Počítačová grafika I
PGR 003

KSVI MFF UK Praha

e-mail: Josef.Pelikan@mff.cuni.cz

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

Obsah a forma

- **základy 2D i 3D grafiky**
  - v letním semestru na ni navazuje Počítačová grafika II (PGR004) a Pokročilá 2D počítačová grafika (PGR007)
- **2/1 Z, Zk**
  - přednáška jednou týdně
  - každý druhý týden cvičení v laboratoři
  - zkouška se dá skládat společně s PGR004 a/nebo PGR007 v létě (ale zapisují se **zvlášť**)

Stručný plán přednášky:

1. **grafický HW, 2D kreslicí algoritmy** (~3)
   - kreslení úseček, kružnic, vyplňování n-úhelníka, záplavové vyplňování, ořezávání, vyhlazování
2. **barvy a jejich zobrazování** (~4)
   - půltónování a rozptylování, barevné vidění, barevné prostory (RGB, CMYK, HSV), zobrazování barev, speciální palety
3. **kódování rastrových obrázků** (~1)
   - kódování obrazu, grafické formáty (TGA, GIF, ..)

Stručný plán přednášky:

4. **matematika pro 3D grafiku** (~2)
   - lineární transformace, homogenní souřadnice, projekce a jejich implementace
5. **reprezentace 3D scén** (~2)
   - výčtové, objemové a povrchové reprezentace
6. **výpočet viditelnosti** (2-3)
   - plovoucí horizont, Appelův algoritmus, malířův algoritmus, Z-buffer, řádkový rozklad, Warnockův algoritmus, BSP strom

Literatura (CZ):

- **Jiří Žára, Bedřich Beneš, Petr Felkel**: ***Moderní počítačová grafika***, Computer Press, Brno, 1998, ISBN: 80-7226-049-9
- **Jiří Sochor, Jiří Žára**: ***Algoritmy počítačové grafiky***, skriptum ČVUT FEL, Praha, 1992
- **Jiří Žára a kol.**: ***Počítačová grafika***, principy a algoritmy, GRADA, Praha, 1992

Literatura (US):

- **J. Foley, A. van Dam, S. Feiner, J. Hughes**: ***Computer Graphics, Principles and Practice***, 2nd edition, Addison-Wesley, Reading, 1990
- **D. F. Rogers**: ***Procedural elements for Computer Graphics***, McGraw-Hill, New York, 1985
- **W. Newmann, R. F. Sproull**: ***Principles of Interactive Computer Graphics***, 2nd edition, McGraw-Hill, New York, 1979

Předpoklady:

- **Základní kurs programování** (jazyk Pascal)
  - Programování I, Programování II
- **Základy programování v jazyku Java**
  - výuková knihovna JaGrLib
- **Základní kurs matematické analýzy a lineární algebry**

Další grafické přednášky (zima):

- **Digitální zpracování obrazu:** 3/0, PGR002 (Jan Flusser, ÚTIA AV ČR)
- **Počítačové vidění a inteligentní robotika:** 2/1, PGR001 (Václav Hlaváč, FEL ČVUT)
- **Virtuální realita:** 2/1, PGR012 (Jiří Žára, FEL ČVUT)
- **Křivky a plochy v počítačové grafice:** 2/0, PGR009 (Zdeněk Töpfer, Auto Škoda M.B., nekoná se každý rok)
- **Úvod do mobilní robotiky**: 2/0, AIL028 (Martin Dlouhý, Zbyněk Winkler, Haptica Ireland, MFF)

Jiné zdroje informací:

- aktuální informace na **WWW:**
  - **http://cgg.ms.mff.cuni.cz/**
  - **http://cgg.ms.mff.cuni.cz/~pepca/**
- **LAN** na MFF:
  - **barbora\usr:\vyuka\pelikan**
- knihovna **JaGrLib**:
  - **http://cgg.ms.mff.cuni.cz/JaGrLib/**

Grafický hardware

KSVI MFF UK Praha

e-mail: Josef.Pelikan@mff.cuni.cz

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

Grafický výstup

- podle **trvalosti zobrazení:**
  - zobrazovací zařízení (displej, graf. projektor)
  - tiskové zařízení (tiskárna, plotter, osvitová jedn.)
- podle **barevných schopností:**
  - černo-bílé zobrazení (*2 barvy*)
  - monochromatické zobrazení (odstíny šedi: *256*)
  - barevná paleta (pevná nebo nahrávaná: *16-1024*)
  - plná barevnost (“true-color” - maximální barevné využití zobrazovací technologie: *16.7mil. i více*)

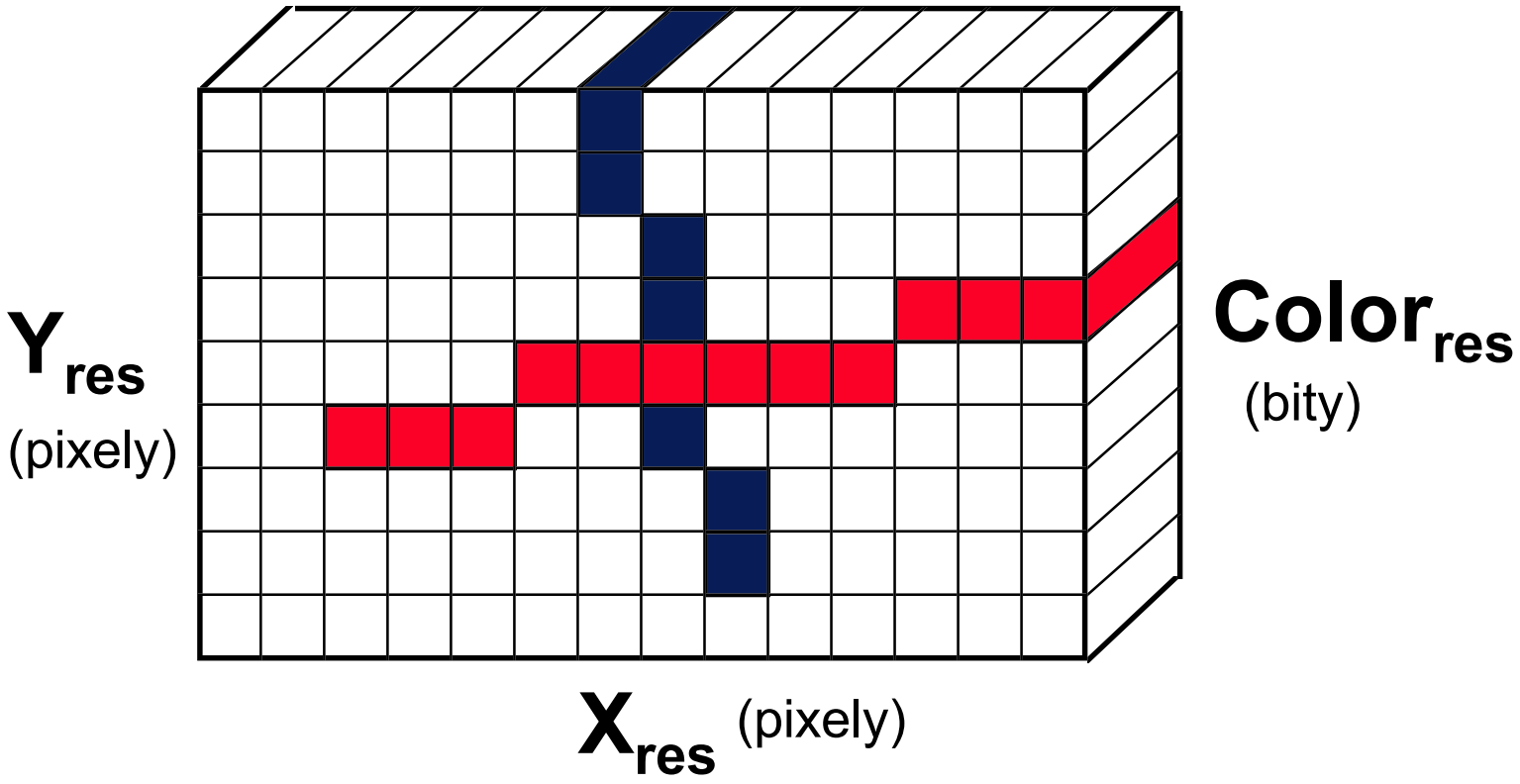
Rastrový / vektorový přístup

- **rastrový výstup:**
  - jsou přímo ovládány (adresovány) jednotlivé pixely
  - data jsou závislá na rozlišení (a nelze je jednoduše škálovat)
- **vektorový výstup:**
  - zobrazují se přímo složitější objekty (čáry, křivky, písmo)
  - data nejsou závislá na rozlišení (lze je škálovat až v zobrazovacím zařízení)

Grafický výstup

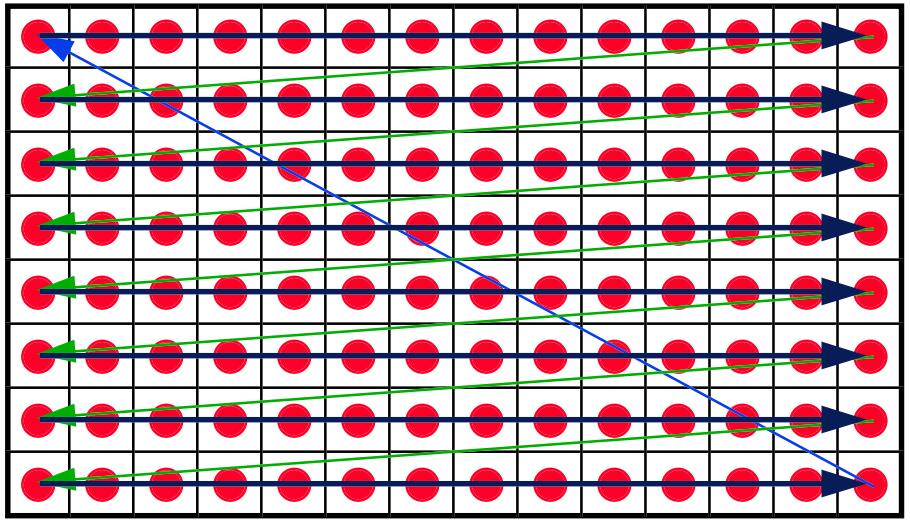
- podle **technologie výstupu:**
  - vektorový výstup (staré displeje, plotter, některé osvitové jednotky)
  - rastrový výstup (displeje, tiskárny, plottery)
- podle **komunikace:**
  - vektorové zařízení (urychlované video-adaptéry, plottery, PostScript®)
  - rastrové zařízení (běžné video-adaptéry, tiskárny v grafickém režimu)

Rastrový displej



Např: **640×480×8** bitů, **1600×1200×24** bitů

Řádkový rozklad



1024 × 768 (72 Hz)

přímý chod
(modulace - zobrazování)

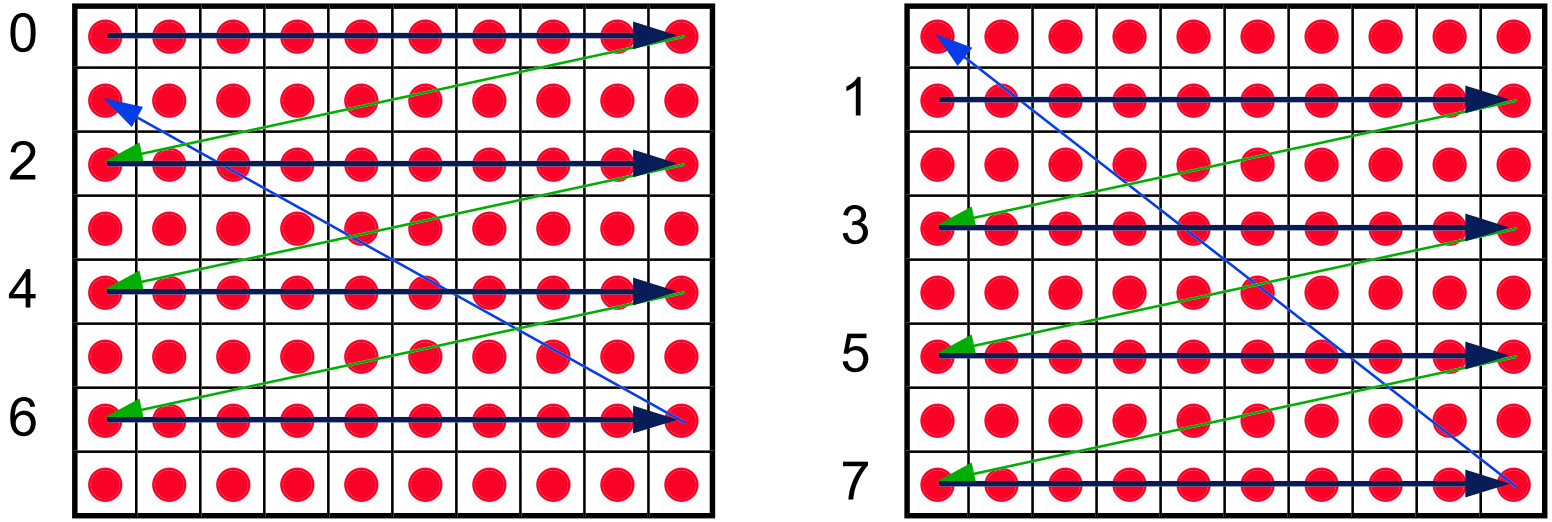
návratový paprsek
(horizontální zatemnění)

návratový paprsek
(vertikální zatemnění)

neprokládaný režim
("non-interlaced" - NI)

