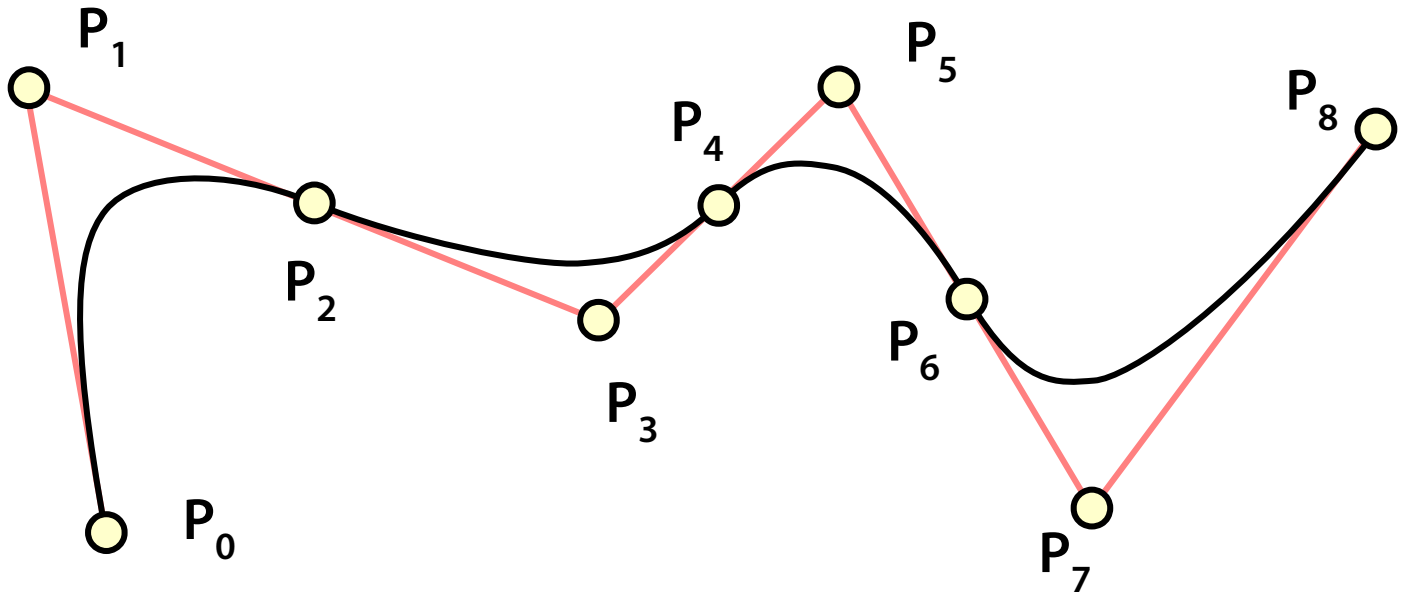
Joining Bèzier curves II

C^1 connection (equal tangent vectors)



Joining Bèzier curves III

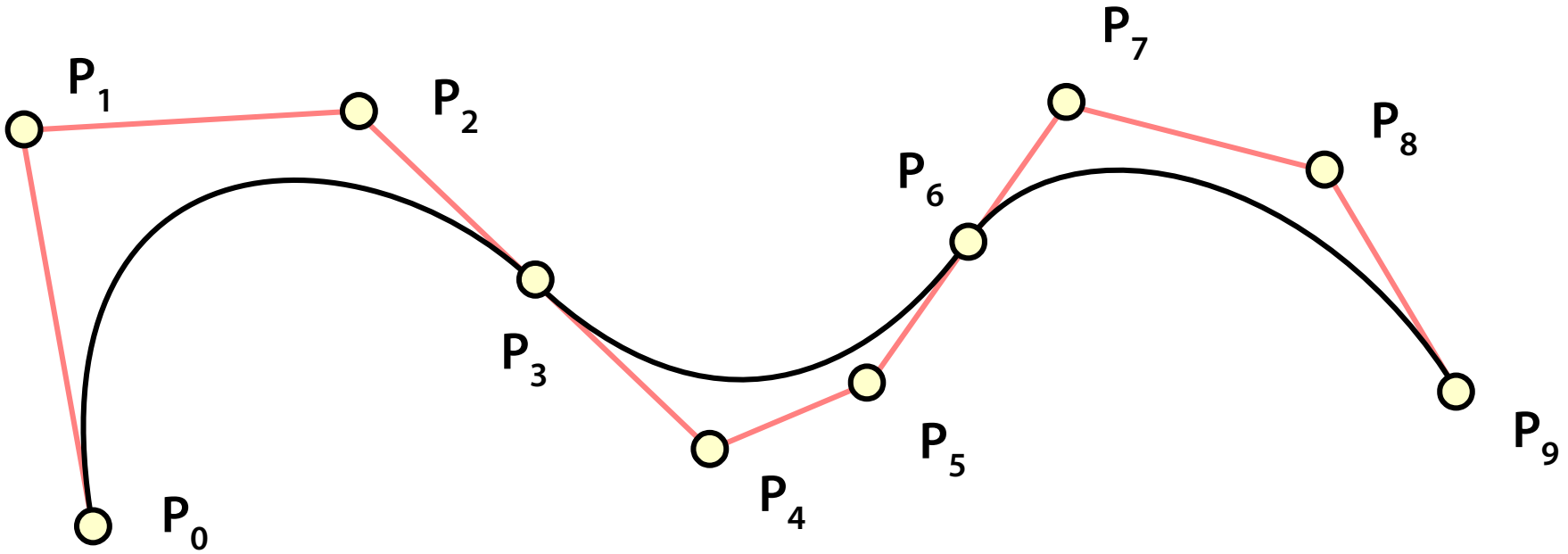
Quadratic spline from Bèzier segments



$$\overrightarrow{P_1P_2} = \overrightarrow{P_2P_3} \quad \overrightarrow{P_3P_4} = \overrightarrow{P_4P_5} \quad \dots \quad \overrightarrow{P_{2k-1}P_{2k}} = \overrightarrow{P_{2k}P_{2k+1}}$$

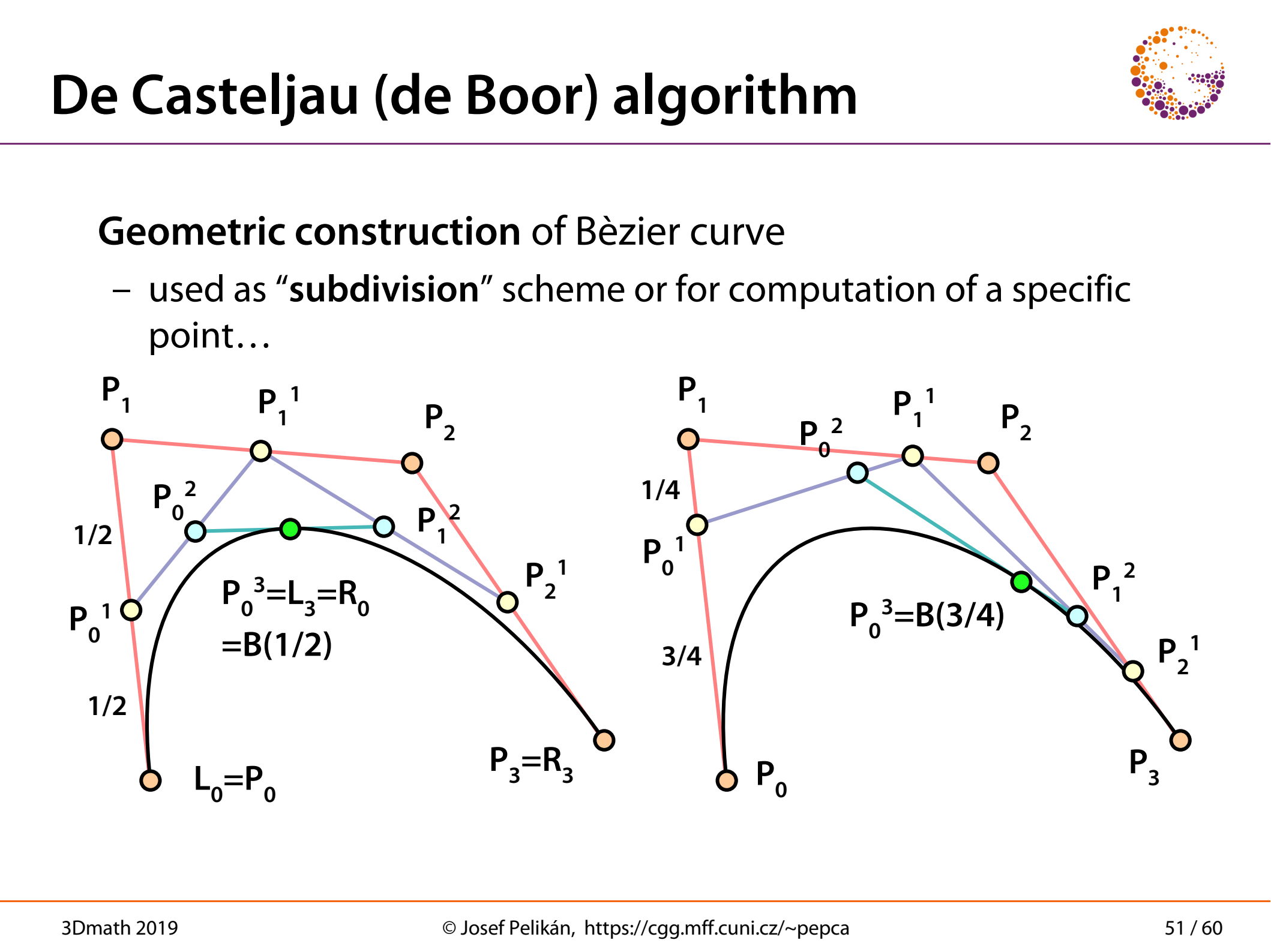
Joining Bèzier curves IV

Cubic spline from Bèzier segments



$$\overrightarrow{P_2P_3} = \overrightarrow{P_3P_4} \quad \overrightarrow{P_5P_6} = \overrightarrow{P_6P_7} \quad \dots \quad \overrightarrow{P_{3k-1}P_{3k}} = \overrightarrow{P_{3k}P_{3k+1}}$$

De Casteljau (de Boor) algorithm



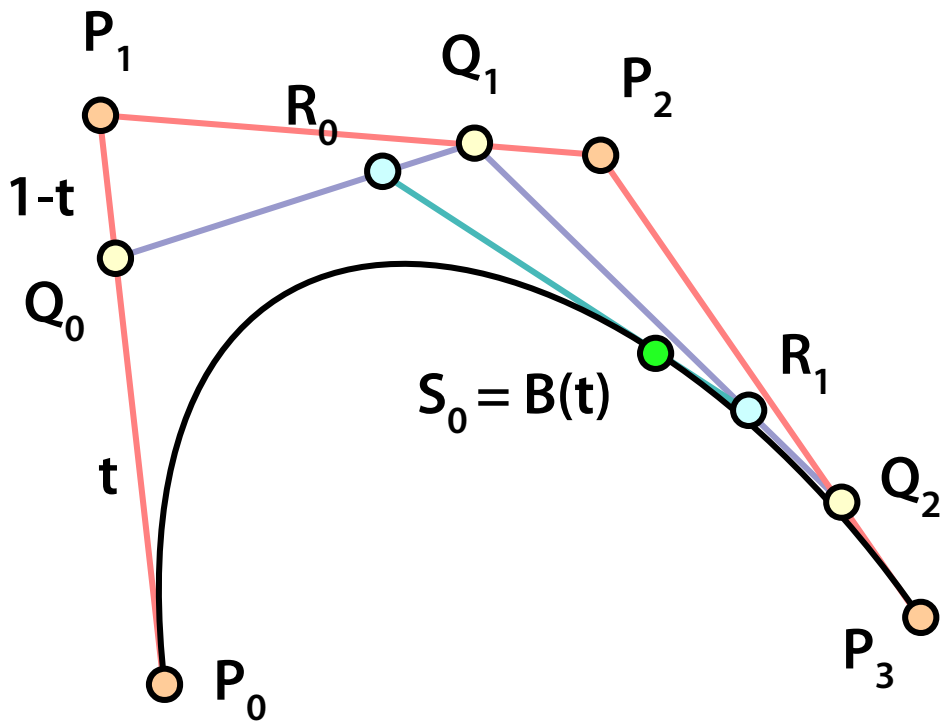
Geometric construction of Bèzier curve

- used as "**subdivision**" scheme or for computation of a specific point…

Using [S]LERP operation

Linear interpolation LERP (SLERP for quaternions)

$$\mathrm{LERP}(A, B, t) = A \cdot (1 - t) + B \cdot t$$



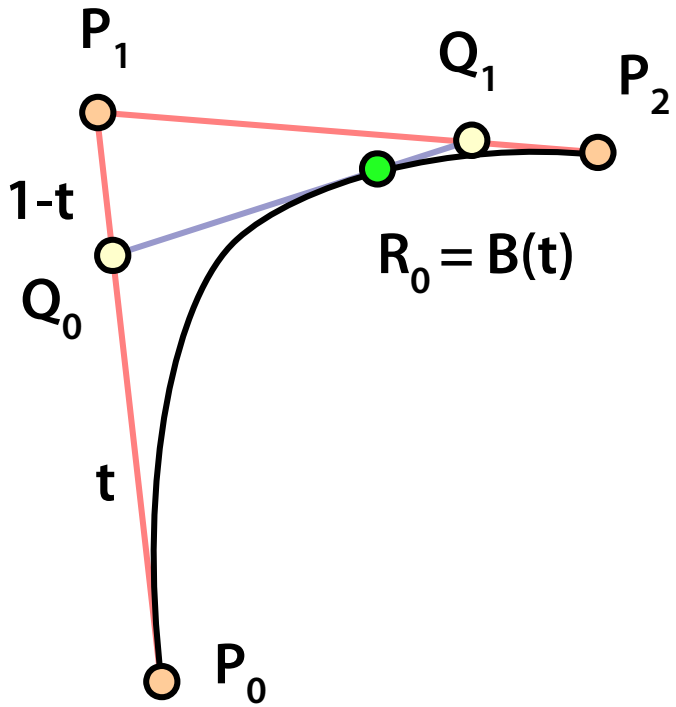
Cubic Bèzier

$$Q_i = \mathrm{LERP}(P_i, P_{i+1}, t)$$

$$R_i = \mathrm{LERP}(Q_i, Q_{i+1}, t)$$

$$S_i = \mathrm{LERP}(R_i, R_{i+1}, t)$$

[S]LERP for quadratic interpolation



Quadratic Bèzier

$$Q_i = \mathrm{LERP}(P_i, P_{i+1}, t)$$

$$R_i = \mathrm{LERP}(Q_i, Q_{i+1}, t)$$

Cubic spline

Function assembled from **cubic polynomials**

- neighbor polynomials have C^2 joint
- elastic "spline-ruler" (see construction)

Interpolating cubic spline

- in knot points $\mathbf{x_0}, \mathbf{x_1}, \ldots \mathbf{x_n}$ function values $\mathbf{y_0}, \mathbf{y_1}, \ldots \mathbf{y_n}$ are prescribed

$$S(x) = S_k(x) = s_{k,0} + s_{k,1}(x - x_k) + s_{k,2}(x - x_k)^2 + s_{k,3}(x - x_k)^3$$
$$x \in [x_k, x_{k+1}], \quad k = 0, 1, \ldots, n-1$$

Condition **A**: $S(x_k) = y_k \qquad k = 0, 1, \ldots, n$

Interpolating cubic spline

Condition **B** (C^0 continuity):

$$S_k(x_{k+1}) = S_{k+1}(x_{k+1}) \qquad k = 0, 1, \ldots, n-2$$

Condition **C** (C^1 continuity):

$$S_k'(x_{k+1}) = S_{k+1}'(x_{k+1}) \qquad k = 0, 1, \ldots, n-2$$

Condition **D** (C^2 continuity):

$$S_k''(x_{k+1}) = S_{k+1}''(x_{k+1}) \qquad k = 0, 1, \ldots, n-2$$

Natural cubic spline has an additional condition **E**:

$$S''(x_0) = S''(x_n) = 0$$

Natural cubic spline

Interpolating spline

- **uniquely determined** by the conditions (solution of linear system of equations $\mathbf{s}_{k,l}$)
- **has no local property** (the whole curve changes after altering one control point)

Open spline

- conditions **A**, **B**, **C**, **D** are not sufficient, two more DoF
- additional condition **E** (second derivatives at endpoints)

Closed (cyclic) **spline:** $\mathbf{x}_0 = \mathbf{x}_n$

- **C** and **D** give us missing conditions for $\mathbf{x}_0$

B-spline (basis spline)

"**Free-form**" curve

- shape is defined by a sequence of **control points**
- parametric form using **basis/blending functions** (dependency of a curve point on control polygon)
- **local property** (only local change after altering one CP)

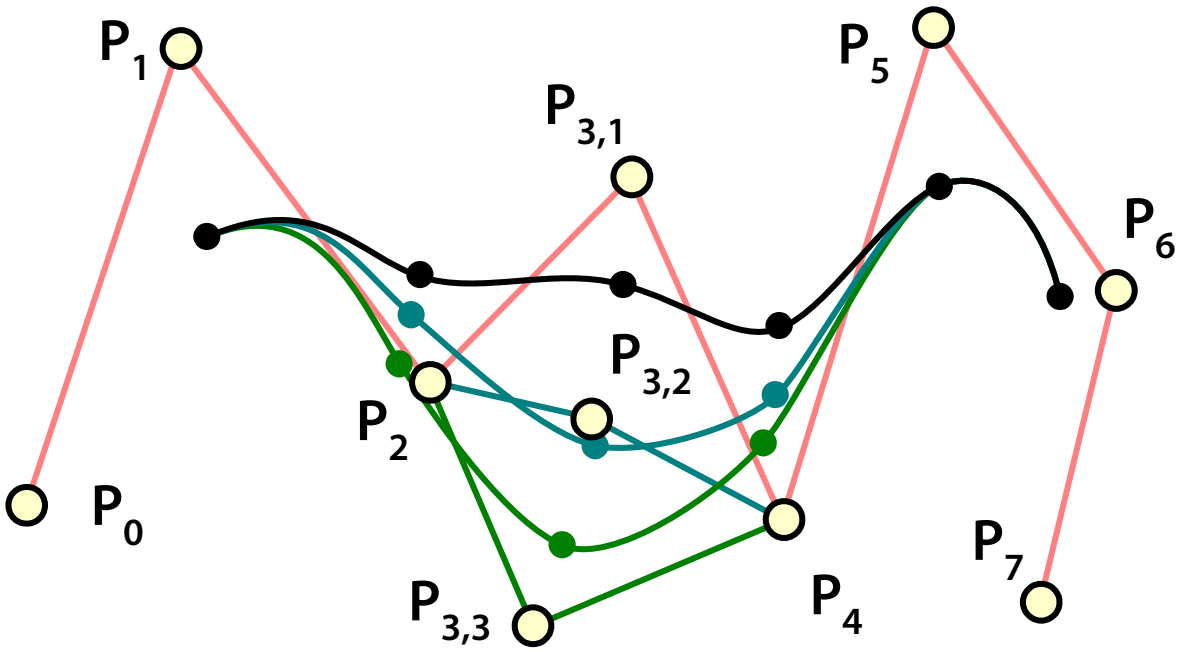
Uniform cubic B-spline (Coons curve)

- unified set of basis functions (cubic polynomials)

Nonuniform B-spline

- more complicated definition using knot vector $[\ \mathbf{t}_i\ ]_i \quad \mathbf{0} \le \mathbf{t}_i \le \mathbf{1}$

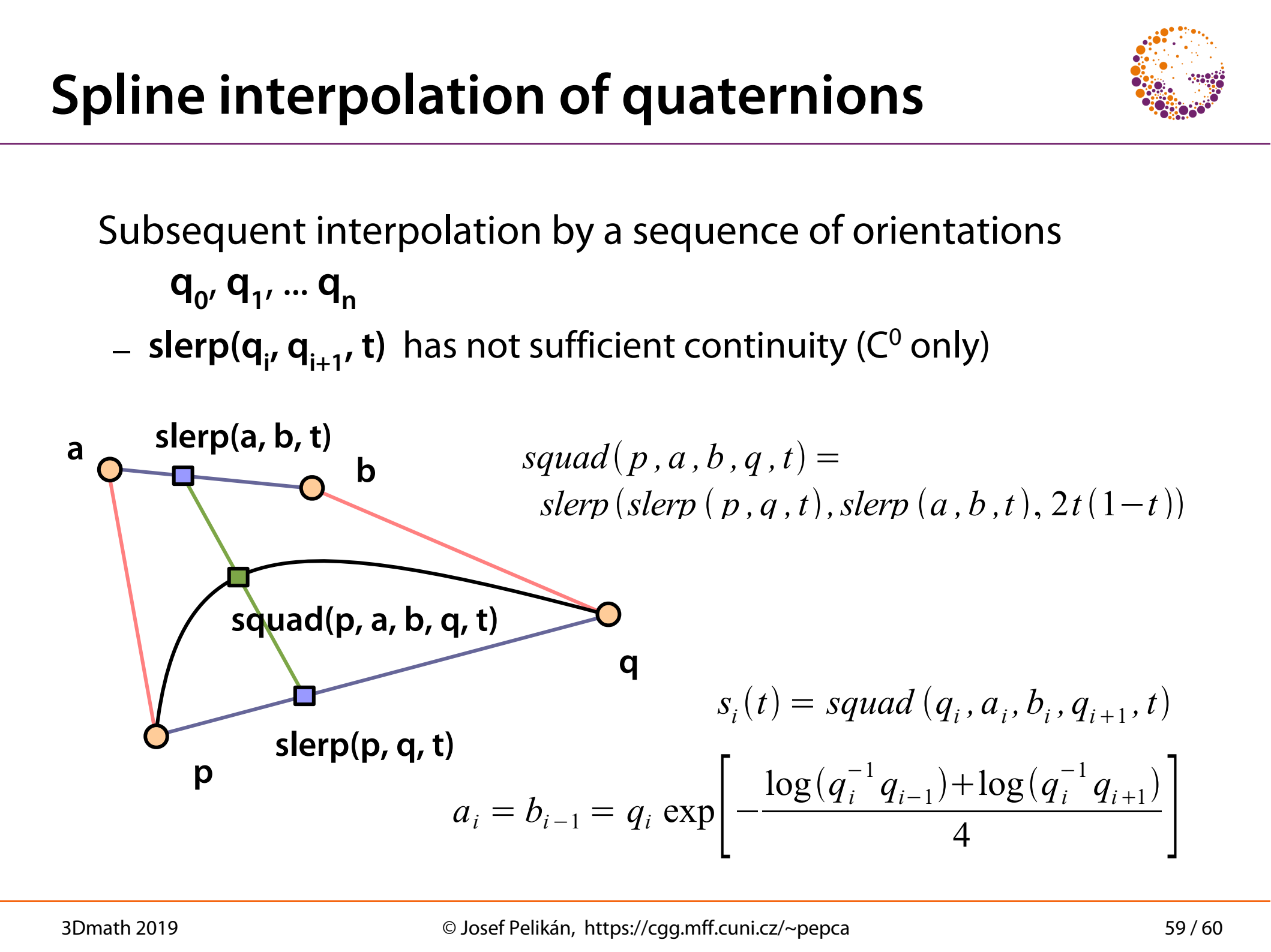
Coons B-spline



- continuity C^2
- **sharing** 3 CP between neighbours
- altering one CP induces change in closest **4 segments**

$$MG = \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}$$

Spline interpolation of quaternions



Subsequent interpolation by a sequence of orientations

$\mathbf{q_0}, \mathbf{q_1}, \ldots \mathbf{q_n}$

– **slerp(q_i, q_{i+1}, t)** has not sufficient continuity (C^0 only)

$$squad(p, a, b, q, t) = slerp(slerp(p, q, t), slerp(a, b, t), 2t(1-t))$$

$$s_i(t) = squad(q_i, a_i, b_i, q_{i+1}, t)$$

$$a_i = b_{i-1} = q_i \exp\left[-\frac{\log(q_i^{-1} q_{i-1}) + \log(q_i^{-1} q_{i+1})}{4}\right]$$

Literature

Tomas Akenine-Möller, Eric Haines et al.: ***Real-time rendering, 4th edition***, A K Peters, 2018, ISBN: 9781138627000

J. Žára, B. Beneš, J. Sochor, P. Felkel: ***Moderní počítačová grafika***, 2. vydání, Computer Press, 2005, ISBN: 8025104540

http://www.geometrictools.com/ (Dave Eberly)

matfyz

Computer Graphics Charles University

Základy OpenGL

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/

Pokroky v hardware

3D akcelerace běžná i v konzumním sektoru

Hry, multimedia i mobilní app. (OpenGL ES)

Vzhled – kvalita prezentace

- velmi důmyslné techniky texturování
- kombinace mnoha textur, modularita zpracování

Vysoký výkon

- nejmodernější čipové technologie pro výrobu GPU (NVIDIA Volta, Turing: 12 nm), **masivní paralelismus**
- velmi rychlé **paměti** (vícecestný přístup, GDDR6, HBM2)
- výjimečné sběrnice mezi GPU a CPU (dnes PCI-E)

Pokroky v software

Dvě hlavní **knihovny** pro 3D grafiku

- **OpenGL/Vulkan** (SGI, open standard) a **Direct3D** (Microsoft)
- přístup je podobný, API je velmi ovlivněno hardwarem

Nastavení parametrů a **úsporný přenos dat** do GPU

- maximální sdílení společných datových polí

Programování grafického řetězce!

- revoluce v programování 3D grafiky
- ***„vertex shader"*** – zpracování vrcholů
- ***„geometry/tesselation/mesh sh."*** – generování dalších elementů
- ***„fragment shader"*** (*„pixel shader"*) – zpracování jednotlivých fragmentů/pixelů před vykreslením

Vývojové nástroje

Příjemné pro programátory i „kreativce“

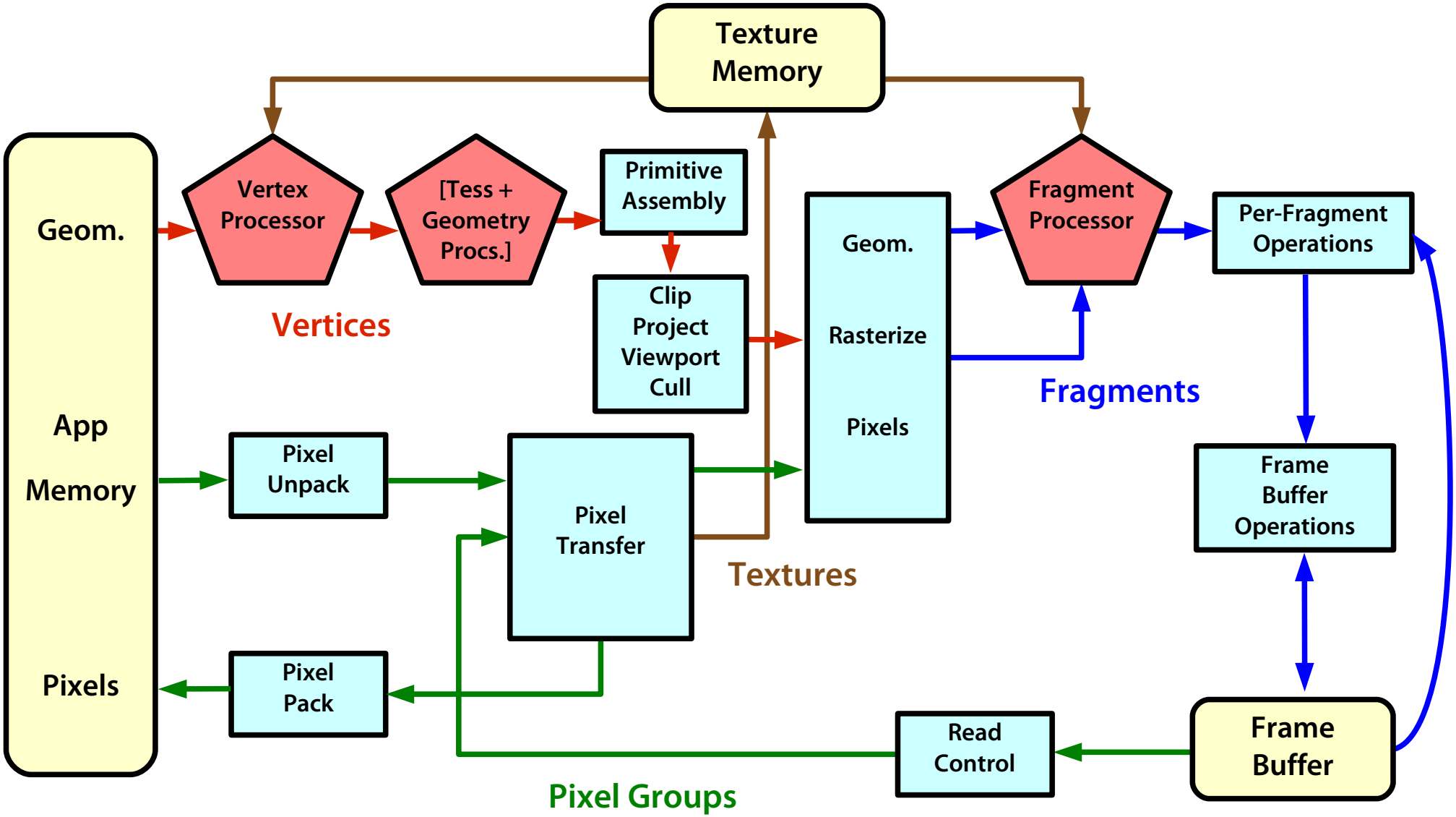
Vyšší jazyky pro programování GPU

- GLSL (OpenGL), HLSL (DirectX), ~~Cg (NVIDIA)~~
- ~~Cg~~ a HLSL jsou téměř shodné

Kompozice grafických efektů

- kompaktní popis celého efektu (GPU programy, odkazy na data) v jednom souboru
- DirectX **.FX** formát, NVIDIA **CgFX** formát
- nástroje: Effect Browser (Microsoft), **FX Composer** (NVIDIA), **RenderMonkey** (ATI)

Schéma OpenGL GPU



OpenGL – geometrická primitiva

Typy geometrických primitiv

- **bod**, **úsečka**, lomená čára, smyčka
- polygon, **trojúhelník**, proužek trojúhelníků, vějíř trojúhelníků, čtyřúhelník, proužek čtyřúhelníků…

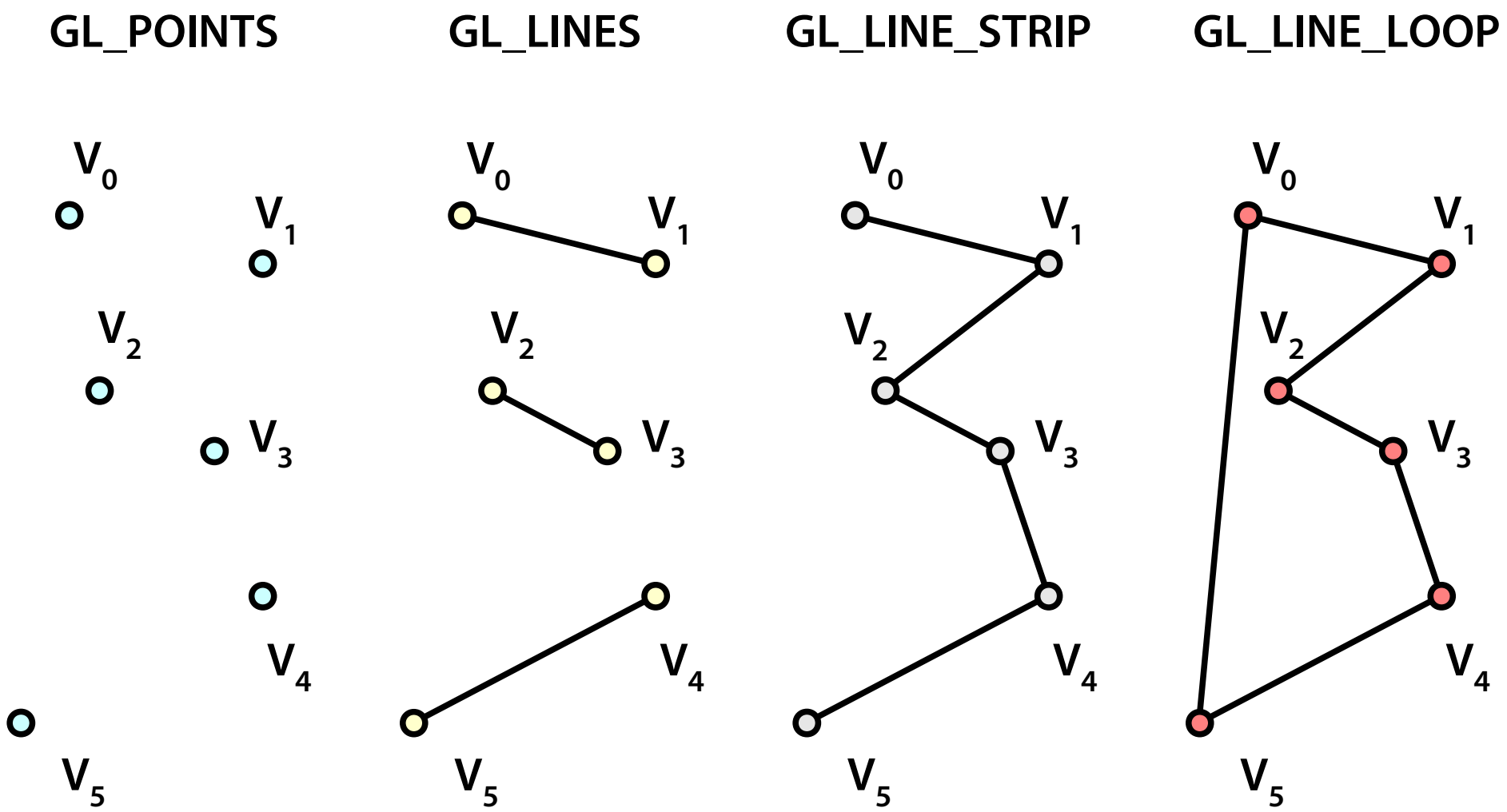
Zpracování **jednotlivých vrcholů**

- glVertex, glColor, glNormal, glTexCoord, ...
- neefektivní (mnoho volání gl* funkcí)