72Hz - 69kHz - 85MHz (12ns/pix)

Prokládání



I. sudé pole

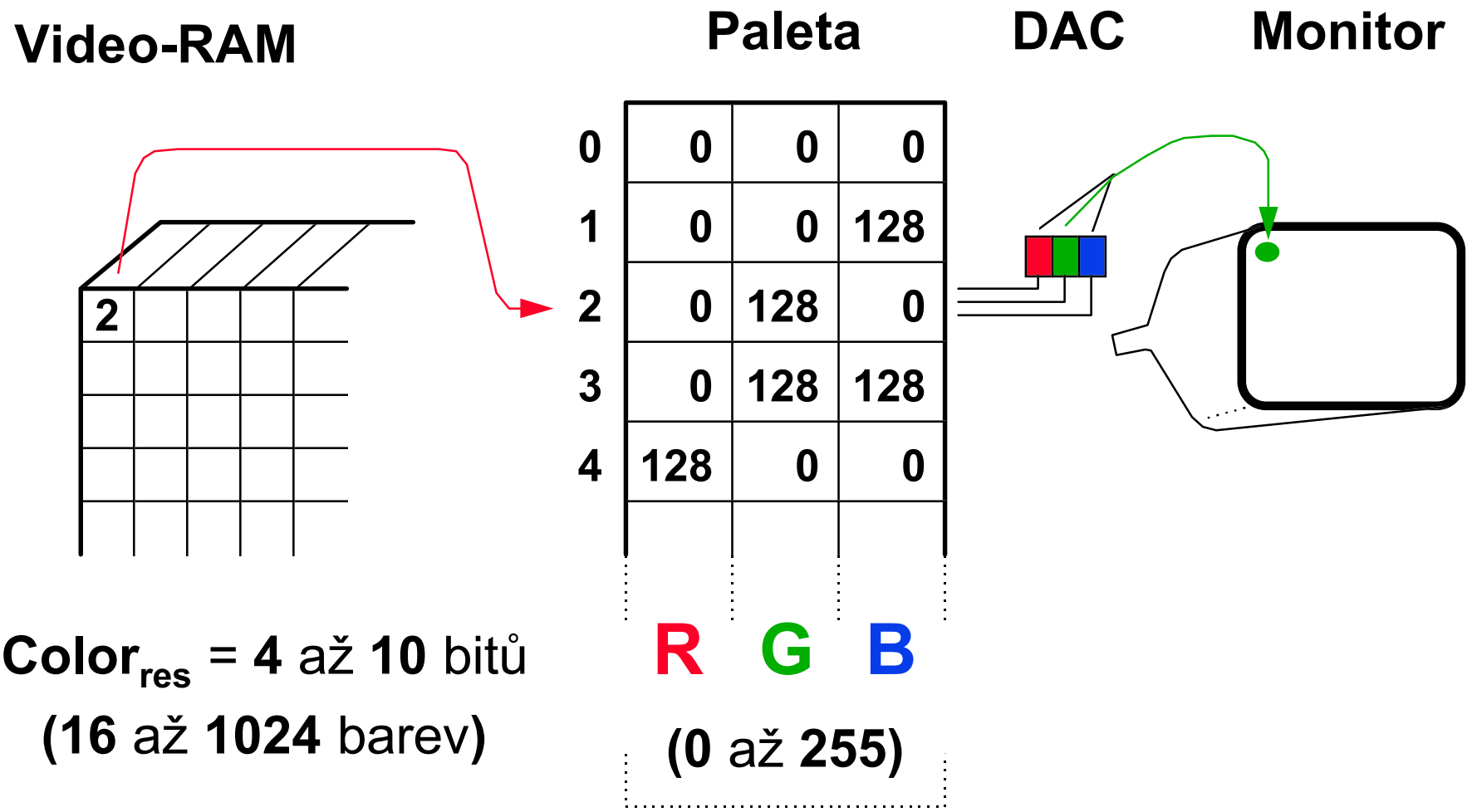
II. liché pole

prokládaný režim

(“interlaced” - I)

72Hz - 35kHz - 42MHz (24ns/pix)

Barevná paleta



Color$_{res}$ = **4** až **10** bitů

(**16** až **1024** barev)

Vektorový displej

- přijímá **vektorové příkazy** a ukládá je ve své paměti (“display list”) - úspora paměti
  - **MoveTo(x,y)**, **LineTo(x,y)**, **Circle(x,y,r)**, ..
- příkazy v seznamu se **cyklicky překreslují**
  - při velkém množství kreslených příkazů obrázek bliká

  ➔ stínítko s dlouhým dosvitem
- kreslené čáry **nejsou zubaté** (jako u rastrových zařízení)

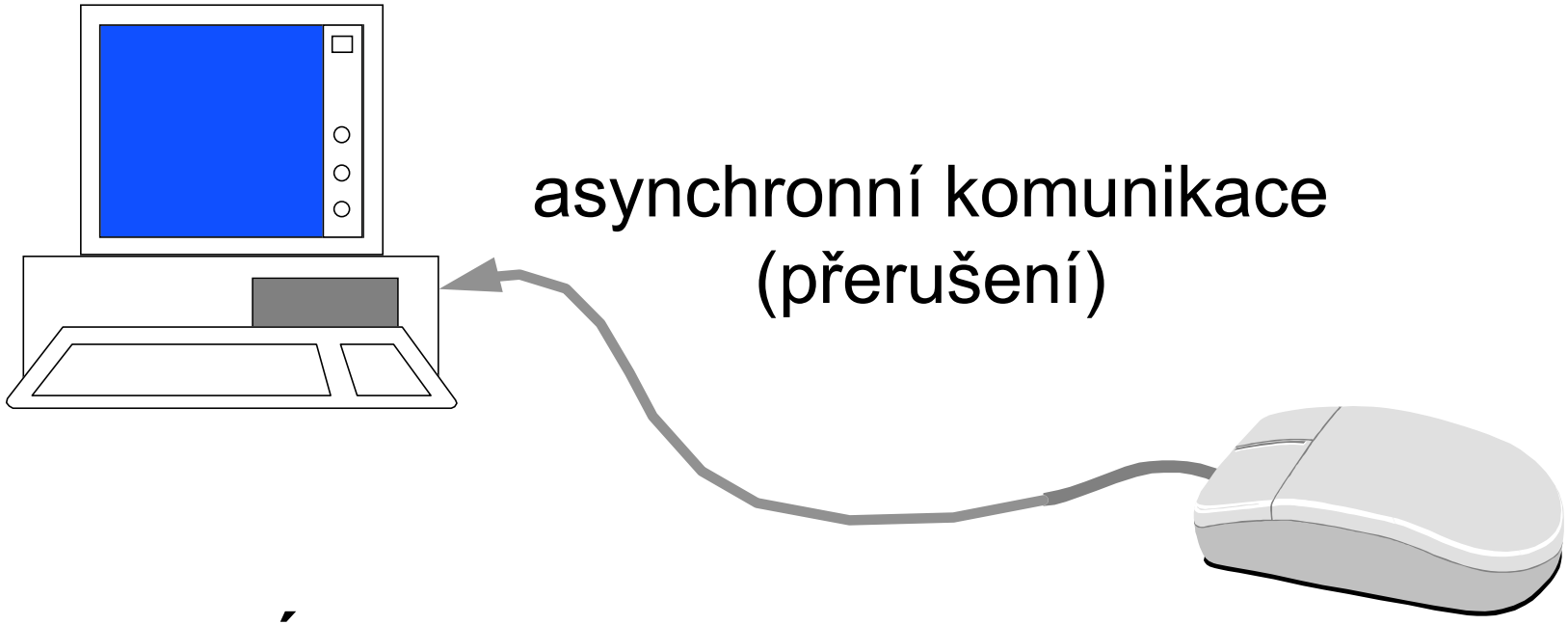
Grafický vstup

- **rastrový vstup:**
  - **scannery** (*300dpi* až *2000dpi*, *16 odstínů šedi* až *3×12 bitů* RGB)
- **interaktivní vstup:**
  - **ukazovátko** (“pick”: výběr z menu, výběr objektu)
  - **vstup polohy** (“locator”: zadání souřadnice **[*x*,*y*]**)

➡ **myš:** levná, méně přesná, relativní vstup

➡ **tablet:** velmi přesný, absolutní vstup

Komunikace myš - PC



zprávy:

Move (dx, dy) relativní pohyb myši

Button (n, down) stisk/uvolnění n-tého tlačítka

Obslužný program pro myš:

- přijímá **zprávy** od myši, pamatuje si stav
  - poskytuje pohodlnější informace aplikačním programům
- **kreslí ukazovátko** na obrazovce
- **on-line** i **off-line** zjišťování stavu myši
- pamatuje si **poslední stisk** každého tlačítka

Konec

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
- **LAN** na Malé Straně:
  - **barbora\usr:\vyuka\pelikan\1**