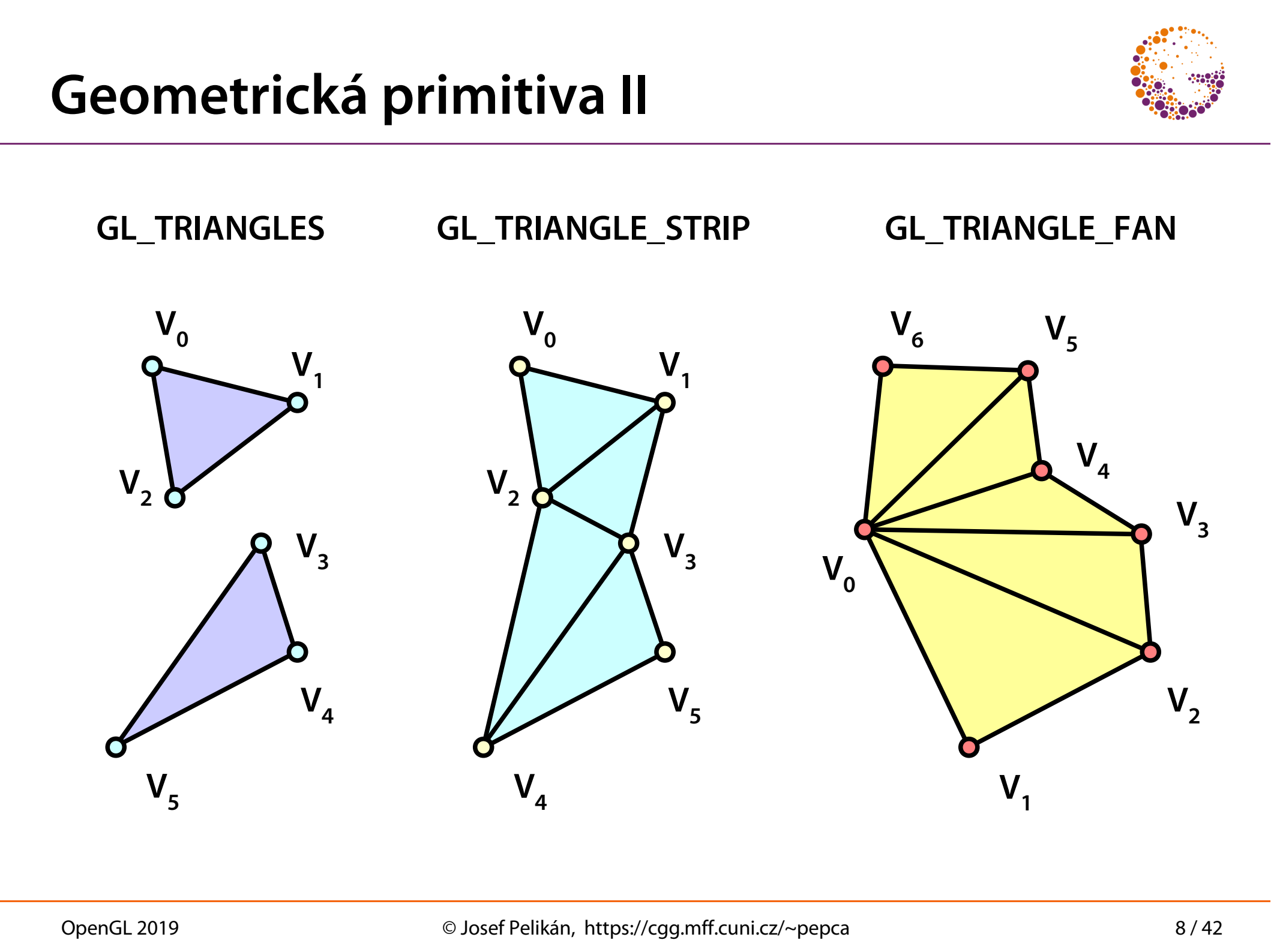
Pole vrcholů

- glDrawArrays, glMultiDrawArrays, glDrawElements …
- glColorPointer, glVertexPointer… nebo **prokládání**

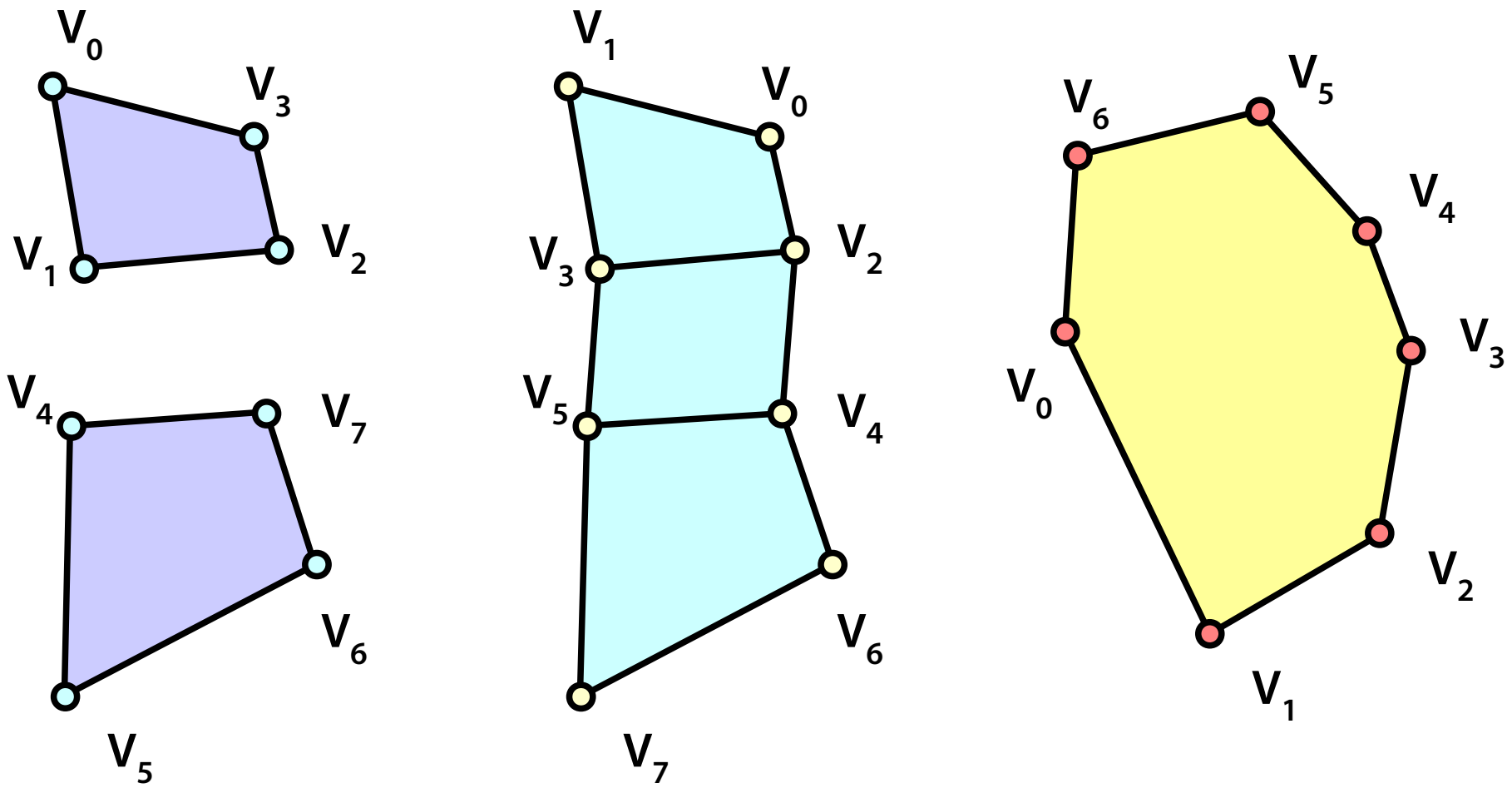
Geometrická primitiva I



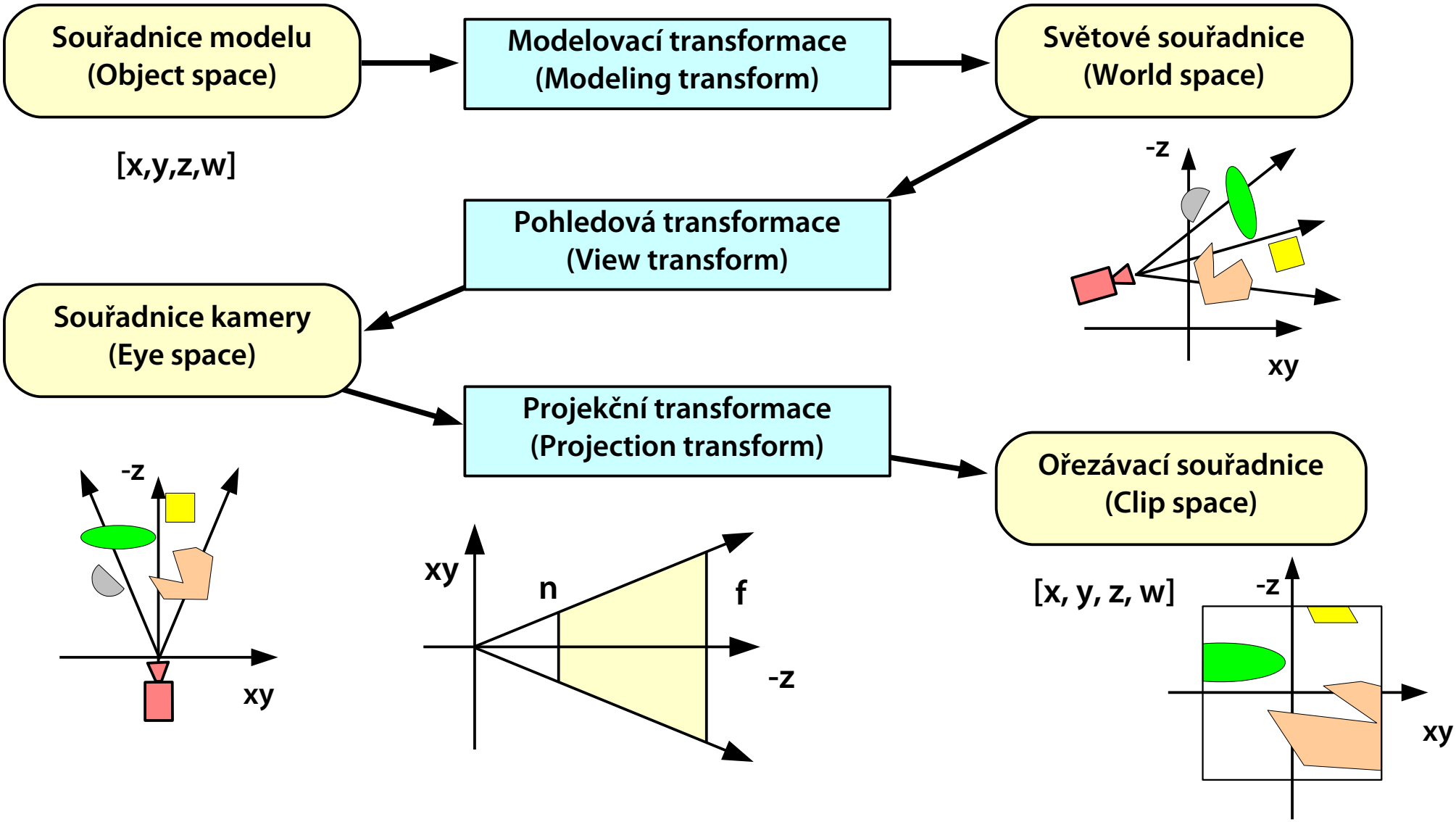
Geometrická primitiva II



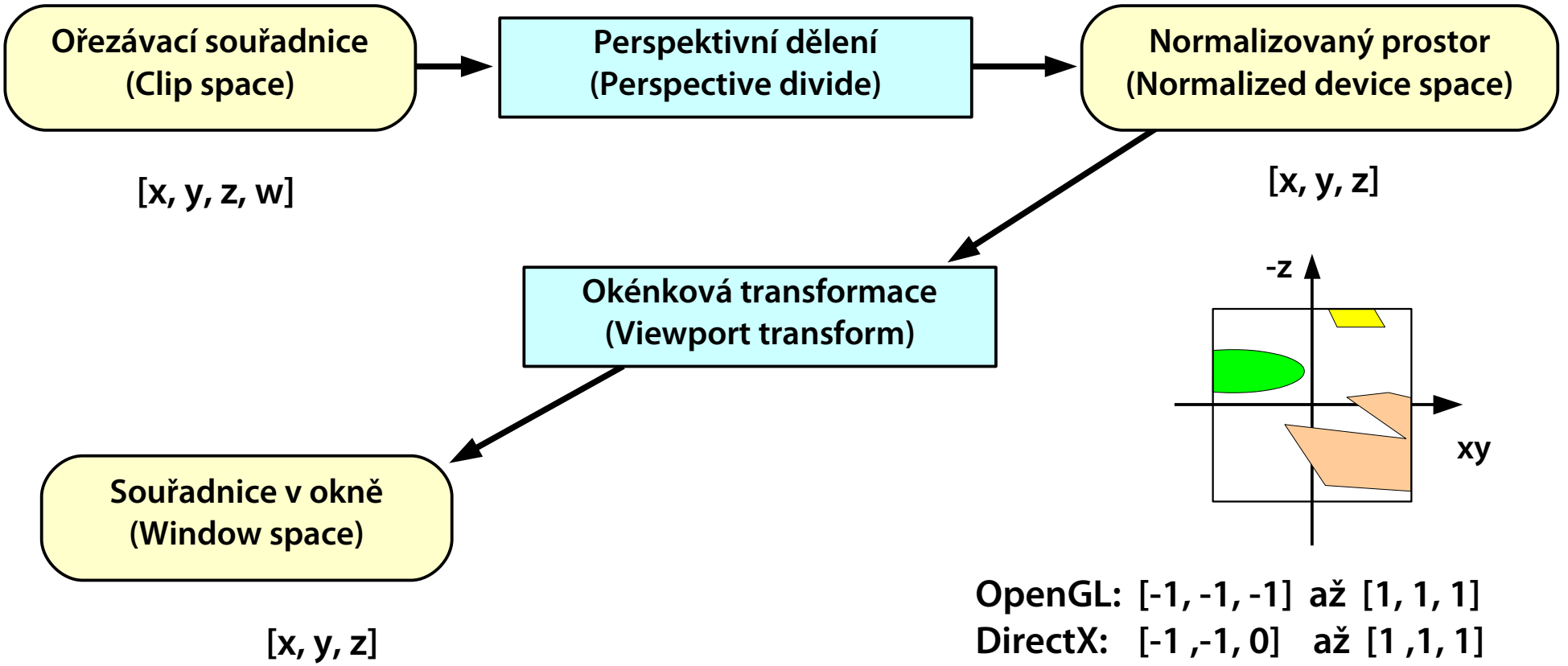
Geometrická primitiva III



Souřadné soustavy



Souřadné soustavy II



[x, y] skutečná velikost v pixelech na obrazovce (fragmenty)
z hloubka kompatibilní s z-bufferem

Souřadné soustavy III

Souřadnice modelu

- databáze objektů, ze kterých se skládá scéna
- 3D modelovací programy (3DS Max, Blender, Maya…)

Světové souřadnice

- absolutní souřadnice virtuálního 3D světa
- vzájemná poloha jednotlivých **instancí objektů**

Souřadnice kamery

- 3D svět se transformuje do relativních souřadnic kamery
- střed projekce: **počátek**, směr pohledu: **-z** (nebo **z**)

Souřadné soustavy a transformace

Transformace model → kamera

- společná transformační matice „model-view"
- světové souřadnice nejsou moc důležité

Projekční transformace

- definuje zorný objem = „**frustum**" [**l**, **r**, **b**, **t**, **n**, **f**]
- přední a zadní ořezávací vzdálenost: **n**, **f**
- výsledkem je homogenní souřadnice (před ořezáním)

Ořezávací souřadnice („clip space")

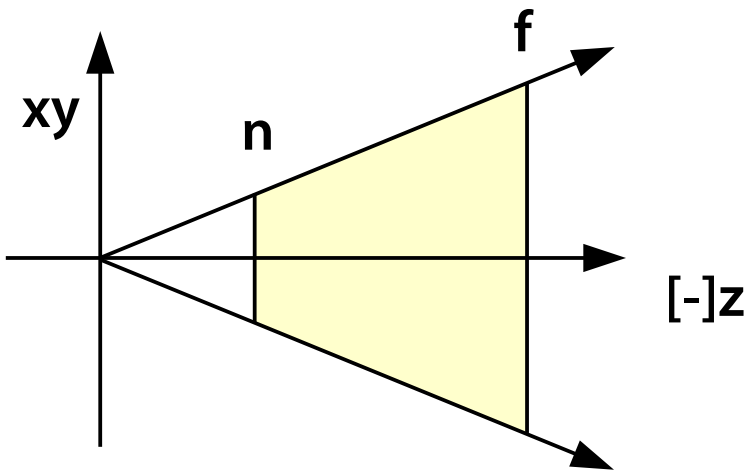
- **výstupní souřadnice** vertex shaderu!

Projekční transformace (perspektiva)

Vzdálený bod **f** může být i v nekonečnu

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ -\frac{r+l}{r-l} & -\frac{t+b}{t-b} & \frac{f+n}{f-n} & 1 \\ 0 & 0 & -\frac{2fn}{f-n} & 0 \end{bmatrix}$$

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ -\frac{r+l}{r-l} & -\frac{t+b}{t-b} & 1 & 1 \\ 0 & 0 & -2n & 0 \end{bmatrix}$$



Souřadné soustavy a transformace

Perspektivní dělení

- pouze převádí **homogenní** souřadnice do **kartézských**

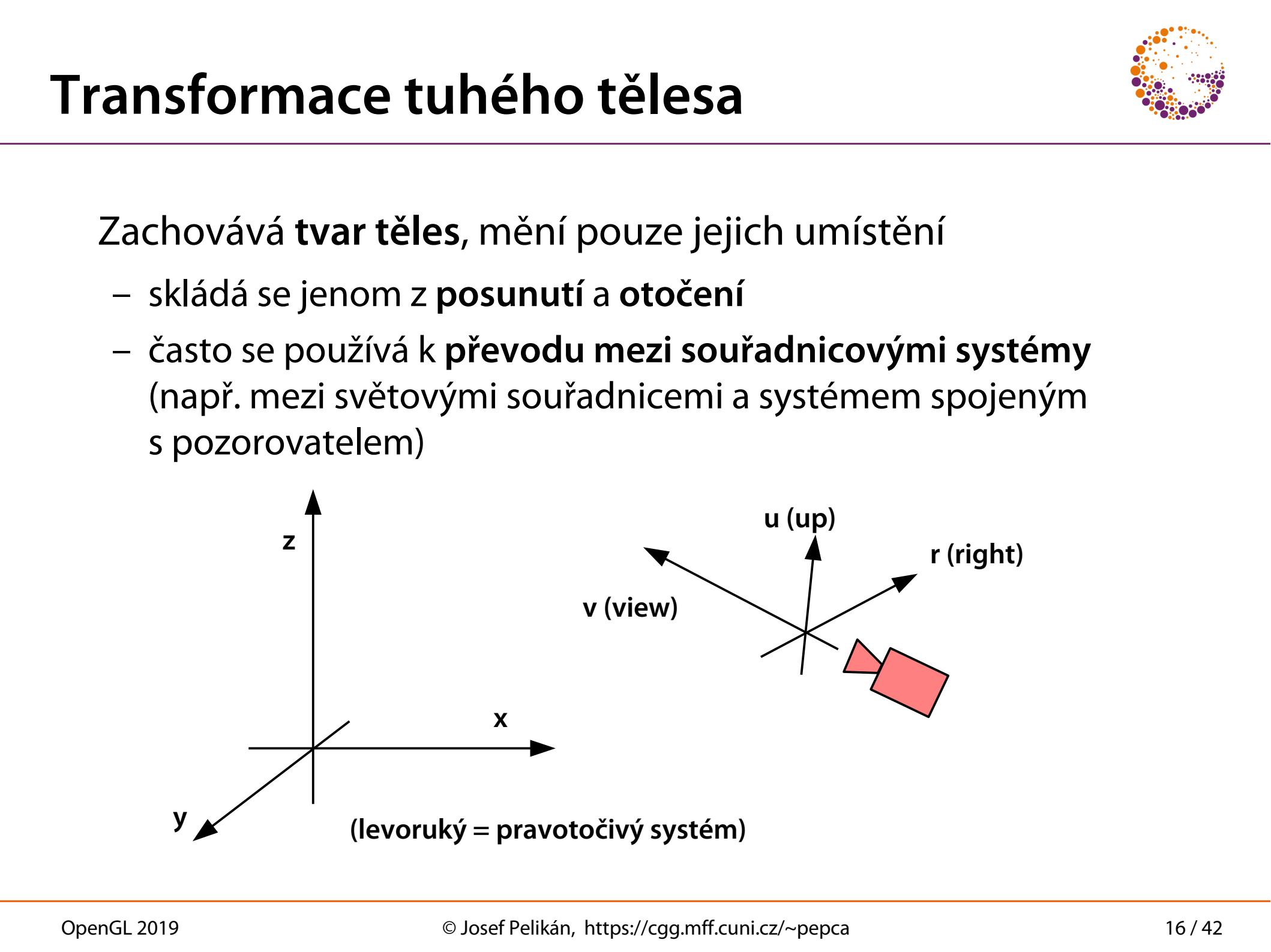
Normalizované souřadnice zařízení („**NDC**”)

- kvádr standardní velikosti
- OpenGL: [-1, -1, -1] až [1, 1, 1]
- DirectX: [-1, -1, 0] až [1, 1, 1]

Souřadnice v okně („window space”)

- výsledkem okénkové transformace a transformace hloubky
- používají se při **rasterizaci** a práci s **fragmenty**

Transformace tuhého tělesa

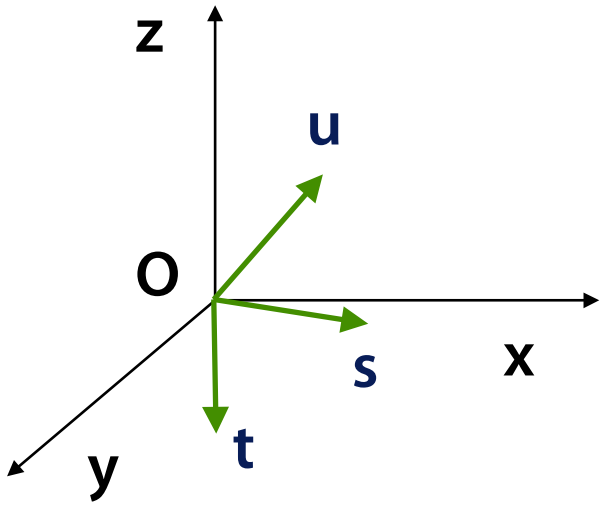


Zachovává **tvar těles**, mění pouze jejich umístění

- skládá se jenom z **posunutí** a **otočení**
- často se používá k **převodu mezi souřadnicovými systémy** (např. mezi světovými souřadnicemi a systémem spojeným s pozorovatelem)

(levoruký = pravotočivý systém)

Převod mezi dvěma orientacemi



Souřadný systém má počátek v **O** a je zadán trojicí jednotkových vektorů [**s, t, u**]

$$[1,0,0]\cdot M_{stu\to xyz} = s$$
$$[0,1,0]\cdot M_{stu\to xyz} = t$$
$$[0,0,1]\cdot M_{stu\to xyz} = u$$

$$M_{stu\to xyz} = \begin{bmatrix} s_x & s_y & s_z \\ t_x & t_y & t_z \\ u_x & u_y & u_z \end{bmatrix}$$

$$M_{xyz\to stu} = M^T_{stu\to xyz}$$

Geometrická data na GPU

VBO, **VAO**, počátky od **OpenGL 1.5** (2003)

- pro .NET nutností (klientská paměť není fixována)

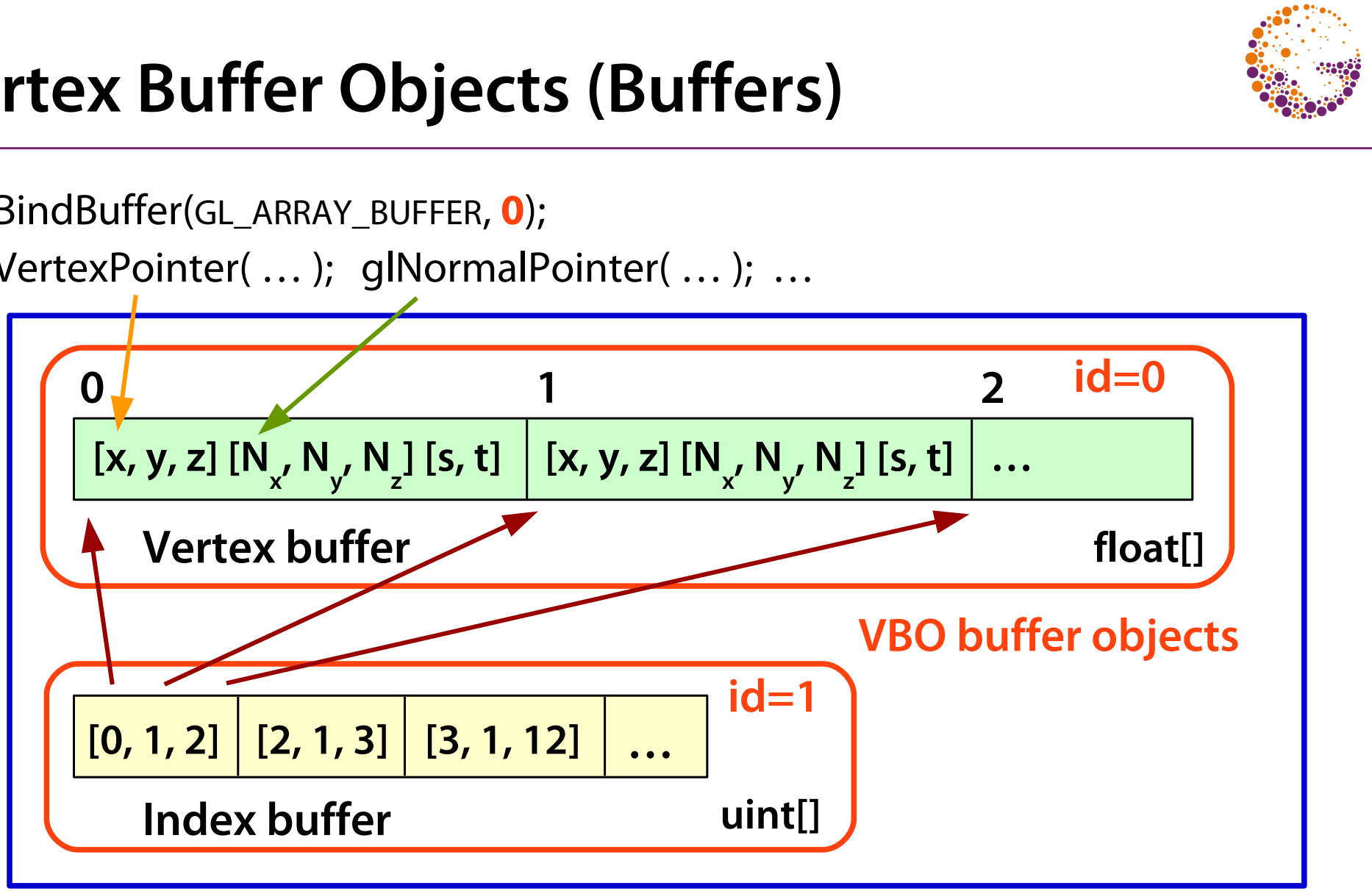
Buffer **na straně GPU (grafického serveru)** obsahující geometrická data

- založení bufferu: glBindBuffer
- **zadání dat** z pole: glBufferData, glBufferSubData
- **mapování** do paměti aplikace: glMapBuffer, glUnmap…

Práce s **klientskou pamětí** nebo s **bufferem**

- glColorPointer, glNormalPointer, glVertexPointer…

Vertex Buffer Objects (Buffers)



Zpracování vrcholu (shaders)

Povinný „**Vertex shader**“

Transformace vrcholů modelovacími a projekčními maticemi

- glMatrixMode
- glLoadIdentity, glLoadMatrix, [glMultMatrix]
- [glRotate, glScale, glTranslate…]

Sestavení a zpracování primitiv

Volitelné „**Tesselation**“ a/nebo „**Geometry shader**“

Sestavení (Assembly)

- určení, kolik vrcholů primitivum potřebuje
- shromáždění balíčku dat a odeslání dál

Zpracování primitiv

- ořezávání („clipping”)
- projekce do zorného objemu („frustum”) – dělení „w“
- projekce a ořezání do 2D okénka („viewport”)
- odstranění odvrácených stěn („culling”)
  - » jednostranná vs. oboustranná primitiva

Rasterizace, fragmenty

Rasterizace = vykreslení vektorových primitiv

- rozklad geometrických objektů na **fragmenty**
- geometrické objekty: body, úsečky, trojúhelníky, bitmapy

Fragment

- **rastrový element**, který **potenciálně** přispívá k barvě nějakého px.
- velikost: stejná nebo menší než u pixelu (anti-aliasing)
- „balíček dat" procházející rasterizační jednotkou GPU
  - » vstup/výstup: **x**, **y**, **z** (pouze hloubku lze měnit!)
  - » texturovací souřadnice $\mathbf{t}_0$ až $\mathbf{t}_n$
  - » lesklá a difusní barva, koeficient mlhy, uživatelská data…
  - » výstupní barva **RGB** a neprůhlednost **α** (frame-buffer op.)

Interpolace ve fragmentech

Atributy fragmentů se automaticky **interpolují z hodnot ve vrcholech**

- hloubka (**z** nebo **w**)
- texturové souřadnice
- barvy (lesklá a difusní složka)
- uživatelské atributy…

Rychlé **HW interpolátory**

Perspektivně korektní interpolace

- jen [**x, y**] se mění lineárně
- ostatní veličiny vyžadují jedno dělení na každý fragment

Zpracování fragmentů (shader)

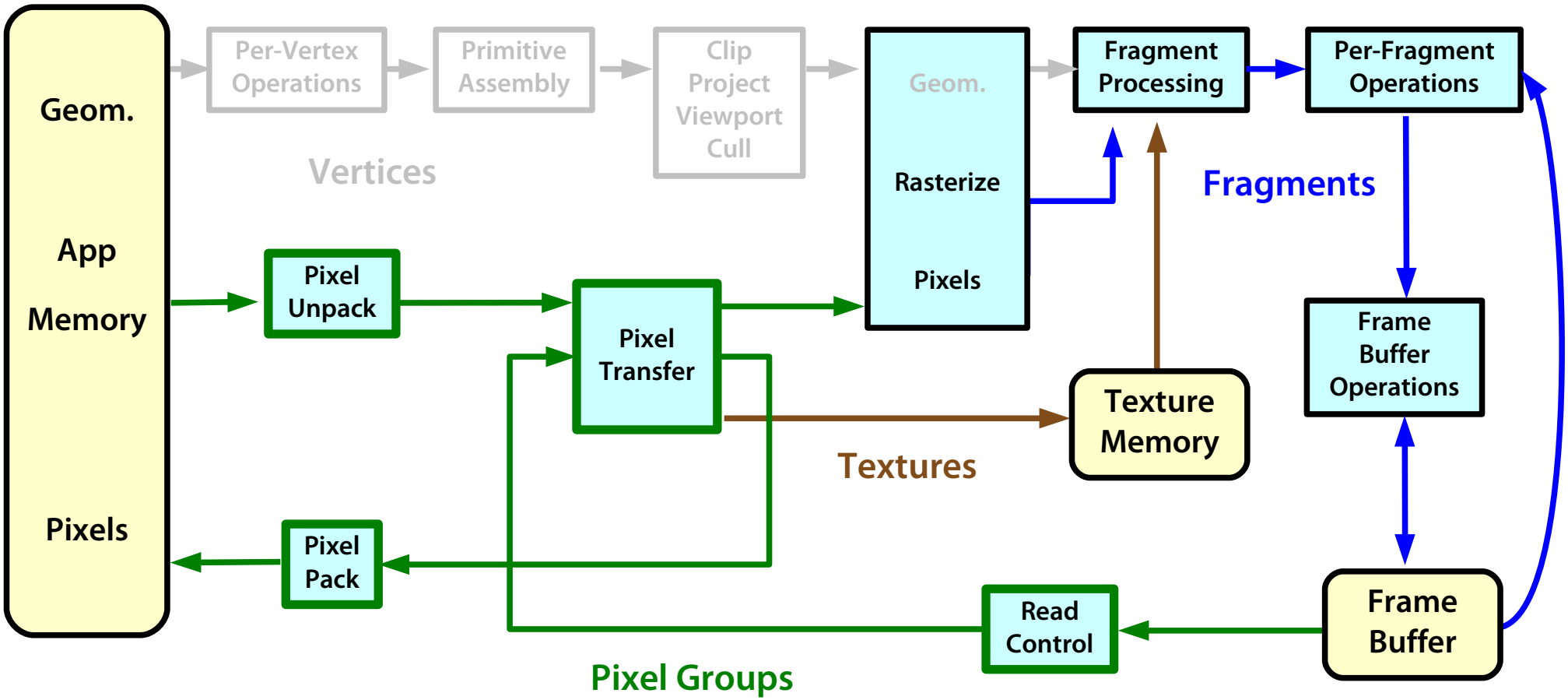
Povinný „**Fragment shader**“

Texturovací operace

- maximálně **optimalizované** operace
- výběr barvy z texturovací paměti
- interpolace texelů
  - » mip-mapping, anisotropic filtering…
- kombinace několika textur (výběr z mnoha operací)
- zvláštní efekty (bump-mapping, environment mapping)

Kombinace primární a sekundární barvy (diffuse, specular)

Rastrové obrázky v OpenGL



Vertex shader

Modul zpracování vrcholů

- transformace vrcholů
- transformace a normalizace normálových vektorů
- výpočet/transformace texturovacích souřadnic
- výpočet osvětlovacích vektorů
- nastavení materiálových konstant do vrcholů

Nemůže ovlivnit

- **počet vrcholů!** (nelze přidat ani ubrat vrchol*)
  - » částečné řešení: degenerace primitivu
- typ / topologii geometrických primitiv
  - » řešení: geometry shader

Fragment shader

Modul zpracování fragmentů

- aritmetické operace s interpolovanými hodnotami
- čtení dat z textur
- aplikace textur
- výpočet mlhy
- závěrečná syntéza barvy fragmentu
- možnost modifikace hloubky fragmentu

Nemůže ovlivnit

- **počet fragmentů!** (nelze přidat fragment*)
  - » zrušení fragmentu: discard
- **polohu fragmentu** na obrazovce [x, y]

Shader uniforms (constants)

Hodnoty, které se nemění příliš často

- jsou **konstantní během jedné kreslicí dávky** (batch)
- aplikace je předává do GPU jednotlivě nebo přes buffery
- omezený objem dat (až 1024 vektorů)
  - » data většího rozsahu se už musí předávat přes textury/buffery