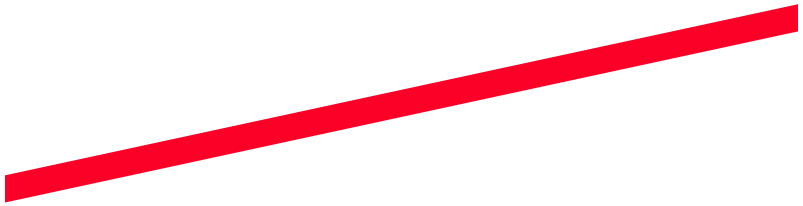
Kreslení čar

KSVI MFF UK Praha

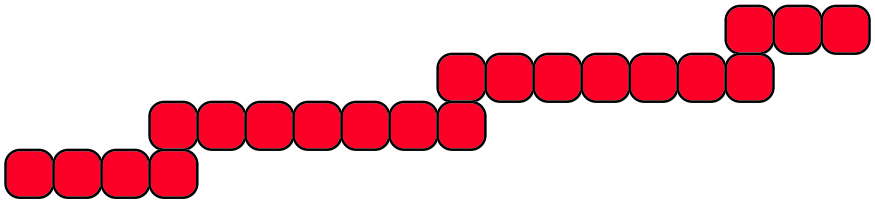
e-mail: Josef.Pelikan@mff.cuni.cz

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

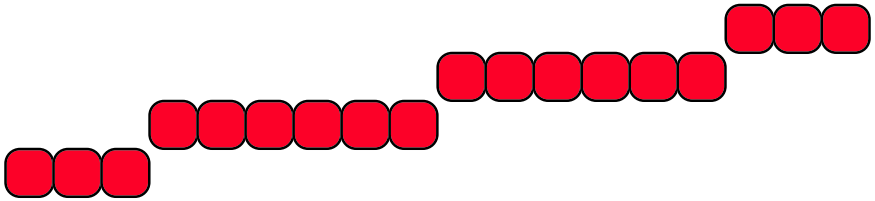
Kreslení úseček



vektorové zařízení

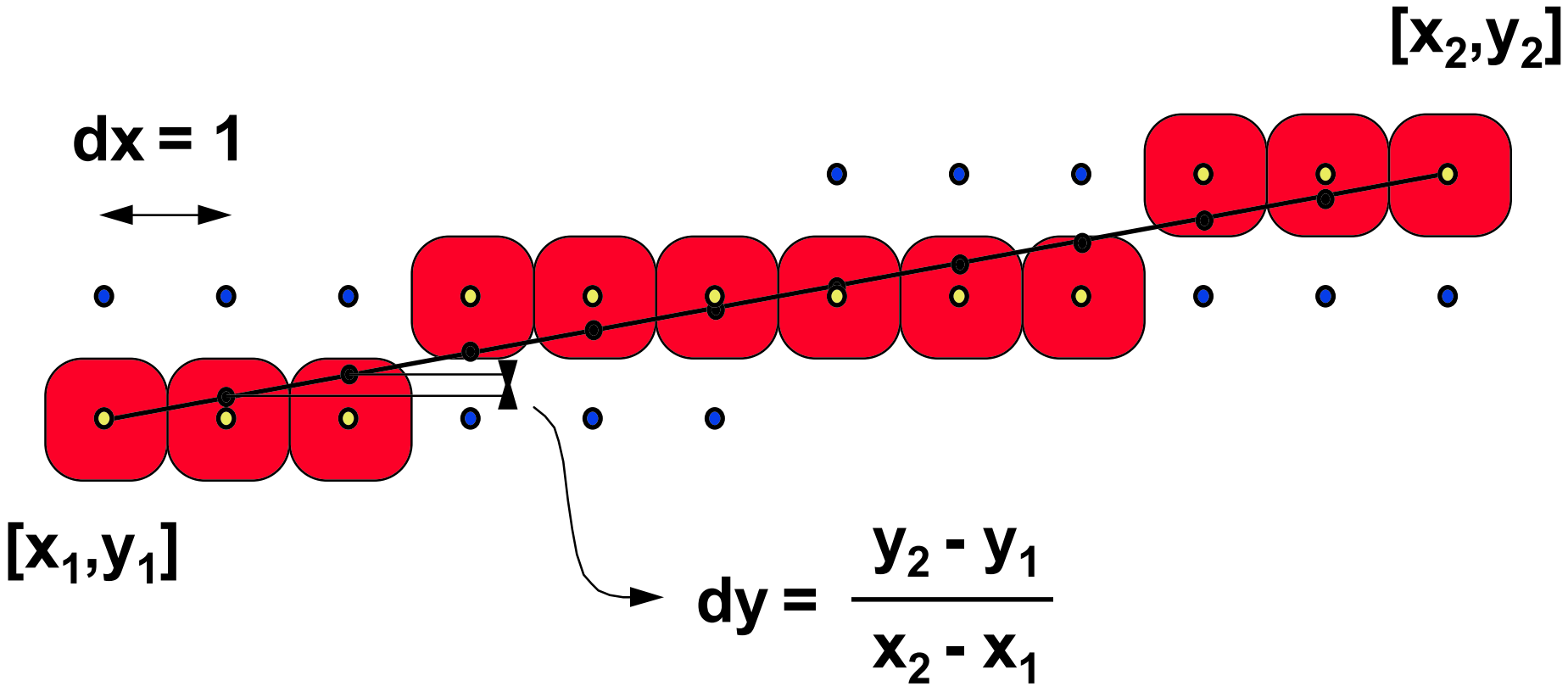


rastrové zařízení



OK

DDA algoritmus



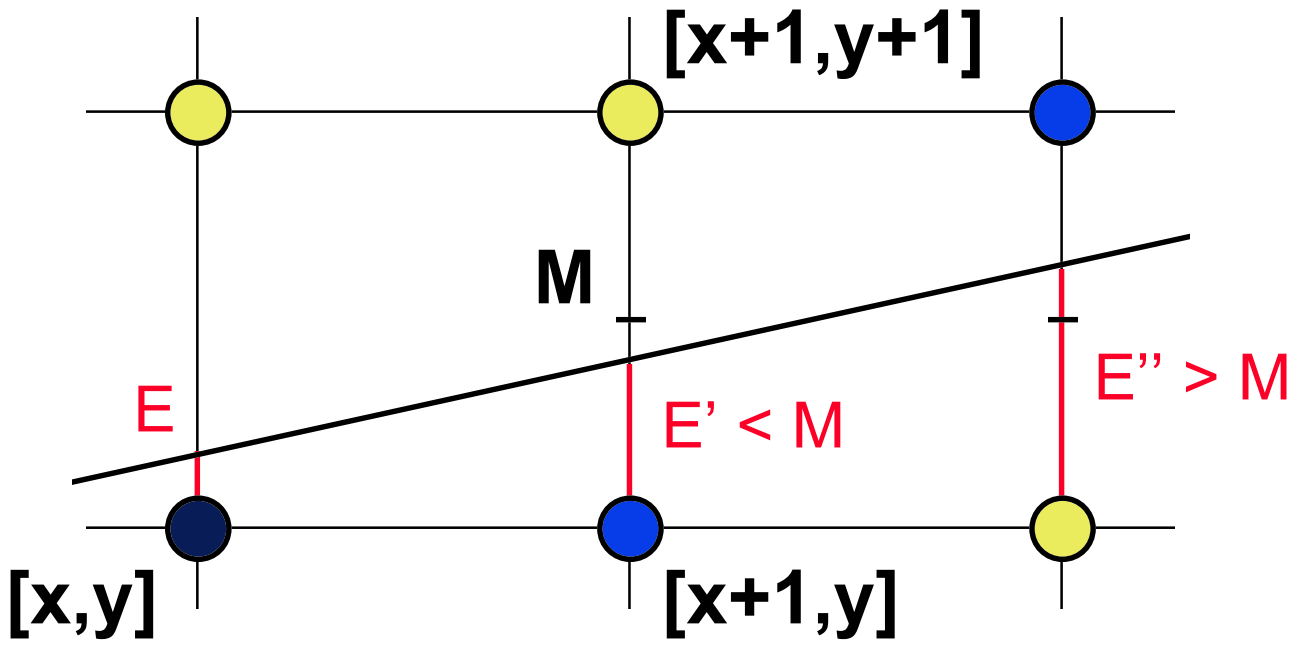
DDA algoritmus

```
procedure LineDDA ( x1, y1, x2, y2, color : integer );
                { předpoklady: x1 < x2, |y2-y1| < |x2-x1| }
var y, dy : real;
begin
  y := y1;
  dy := (y2-y1)/(x2-x1);
  PutPixel(x1,y1,color);
  while x1 < x2 do
    begin
      x1 := x1 + 1;
      y := y + dy;
      PutPixel(x1,round(y),color);
    end;
end;
```

DDA algoritmus

- ✚ **výhody:**
  - snadná implementace (HW)
- ➔ **nevýhody:**
  - nutno počítat s velkou **přesností** (real, 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

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

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

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

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

$$D_0 = 2dy - dx$$

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

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

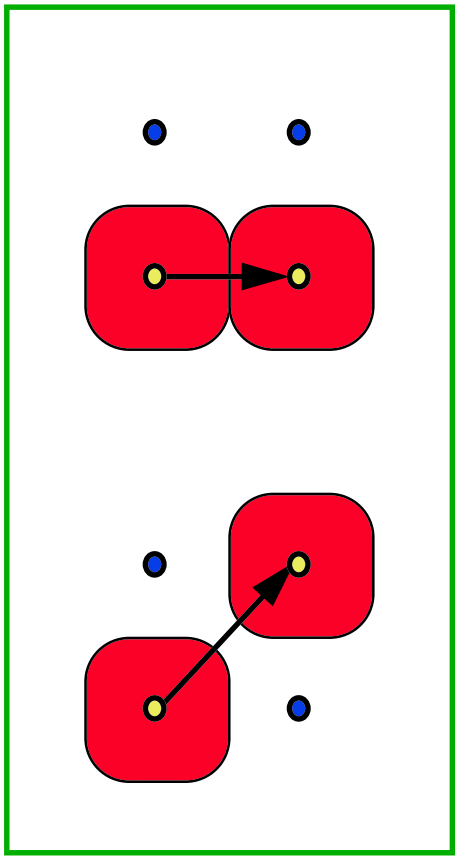
Bresenhamův algoritmus

```
procedure LineBres ( x1, y1, x2, y2, color : integer );
                { předpoklady: x1 < x2, |y2-y1| < |x2-x1| }
var dx, dy, D, inc0, inc1 : integer;
begin
  dx := x2 - x1;
  dy := y2 - y1;
  D  := 2*dy - dx;
  inc0 := 2*dy;
  inc1 := 2*(dy - dx);
  PutPixel(x1,y1,color);
  ...
```

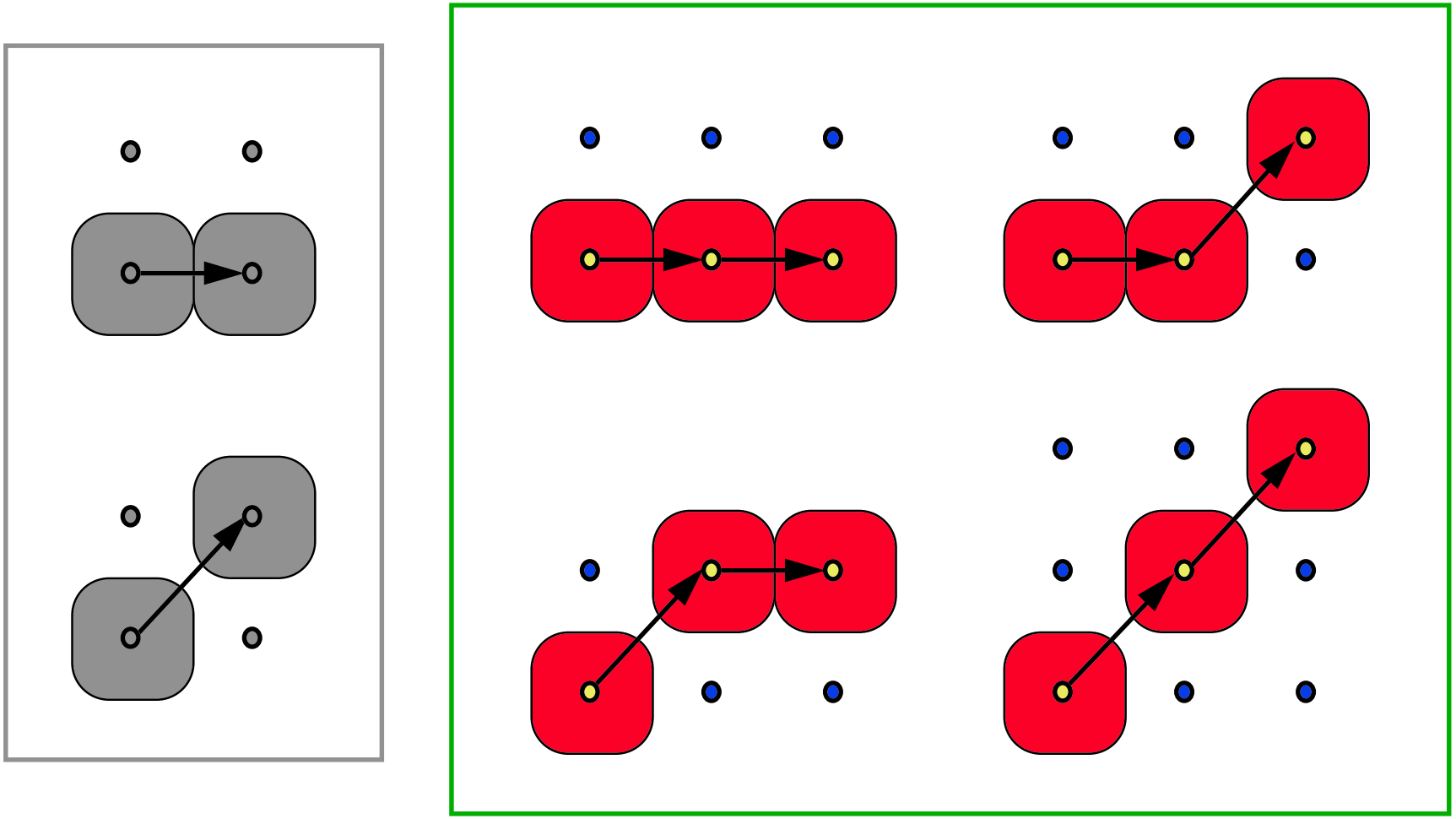
Bresenhamův algoritmus

```
  ...
  while x1 < x2 do
    begin
      if D <= 0 then D := D + inc0
                else
        begin
          D := D + inc1;
          y1 := y1 + 1;
        end;
      x1 := x1 + 1;
      PutPixel(x1,y1,color);
    end;
end;
```

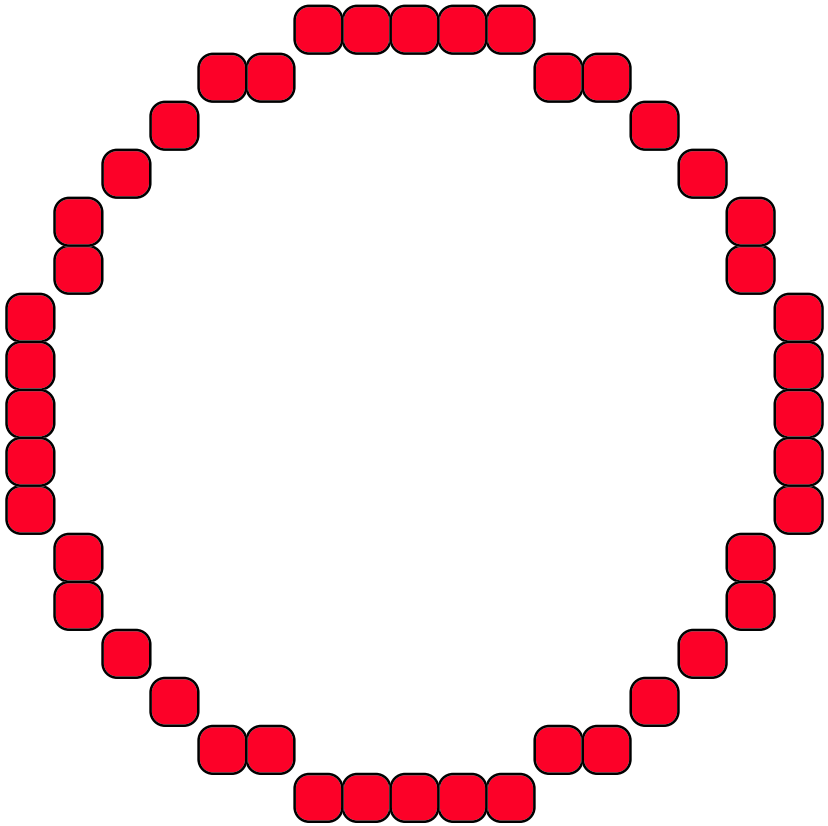
Jednokrokový algoritmus



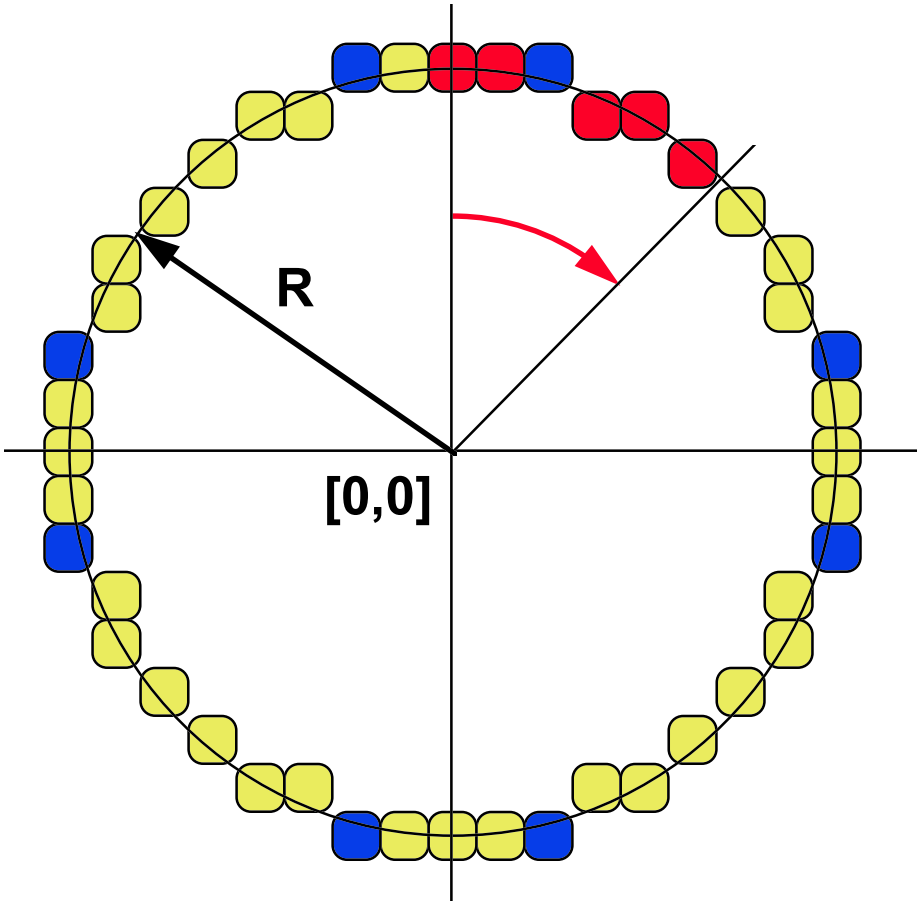
Vícekrokové algoritmy



Kreslení kružnice

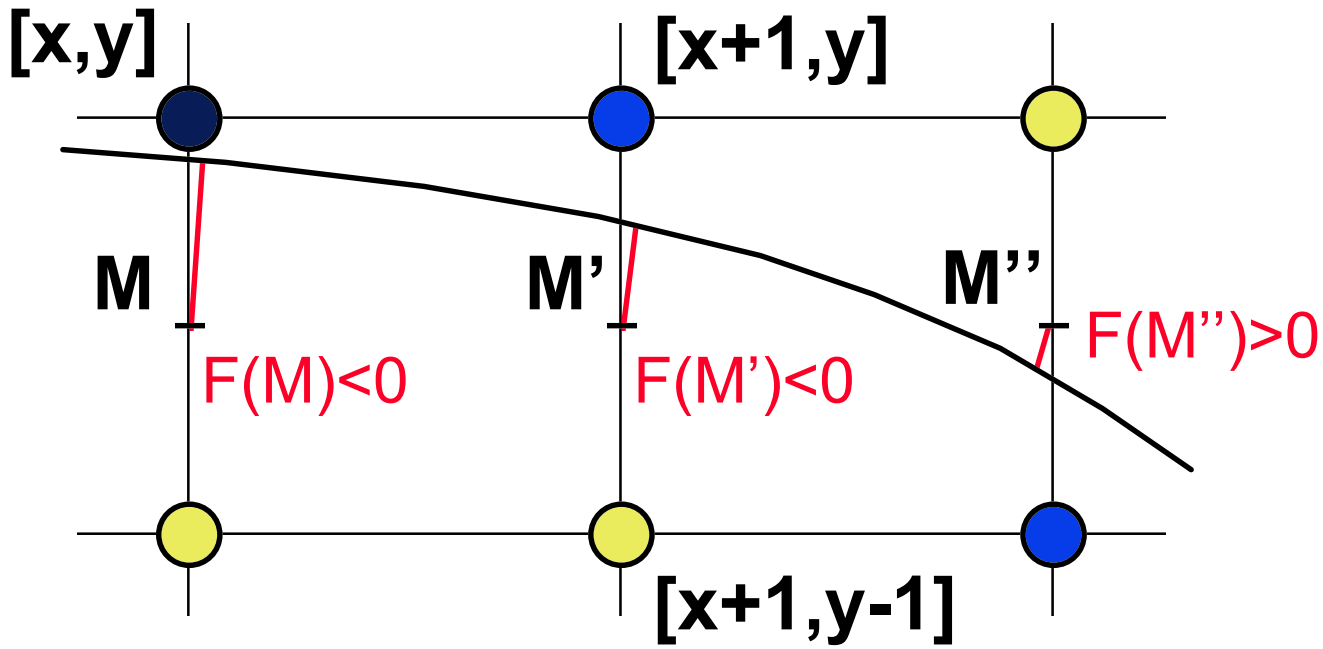


Kreslení kružnice



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

```
procedure CirclePoints ( x, y, color : integer );
                { předpoklad: střed kružnice je v počátku }
begin
  PutPixel( x, y,color);
  PutPixel( y, x,color);
  PutPixel( x,-y,color);
  PutPixel( y,-x,color);
  PutPixel(-x, y,color);
  PutPixel(-y, x,color);
  PutPixel(-x,-y,color);
  PutPixel(-y,-x,color);
end;

...
```

Kreslení kružnice

```
procedure CircleBres ( R, color : integer );
var x, y, D : integer;
begin
  x := 0; y := R; D := 1 - R;
  CirclePoints(0,R,color);
  while y > x do
    begin
      if D < 0 then D := D + 2*x + 3
               else
        begin
          D := D + 2*(x - y) + 5;
          y := y - 1;
        end;
      x := x + 1;
      CirclePoints(x,y,color);
    end;
end;
```

Konec

Další informace:

- **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
- **LAN** na Malé Straně:
  - **barbora\usr:\vyuka\pelikan\2**