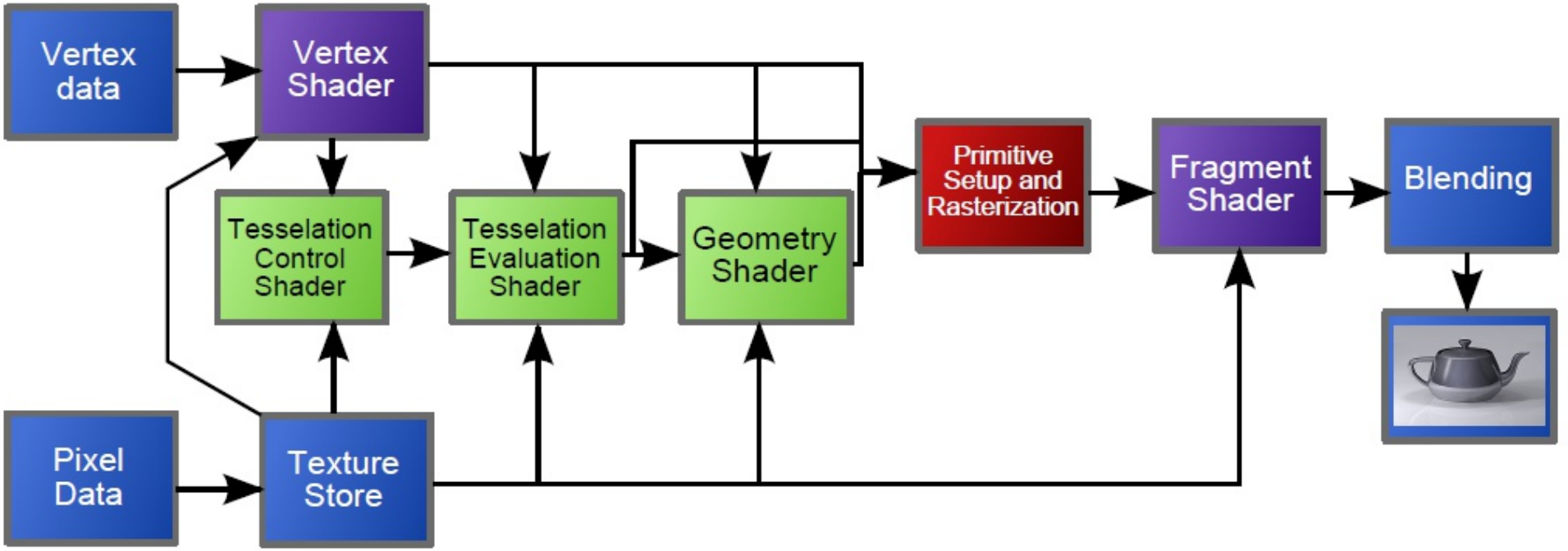
Typické uniforms/konstanty

- **transformační matice** (matrix stack)
  - » **model matrix** (mění se s každou instancí objektu)
  - » **view-projection matrix** (mění se jednou za snímek)
- mody vykreslování
- poloha světel, kamery a další hodnoty pro stínování

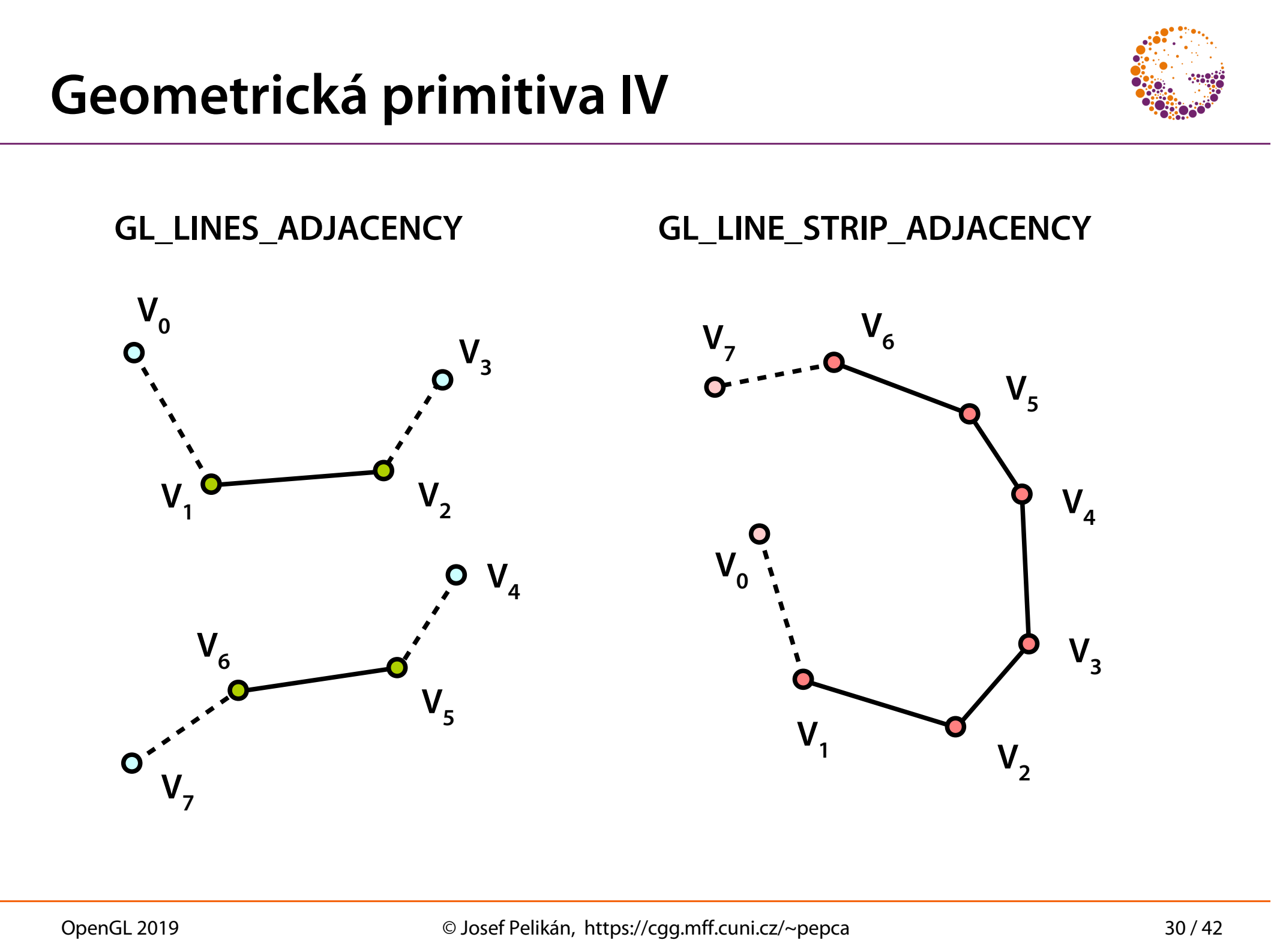
„Novinky“ v OpenGL 3.x+ (2009-)

Dva další kroky **zpracování geometrie na GPU**

- Geometry shader (OpenGL 3.2+)
- Tesselation shaders (OpenGL 4.0+)



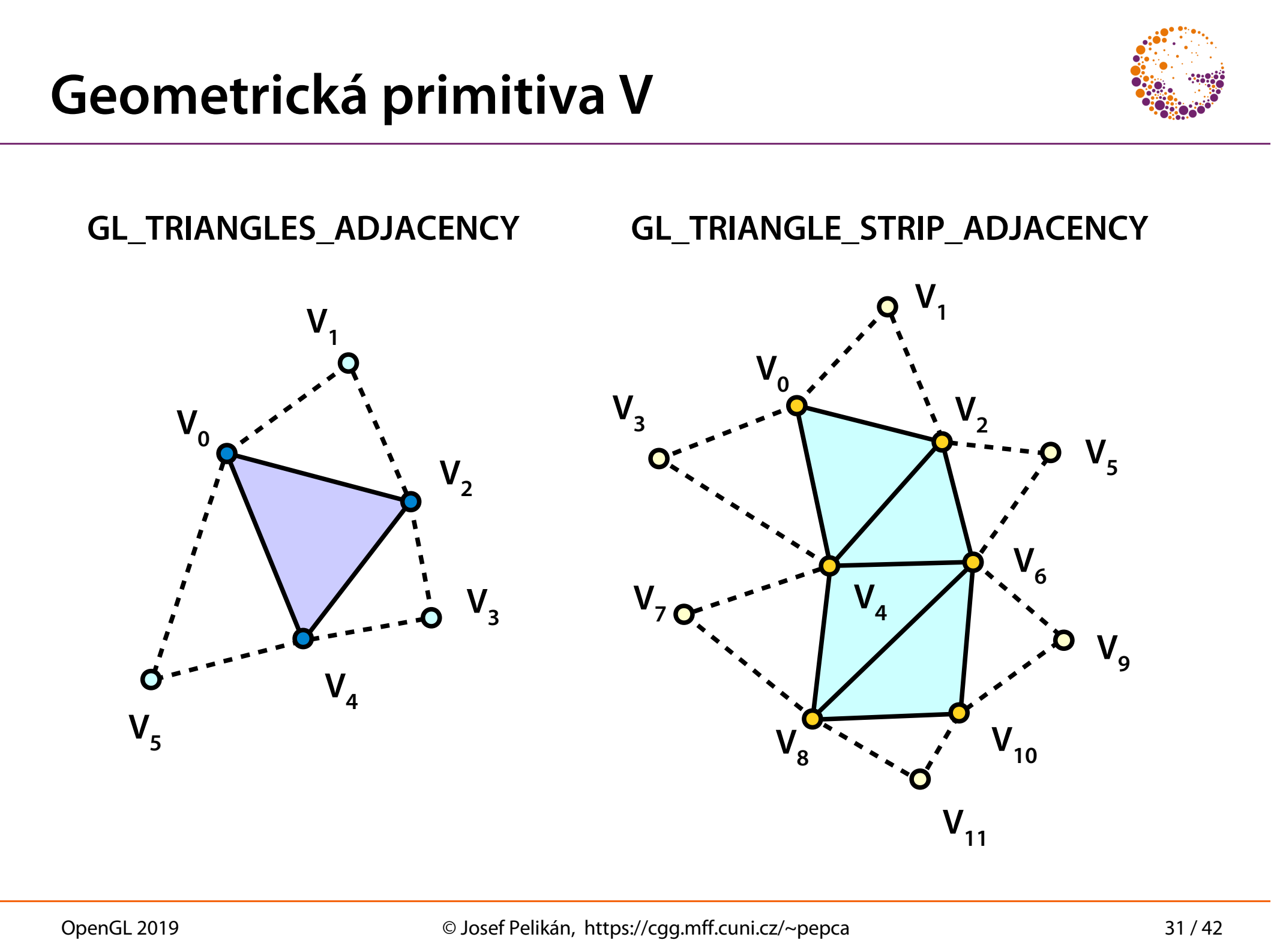
Geometrická primitiva IV



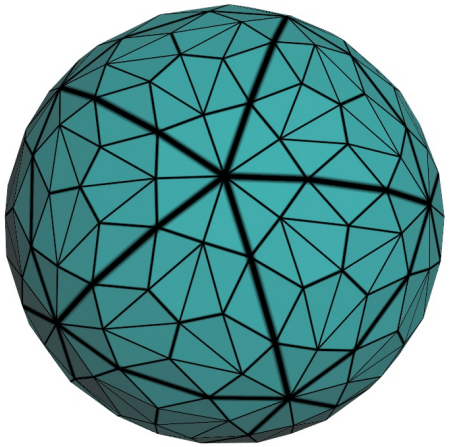
GL_LINES_ADJACENCY

GL_LINE_STRIP_ADJACENCY

Geometrická primitiva V



Geometrické procesory



Tesselation shaders

- nově v OpenGL 4.0
- HW podporované dělení ploch, subdivision (spline pláty…)
- dva shadery: „tesselation control“ a „tesselation evaluation“
- první definuje topologii, druhý počítá geometrii (koeficienty)

Geometry shader

- od OpenGL 3.2
- těsně před rasterizační jednotkou
- možnost pracovat s celými primitivy
- obecnější než TS, avšak pomalejší (na jednoduchá schémata dělení se nehodí)

Programování procesorů v GPU

Vertex shader, **Fragment shader**…

– kód zavedený ve vrcholovém, fragmentovém… procesoru

Aplikační programátor může tyto kódy měnit!

– **HW nezávislé*** programovací jazyky

– **strojový kód pro GPU** se kompiluje až v době běhu aplikace (RT optimalizace, různé profily/verze)

– low-level instrukce (jazyk se podobá assembleru)

– nebo **vyšší jazyky** ~~Cg~~, HLSL, GLSL (podobné)

NVIDIA Microsoft OpenGL

GLSL example (vertex shader)

```
#version 130

in vec4 position;
in vec3 normal;
in vec2 texCoords;
in vec3 color;

out vec2 varTexCoords;
out vec3 varNormal;
out vec3 varWorld;
out vec3 varColor;
flat out vec3 flatColor;

uniform mat4 matrixModelView;
uniform mat4 matrixProjection;

void main ()
{
  gl_Position  = matrixProjection * matrixModelView * position;
  // Propagated quantities.
  varTexCoords = texCoords;
  varNormal    = normal;
  varWorld     = position.xyz;
  varColor     = flatColor = color;
}
```

GLSL example (Phong shader + texture)

```
#version 130

in vec2 varTexCoords;       // [s, t]
in vec3 varNormal;          // world coordinate system
in vec3 varWorld;
in vec3 varColor;           // Gouraud color
flat in vec3 flatColor;

uniform bool useShading;
uniform vec3 globalAmbient;
uniform vec3 lightColor;
uniform vec3 lightPosition; // world coordinate system
uniform vec3 eyePosition;   // world coordinate system
uniform vec3 Ks;
uniform float shininess;
uniform bool useTexture;
uniform sampler2D texSurface;

out vec3 fragColor;         // output = fragment color

...
```

GLSL example (Phong shader + texture)

```
void main ()
{
  if (useShading)
  {
    vec3 P = varWorld;
    vec3 N = normalize(varNormal);
    vec3 L = normalize(lightPosition - P);
    vec3 V = normalize(eyePosition - P);
    vec3 H = normalize(L + V);

    float cosb = 0.0;
    float cosa = dot(N, L);
    if ( cosa > 0.0 )
      cosb = pow(max(dot(N, H), 0.0), shininess);

    vec3 kakd;
    if (useTexture)
      kakd = vec3(texture2D(texSurface, varTexCoords));
    else
      kakd = varColor;

    fragColor = kakd * globalAmbient +
                kakd * lightColor * cosa +
                Ks   * lightColor * cosb;
  }
  else
    fragColor = flatColor;
}
```

Typický main() v OpenGL

Real-time simulátor (videohra, 3D editor/prohlížeč, apod.)

- vykreslování 3D scény plnou rychlostí (maximální fps)

Nadstavba nad OpenGL (okno + „**swap-chain**“, interakce)

- SDL, OpenTK, freeglut, Qt, wxWidgets…

ColdStart – inicializace nadstavby i OpenGL

- globální start a nastavení OpenGL, mody grafiky (color, depth-buffer, swap-chain), kompilace shaderů, [načtení textur]…

WindowResize – reakce na změnu velikosti okna

- přenastavení **viewport** a **projekční matice**
- změna dalších souvisejících hodnot (např. pro UnProject())

Vnitřní smyčka (pozor na synchronizaci)

ProcessEvents – čtení a zpracování událostí

- interaktivní: **myš**, **klávesnice**, **pad**, ostatní: **síťová komunikace**…
- lze implementovat **asynchronně** (UWP, pozor na ochranu dat!)

Update (Simulate) – simulace scény

- logika simulátoru/hry, simulace fyziky, přehrávání animací…
- komponenta **masivně měnící aktuální stav** simulovaného světa
- lze implementovat **asynchronně** (i ve více vláknech), ale pozor…!

Render – vykreslení scény

- **nakreslení aktuálního stavu** 3D reprezentace a předání výsledku do swap-chainu
- zde je potřeba **OpenGL**

Typický main() v OpenGL I

```
World world(..);            // simulated world
Swapchain swapchain(..);    // object presenting result 2D graphics to the window
bool done = false;          // app-exit request

void main (int argc, char **argv)
{
  CommandlineArguments(argc, argv);

  ColdStart("Test app", swapchain, PIX_FORMAT_R8G8B8A8, DOUBLE_BUFFERING, ..);

  WindowResize();

  world.InitSimulation();

  while (!done)
  {
    // Simulate & render one frame.

    ProcessEvents();

    Update(GetSystemTime());

    if (WindowVisible())
      Render();
  }
}
```

Typický main() v OpenGL II

```
void ProcessEvents ()        // process all available events
{
  Event ev;
  while (PollEvent(ev))
    switch (ev.type):
    {
      case WINDOW_RESIZE:
        WindowResize();
        break;
      case WINDOW_CLOSE:
        done = true;
        break;
      case KEY_DOWN:
        ...
      case MOUSE_BUTTON_DOWN:
        ...
      case MOUSE_MOVE:
        ...
    }
}

void Update (double time)     // simulate the world (to the given target time)
{
  world.UpdateForces(time);   // application-related code
  world.UpdatePositions(time);

  world.PlayAnimations(time); // scripted animations
}
```

Typický main() v OpenGL III

```
void Render ()                  // Render current world's state using OpenGL.
{
  swapchain.FrameStart();

  // Clear frame-buffer & depth-buffer.
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  SetViewProjMatrices();        // camera is constant for the whole frame..

  for (const auto& section : world.GetSections())   // R/O copy of world's data (?)
  {
    section.SetModelMatrix(); // every section has its own model transform
    section.SetShaders();     // change shaders only if necessary
    section.SetTextures();    // change textures only if necessary
    section.SetUniforms();    // either here or before every batch

    // One scene section can have multiple rendering batches.
    for (const auto& batch : section.batches)
    {
      batch.SetBuffers();     // change VB/IB only if necessary (glBindBuffer..)
      batch.Render();         // actual rendering command[s] (glDrawElements,
                              //   glDrawArrays, glDrawElementsInstanced, ..)
    }
  }

  swapchain.Present();        // “SwapBuffers()” ..
}
```

Literatura

Tomas Akenine-Möller, Eric Haines et al.: ***Real-time rendering, 4th edition***, A K Peters, 2018, ISBN: 9781138627000

OpenGL ARB: ***OpenGL Programming Guide, 8th edition***, Addison-Wesley, 2013, ISBN: 0321773039

matfyz

Computer Graphics Charles University

Malířův algoritmus

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/

Malířův algoritmus

Kreslení do bufferu

- video-RAM, GPU, rastrová tiskárna s bufferem

Vyplňování ploch

- lze i stínovat

Kreslení odzadu dopředu

- **překreslování** dříve nakreslených objektů
- kreslení **poloprůhledných objektů na GPU**

→ **Určení správného pořadí ploch**

Zjednodušené varianty

Explicitní pořadí kreslení

- např. u grafu funkce dvou proměnných: $z = f(x,y)$

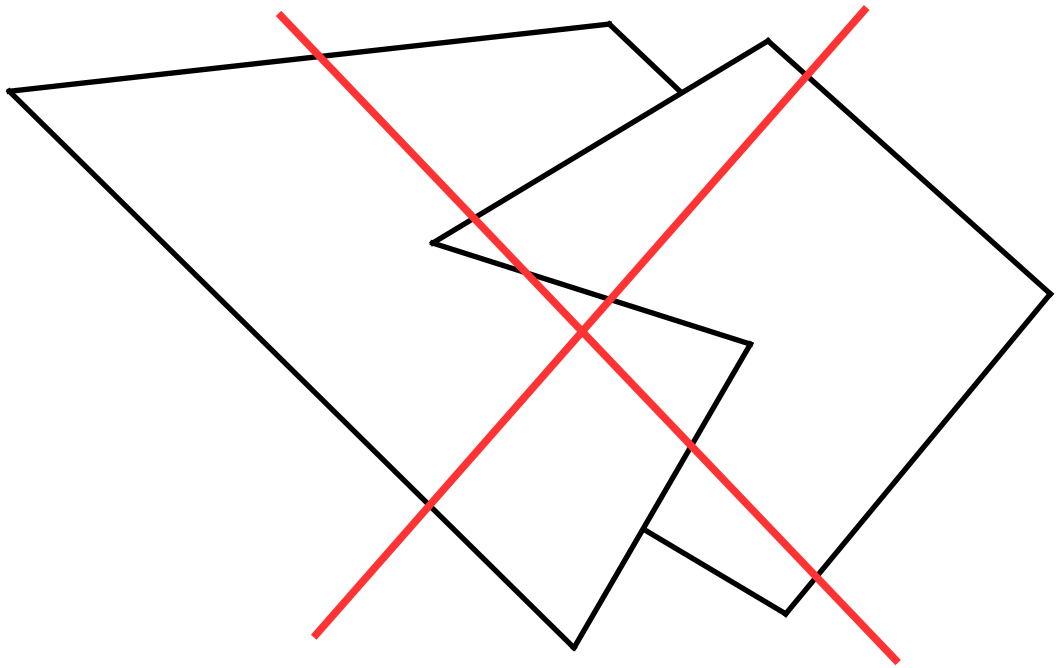
Hloubkové třídění („depth-sort")

- **setřídění objektů podle souřadnice *z*** (střed, těžiště)
- dobře funguje při velkém množství malých objektů
- nesprávná kresba velkých ploch (velká stolní deska s malými předměty)

Korektní algoritmus

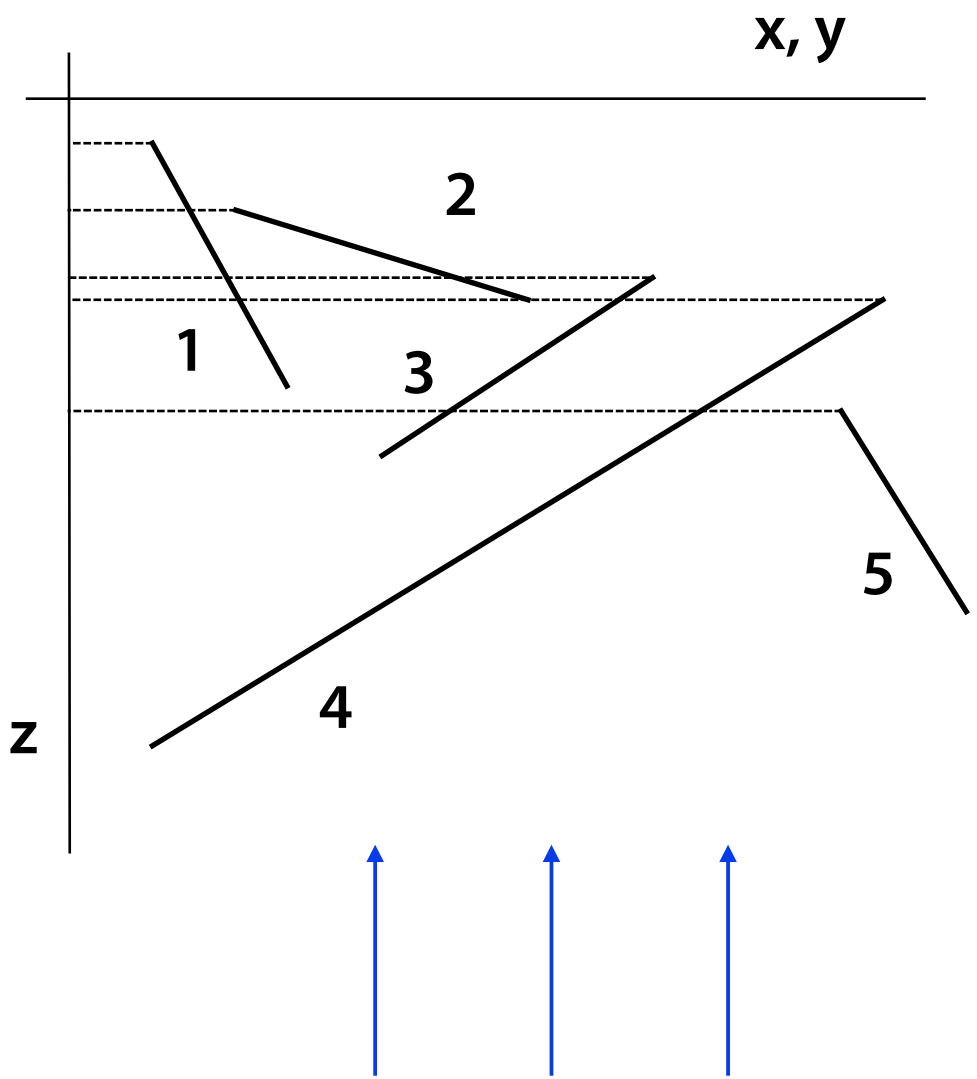
Scéna je složena z **rovinných plošek (stěn)**

- stěny mohou mít společné body **pouze na obvodu** (**nesmějí** se prosekávat)

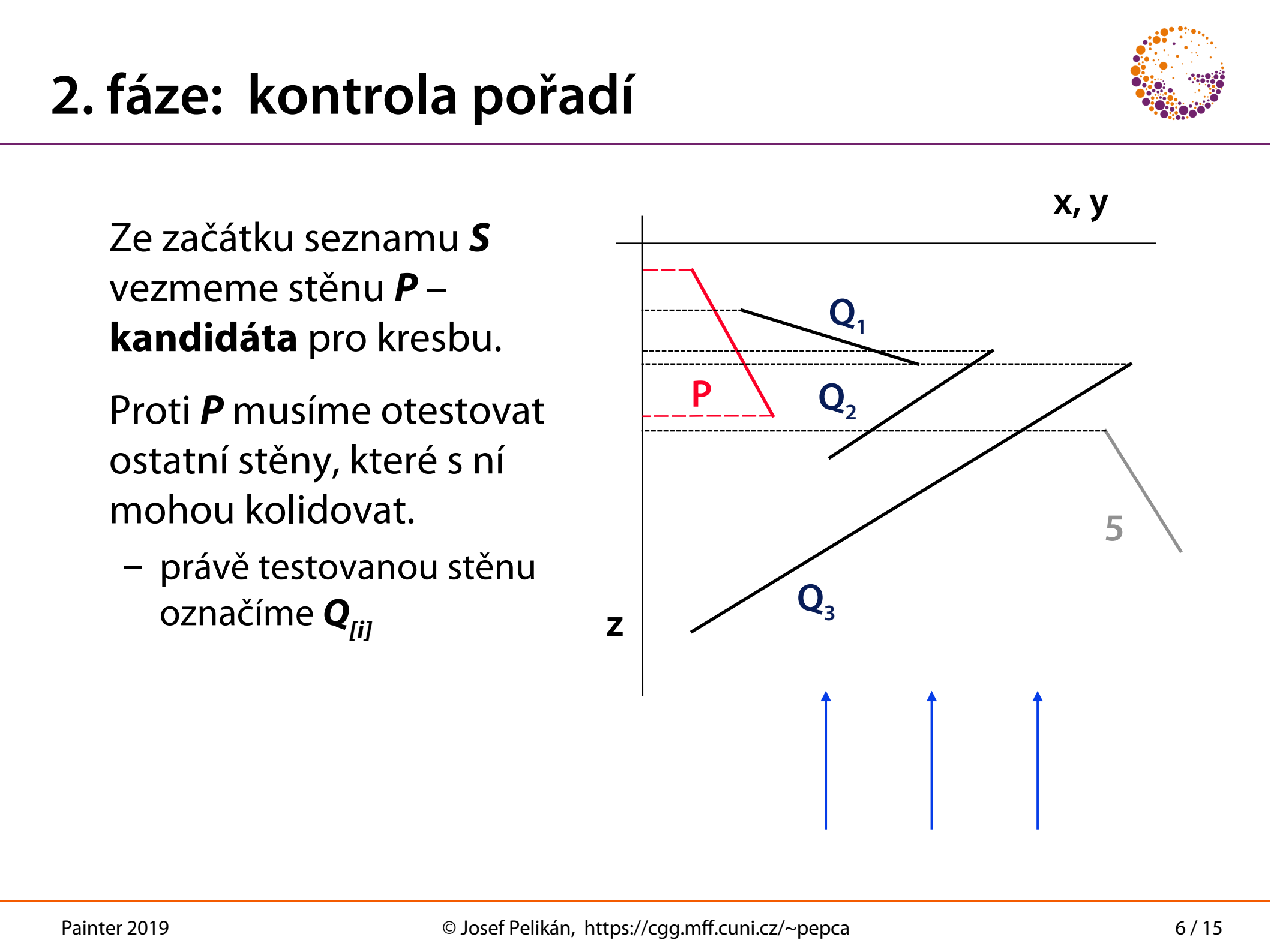


1. fáze: třídění

Stěny setřídíme podle **minimální souřadnice *z* vzestupně** – tj. odzadu dopředu – vytvoříme tak **vstupní seznam *S***



2. fáze: kontrola pořadí



Ze začátku seznamu ***S*** vezmeme stěnu ***P*** – **kandidáta** pro kresbu.

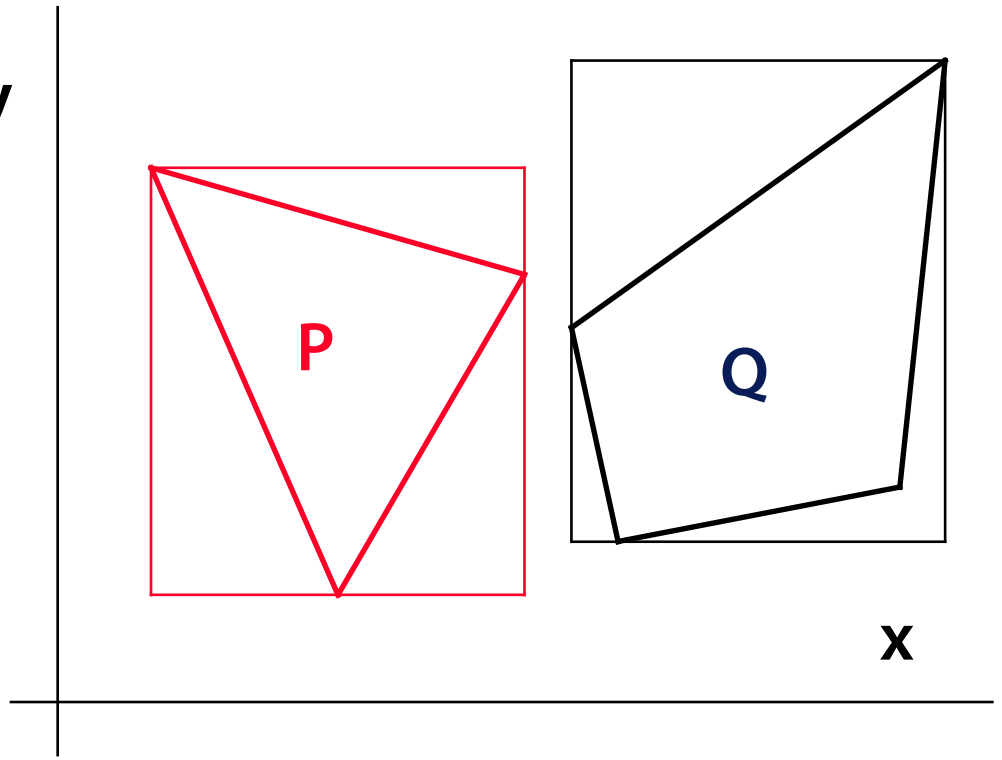
Proti ***P*** musíme otestovat ostatní stěny, které s ní mohou kolidovat.