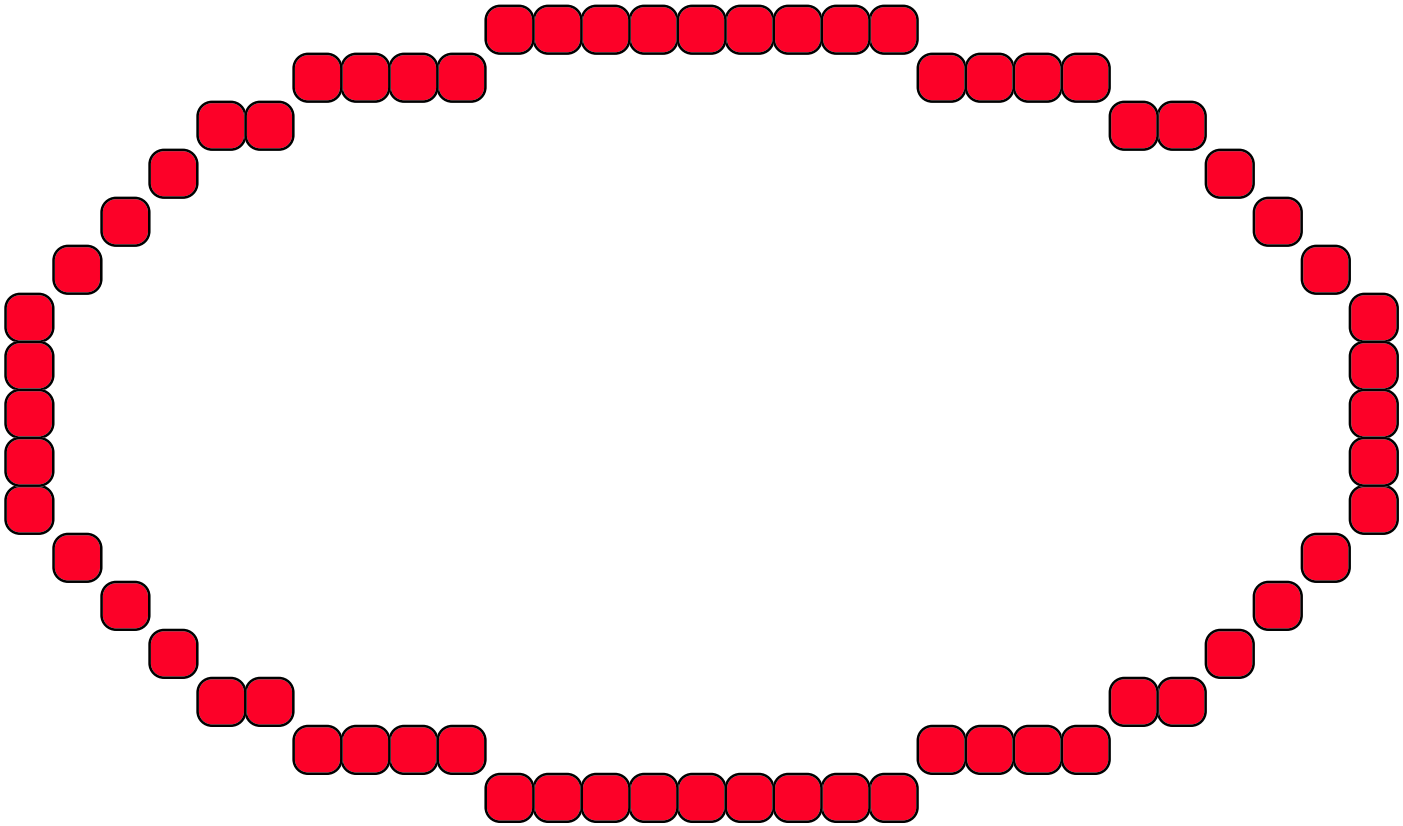
Kreslení křivek

KSVI MFF UK Praha

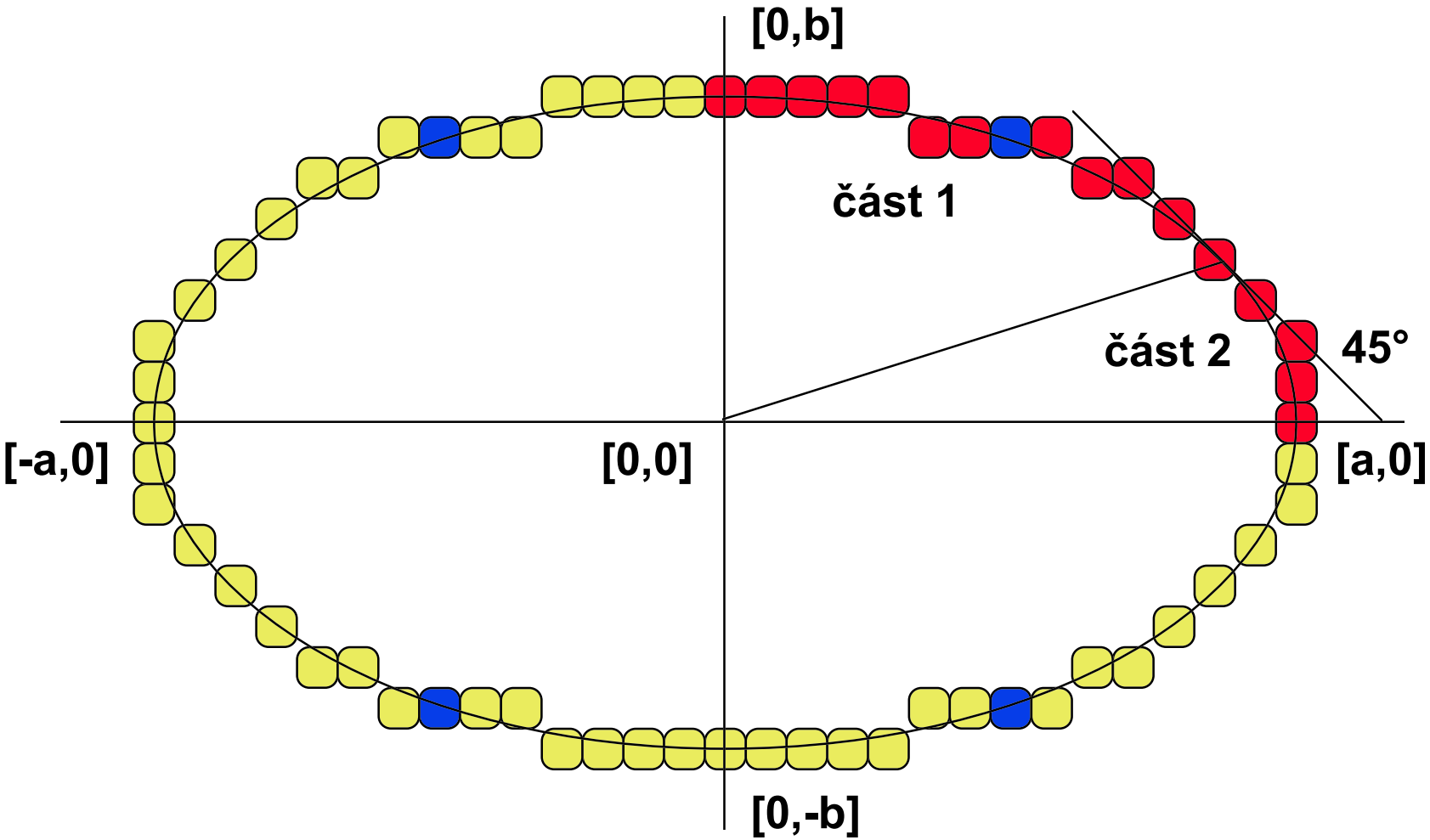
e-mail: Josef.Pelikan@mff.cuni.cz

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

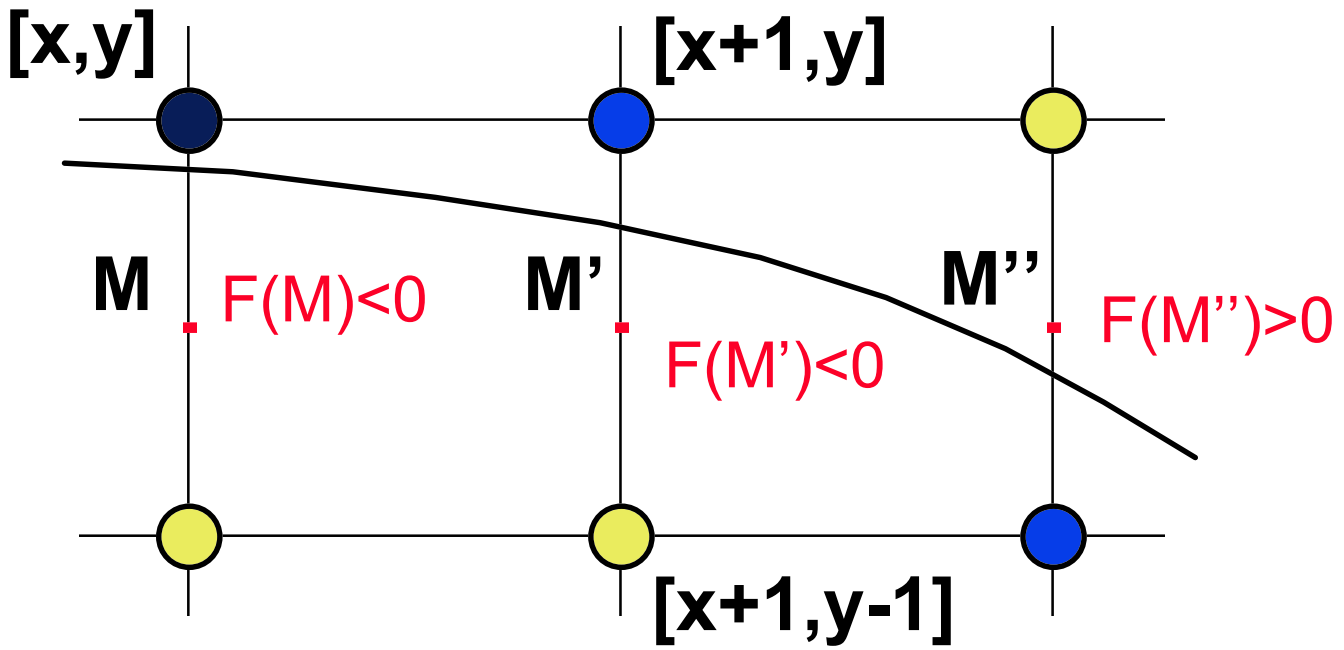
Elipsa



Elipsa se středem v počátku



“midpoint” algoritmus



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

Inkrementální odvození

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

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

$F(M'') = F(M') + b^2(2x+3)$

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

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

$F(M'') = F(M') + b^2(2x+3) + a^2(-2y+2)$

Inkrementální odvození

I) inicializace:

$$F\left[1, b - \tfrac{1}{2}\right] = b^2 - a^2 b + \tfrac{1}{4} a^2$$

II) přechod z části 1 do části 2:

$$\frac{\partial F}{\partial x} = \frac{\partial F}{\partial y}, \quad \text{tj.} \quad 2 b^2 x = 2 a^2 y$$

pomocné diferenční proměnné pro [x+1,y-1/2]:

$$dx = \frac{\partial F}{\partial x} = b^2 (2x + 2), \quad dy = \frac{\partial F}{\partial y} = a^2 (2y - 1)$$

Inkrementální odvození

III) inicializace pro část 2:

$$F\left[x+\tfrac{1}{2}, y-1\right] = b^2\left(x+\tfrac{1}{2}\right)^2 - a^2(y-1)^2 - a^2b^2$$

testy v části 2:

$$F(M') \geq 0 \Rightarrow F(M'') = F(M') + a^2(-2y+3)$$

$$F(M') < 0 \Rightarrow F(M'') = F(M') + b^2(2x+2) + a^2(-2y+3)$$

Kreslení elipsy

```
procedure EllipsePoints ( x, y, color : integer );
                { předpoklad: střed elipsy je v počátku }
begin
  PutPixel( x, y,color);
  PutPixel( x,-y,color);
  PutPixel(-x, y,color);
  PutPixel(-x,-y,color);
end;

procedure EllipseMidpoint ( a, b, color : integer );
var x, y, D, dx, dy, aa, aa2, bb, bb2 : longint;
begin
  x := 0; y := b;          { souřadnice prvního bodu }
  aa := sqr(a); aa2 := 2*aa; bb := sqr(b); bb2 := 2*bb;
  D := bb - aa*b + aa div 4;
  dx := bb2; dy := aa*(2*b - 1);     { dF/dx, dF/dy }
  ...
```

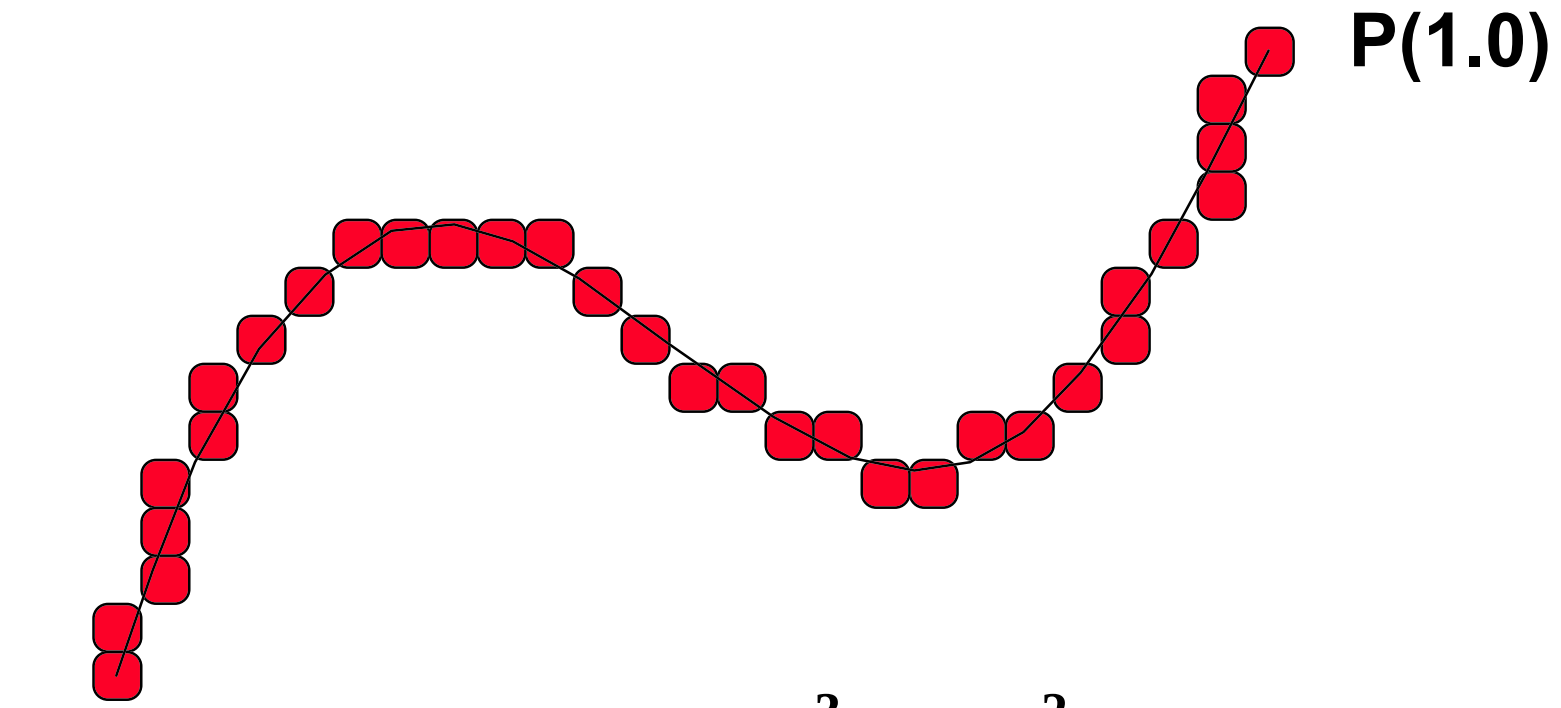
Kreslení elipsy

```
...
EllipsePoints(0,b,color);
while dx < dy do        { část 1 }
  begin
    if D >= 0 then
      begin             { klesám o jednu řádku }
        D := D - dy + aa;
        dy := dy - aa2;
        y := y - 1;
      end;
    D := D + dx + bb;
    dx := dx + bb2;
    x := x + 1;
    EllipsePoints(x,y,color);
  end;
...
```