- právě testovanou stěnu označíme $Q_{[i]}$

2.a fáze: „minimax test”

Nejprve provedeme **nejjednodušší test** – **v průmětu** porovnáme obdélníky opsané oběma stěnám

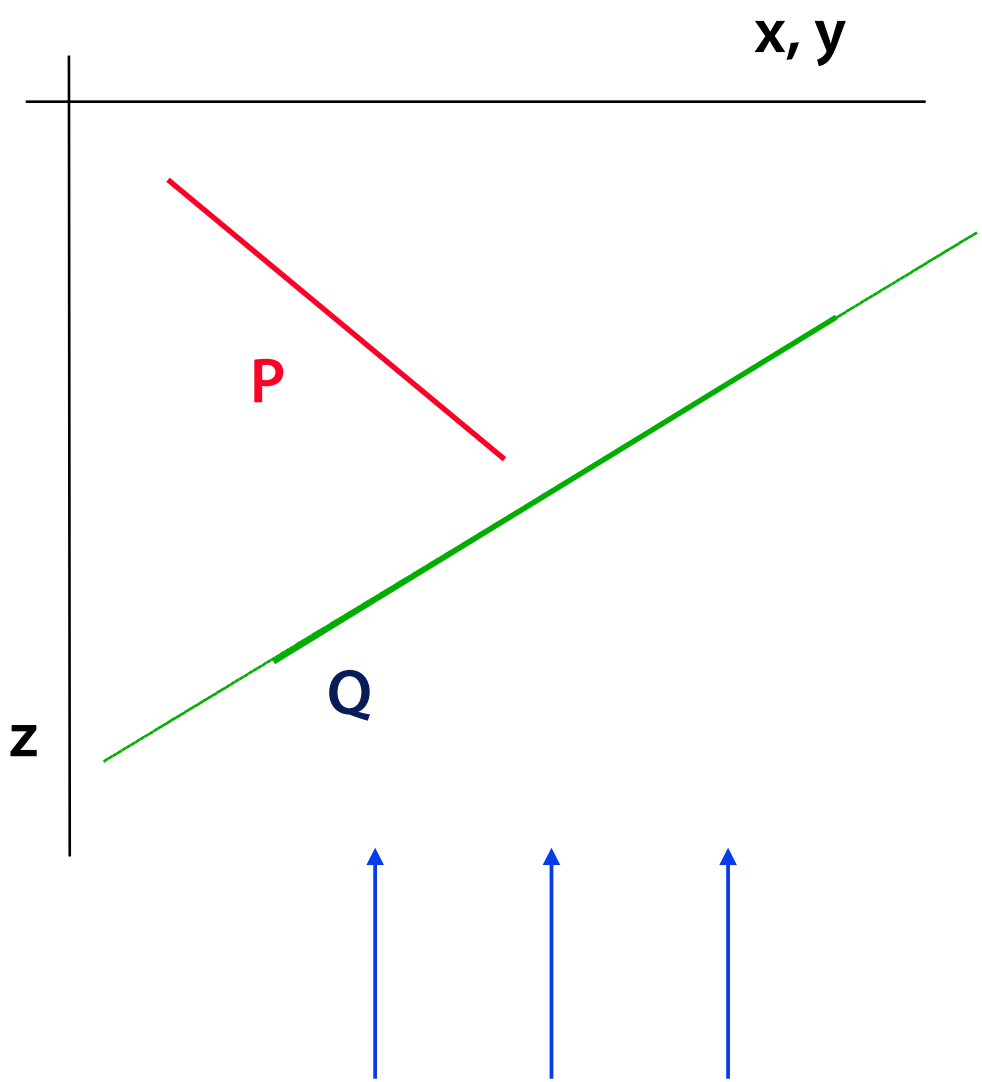
- jestliže nemají společný bod, testování **Q** končí (pass)
- jinak pokračujeme dalším testem **P** a **Q**



2.b fáze: P versus rovina Q

Testujeme, zda stěna **P** neleží celá **za rovinou** stěny **Q**

- v kladném případě testování **Q** končí (pass)
- jinak pokračujeme dalším testem **P** a **Q**

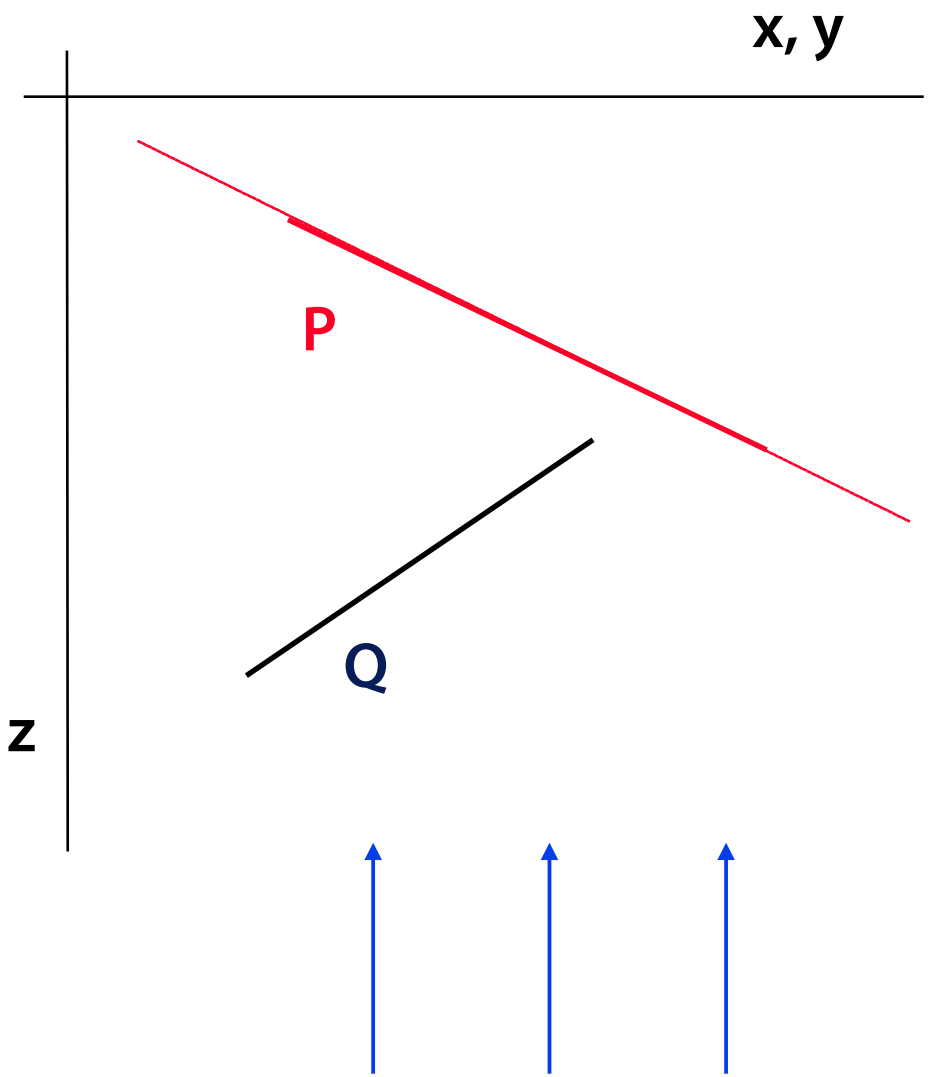


$$a \cdot x + b \cdot y + c \cdot z + d < 0$$

2.c fáze: Q versus rovina P

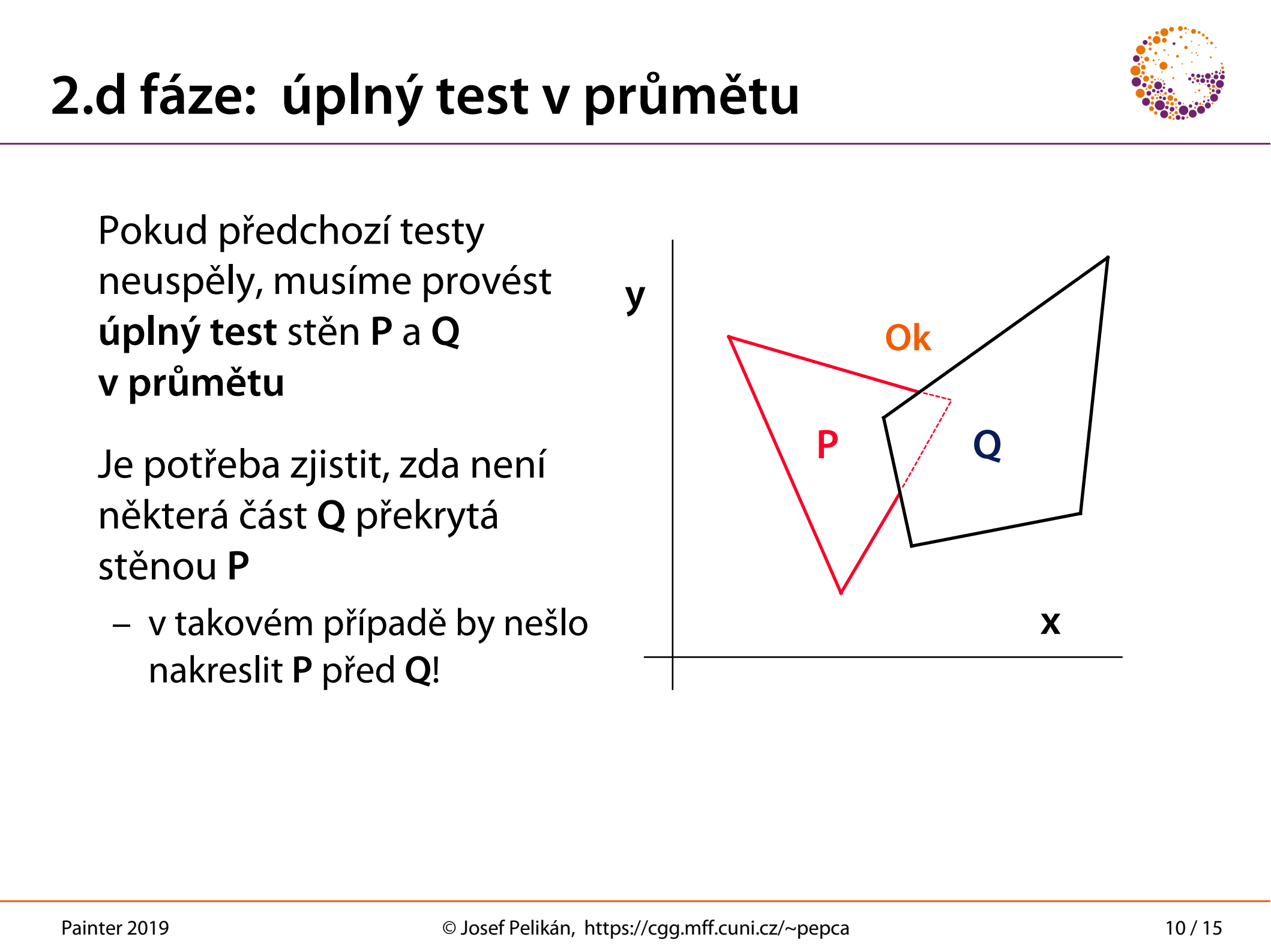
Test, zda stěna **Q** neleží celá **před rovinou** stěny **P**

- v kladném případě testování **Q** končí (pass)
- jinak pokračujeme dalším testem **P** a **Q**



$$\mathbf{a} \cdot x + \mathbf{b} \cdot y + \mathbf{c} \cdot z + \mathbf{d} > 0$$

2.d fáze: úplný test v průmětu



Pokud předchozí testy neuspěly, musíme provést **úplný test** stěn **P** a **Q** **v průmětu**

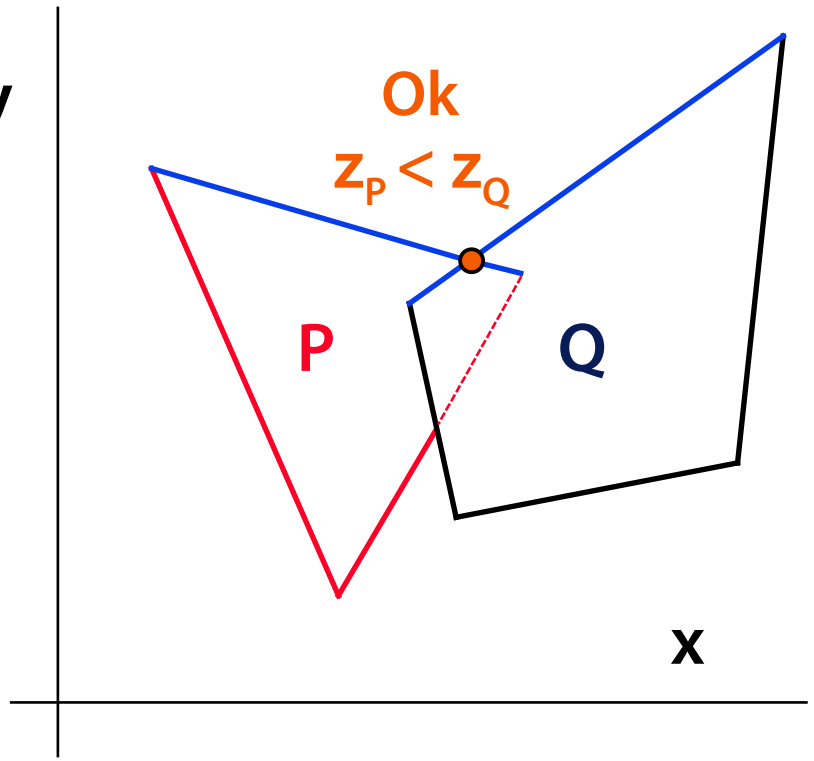
Je potřeba zjistit, zda není některá část **Q** překrytá stěnou **P**

- v takovém případě by nešlo nakreslit **P** před **Q**!

2.d fáze: úplný test v průmětu

Testujeme proti sobě **všechny hrany P** a **Q**

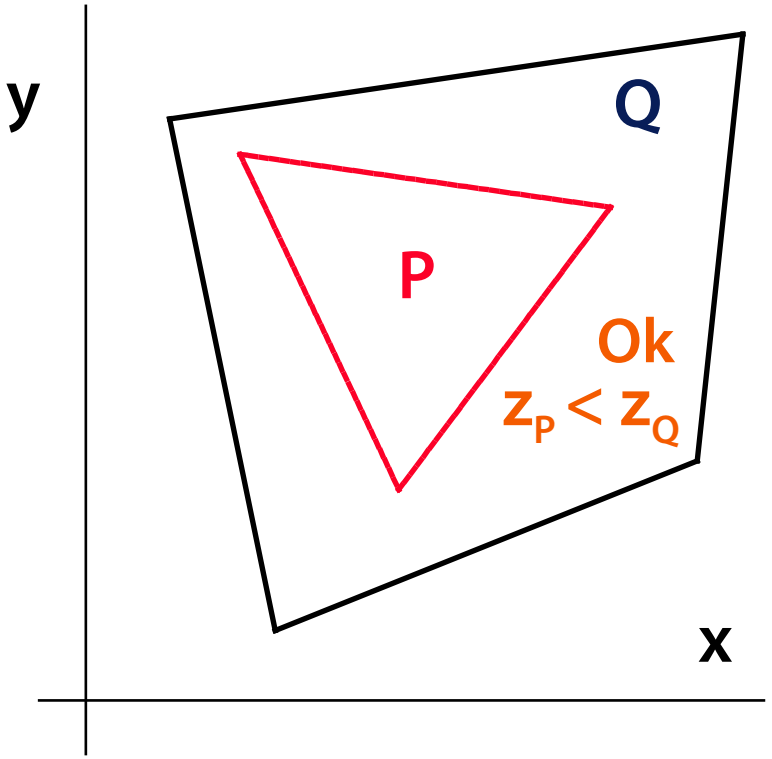
- najdeme-li průsečíky, porovnáme v nich souřadnice **z**
- je-li vždy **P** za **Q**, test **Q** končí úspěšně (pass)
- v opačném případě **nelze P nakreslit jako první!**



2.d fáze: úplný test v průmětu

Neexistuje-li průsečík hran **P** a **Q**, je třeba ještě zkontrolovat, zda neleží stěna **P celá uvnitř Q** nebo naopak

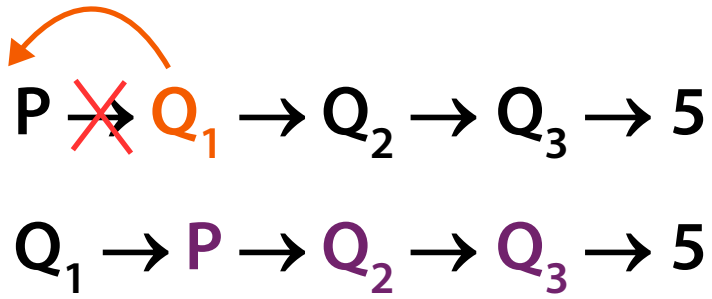
- to by se opět musely testovat souřadnice **z**



2. fáze: změna pořadí

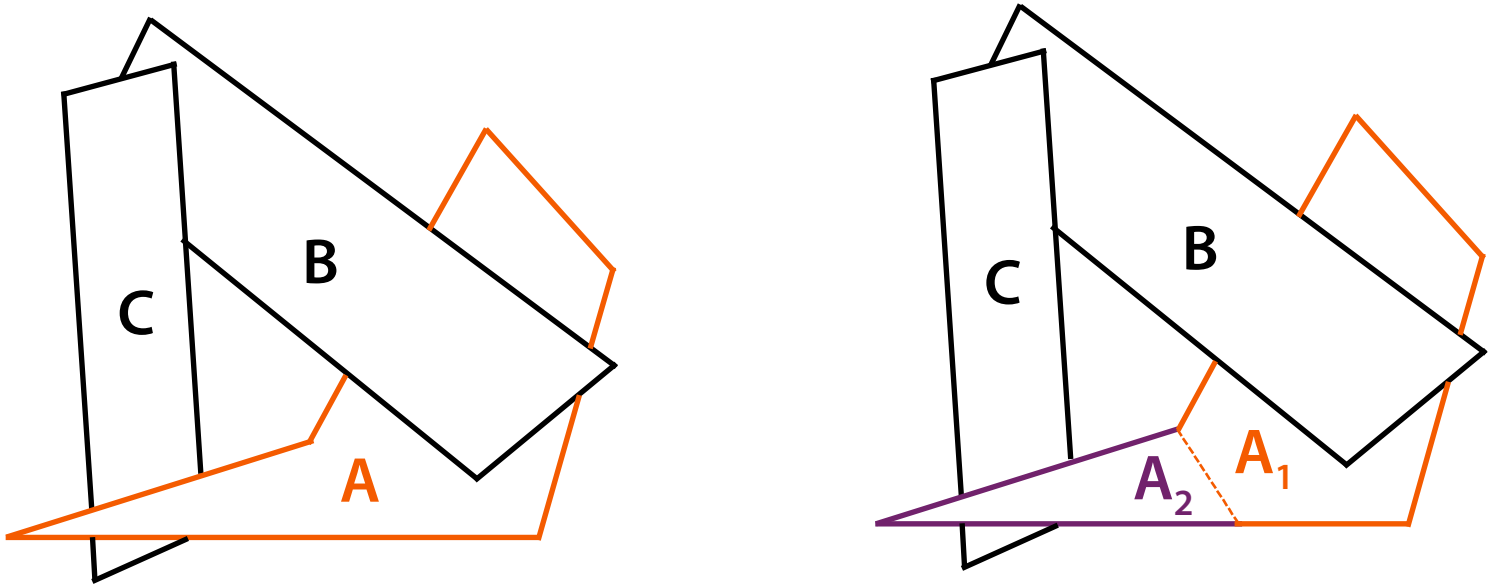
Jestliže nelze z nějakého důvodu nakreslit P před Q, zkusíme přesunout stěnu **Q na začátek seznamu S** (ještě před **P**)

- pro **Q** budeme opět provádět všechny testy 2. fáze (jak jsme je popsali se stěnou **P**)
- testy nového kandidáta **Q** proti **P** už byly z velké části provedeny, stačí pouze doplnit obrácené testy **2.b** a **2.c**



Kvůli možnosti **zacyklení** se musí každý kandidát označit zvláštním příznakem

2. fáze: zacyklení



Jestliže je testován některý kandidát podruhé, došlo k **zacyklení**

Cyklus lze odstranit **rozdělením** některé stěny

– správné pořadí: A_1, **B**, **C**, A_2

Literatura

J. Foley, A. van Dam, S. Feiner, J. Hughes: ***Computer Graphics, Principles and Practice***, 672-675

Jiří Žára a kol.: ***Počítačová grafika, principy a algoritmy***, 302-304

matfyz

Computer Graphics Charles University

Z-buffer, depth-buffer

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/

Z-buffer, depth-buffer (paměť hloubky)

Kreslení do bufferu

- video-RAM, GPU, rastrová tiskárna s bufferem

Vyplňování ploch

- lze i stínovat

Není třeba nic třídit!

Korektní vykreslení nestandardních situací

- prosekávání stěn, cyklické zákryty…

Paměť hloubky

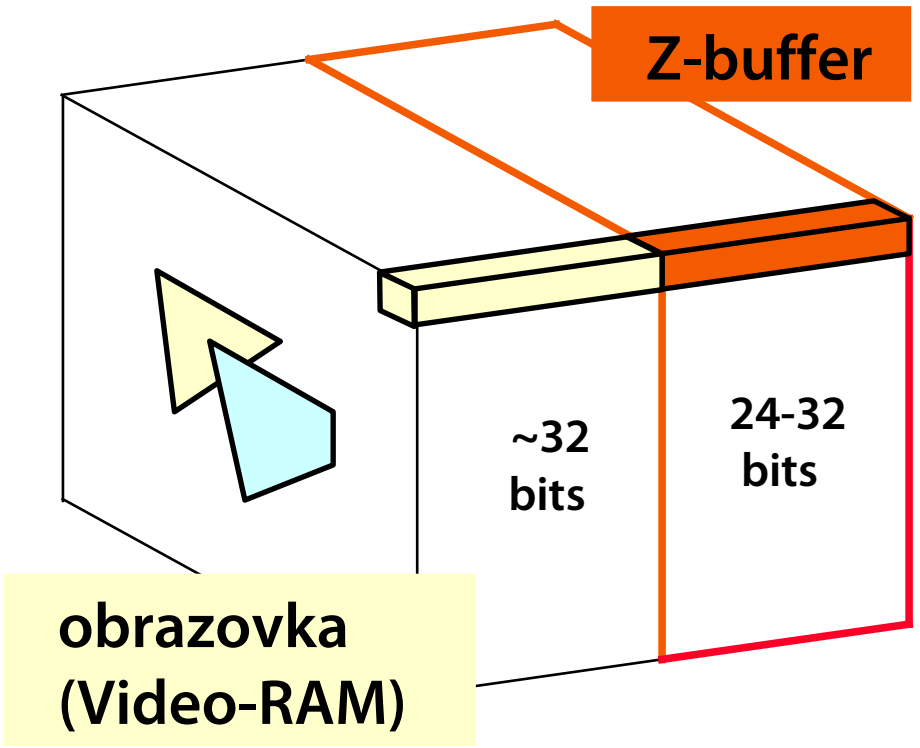
Pro každý pixel se ukládá

- **barva** (Video-RAM)
- **hloubka** = vzdálenost od pozorovatele (**Z-buffer**)

float – jednodušší (nové GPU)

Integer – rychlejší (24-32 bitů)

- pozor na přesnost!



Algoritmus

Inicializace

- fillBuffer(VideoRAM, „barva pozadí“);
- fillBuffer(Zbuf, „nekonečno”);

Zápis všech objektů do Z-bufferu

- rozložení na jednotlivé pixely (**rasterizace**, vyplňování)
- **test hloubky** jednoho pixelu:

```
void WritePixel (int x, int y, float z, Color color)
{
  if (z < Zbuf[x, y])
  {
    Zbuf[x, y] = z;
    VideoRAM[x, y] = color;
  }
}
```

Výhody Z-bufferu

Jednoduchost výpočtů

- celočíselná/fixed-point aritmetika
- možnost **masivní paralelizace** (viz GPU)
- HW implementace 10M až 10G plošek/s

Není nutné třídění

Správné vykreslení všech **obtížných situací**

Nemusí se kreslit pouze rovinné plošky

- rutina pro rozklad objektu na pixely (s výpočtem hloubky **z**)

Nevýhody Z-bufferu

Mazání bufferu na začátku zpracování snímku

Některé pixely ve Video-RAM se **několikanásobně překreslují**

- zbytečné vyhodnocování jejich barvy (fragment shader)

„Z-fighting“

- nepříjemné artefakty při kreslení objektů ve stejné rovině
- velmi rušivé při animacích
- je to často chybou modelu nebo špatně nastavených parametrů projekce

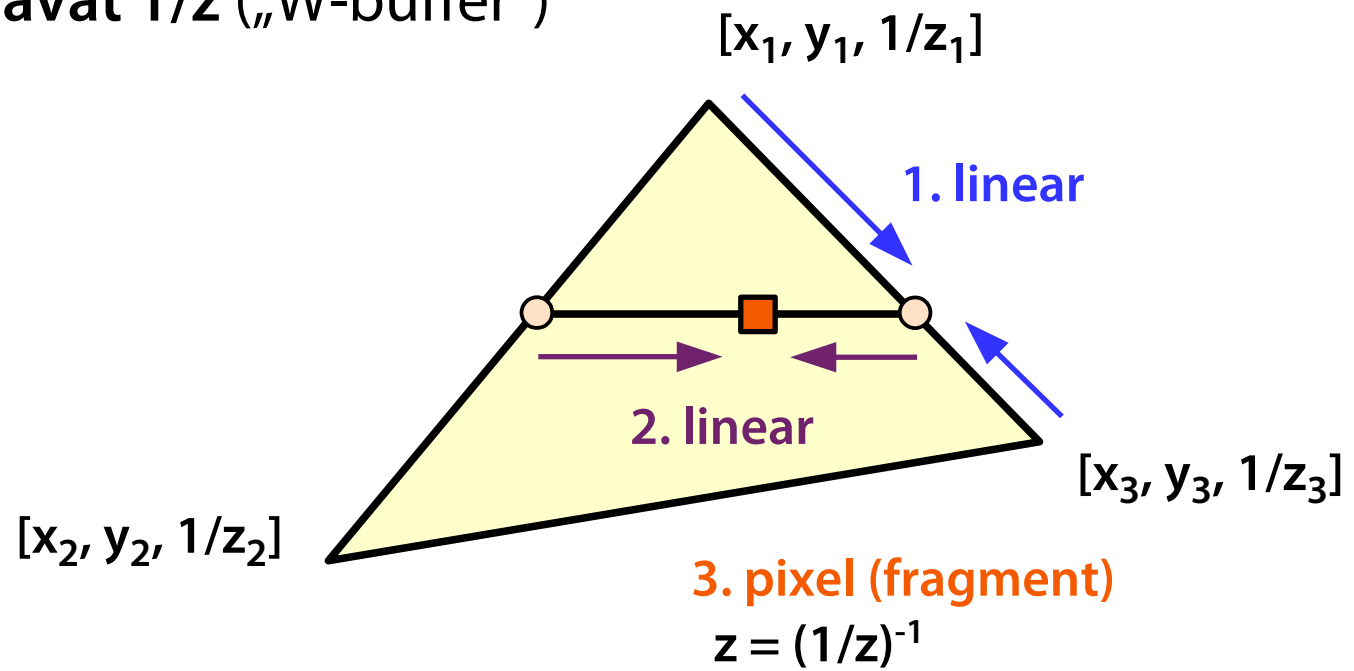
Neřeší jednoduše **poloprůhledné 3D scény**

- nutnost předzpracování (3D třídění primitivů)

Perspektivně korektní interpolace hloubky

V perspektivě nemůže interpolátor (rasterizer) na ploše průmětny přímo lineárně interpolovat hloubku **z**

- místo toho lze interpolovat převrácené hodnoty **1/z**
- v každém pixelu můžeme **z** spočítat dělením nebo přímo **porovnávat 1/z** („W-buffer“)



Literatura

J. Foley, A. van Dam, S. Feiner, J. Hughes: ***Computer Graphics, Principles and Practice***, 668-672

Jiří Žára a kol.: ***Počítačová grafika, principy a algoritmy***, 298-300

Kok-Lim Low: ***Perspective-Correct Interpolation*** (report, proof), University of North Carolina at Chapel Hill, 2002

matfyz

Computer Graphics Charles University

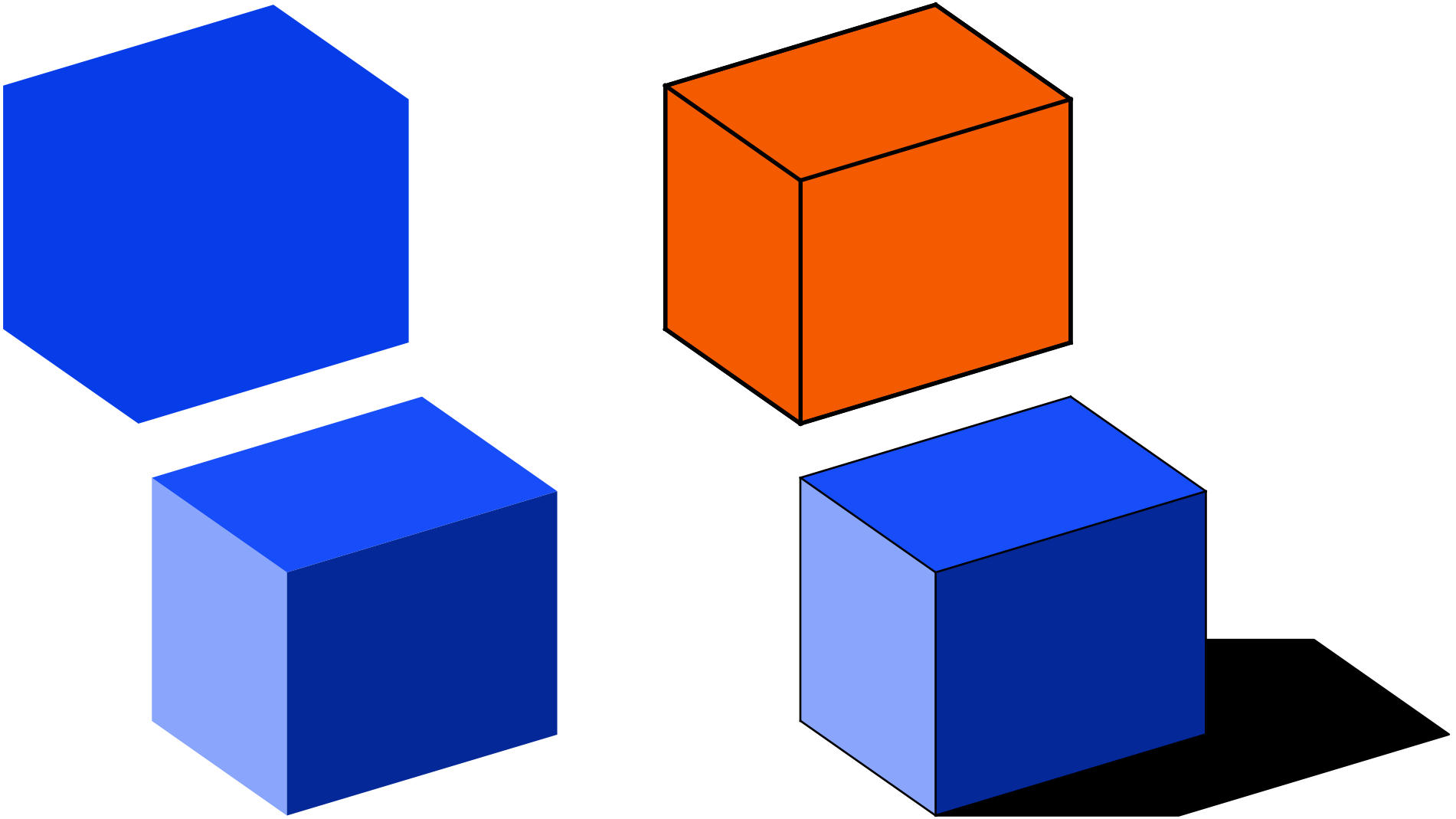
Phongův osvětlovací model

CGG MFF UK Praha

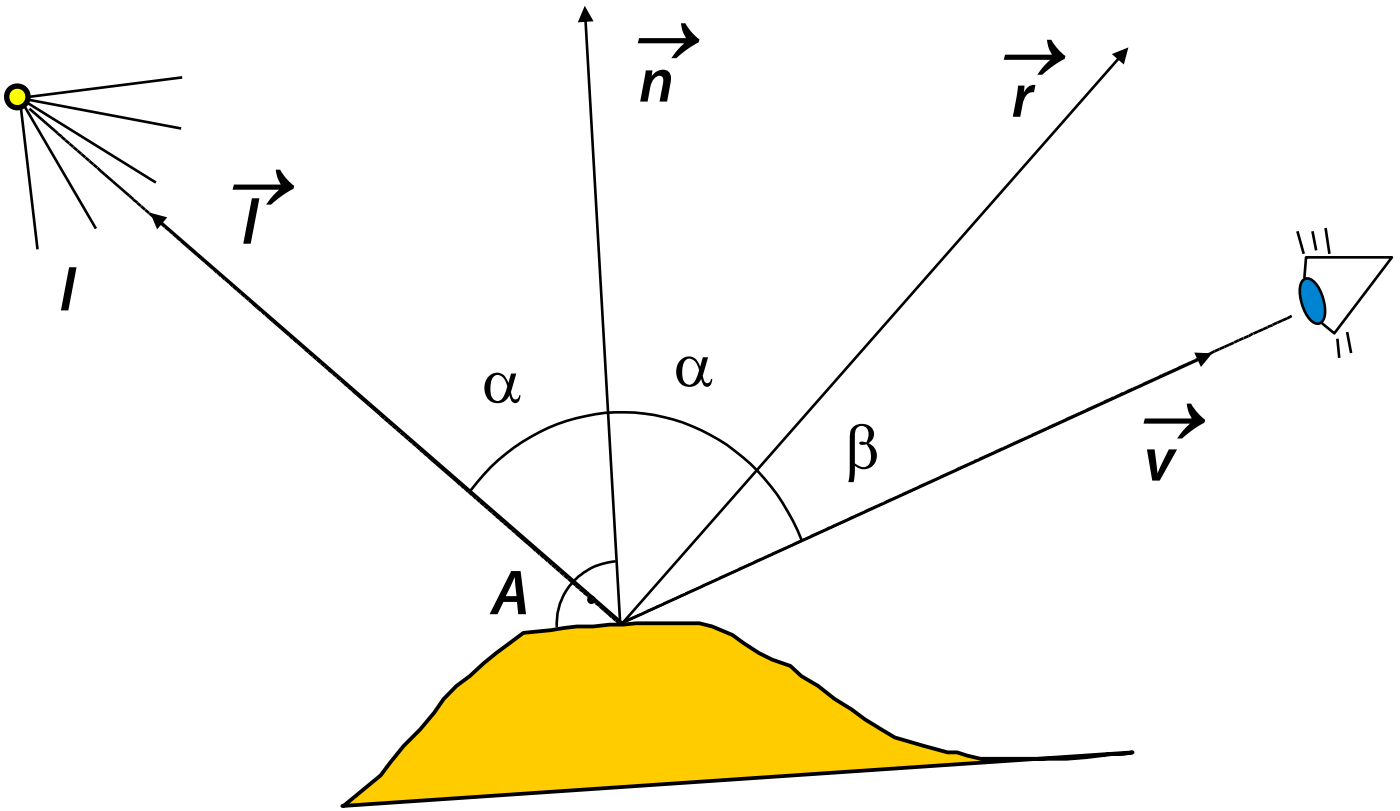
pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/

Stínování a vržené stíny



Světelný model



Difusní složka E_D

Odpovídá **ideálně matnému tělesu**

$$\mathbf{E_D = I_i \cdot C_D \cdot k_D \cdot \cos \alpha}$$

$\mathbf{I_i}$ … intenzita světelného zdroje

$\mathbf{C_D}$ … barva difusní složky (RGB)

$\mathbf{k_D}$ … koeficient difusního světla (0.0 až 1.0)

$\mathbf{\cos \alpha = l \cdot n}$ … skalární součin normovaných vektorů

Okolní světlo E_A

Všesměrové konstantní osvětlení

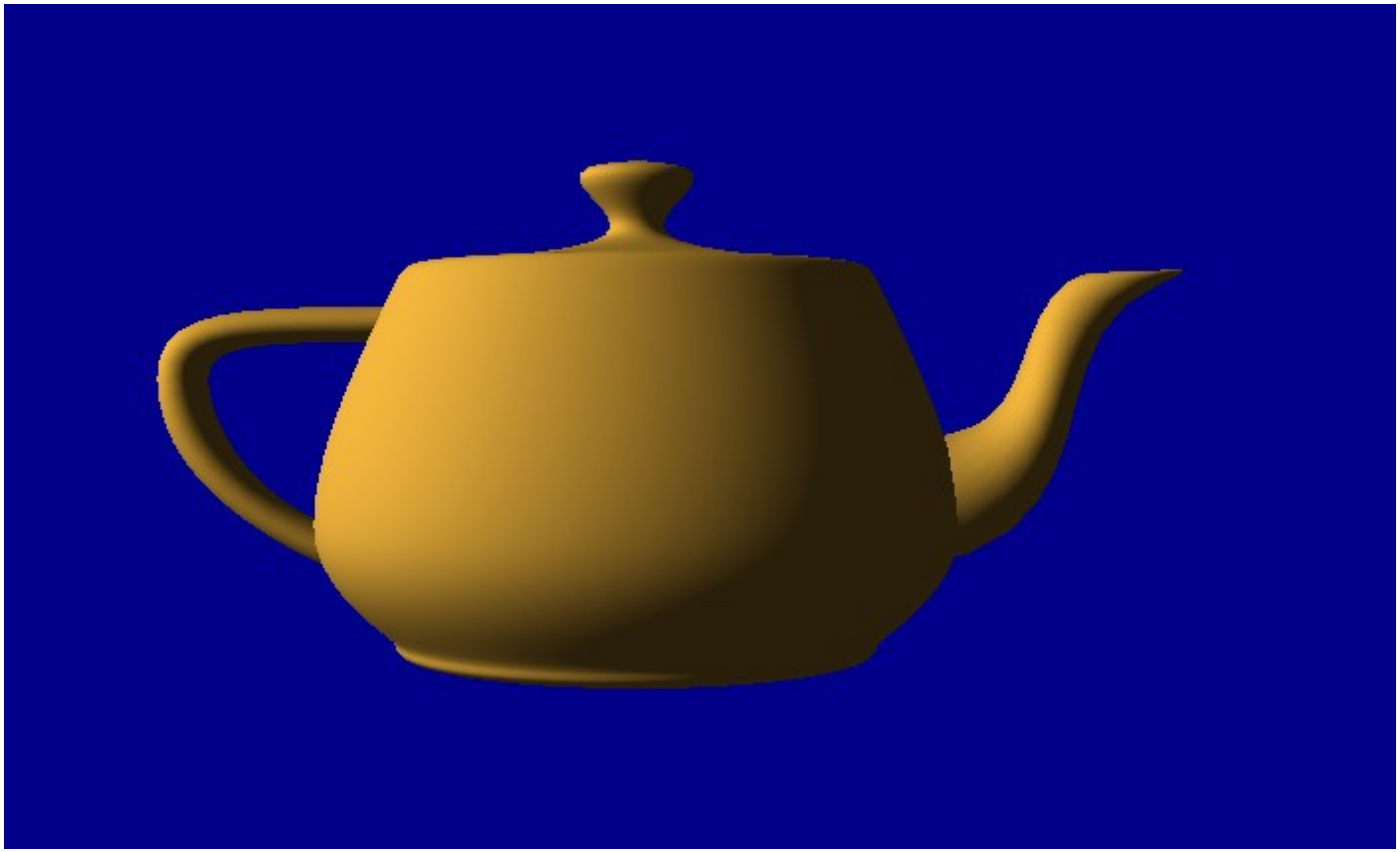
Napodobuje **sekundární odražené světlo**

$$\mathbf{E_A = C_D \cdot k_A}$$

$\mathbf{C_D}$ … barva stejná jako u difusní složky (RGB)

$\mathbf{k_A}$ … koeficient okolního světla (0.0 až 1.0)

Difusní a okolní světlo



Lesklý odraz E_s

Simuluje **odlesk** na povrchu lesklých těles

$$\mathbf{E_s = I_i \cdot C_s \cdot k_s \cdot \cos^h \beta}$$

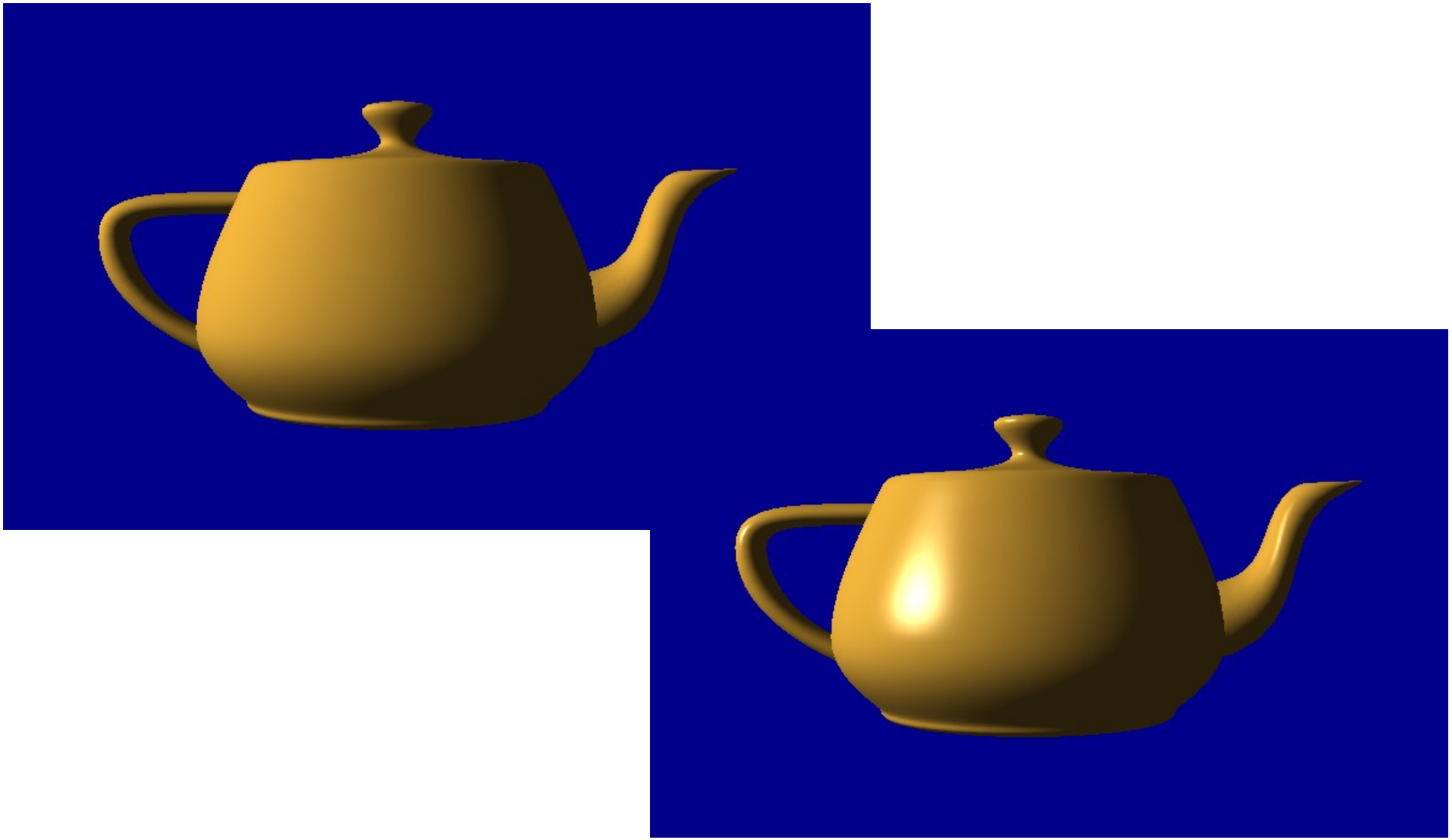
$\mathbf{C_s}$ … barva lesklého odrazu (RGB)

$\mathbf{k_s}$ … koeficient lesklého odrazu (0.0 až 1.0)

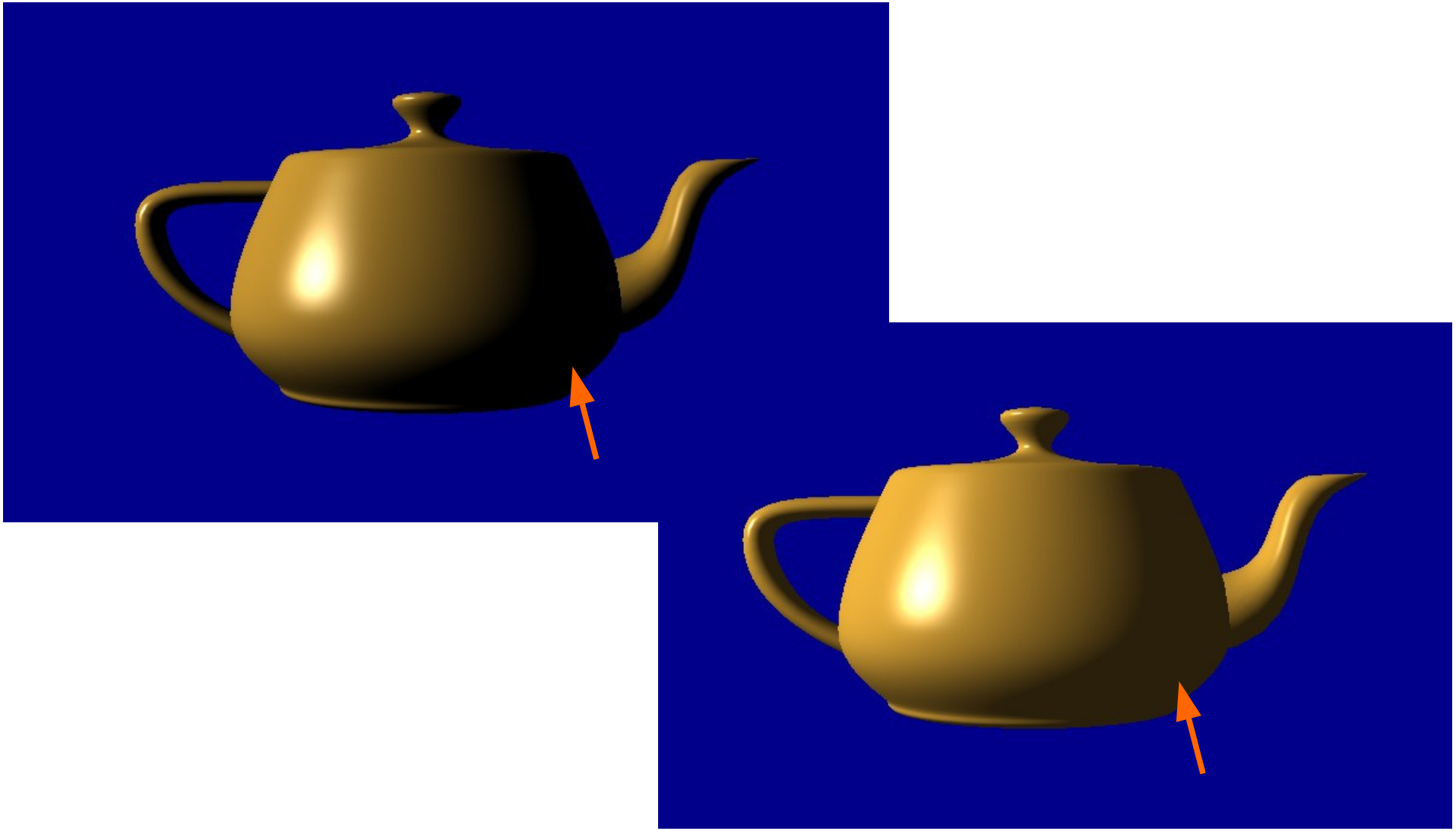
$\mathbf{\cos \beta = \vec{r} \cdot \vec{v}}$ … skalární součin normovaných vektorů

h … ovlivňuje velikost odlesku (5 … 500)

Vliv lesklé složky odrazu



Vliv okolního světla



Osvětlení od jednoho zdroje

$$\mathbf{E} = \mathbf{E}_A + \mathbf{E}_D + \mathbf{E}_S$$

Barvy

$\mathbf{C}_D = \mathbf{C}$ … barva materiálu (RGB)

$\mathbf{C}_S = \mathbf{C}_L$ … barva světelného zdroje (RGB)

Konzistence

$\mathbf{k}_A + \mathbf{k}_D + \mathbf{k}_S = \mathbf{1}$ (proti přetečení)

Více světelných zdrojů

$$E = E_A + \sum_i (E_D + E_S)$$

Výpočet **vektoru odrazu**

$$R = 2\vec{n}(\vec{n} \cdot \vec{l}) - \vec{l}$$

Původní **Phongův** vzorec pro lesklý odraz

- místo konstantního členu $C_s \cdot k_s$ obsahuje funkci $W(\alpha)$ (silnější odraz pro velké úhly)

Oprava na vzdálenost zdroje

Měla by být … $\mathbf{1/d^2}$

– příliš velký rozsah hodnot (monitor počítače není schopen zobrazit)

Používá se … $\mathbf{1/(c_0 + c_1 d + c_2 d^2)}$

$$\mathbf{E = E_A + \sum_i (E_D + E_S)/(c_0 + c_1 d_i + c_2 d_i^2)}$$

Zjednodušení výpočtů (Blinn)

Světelné zdroje v nekonečnu (směrové světelné zdroje)

- v celé scéně budou konstantní vektory $\vec{\mathbf{l}}_i$

Rovnoběžná projekce (pozorovatel v nekonečnu)

- v celé scéně bude konstantní vektor $\vec{\mathbf{v}}$

Zjednodušení (Blinn)

Pokud platí obě předchozí podmínky, lze místo $(\vec{r}_i \cdot \vec{v})^h$ použít $(\vec{h}_i \cdot \vec{n})^{4h}$

Půlící vektor $\vec{h}_i = (\vec{l}_i + \vec{v}) / |\vec{l}_i + \vec{v}|$

- $\vec{h}_i$ je konstantní v celé scéně

Někdy se nazývá „**Blinn-Phong model**“

Literatura

J. Foley, A. van Dam, S. Feiner, J. Hughes: ***Computer Graphics, Principles and Practice***, 721-734

Jiří Žára a kol.: ***Počítačová grafika, principy a algoritmy***, 343-346

matfyz

Computer Graphics Charles University

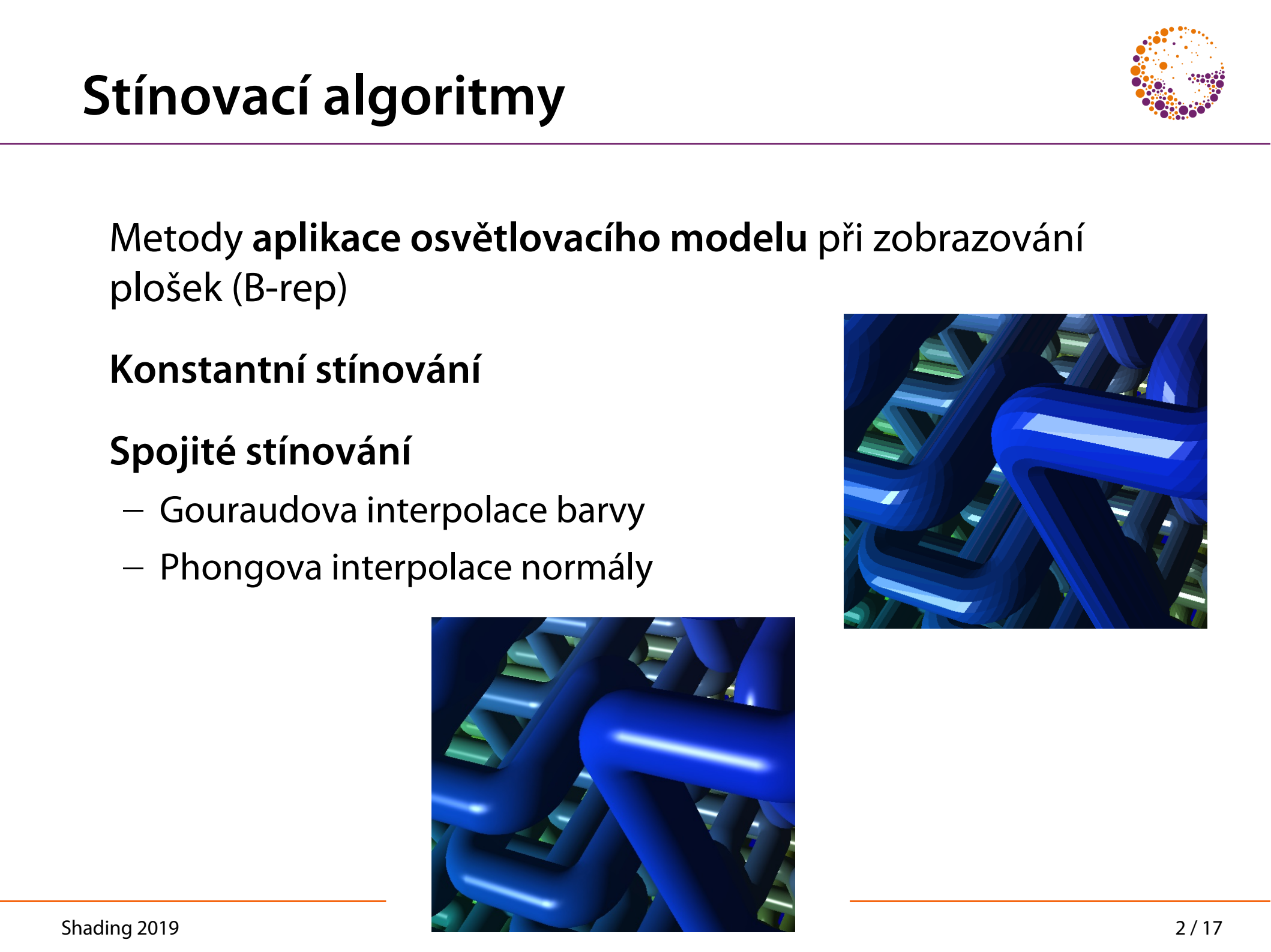
Spojité stínování

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/

Stínovací algoritmy



Metody **aplikace osvětlovacího modelu** při zobrazování plošek (B-rep)

Konstantní stínování

Spojité stínování

- Gouraudova interpolace barvy
- Phongova interpolace normály

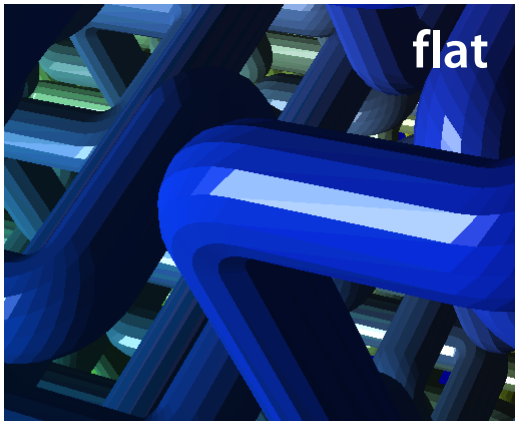
Konstantní stínování („flat shading“)

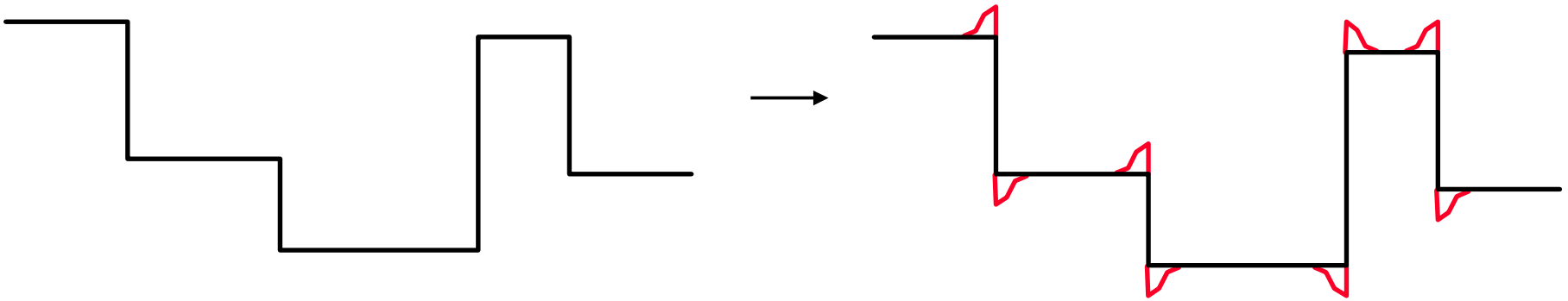
Spočítá se **osvětlení** jednou na každé stěně (např. v těžišti) a celá stěna se vyplní **touto barvou**

Funguje dobře u **hranatých těles**

Oblé plochy aproximované trojúhelníky

- příliš se zvýrazní umělé hrany tělesa
- tzv. **Machův efekt** (zrakový systém člověka)





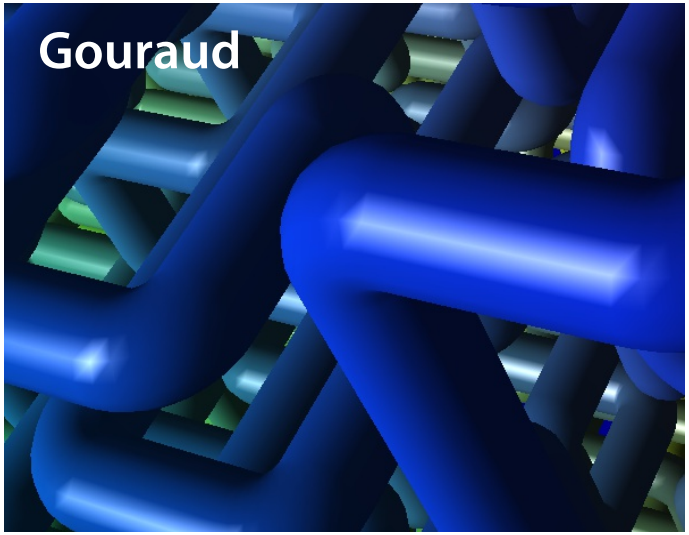
Spojité stínování

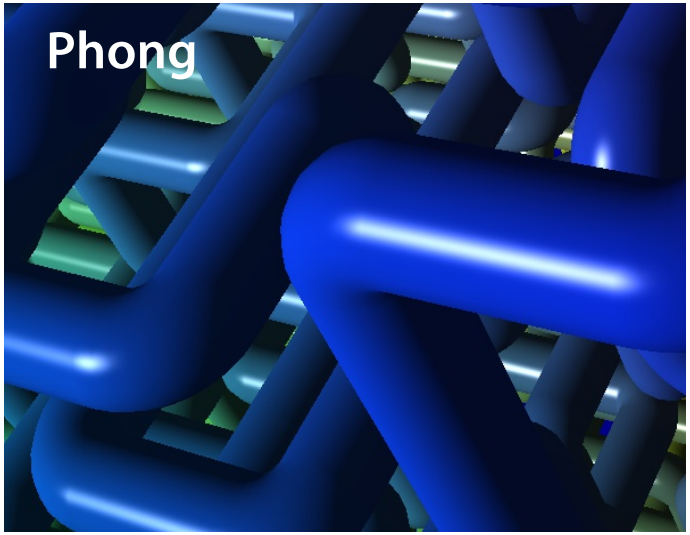
Interpolace barvy – **Gouraudovo stínování**

- rychlejší, vhodné jen pro matné povrchy
- HW implementace (byla součástí „fixed pipeline“ na GPU)

Interpolace normály – **Phongovo stínování**

- pomalejší, realističtější, vhodné i pro lesklá tělesa
- na GPU se implementuje ve fragment/pixel shaderu





Gouraudovo stínování

V **umělých vrcholech** tělesa se spočítá **normálový vektor** a z něj **osvětlení** (barva)

- aplikace zvoleného modelu osvětlení

Uvnitř stěn se počítá barva **bilineární interpolací**

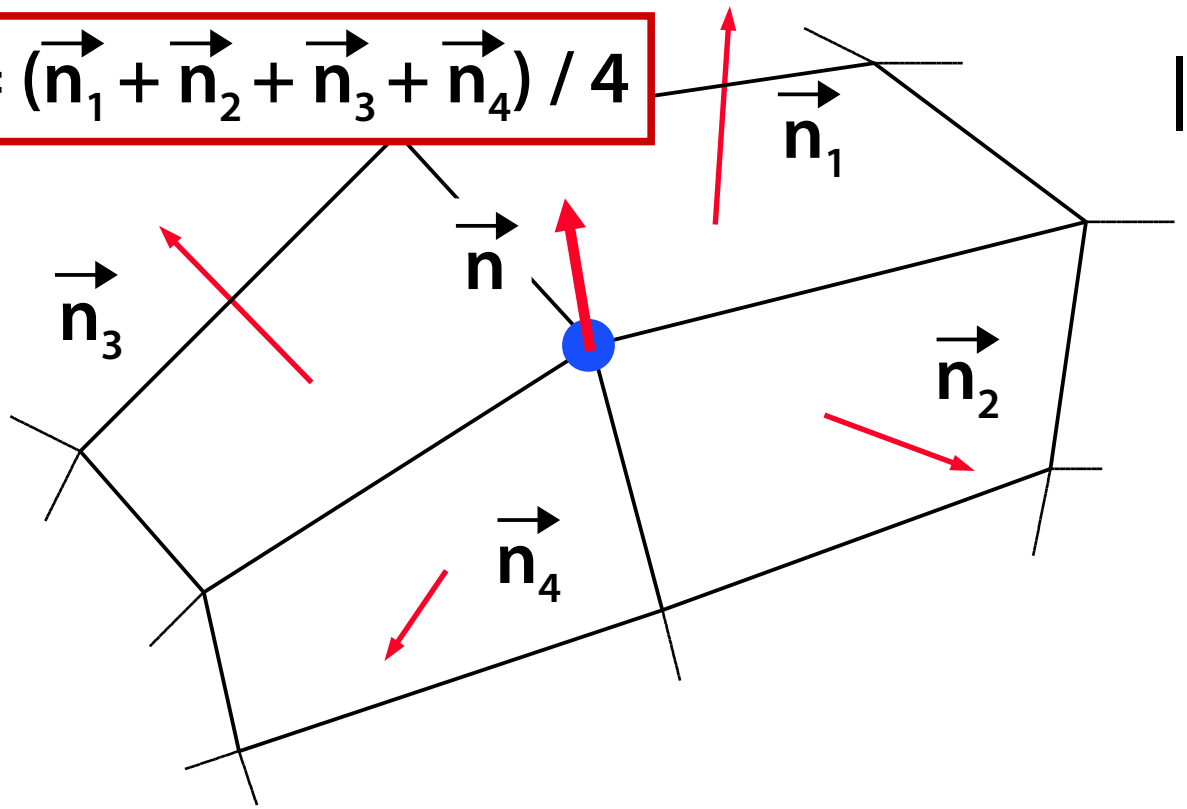
- vyplňování řádkovým rozkladem
- GPU rasterizer

Výpočet normál ve vrcholech

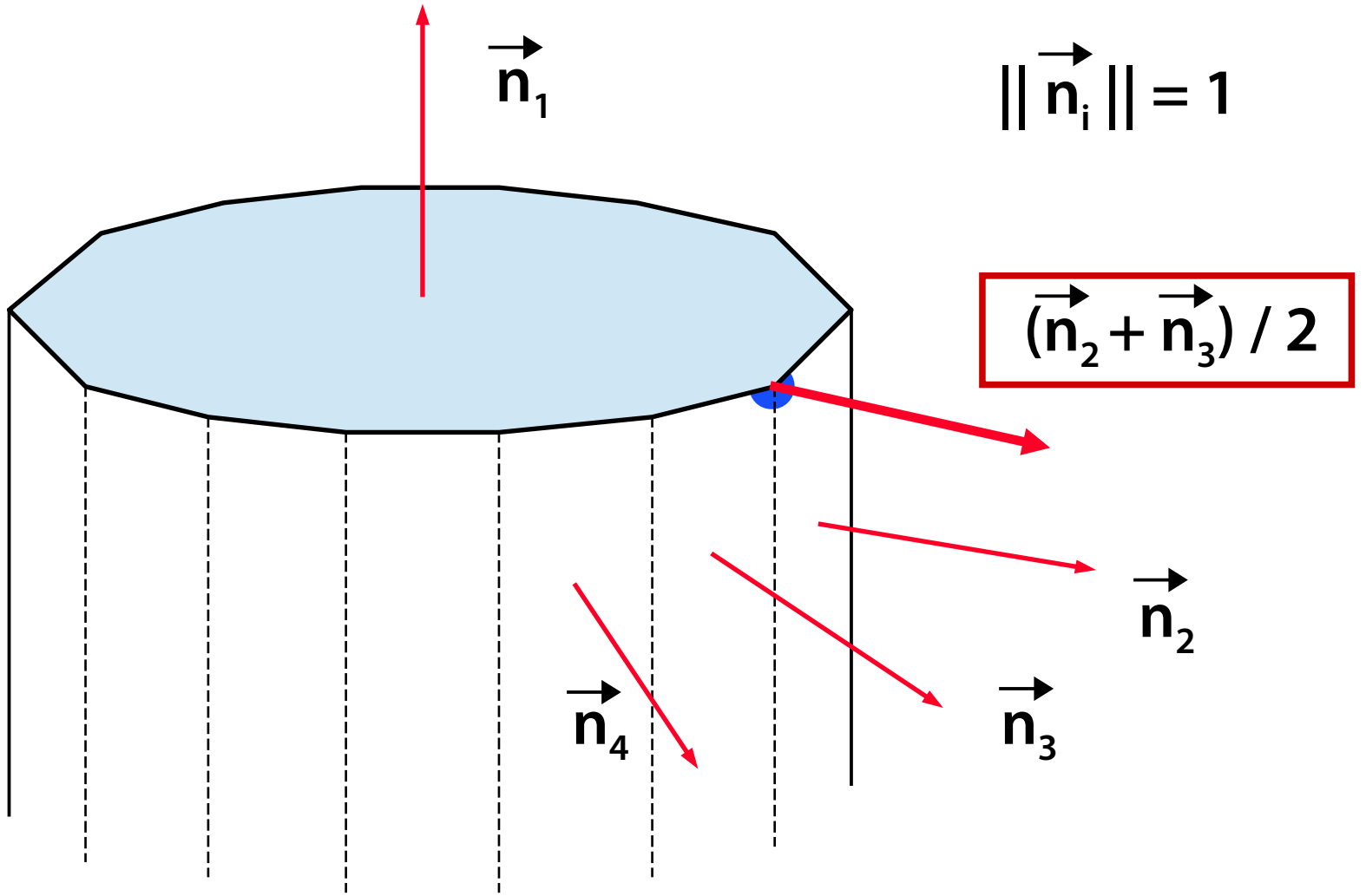
1. **Analyticky** – podle přesných vzorců původní plochy
2. **Aproximací** normál sousedních stěn

$$\vec{n} = (\vec{n}_1 + \vec{n}_2 + \vec{n}_3 + \vec{n}_4) / 4$$

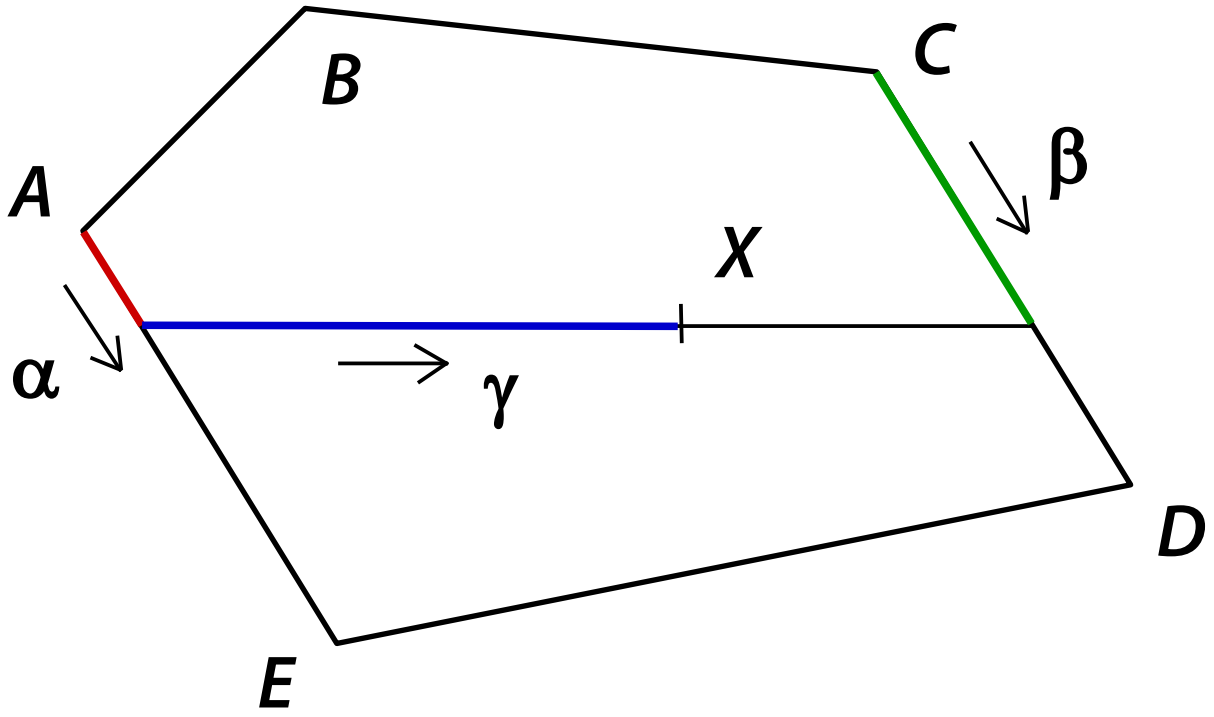
$$\| \vec{n}_i \| = 1$$



Skutečné a pomocné hrany



Bilineární interpolace



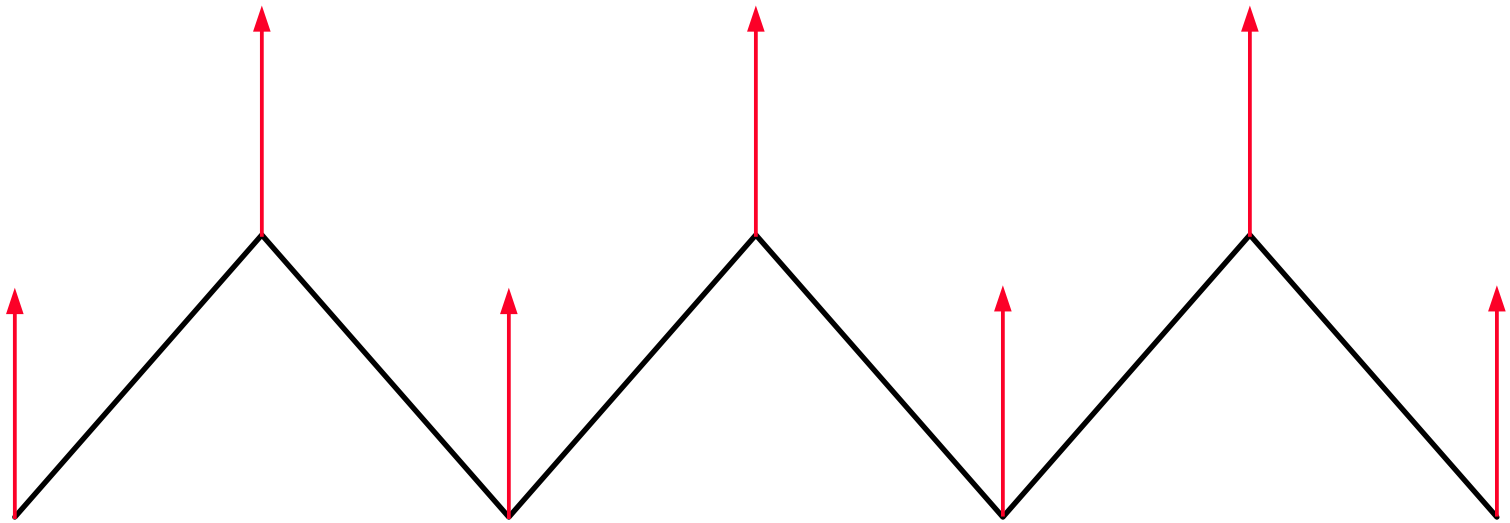
$$f_X = (1 - \gamma) \cdot [(1 - \alpha) \cdot f_A + \alpha \cdot f_E] + \gamma \cdot [(1 - \beta) \cdot f_C + \beta \cdot f_D]$$

Problémy interpolace (barvy)

Špatně vystihuje **maximum odrazu** (zejména zrcadlového odlesku)

Není invariantní k **otočení!**

Pozor na **špatný výpočet normál**!