Kreslení elipsy

```
  ...
  D := bb*(sqr(x)+x) + bb div 4 + aa*(sqr(y-1)-bb);
  while y < 0 do          { část 2 }
    begin
      if D < 0 then
        begin             { posun doprava }
          D := D + dx;
          dx := dx + bb2;
          x := x + 1;
        end;
      D := D - dy + aa2;
      dy := dy - aa2;
      y := y - 1;
      EllipsePoints(x,y,color);
    end;
end;
```

Kubická křivka v rovině



$$P_x(t) = A_x t^3 + B_x t^2 + C_x t + D_x$$

$$P_y(t) = A_y t^3 + B_y t^2 + C_y t + D_y$$

Diferenční algoritmus

- výpočet hodnot **polynomu P** v bodech **P(0)**, **P(h)**, **P(2h)**, .. **P(ih)**, .. pouze pomocí **sčítání**

- pro **kubický polynom**:
  1. inicializace proměnných **a**, **b**, **c**, **d**
  2. krok: **Output(d)**
     d := d + c
     c := c + b
     b := b + a

Maticová notace

- **inicializace**:

$$\begin{bmatrix} d \\ c \\ b \\ a \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & h & h^2 & h^3 \\ 0 & 0 & 2h^2 & 6h^3 \\ 0 & 0 & 0 & 6h^3 \end{bmatrix} \cdot \begin{bmatrix} D \\ C \\ B \\ A \end{bmatrix}$$

- **krok**:

$$\begin{bmatrix} d \\ c \\ b \\ a \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} d \\ c \\ b \\ a \end{bmatrix}$$

Úpravy kroku **h**:

- **U** - zvětšení kroku na dvojnásobek (“Up”):

$$\mathbf{U} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 8 \end{bmatrix}$$

- **D** - zmenšení kroku na polovinu (“Down”):

$$\mathbf{D} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1/2 & -1/8 & 1/16 \\ 0 & 0 & 1/4 & -1/8 \\ 0 & 0 & 0 & 1/8 \end{bmatrix}$$

Adaptivní diferenční algoritmus

```
procedure CubicCurve;
var B1, B2, B3 : POINT2D; { x, y : real }
    X, Y : DiffRecord;    { a, b, c, d, h : real }
    t : real;             { parametr křivky }
begin
  "inicializace X, Y"   { inicializace diferencí }
  t := 0.0; B1.x := X.d; B1.y := Y.d;
  B2 := B1; B3 := B1;   { B1=B2=B3=začátek křivky }
  repeat                { jeden krok křivky }
    if |B3-B1| > 1 then Output(B1); B1 := B2
    B2 := B3;           { zapomenu B2 }
    while (X.c > 1) or (Y.c > 1) do Down(X); Down(Y);
    while (X.c < 0.5) and (Y.c < 0.5) do Up(X); Up(Y);
    Step(X); Step(Y); B3.x := X.d; B3.y := Y.d;
    t := t + X.h;
  until t > 1.0;
  Output(B1); Output(B2);  { dokreslím zbytek křivky }
end;
```

Celočíselná implementace

- pro uložení proměnných **a**, **b**, **c**, **d** použiji **32-bitové** registry (v pevné desetinné čárce)

a	celá část	desetinná část
b	12 bitů	20 bitů
c		
d		

- díky **akumulaci chyb** nelze nakreslit křivku delší než cca **100 pixelů**

Zvětšená přesnost členu **d**:

- pro uložení proměnných **a**, **b**, **c** použiji **více desetinných bitů** (28)

- přibyla jedna **operace "shift"**, ale již lze nakreslit až **512 kroků křivky**

Dynamické řízení přesnosti:

- pro uložení proměnných **a**, **b**, **c** použiji **více desetinných bitů** (**n** mohu adaptivně měnit)

- přibyly dvě **operace "shift"**, ale např. pro **n=14** lze již nakreslit **8K kroků křivky**

Literatura

- **J. Foley, A. van Dam, S. Feiner, J. Hughes**: ***Computer Graphics, Principles and Practice***, 88-91
- **D. Da Silva: *Raster Algorithms for 2D Primitives***, Master's Thesis, Brown University, 1989
- **Jiří Žára a kol.**: ***Počítačová grafika***, principy a algoritmy, 112-115

Konec

- **S.-L. Chang, M. Chantz, R. Rocchetti:** ***Rendering Cubic Curves and Surfaces with Integer Adaptive Forward Differencing***, Computer Graphics, vol. 23, #3, 157-166

- **LAN** na Malé Straně:
  - **barbora\usr:\vyuka\pelikan\3**

Vyplňování n-úhelníka

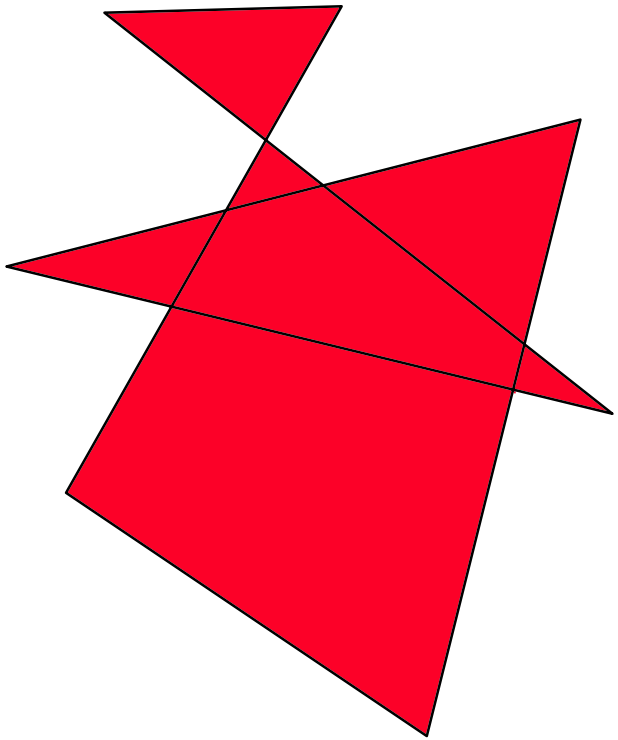
© 1995-2001 Josef Pelikán

KSVI MFF UK Praha

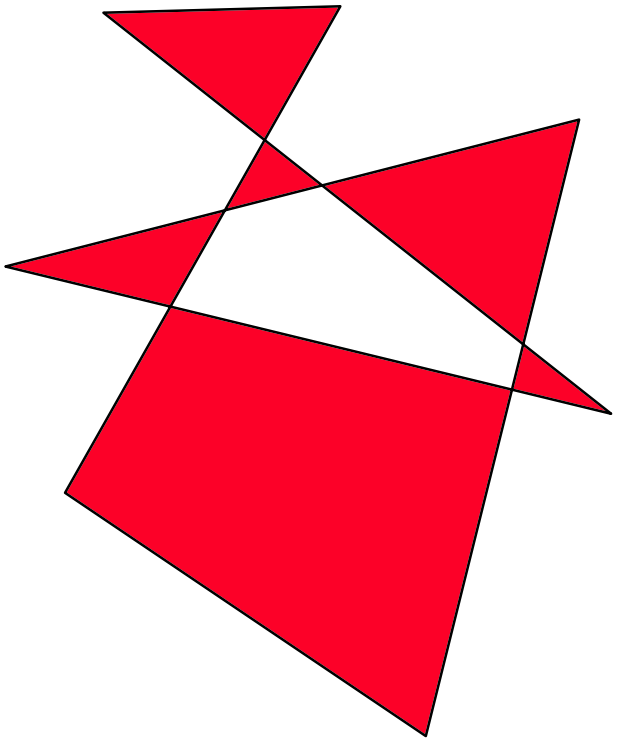
e-mail: Josef.Pelikan@mff.cuni.cz

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

Pravidla vyplňování:

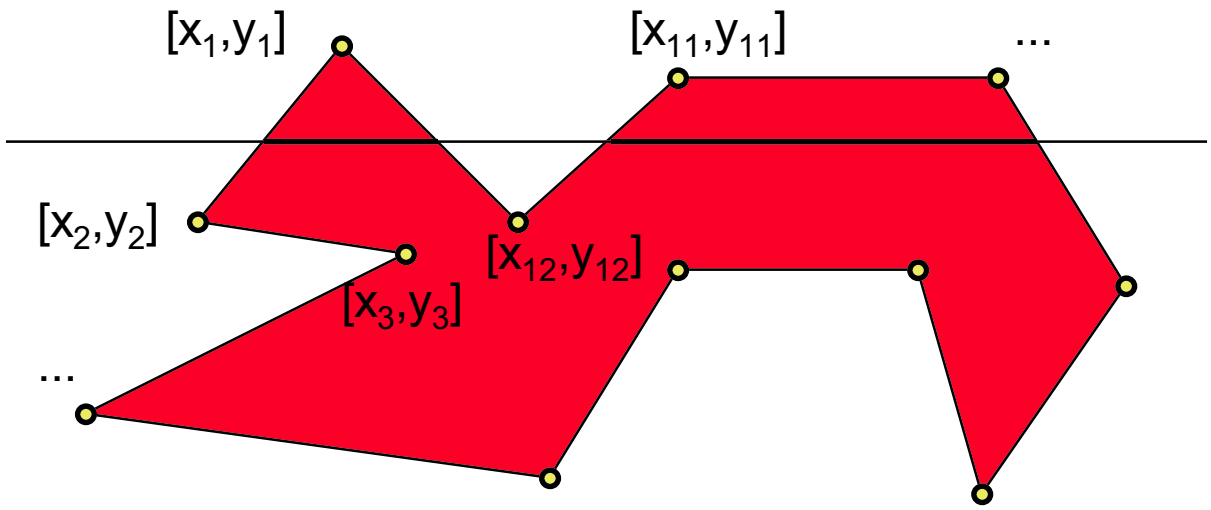


všechny vnitřní body



jen liché body
(“odd-even” rule)

Řádkový algoritmus

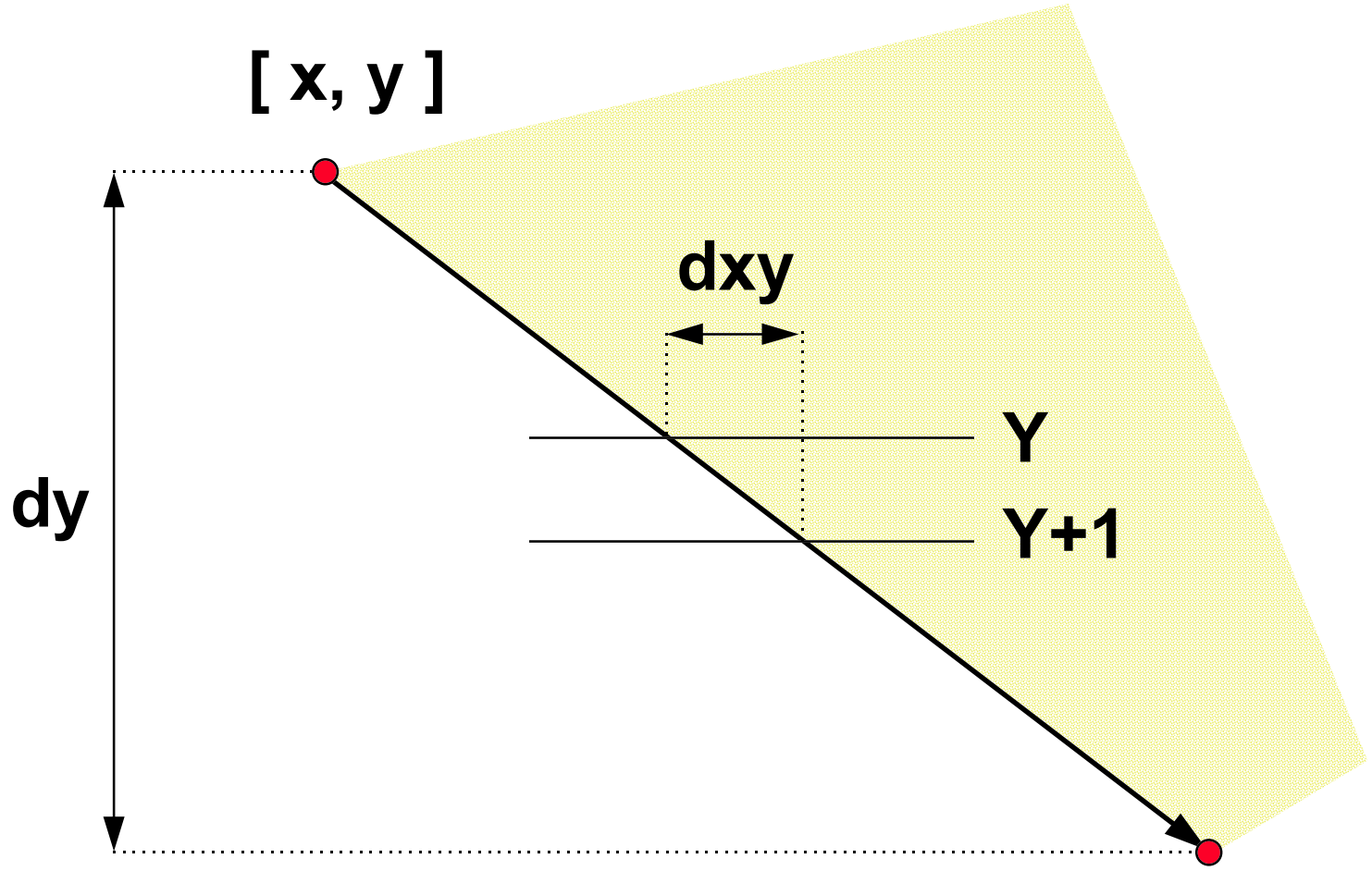


- **n-úhelník** je zadán posloupností svých vrcholů
- může být **nekonvexní**
- zjednodušení: vyplňují se jen **liché body**

1. předzpracování

- n-úhelník rozložíme na **jednotlivé hrany**
- **vodorovné hrany** odstraníme
- pro ostatní hrany vytvoříme **pracovní záznamy**
  - hrany orientujeme směrem dolů

pracovní záznam pro hranu:



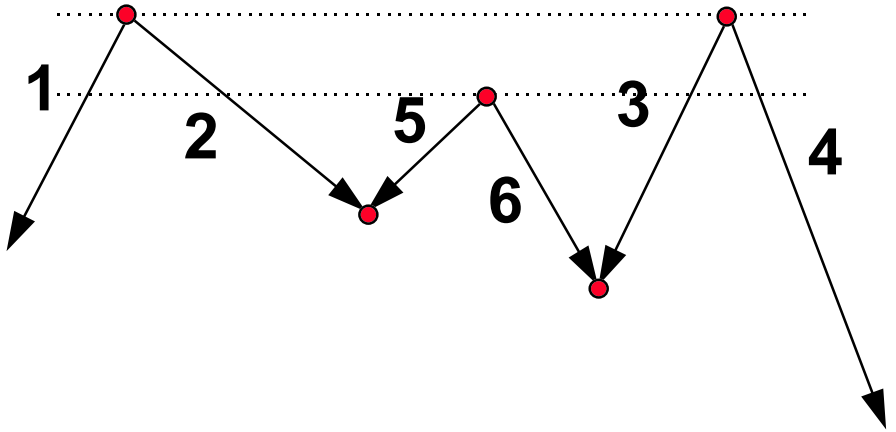
pracovní záznam pro hranu:

- `x : real;` { ***x*** horního koncového bodu, později souřadnice průsečíku s aktuální řádkou }
- `y : integer;` { ***y*** horního koncového bodu }
- `dy : integer;` { výška hrany v pixelech: $|y_2-y|$ }
- `dxy : real;` { změna ***x*** při posunutí na následující řádku (směrnice pro ***x***): $(x_2-x)/dy$ }

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***

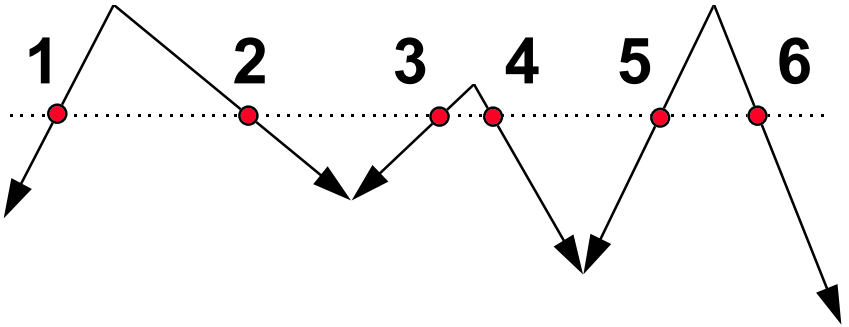


3. aktuální seznam *A*

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

❷ vzestupně podle ***x***

❸ vzestupně podle ***dxy***

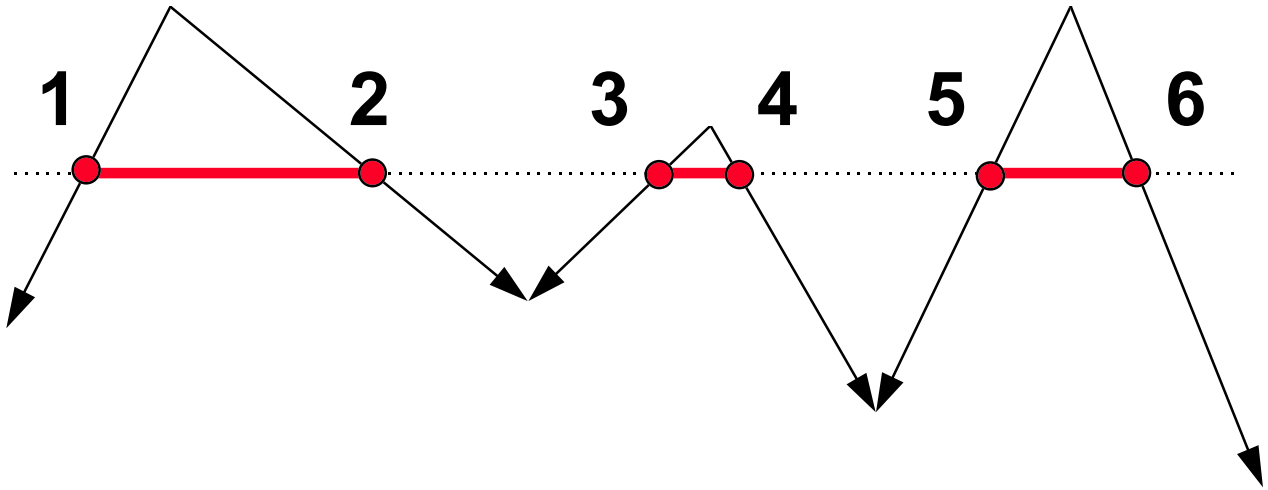


Na začátku zařadíme do ***A*** **počáteční úsek seznamu** ***S*** - hrany se shodným (tj. minimálním) ***y***.

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 := dy - 1;`
- `if dy = 0 then` "odstraň hranu ze seznamu A"
- `x := x + dxy;`

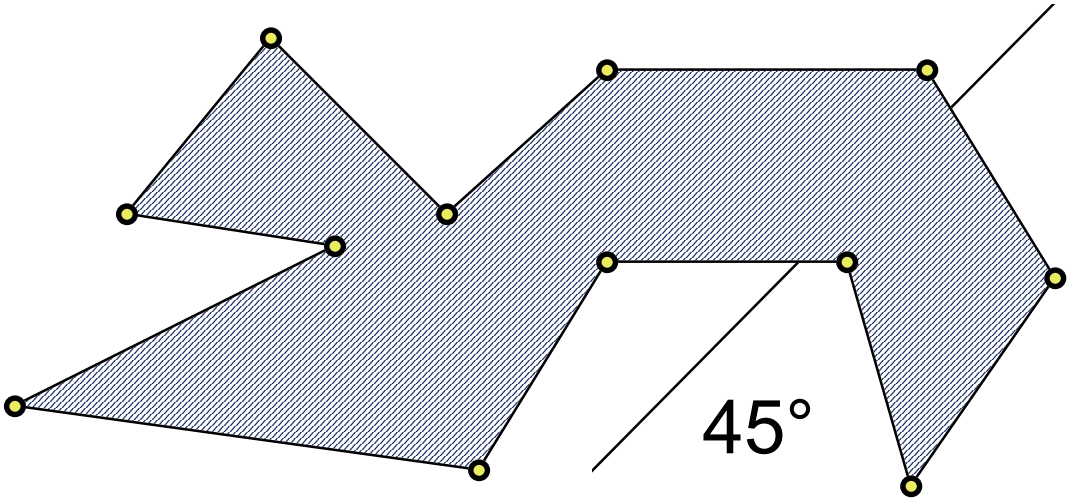
- kontrola **setřídění *A***
- zatřídění **nových hran** z ***S*** do ***A*** (počáteční úsek ***S***)