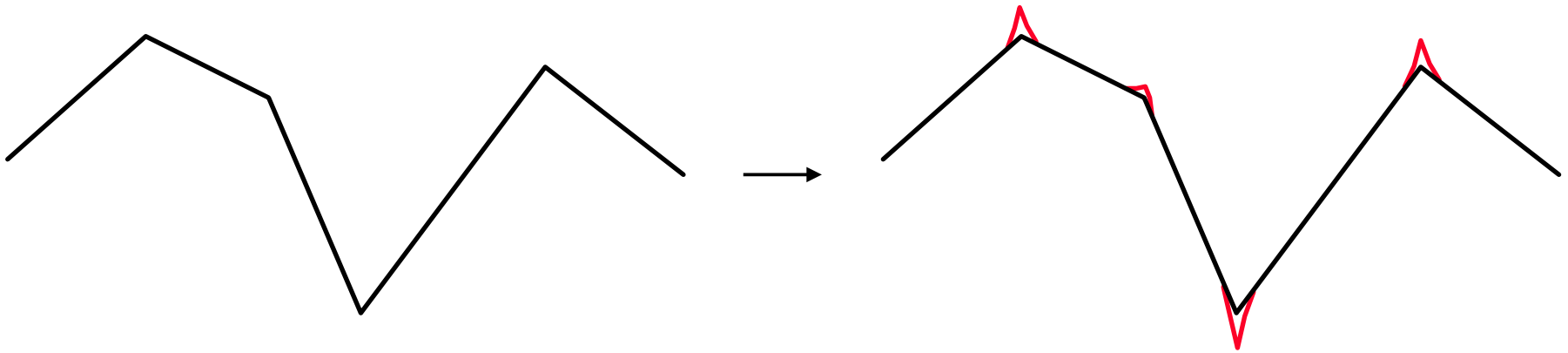


Machův efekt (1865)

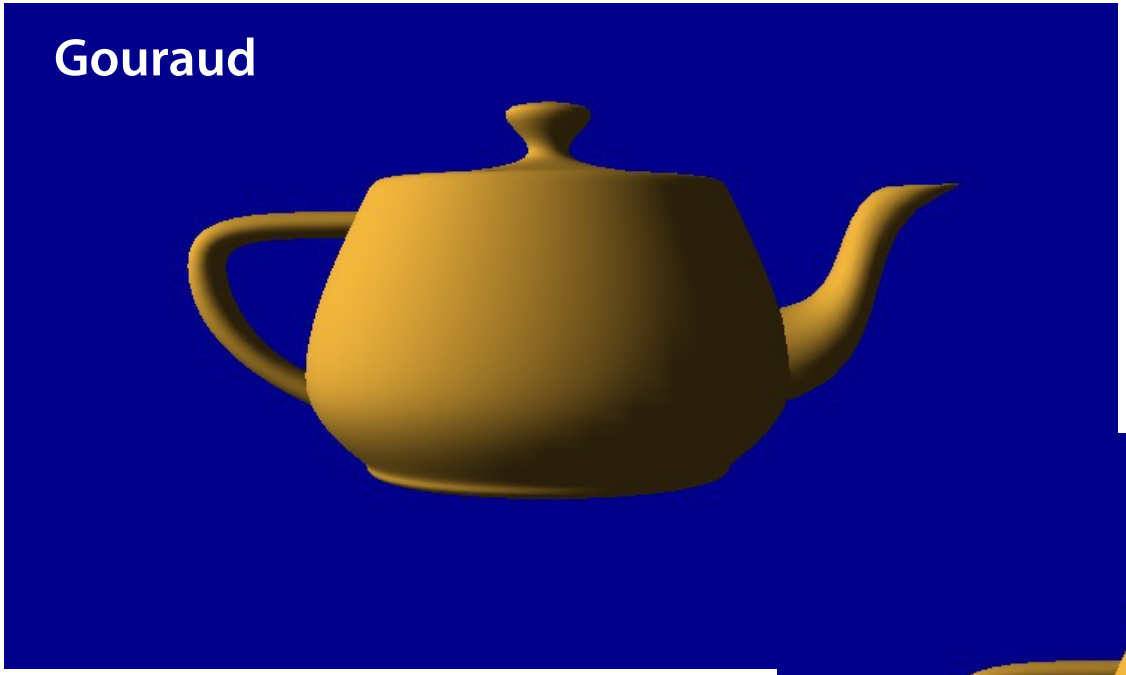
Zvýraznění **nespojitostí intenzity** nebo její **derivace**!

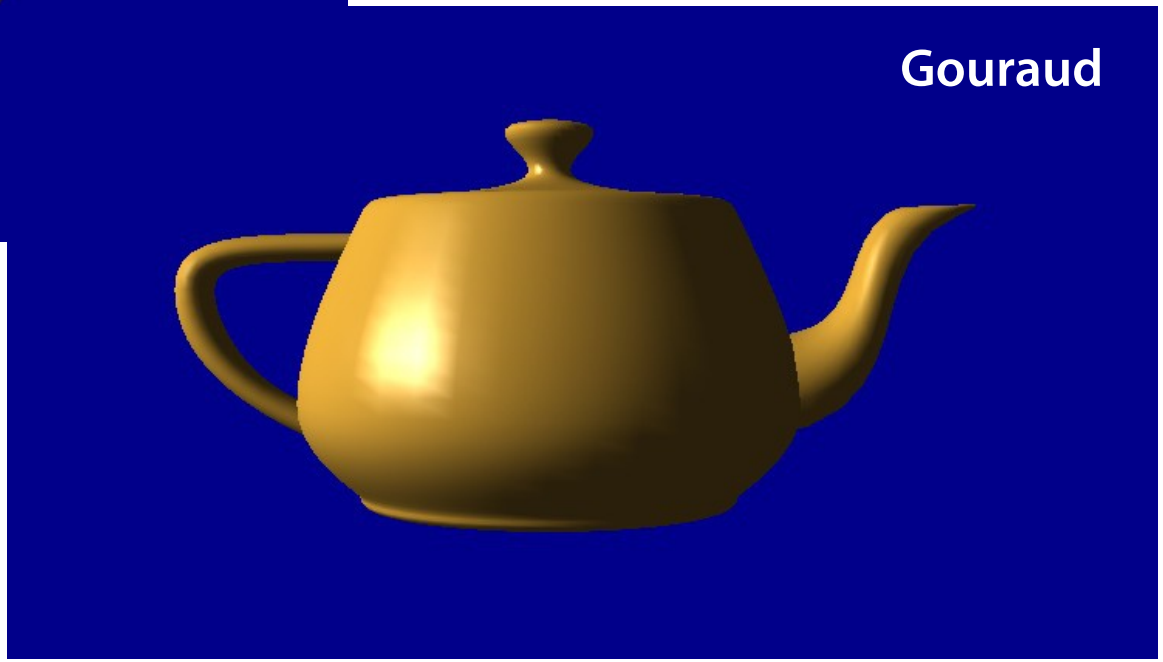
Je způsoben **laterální inhibicí** fotoreceptorů (sousedních neuronů) na sítnici

- aktivovaná buňka potlačuje citlivost sousedních buněk

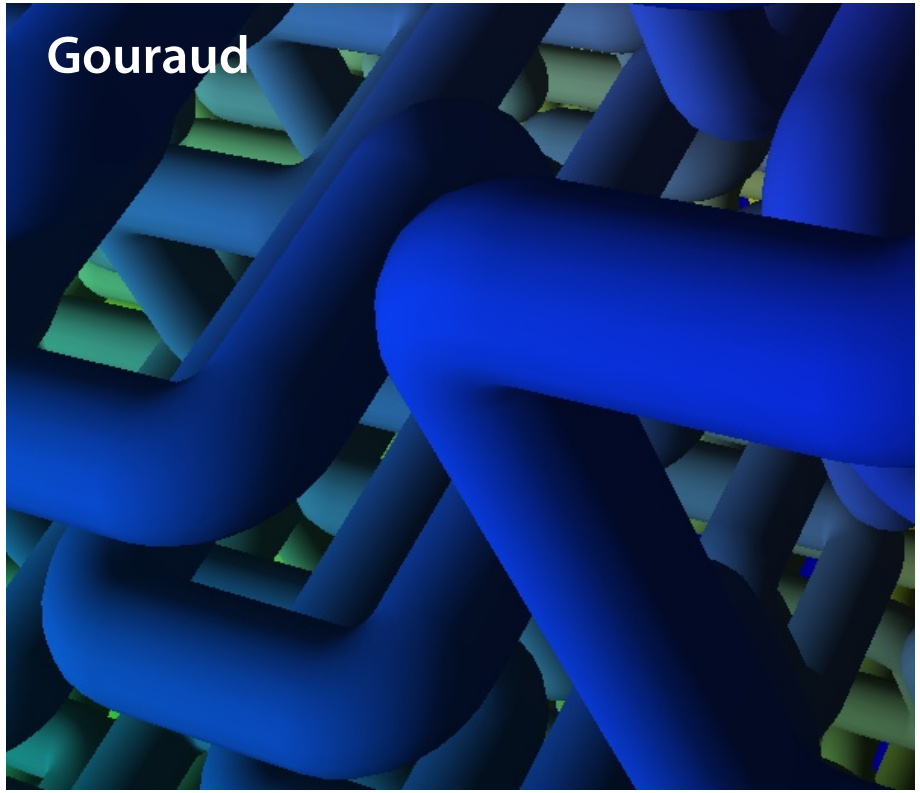


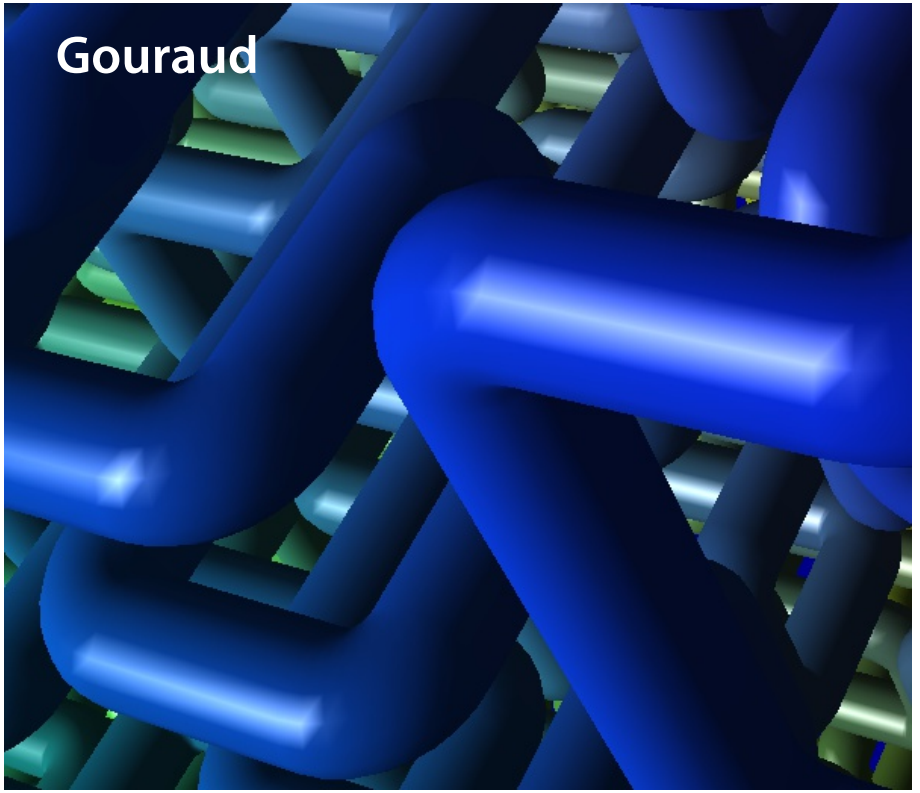
Interpolace barvy – mat a lesk





Interpolace barvy – mat a lesk





Phongovo stínování

V **umělých vrcholech** tělesa spočítám **normálové vektory**

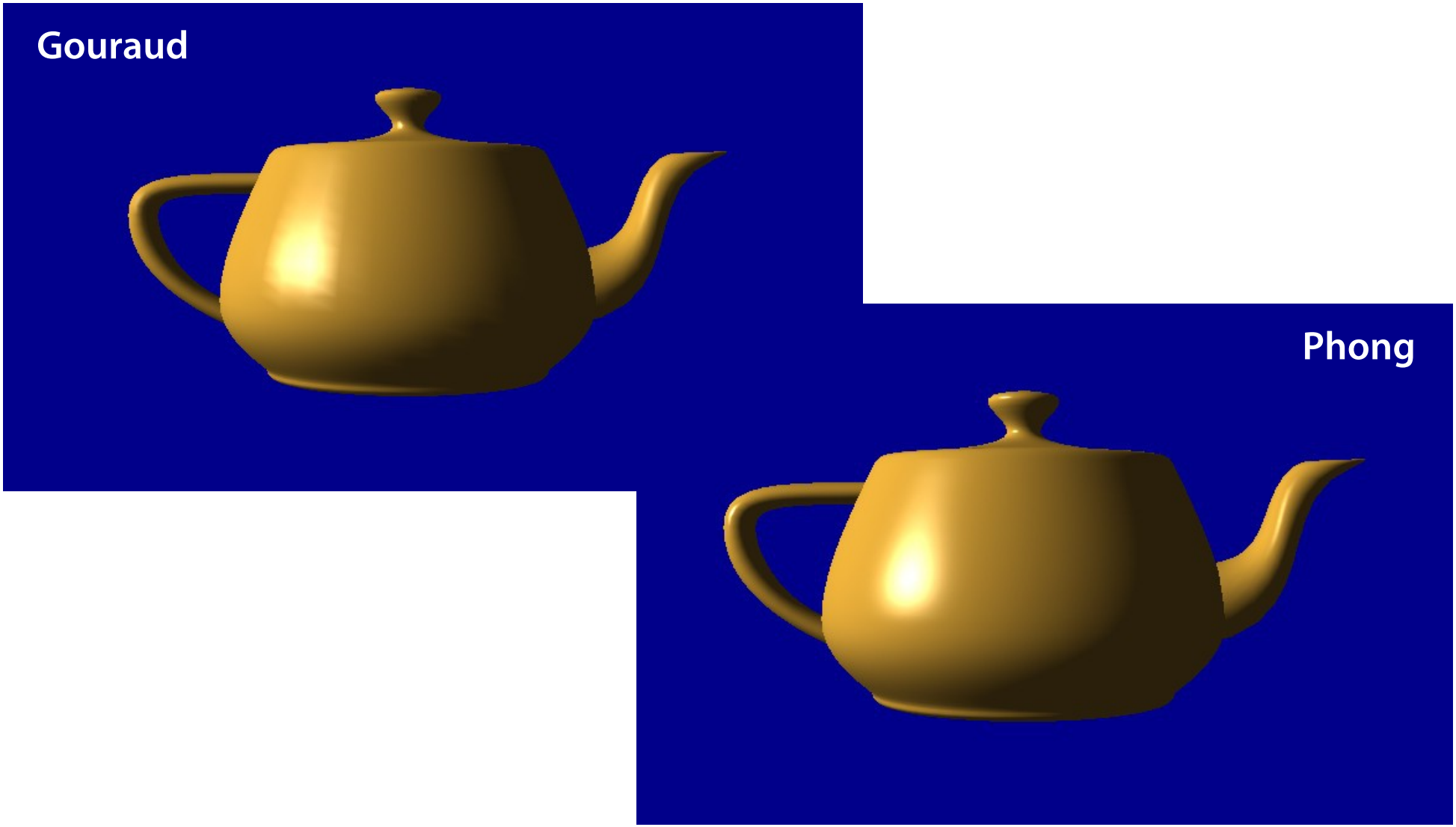
Uvnitř stěn dopočítávám normálu v každém pixelu **bilineární interpolací**

- vyplňování řádkovým rozkladem

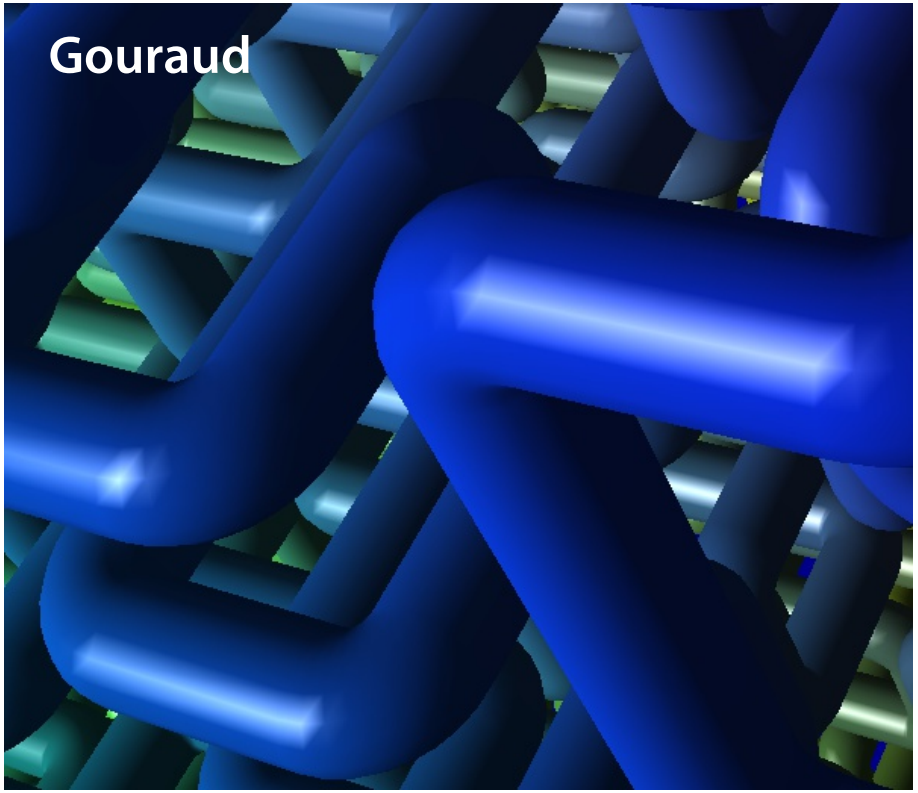
V **každém vnitřním pixelu** plochy počítám **osvětlení** (barvu)

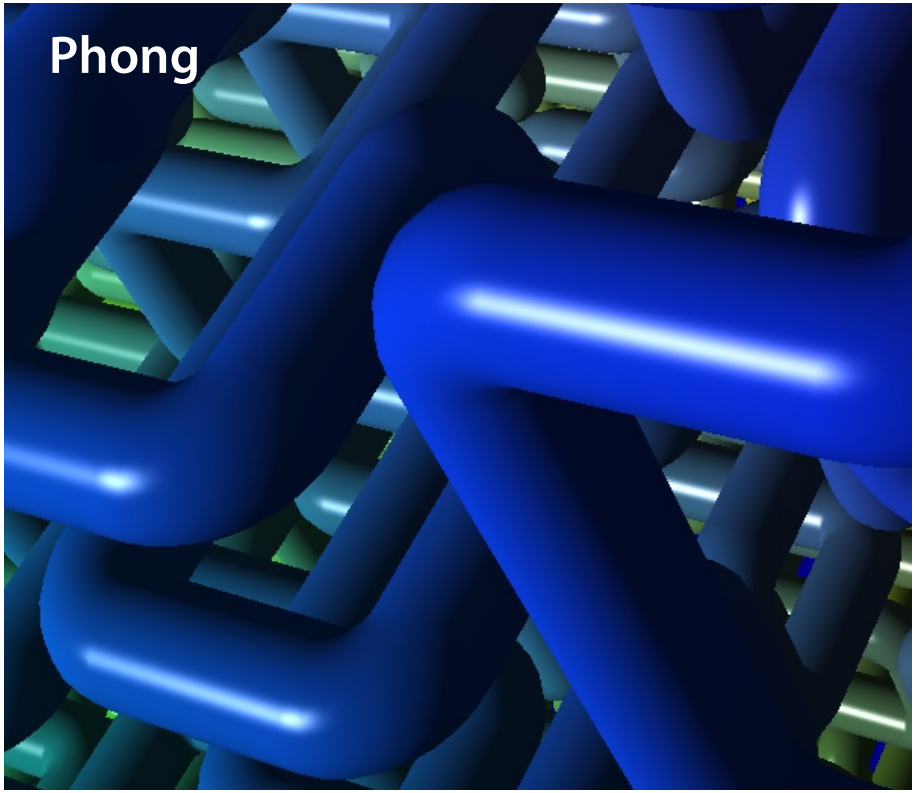
- aplikace zvoleného modelu osvětlení

Interpolace barvy vs. normály



Interpolace barvy vs. normály





Větší výpočetní náročnost (Phong)

Normála se dopočítává **v každém pixelu**

- bilineární interpolace a normalizace vektoru – výpočet **odmocniny**
- existují přibližné metody interpolace bez odmocňování

V každém pixelu se počítá **model osvětlení**

- skalární součiny, umocňování, dělení

Literatura

J. Foley, A. van Dam, S. Feiner, J. Hughes: ***Computer Graphics, Principles and Practice***, 734-741

Jiří Žára a kol.: ***Počítačová grafika, principy a algoritmy***, 355-361

matfyz

Computer Graphics Charles University

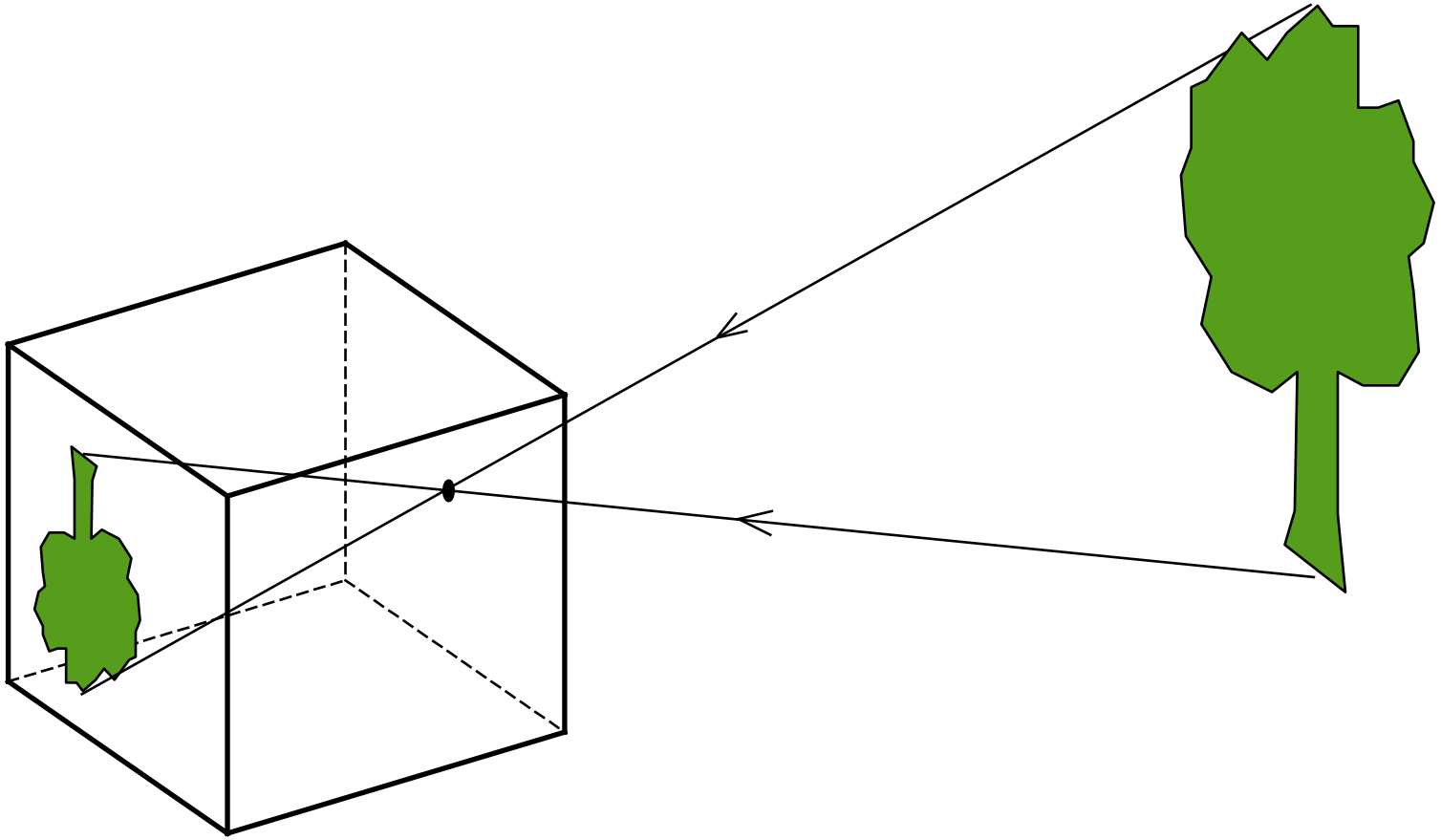
Vrhání paprsku (CSG)

CGG MFF UK Praha

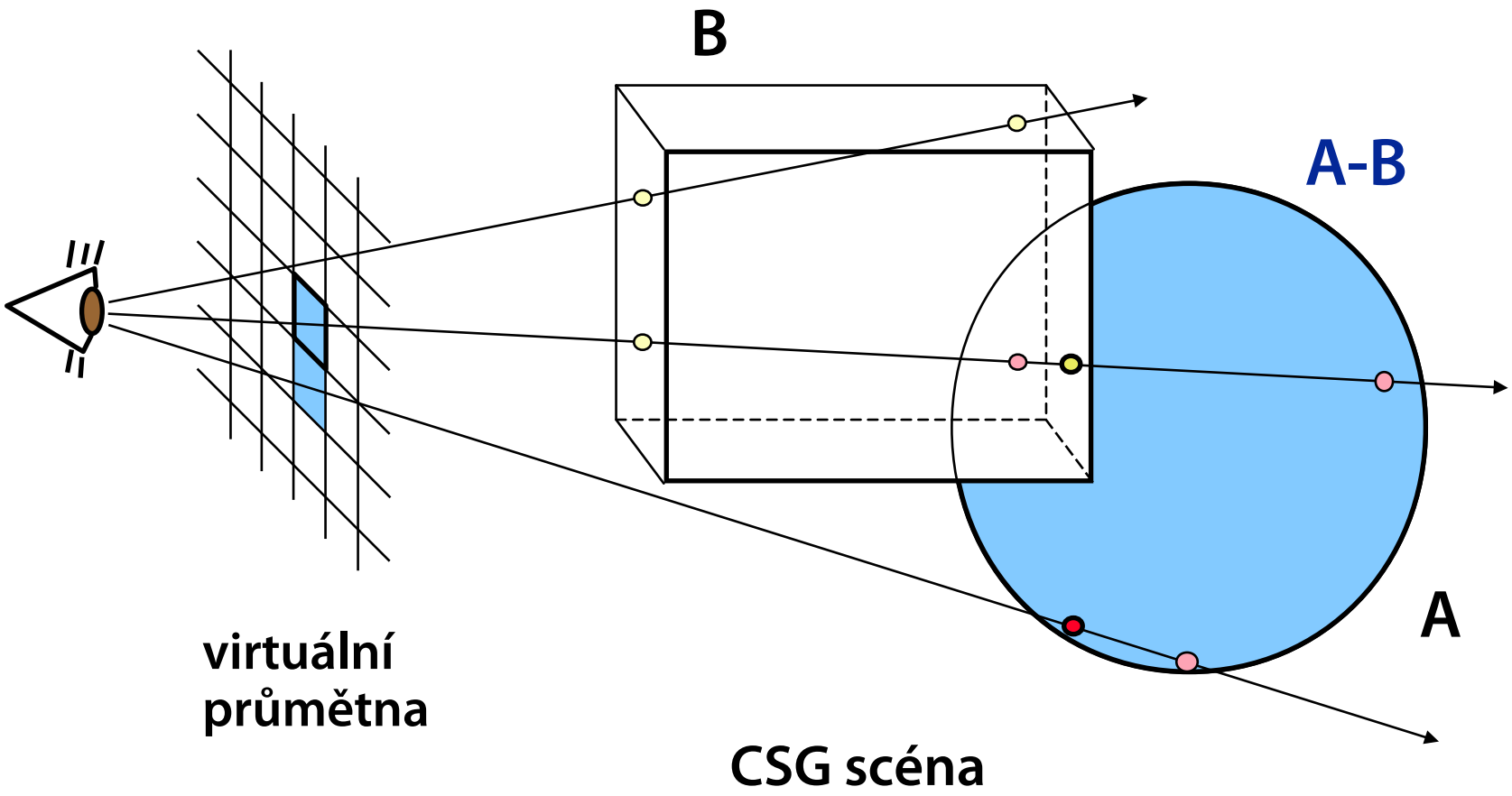
pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/

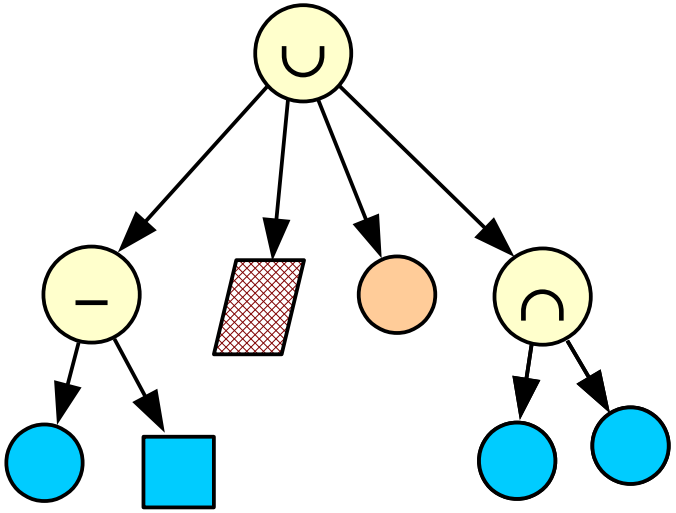
Model dírkové kamery

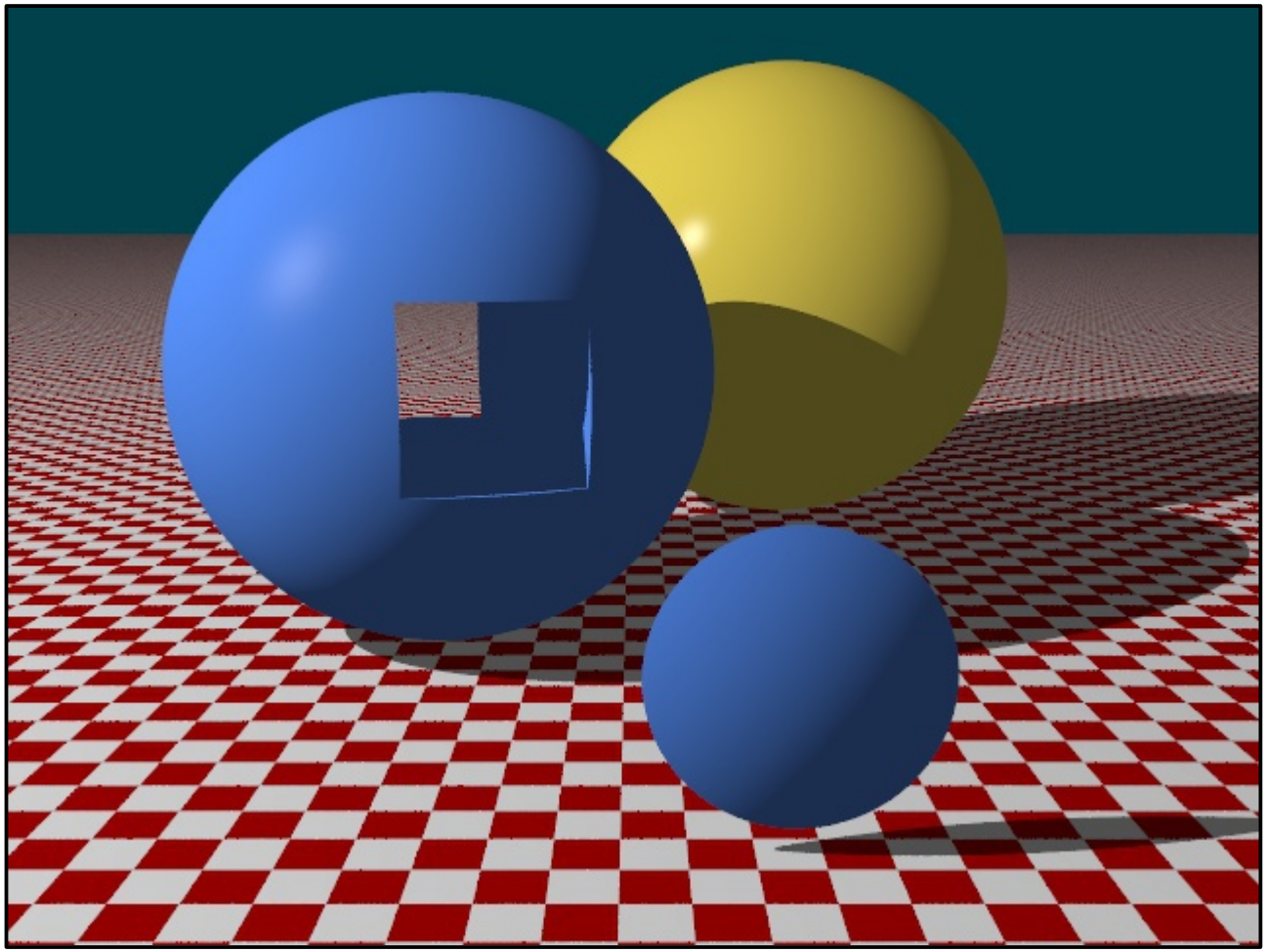


Zobrazování vrháním paprsku



Příklad CSG scény (i rozdíl a průnik)





Průsečík paprsku s CSG

Pro **elementární tělesa** umím průsečíky spočítat

- začátek a konec průniku paprsku s tělesem pro konvexní tělesa

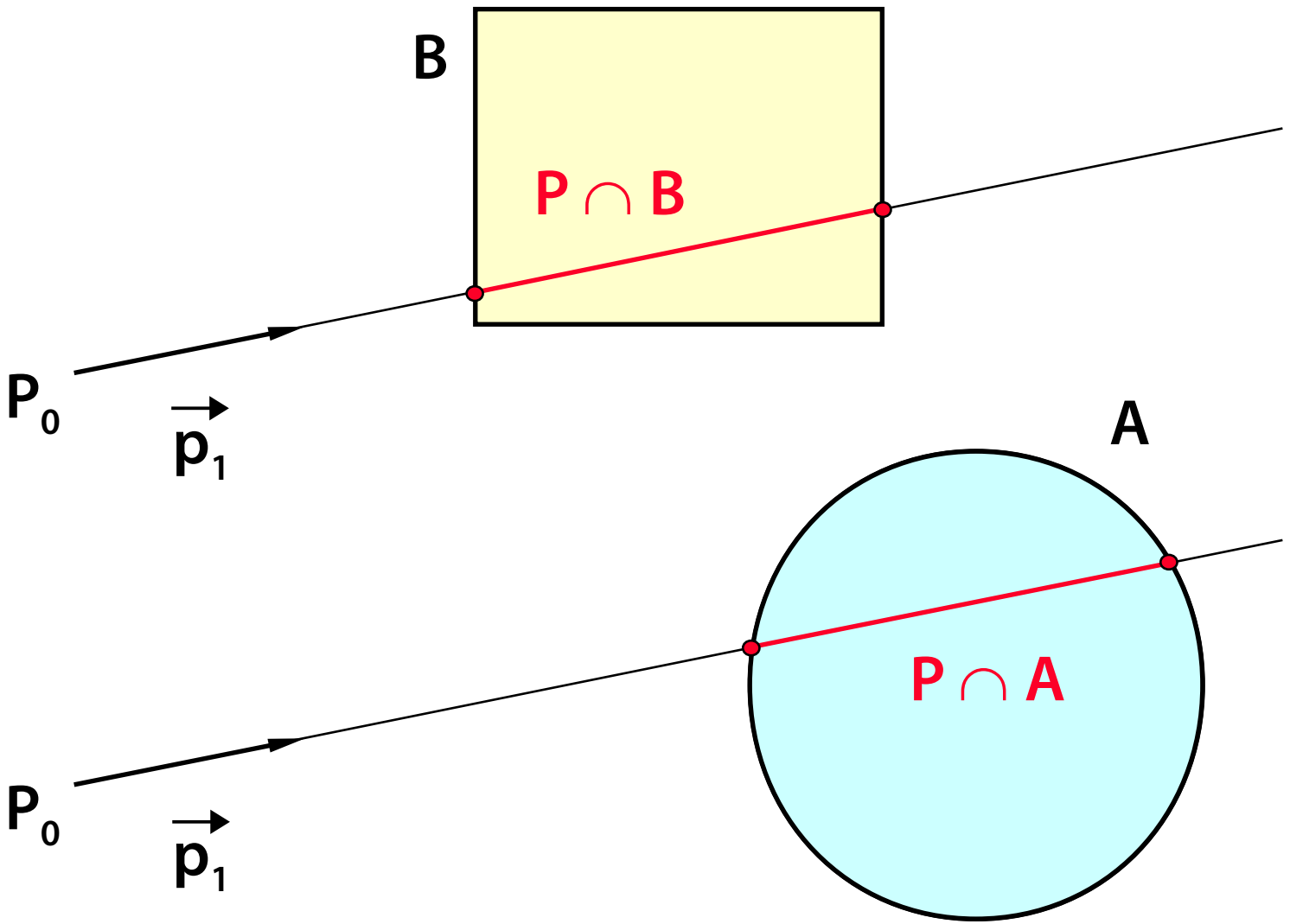
Množinové operace provádím na polopřímce paprsku

- distributivita: $\mathbf{P \cap (A - B) = (P \cap A) - (P \cap B)}$
- obecný průnik paprsku se scénou je množina intervalů

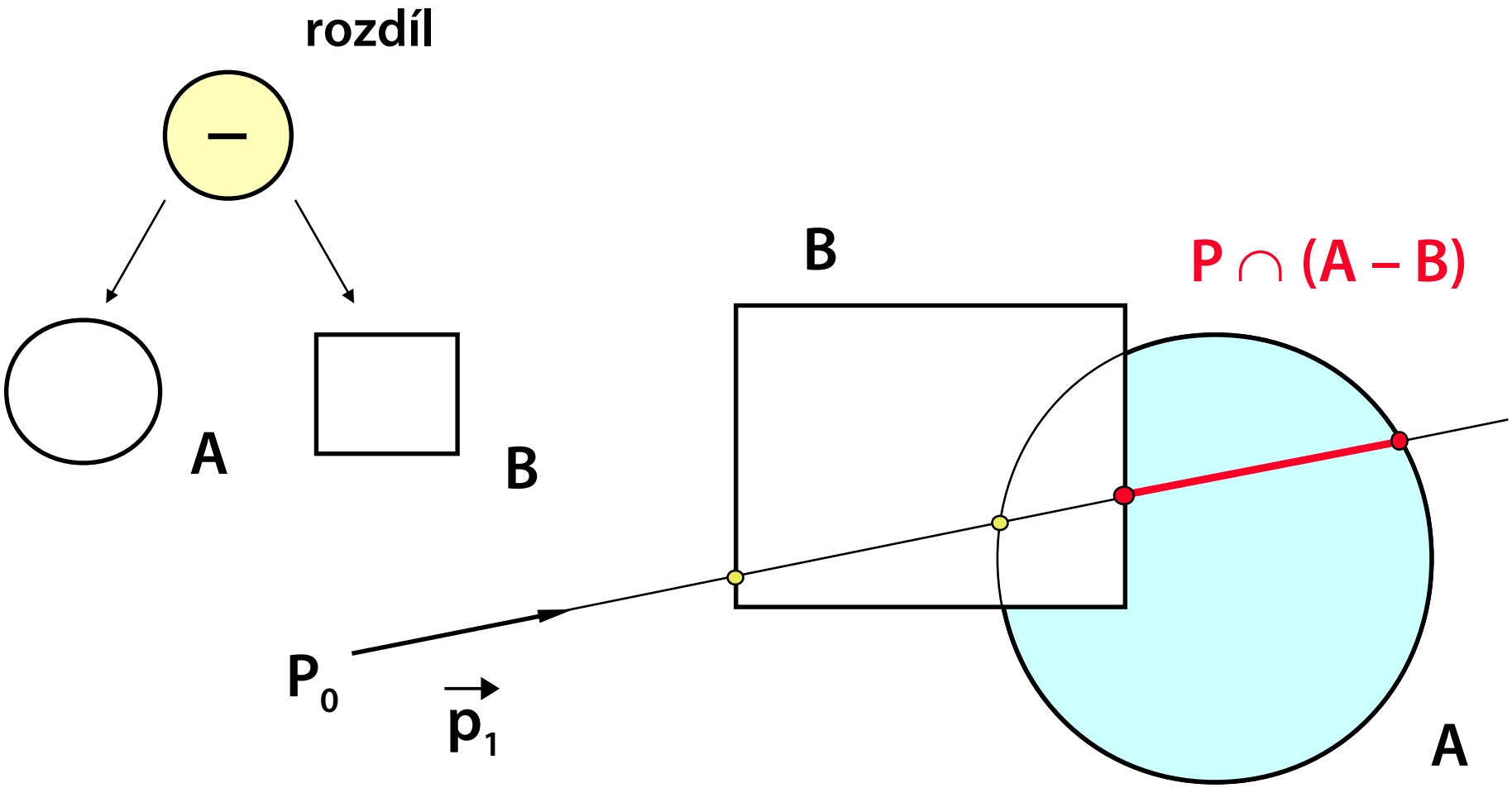
Geometrické transformace

- na paprsek aplikuji inverzní transformace

Průsečíky $P \cap A$, $P \cap B$



Průsečík $P \cap (A - B)$



Implementace

Paprsek

- počáteční bod $\mathbf{P}_0$ a směrový vektor $\overrightarrow{\mathbf{p}_1}$
- transformuje se inverzními maticemi $\mathbf{T}_i^{-1}$

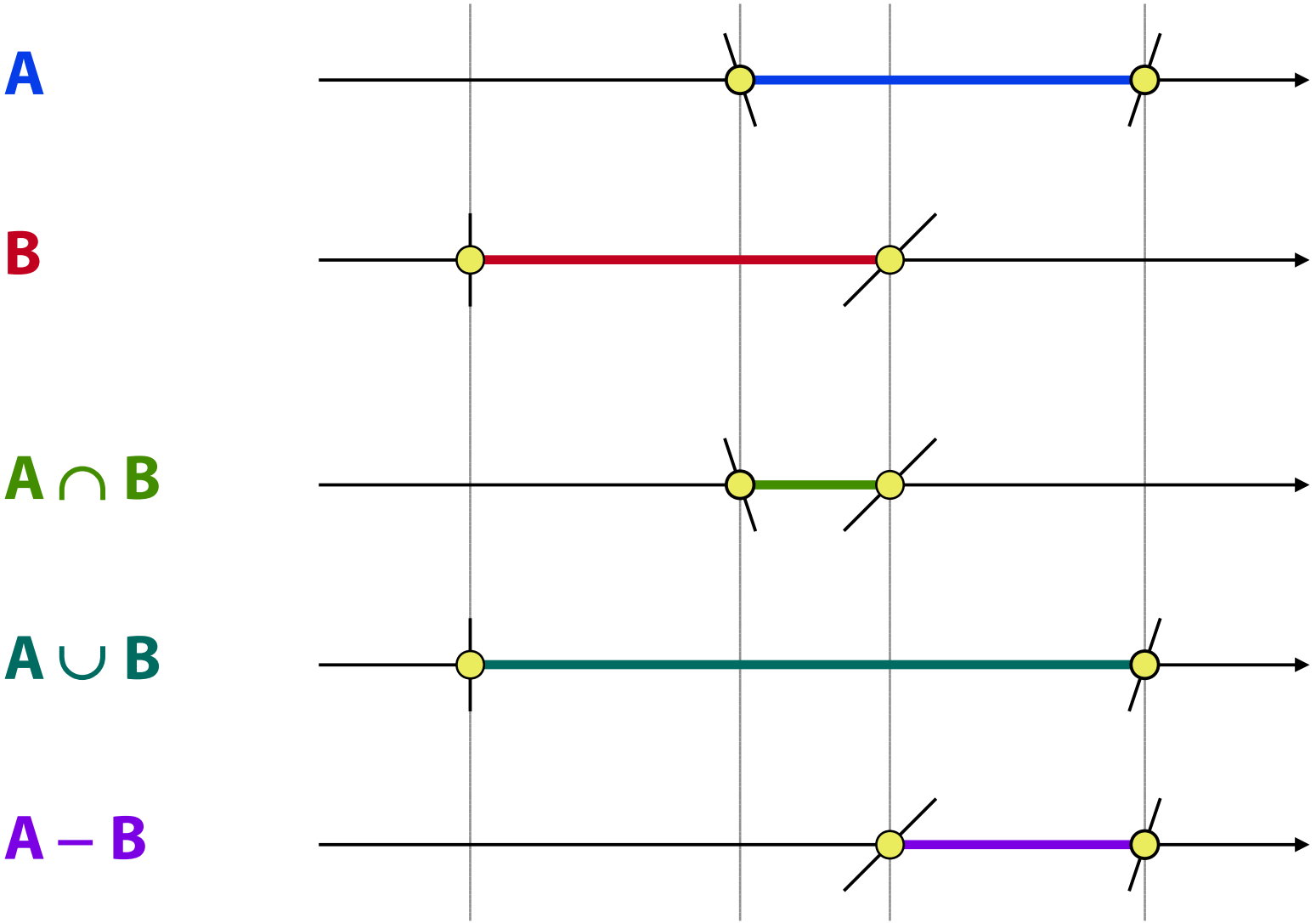
Průnik paprsku se scénou (částí scény)

- uspořádaný seznam hodnot parametru **t**: $[\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3 \ldots]$

Množinové operace

- zobecněné slévání vstupních seznamů – např. $[\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3 \ldots]$ a $[\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3 \ldots]$
- viz seznam řádkových změn („X-transition list")

Množinové operace na paprsku



Určení barvy pixelu

Průnik paprsku s CSG scénou **je prázdný**

- barva pozadí

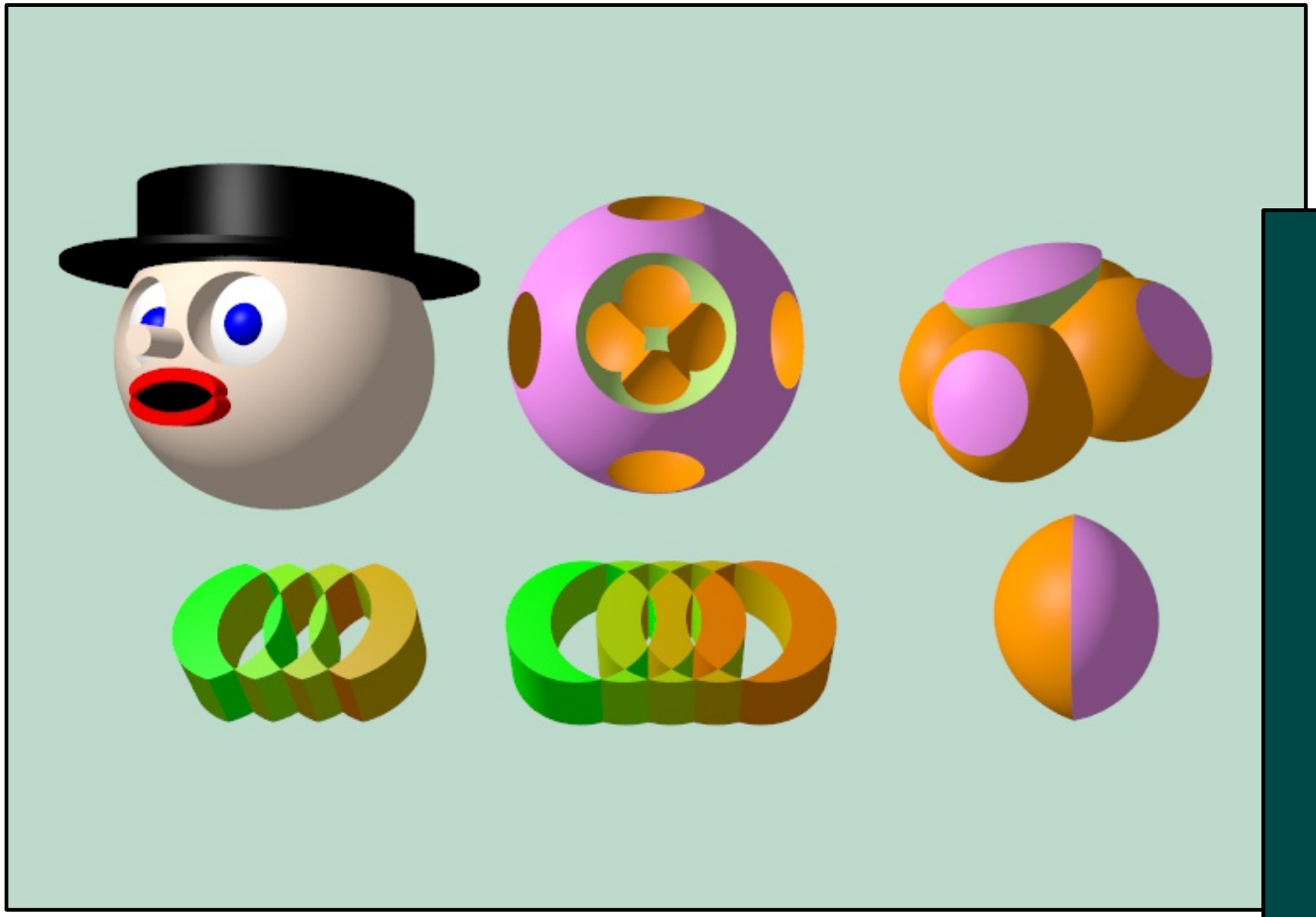
Průnik je neprázdný

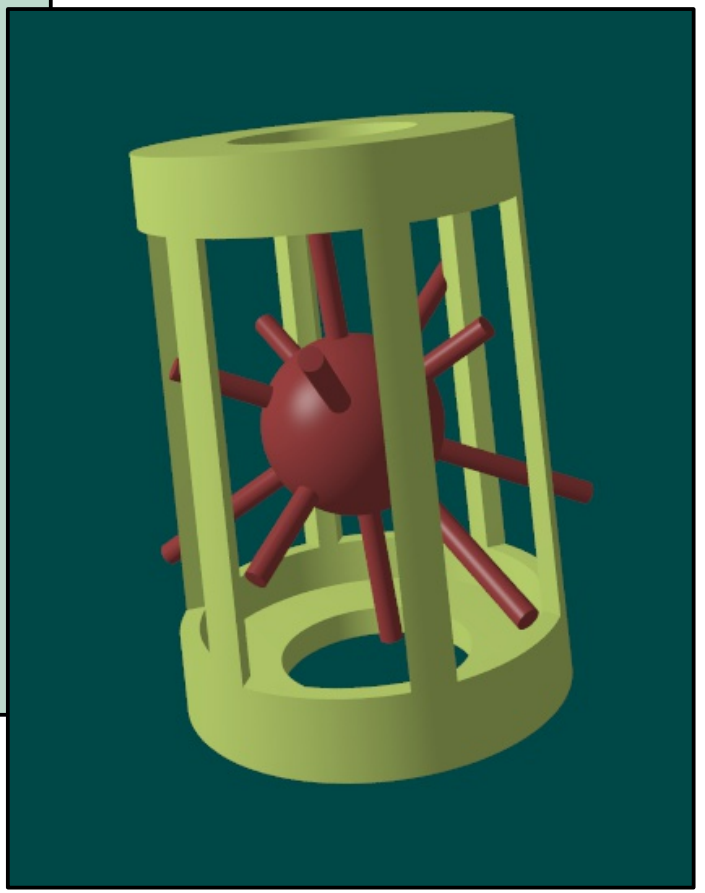
- barva tělesa (podle prvního záznamu – $\mathbf{t}_1$)
- možnost stínování (normálový vektor v místě průsečíku)

Obarvení podle **typu množinové operace**

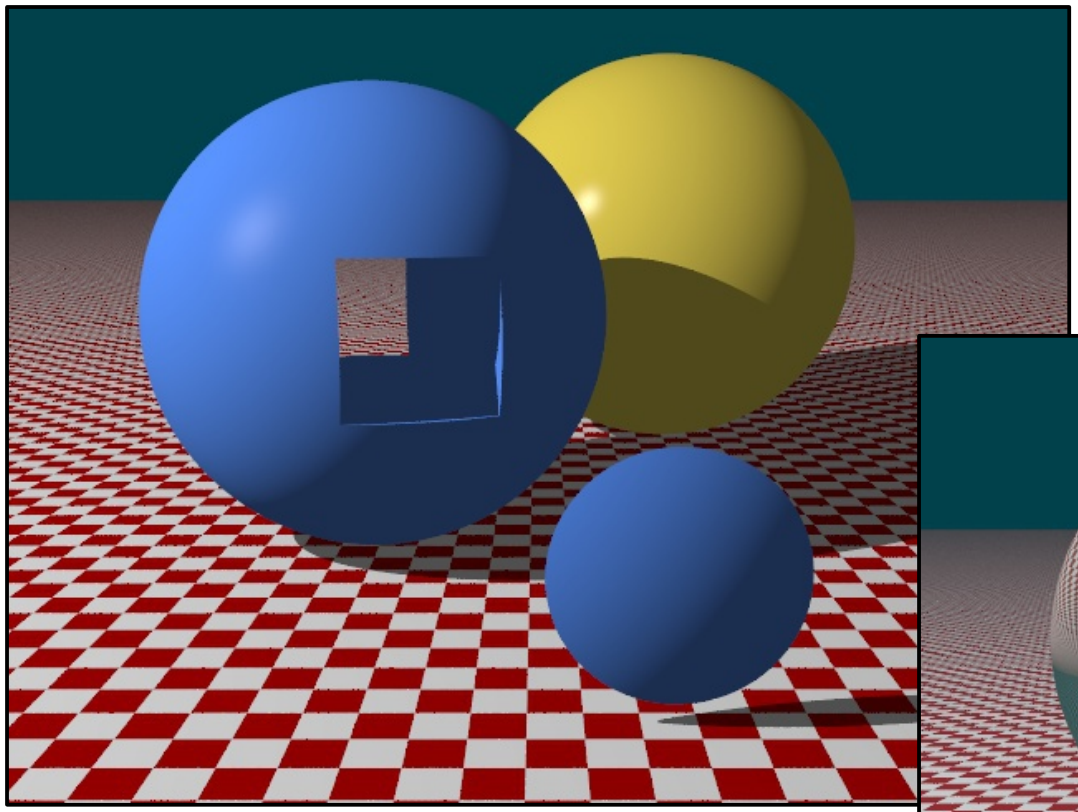
- složitější pravidla přenášení barev při výpočtu množinových operací
- např. speciální barva pro odečtenou část tělesa

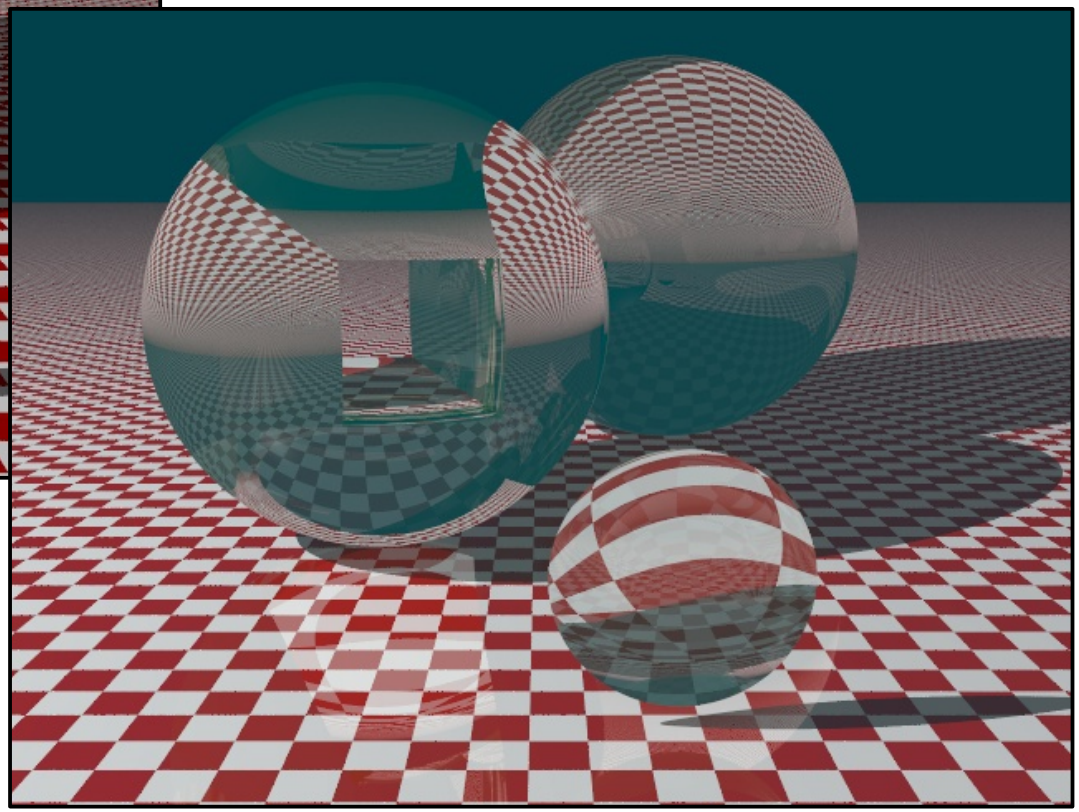
Další ukázky





Ray tracing: více paprsků, odrazy, lomy…





Literatura

J. Foley, A. van Dam, S. Feiner, J. Hughes: ***Computer Graphics, Principles and Practice***, 712-714

matfyz

Computer Graphics Charles University

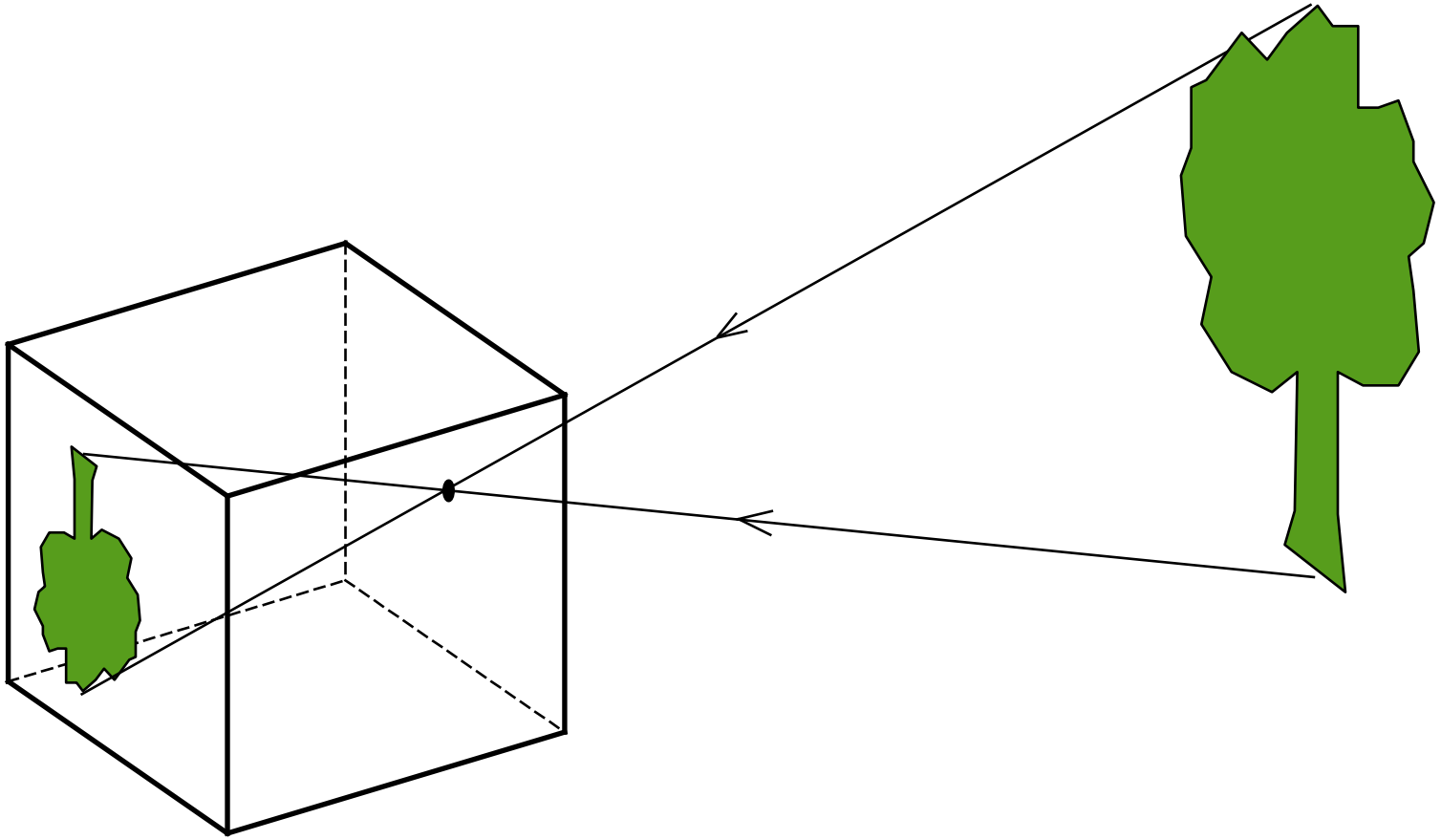
Rekurzivní sledování paprsku (Ray-tracing)

CGG MFF UK Praha

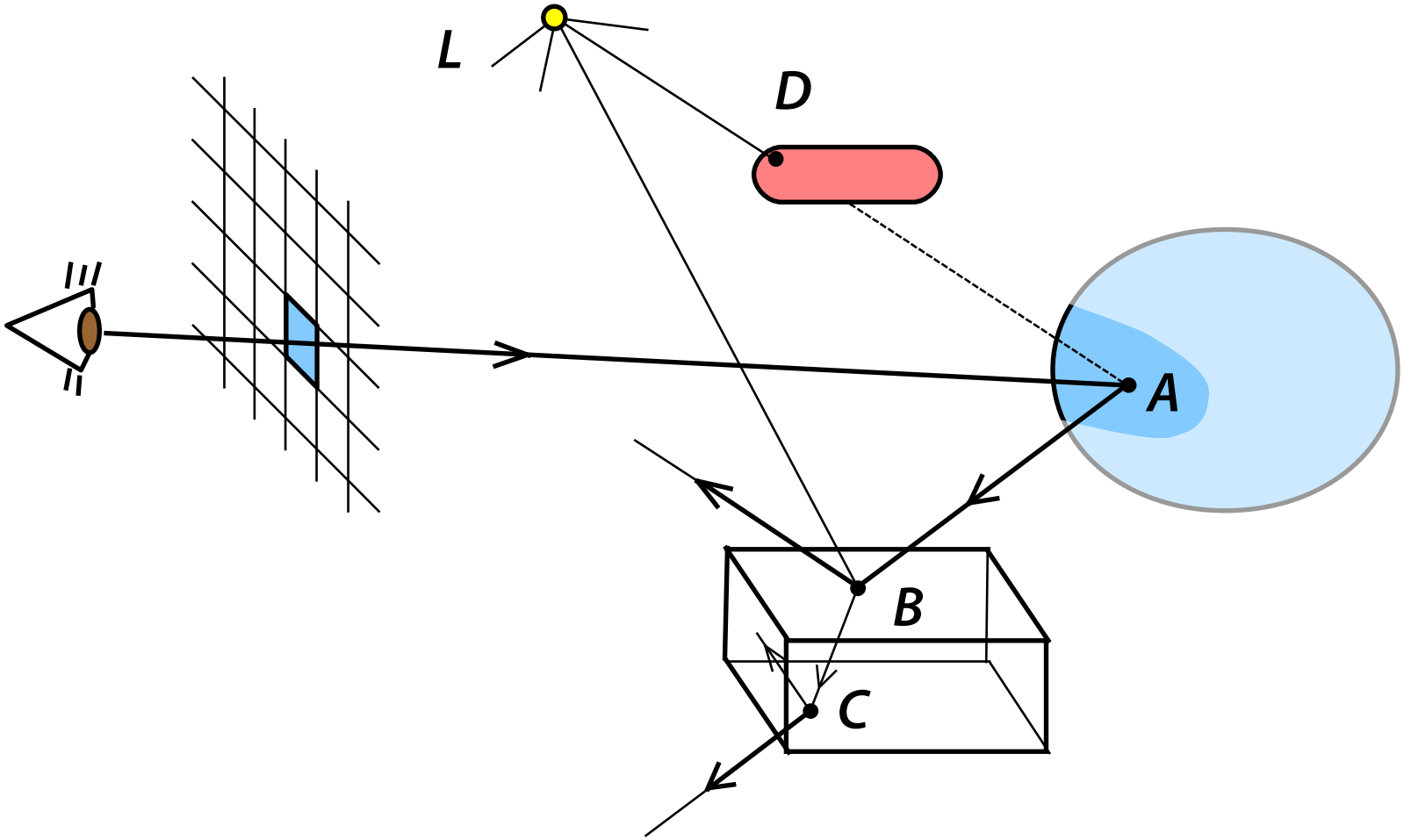
pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/

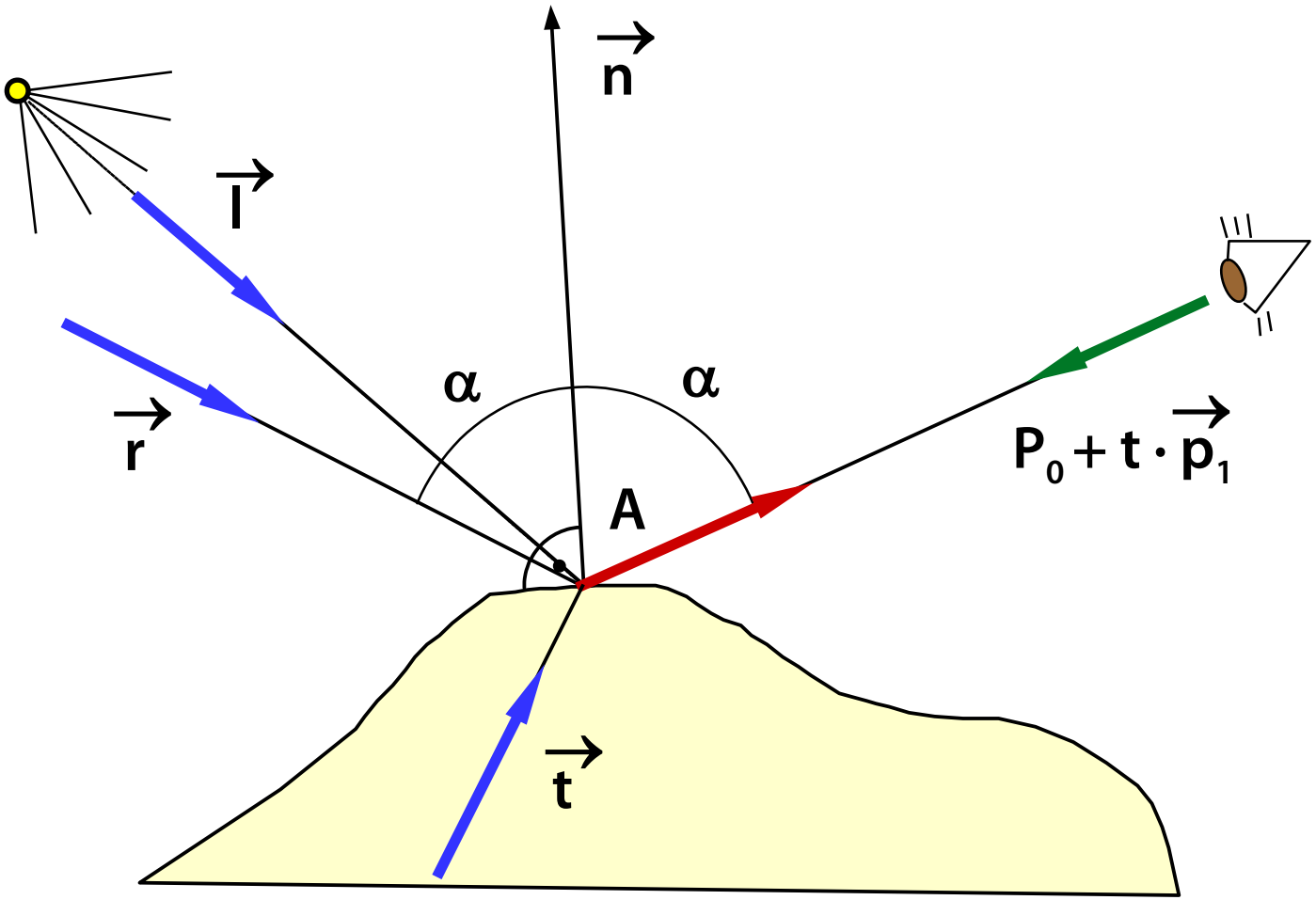
Model dírkové kamery



Zpětné sledování paprsku



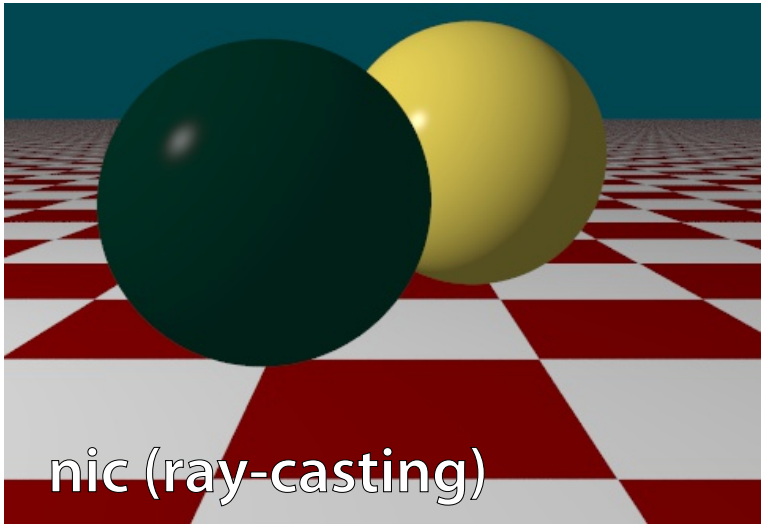
Skládání světla

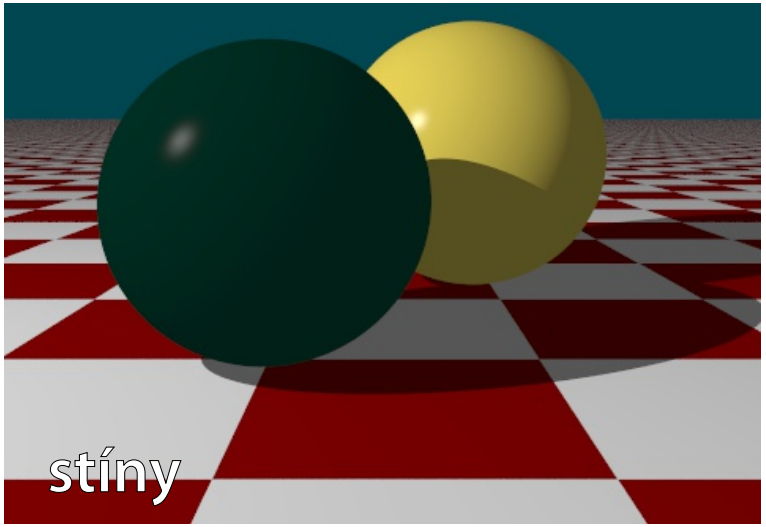


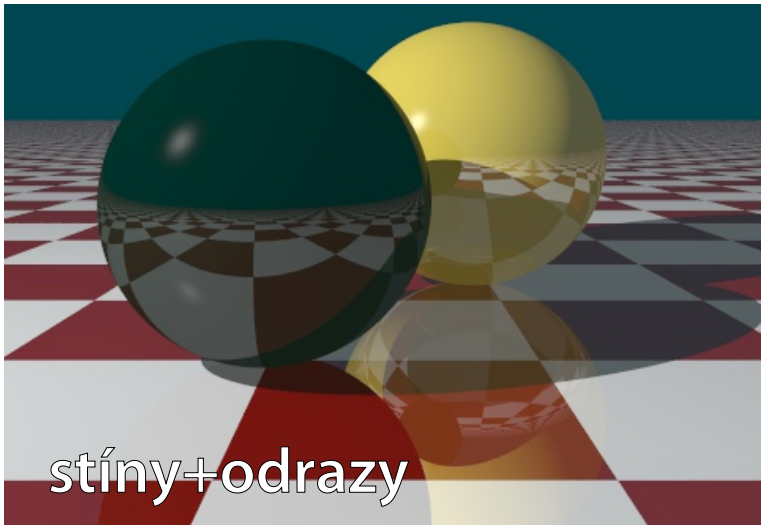
Rekurzivní implementace

```
int maxDepth = 10;              // Maximum recursion level.
RayScene scene;                 // Global scene object (geometry, materials, light sources…)

RGB shade (Vector3d P0, Vector3d p1, int depth)
  // P0 – ray origin, p1 – ray direction, depth – interactions so far
{
  Vector3d A = intersection(scene, P0, p1);
  if (!isValid(A)) return scene.background; // No intersection at all.

  RGB color{0};                 // Result color.
  for (const auto& light : scene.lightSources)
    if (!isValid(intersection(scene, A, light.point - A)))
      color += scene.kL(A) * light.contribution(A, -p1, scene.material(A), scene.normal(A));

  if (++depth >= maxDepth) return color;

  if (scene.isGlossy(A))        // Recursion - reflection.
  {
    Point3d r = reflection(p1, scene.normal(A));
    color += scene.kR(A) * shade(A, r, depth);
  }

  if (scene.isTransparent(A)) // Recursion – refraction.
  {
    Point3d t = refraction(p1, scene.normal(A), scene.index(A));
    color += scene.kT(A) * shade(A, t, depth);
  }

  return color;
}
```

Jednotlivé složky







Řízení hloubky rekurze

Statické – omezení konstantou (nehodí se pro scény obsahující zrcadla i méně odrazivé lesklé povrchy)

Dynamické – podle „významu“ (importance, performance) paprsku

- „**význam**” je procentuální podíl právě sledovaného paprsku na výsledné barvě pixelu (pro primární paprsky: 100%)
- omezení „významu” konstantou (např. 1-2%)

Kombinované – omezení hloubky i „významu” paprsku

Výpočet průsečíku

Geometrický výpočet, jehož výsledkem jsou

- souřadnice průsečíku (stačí 1D, speciální hodnota: „nekonečno”)
- normálový vektor povrchu tělesa
- 2D texturové souřadnice
- číslo tělesa (plochy), odkaz na těleso (barva, materiál…)

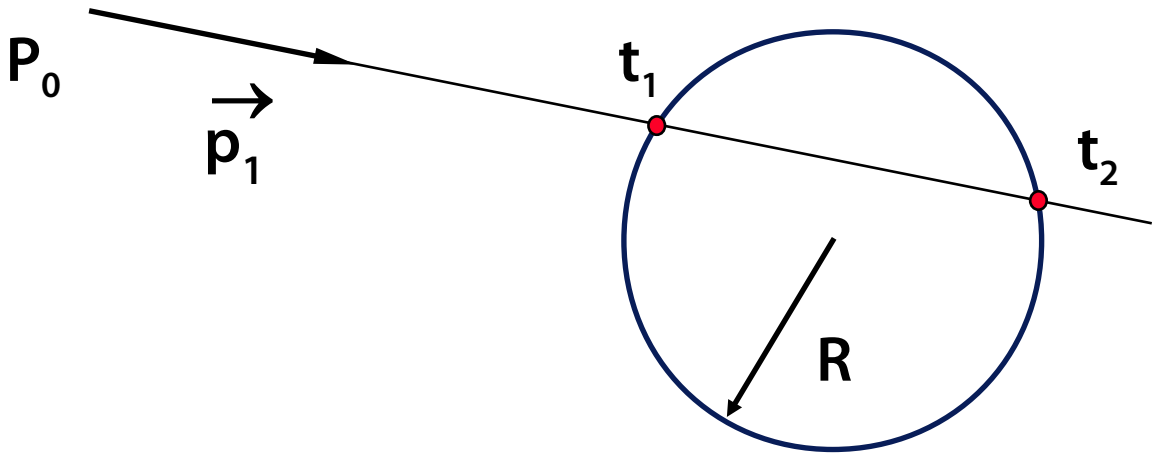
Časově nejnáročnější operace (90-95% času)

- urychlovací metody

Analytický výpočet (koule, válec, kvádr…)

Numerický výpočet (aproximační plochy, rotační tělesa, implicitní povrchy…)

Průsečík paprsku s koulí



$P_0 = [x_0, y_0, z_0]$

$\vec{p}_1 = [x_1, y_1, z_1]$

Paprsek: $\mathbf{P(t) = P_0 + t\,\vec{p}_1,\quad t > 0}$ (1)

Koule (střed v počátku): $\mathbf{x^2 + y^2 + z^2 - R^2 = 0}$ (2)

Po dosazení (1) do (2) vyjde **kvadratická rovnice (*t*)**

$$\mathbf{t^2 (x_1^2 + y_1^2 + z_1^2) + 2t (x_0x_1 + y_0y_1 + z_0z_1) + x_1^2 + y_1^2 + z_1^2 - R^2 = 0}$$

Průsečík s CSG scénou

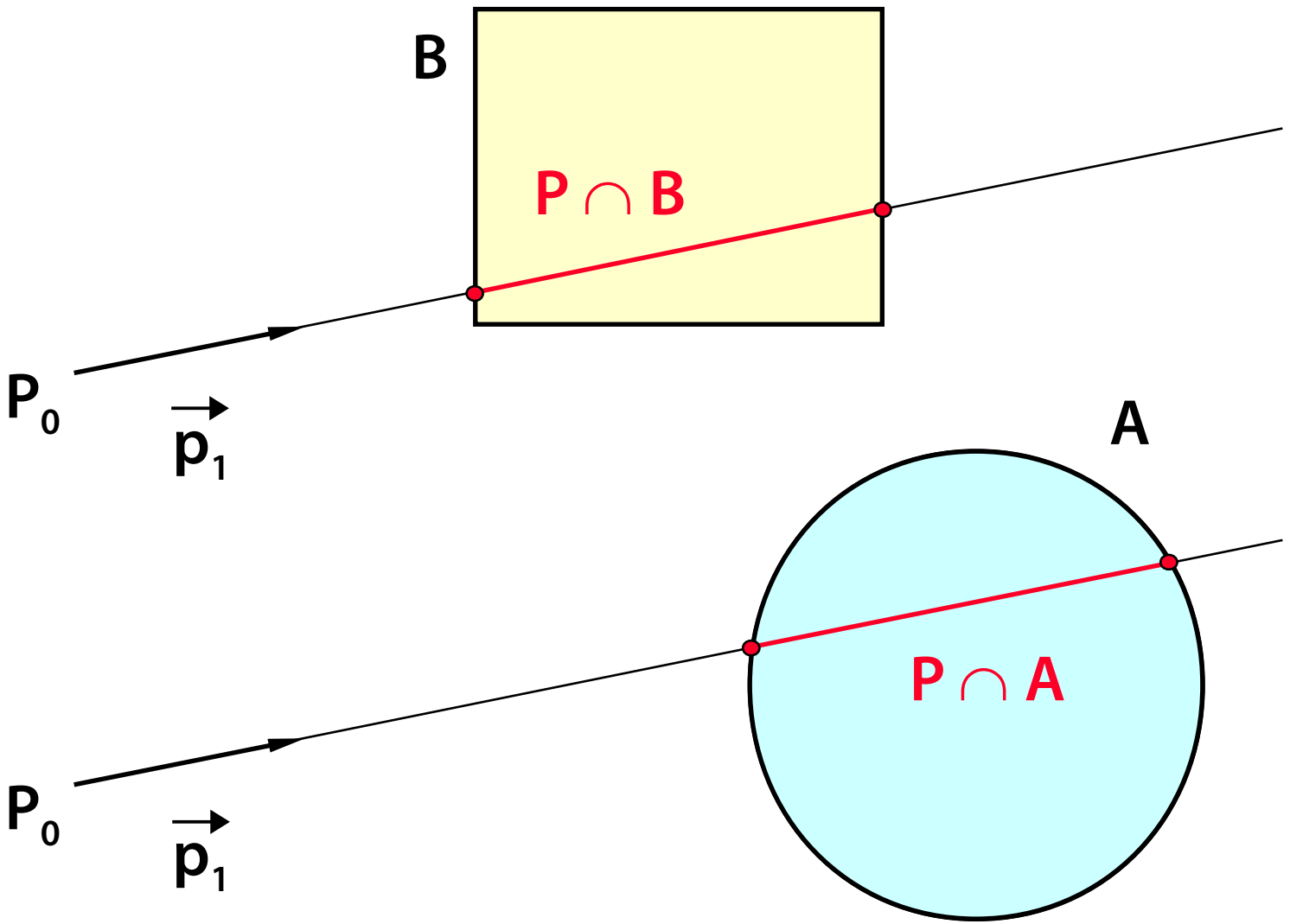
Pro **elementární tělesa** lze snadno průsečíky spočítat

- začátek a konec průniku paprsku s tělesem pro konvexní tělesa

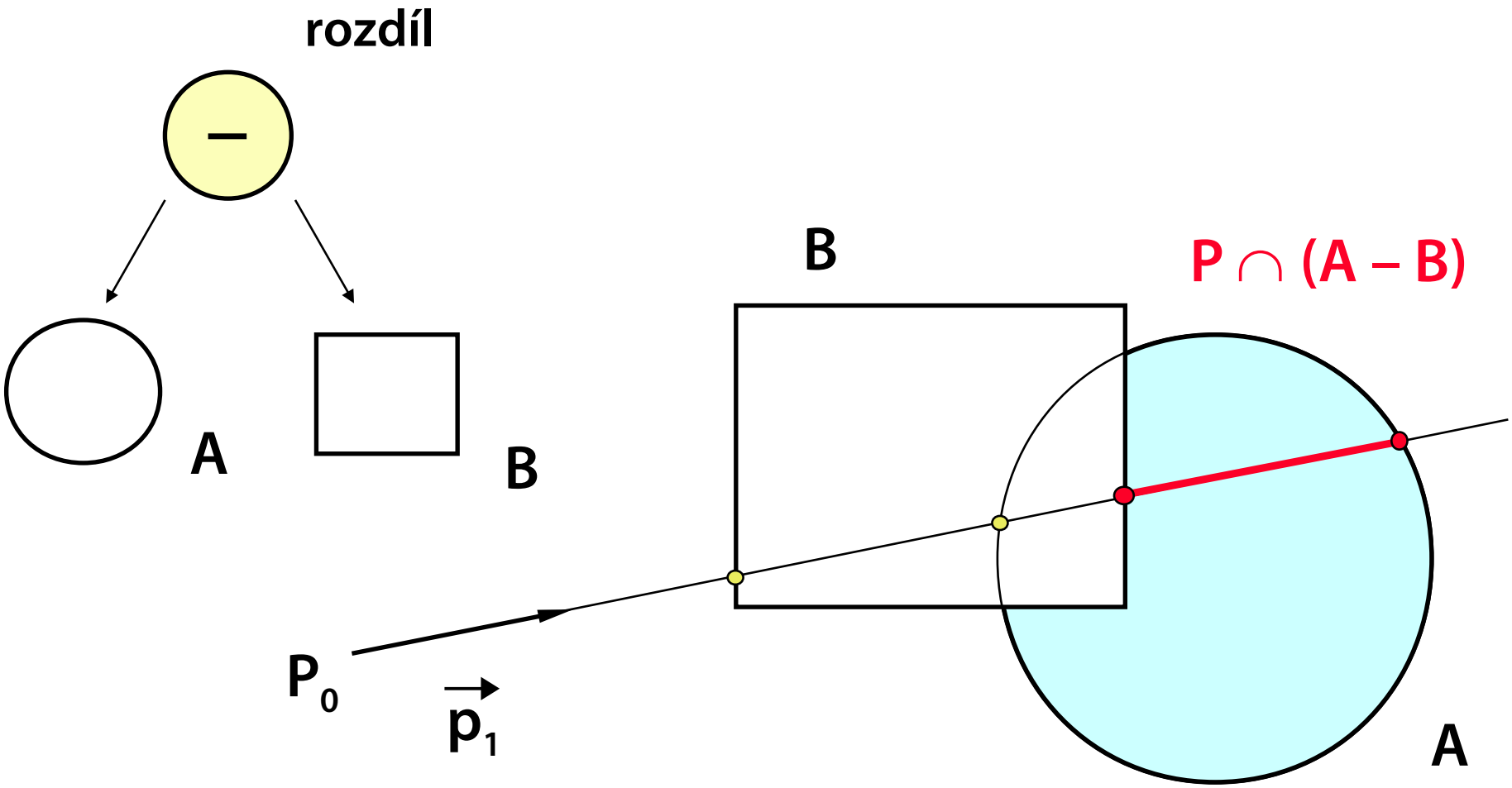
Množinové operace se provádí na polopřímce paprsku

- díky distributivitě: $\mathbf{P} \cap (\mathbf{A} - \mathbf{B}) = (\mathbf{P} \cap \mathbf{A}) - (\mathbf{P} \cap \mathbf{B})$
- obecný průnik paprsku se scénou je množina intervalů

Průsečíky $P \cap A$, $P \cap B$



Průsečík $P \cap (A - B)$



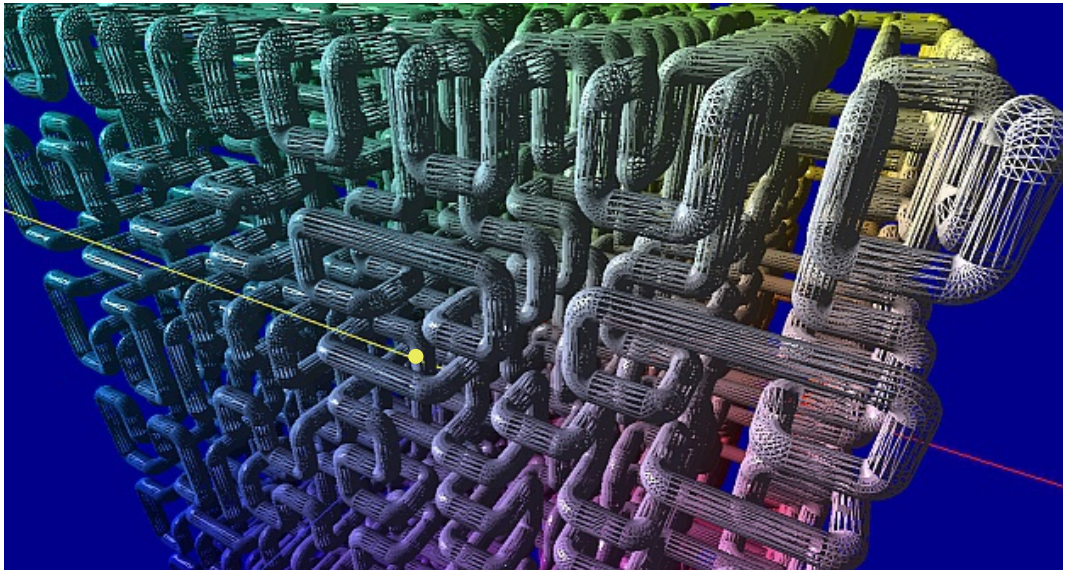
Průsečík s trojúhelníkovou sítí

Scéna reprezentována **sítí trojúhelníků**

- jednoduchý koncept (jednoduché API)
- jakoukoli geometrii lze pomocí trojúhelníků aproximovat

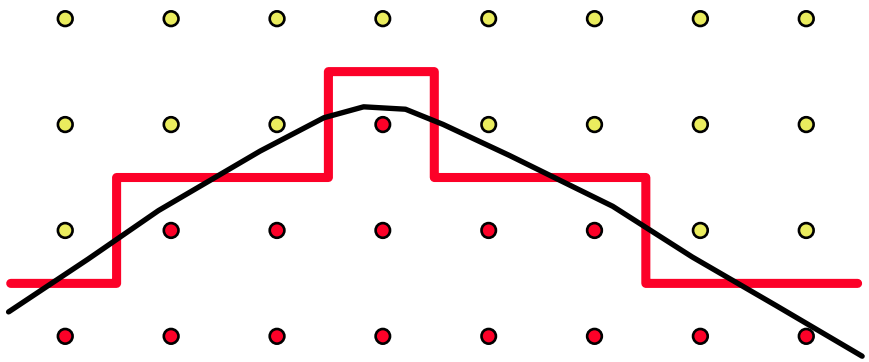
Jednoduchý test **paprsek – trojúhelník**

- velké množství trojúhelníků (10^6 až 10^{10}), ***O(N)*** je moc pomalé
- urychlovací techniky se snaží o lepší složitost, např. ***O(log N)***



N ≈ 10^6

Vyhlazování (anti-aliasing)

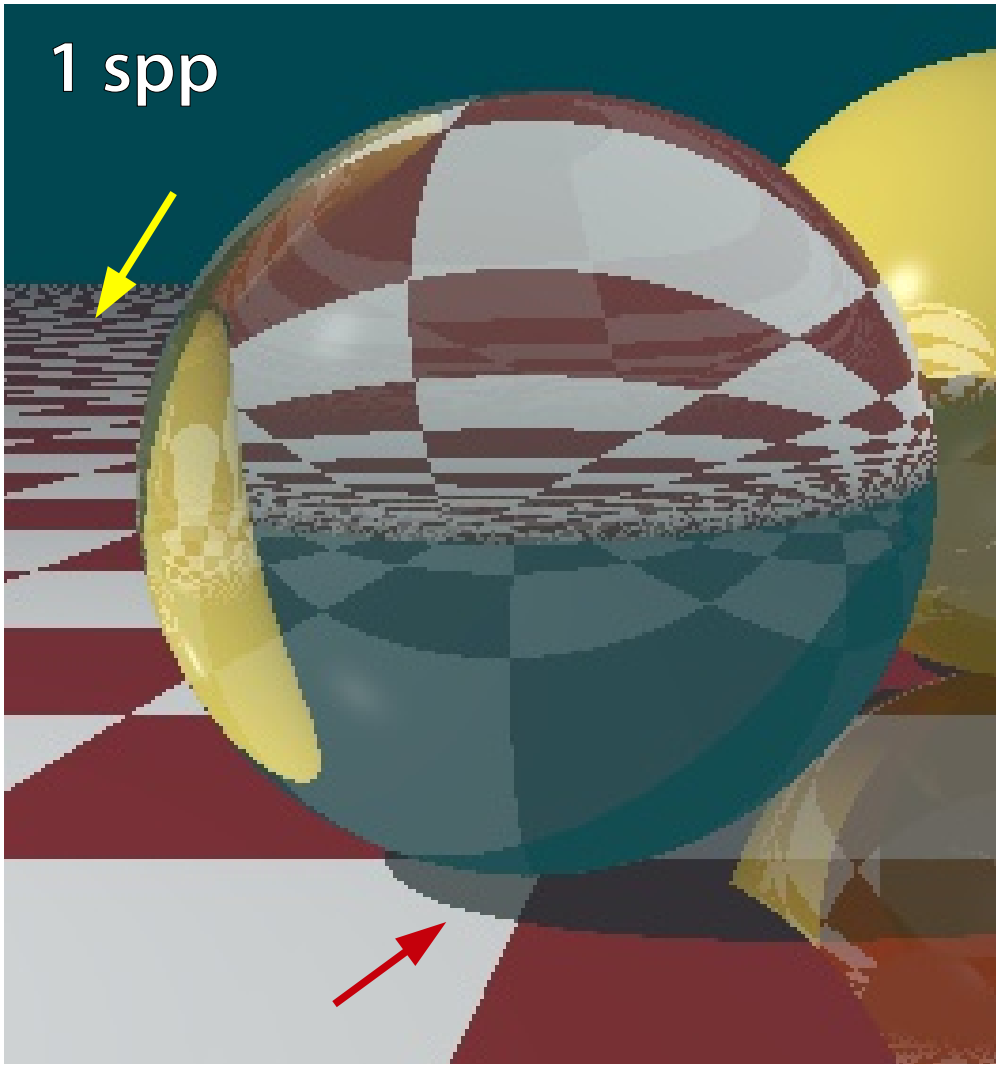


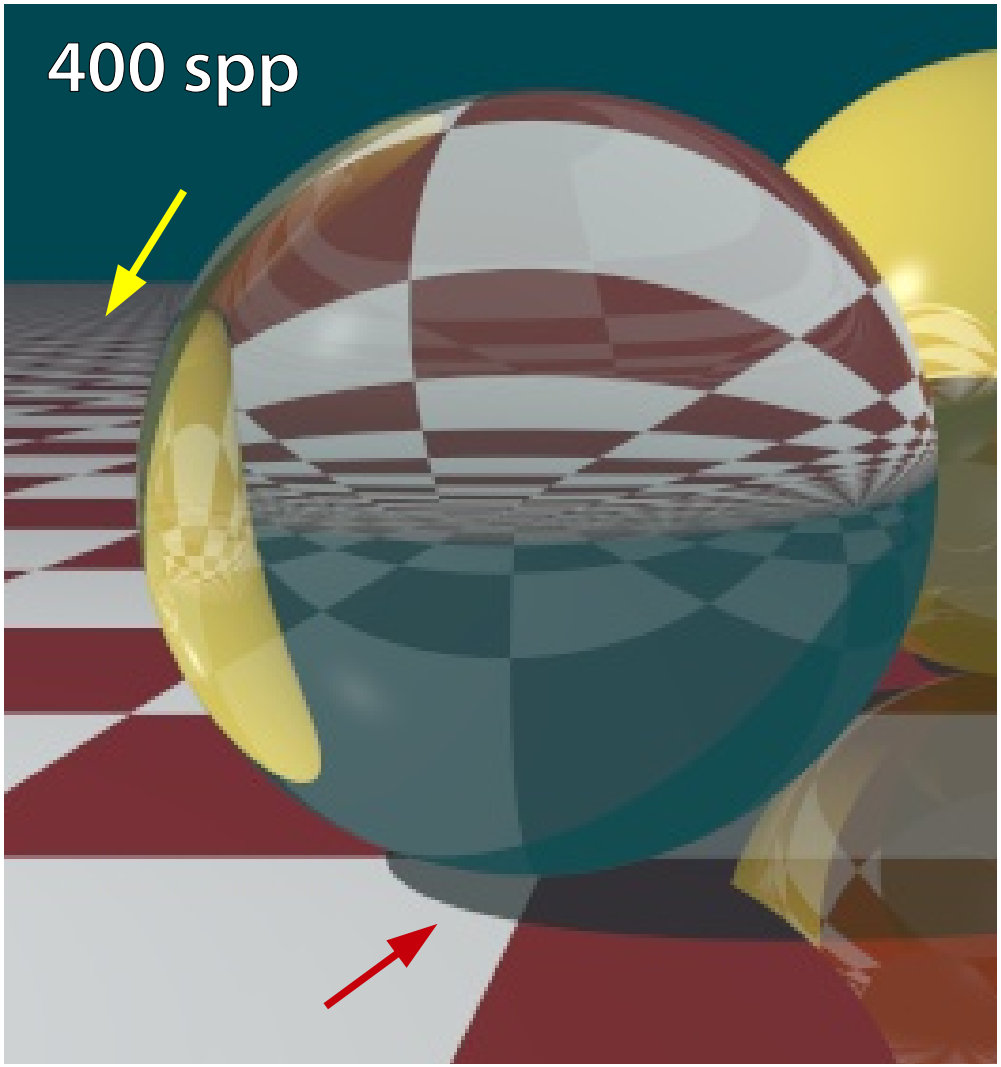
Pouze jeden paprsek na jeden pixel – vzniká tzv. „**alias**"

- zubaté okraje
- interference

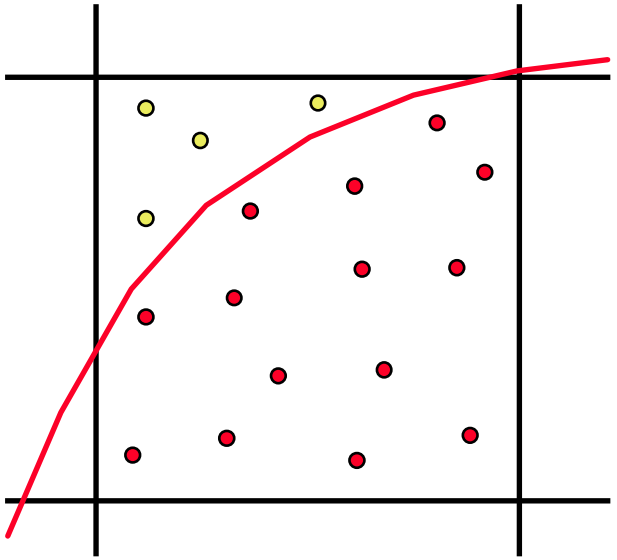
Zvětšením rozlišení se problém nevyřeší

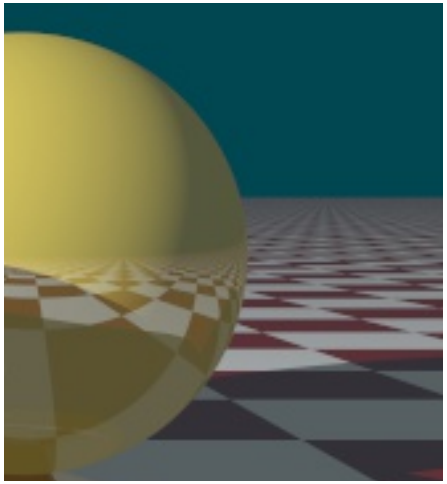
Ukázka vyhlazování (super-sampling)





Převzorkování (super-sampling)





Posílá se **více paprsků** jedním pixelem

Výsledná barva se spočte jako **aritmetický průměr**

Přechody budou **jemnější** (bez zubů)

Paprsky by měly pokrývat plochu pixelu **rovnoměrně**, ale ne úplně pravidelně!

Textury

Změna **barvy** na povrchu předmětů

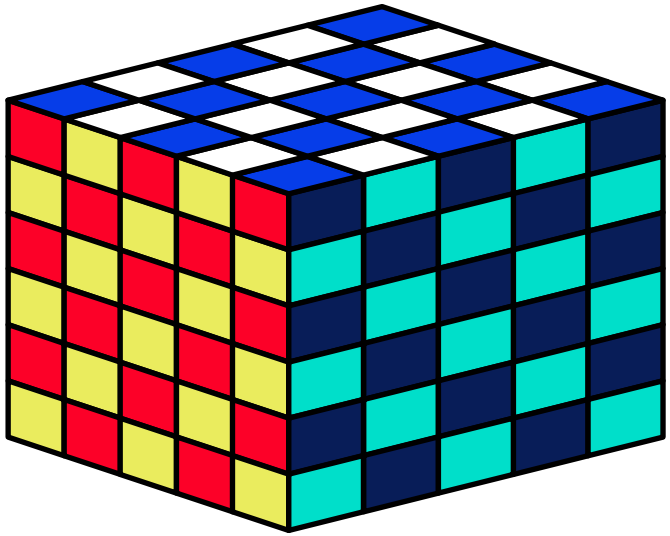
Mohou ovlivňovat též **odrazivost** ($\mathbf{k_D}$ a $\mathbf{k_S}$), **normálový vektor** („normal map“)…

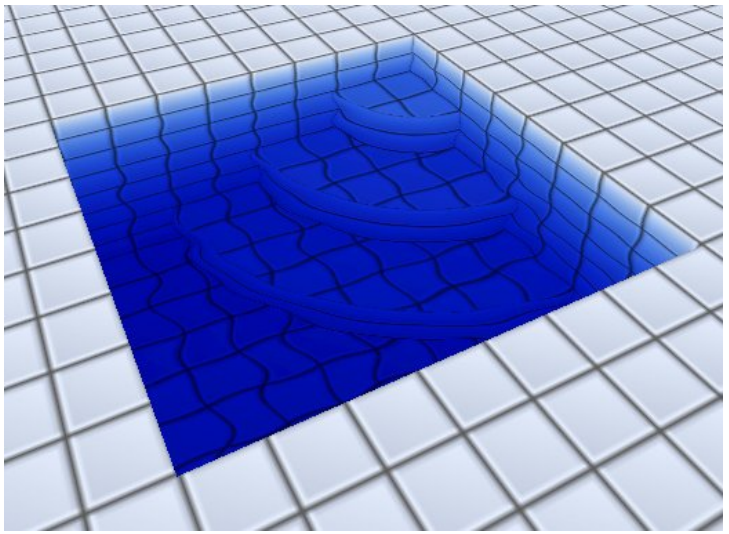
Realistické napodobení **fyzikálních vlastností materiálu** (barevný vzorek, mikro- i makro-struktura povrchu)

- dřevo, kůra pomeranče, omítka, leštěný kov…

Nahrazení složité **geometrie** (vlny na vodě…)

2D textura





Pokrývá **povrch** tělesa (jako tapeta, samolepka)

Mapování textury: $[x, y, z] \rightarrow [u, v]$

Vlastní textura: $[u, v] \rightarrow$ **barva** (normála, materiál…)

3D textury

Reprezentují změny veličin **uvnitř tělesa**

Napodobují **vnitřní strukturu materiálu**

- dřevo, mramor…

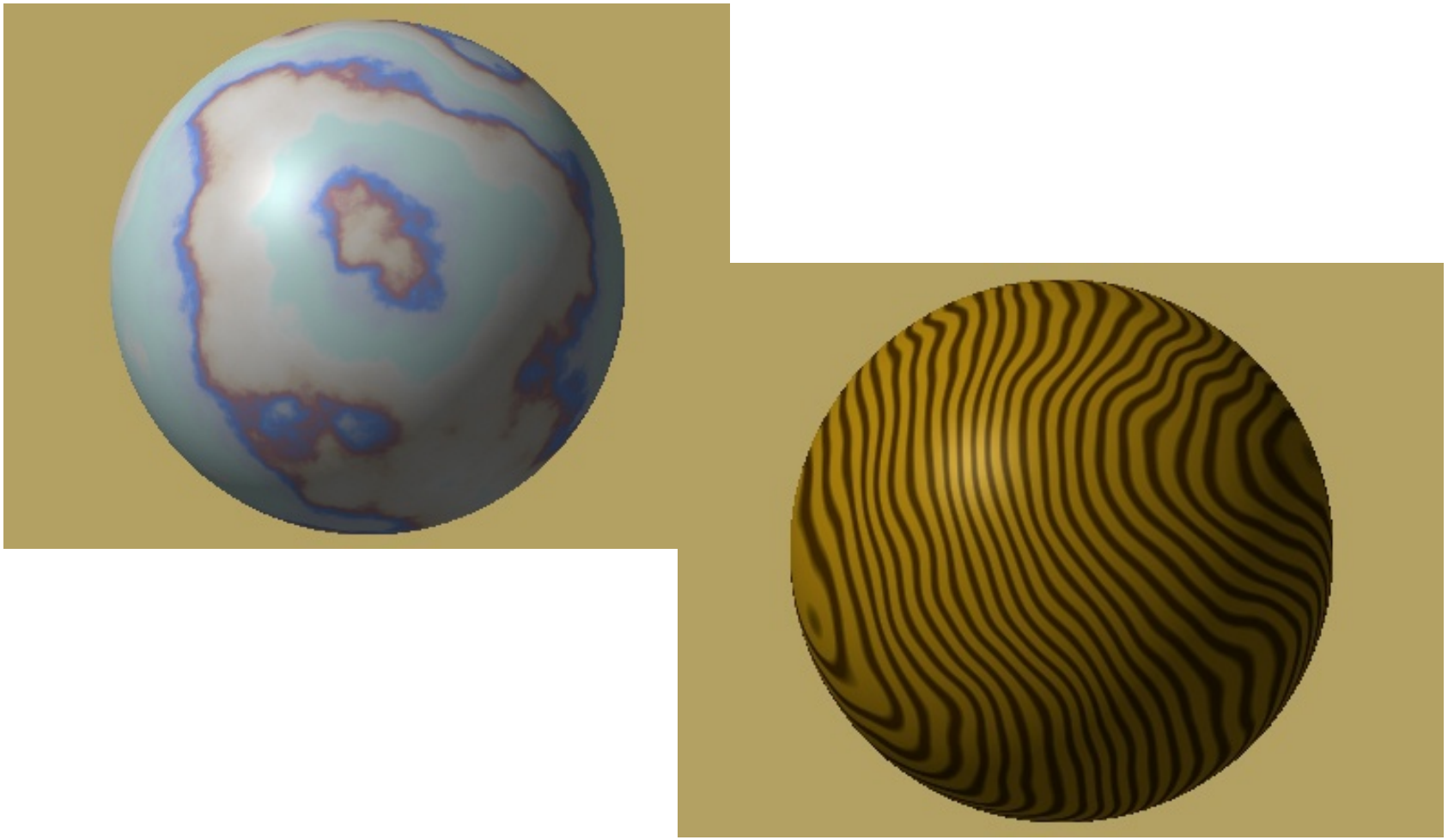
Není třeba **mapování**

3D textura: $[x, y, z] \rightarrow$ **barva** (materiál…)

Často se využívají 3D **šumové funkce**

- napodobení náhodné turbulence/vrásnění

Příklady 3D textur



Literatura

A. Glassner: ***An Introduction to Ray Tracing***, Academic Press, London 1989, 1-31

Jiří Žára a kol.: ***Počítačová grafika, principy a algoritmy***, 374-378