6. podmínka ukončení cyklu

- jestliže je seznam ***A*** **neprázdný**, výpočet pokračuje krokem **4**

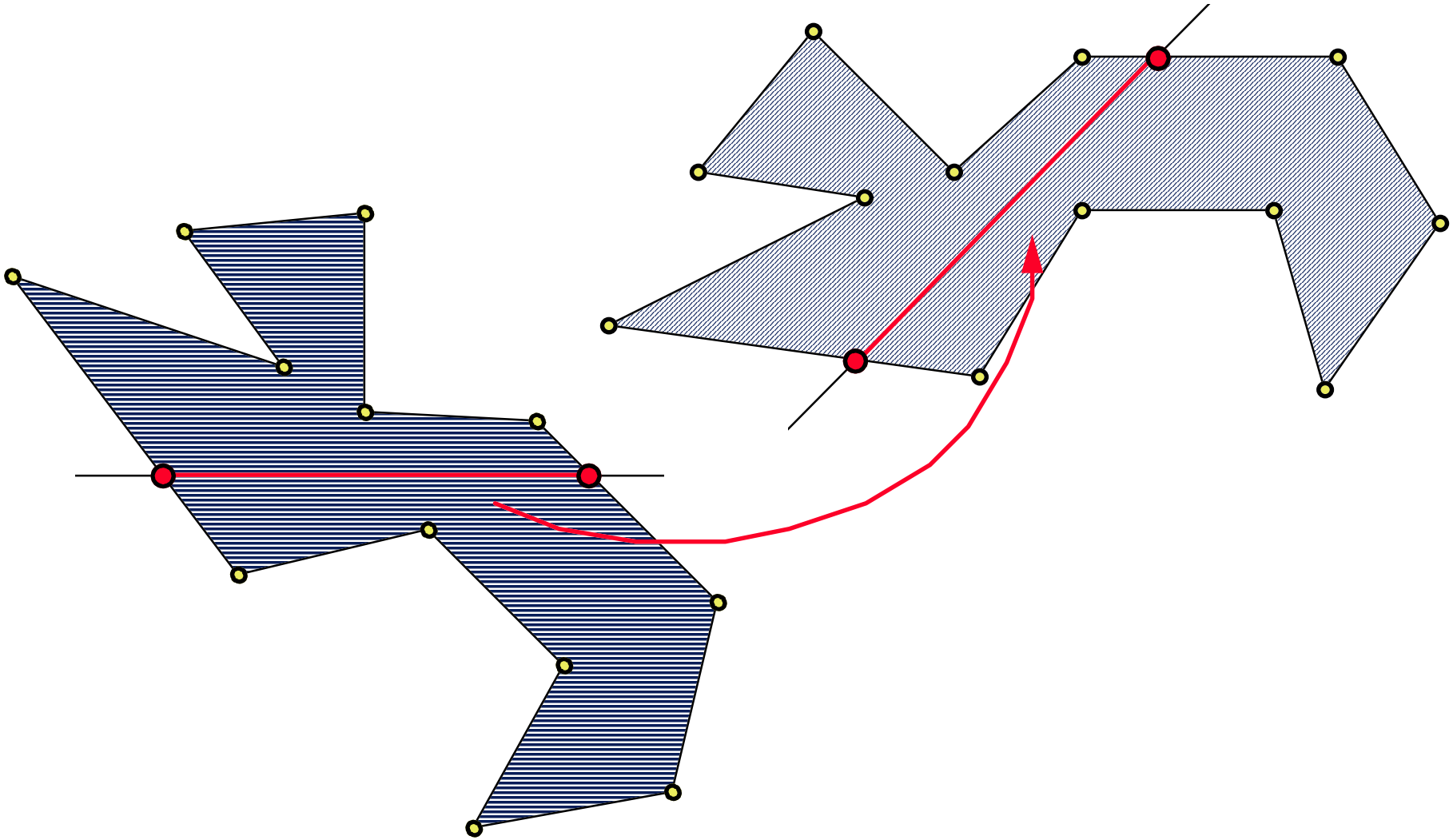
- jinak algoritmus **končí**

Šrafování

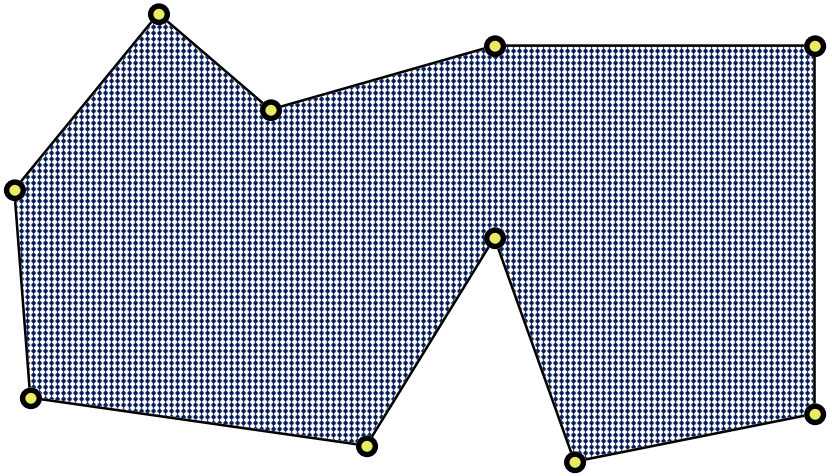


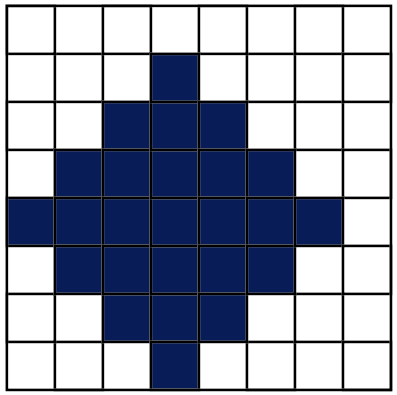
- **otočím** vrcholy n-úhelníka o opačný úhel
- kreslím každý **k-tý řádek**
- před kreslením každou úsečku **otočím zpátky**

Šrafování



Vyplňování vzorkem





matice M[8×8]

- vzorek je zadán **maticí pixelů** (např. 8×8)
- každý pixel kreslím podle předpisu:
  - PutPixel(x, y, M[y mod 8, x mod 8]);

Konec

Další informace:

- **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
- **LAN** na Malé Straně:
  - **barbora\usr:\vyuka\pelikan\2**

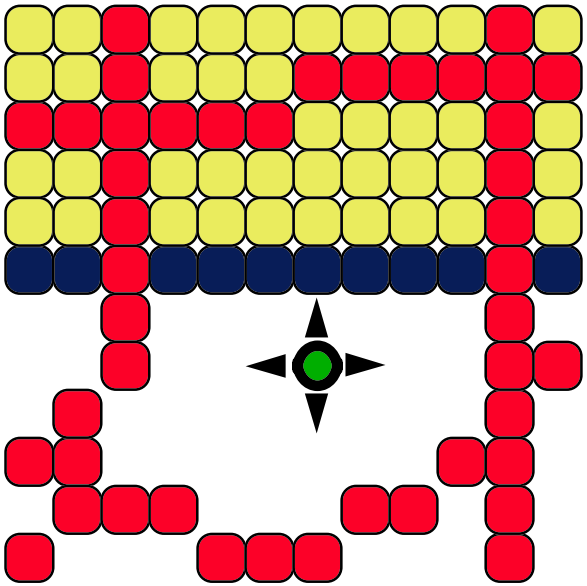
Vyplňování souvislé oblasti

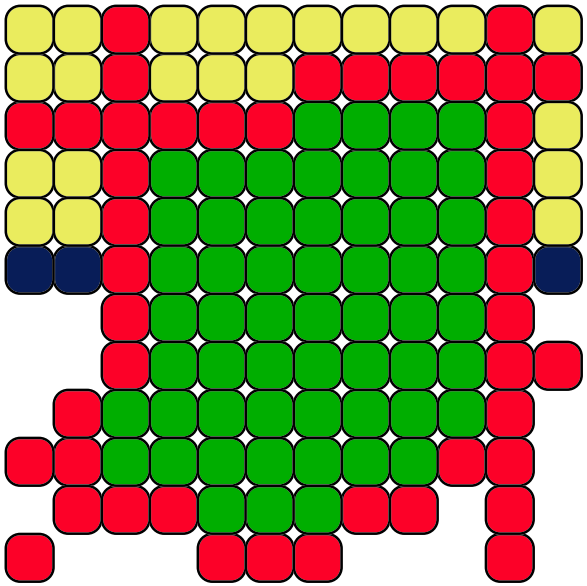
KSVI MFF UK Praha

e-mail: Josef.Pelikan@mff.cuni.cz

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

Hraniční vyplňování

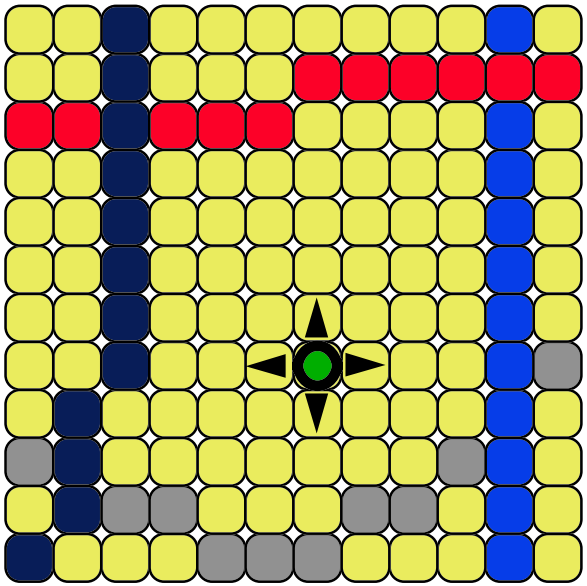


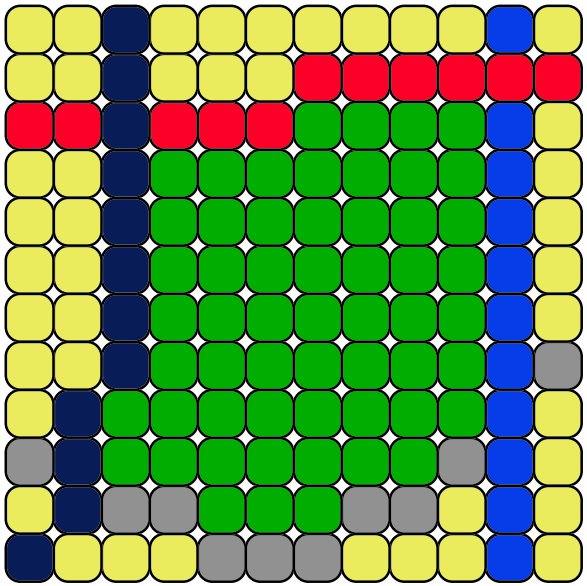


vyplnění až ke hranici dané barvy

GetPixel(x,y) <> barva_hranice

Záplavové vyplňování

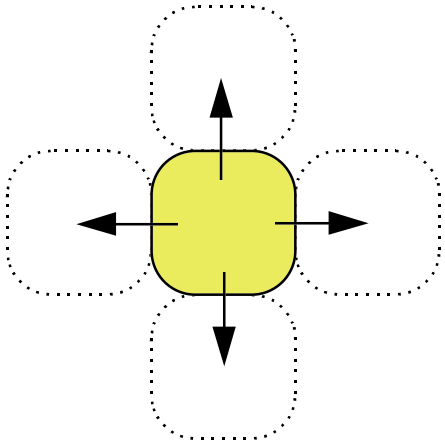


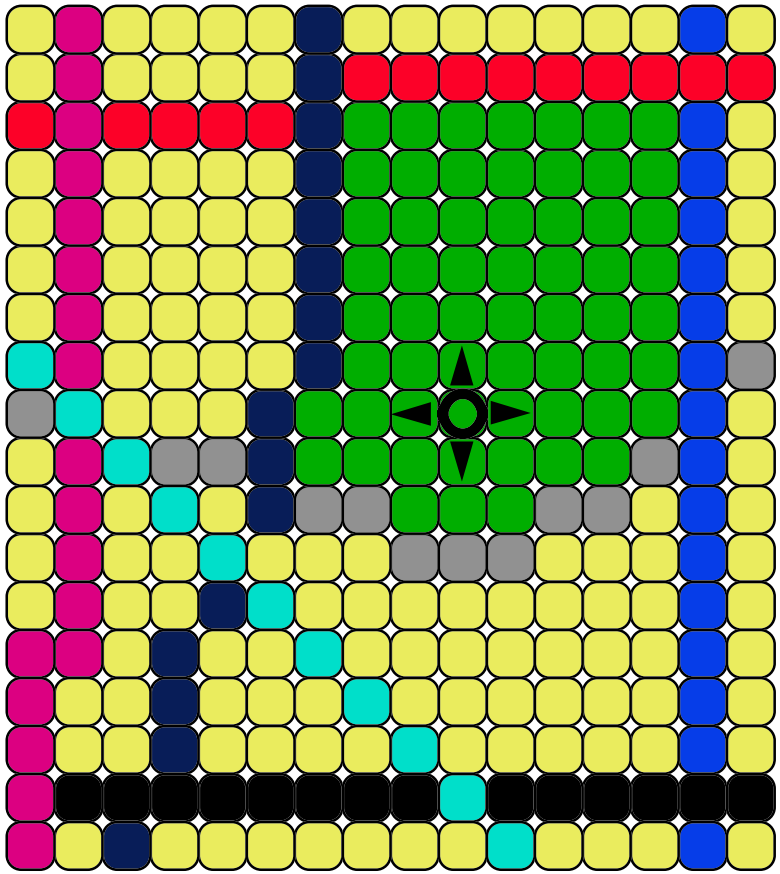


přebarvení pixelů dané barvy

GetPixel(x,y) = původní_barva

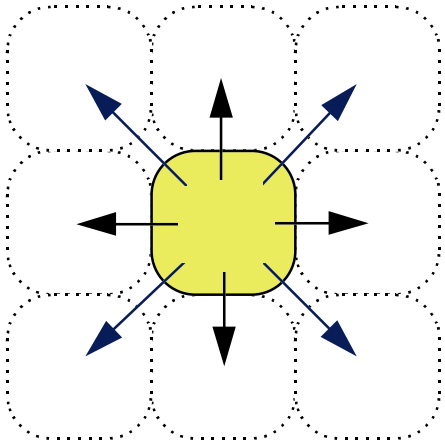
4-souvislá oblast

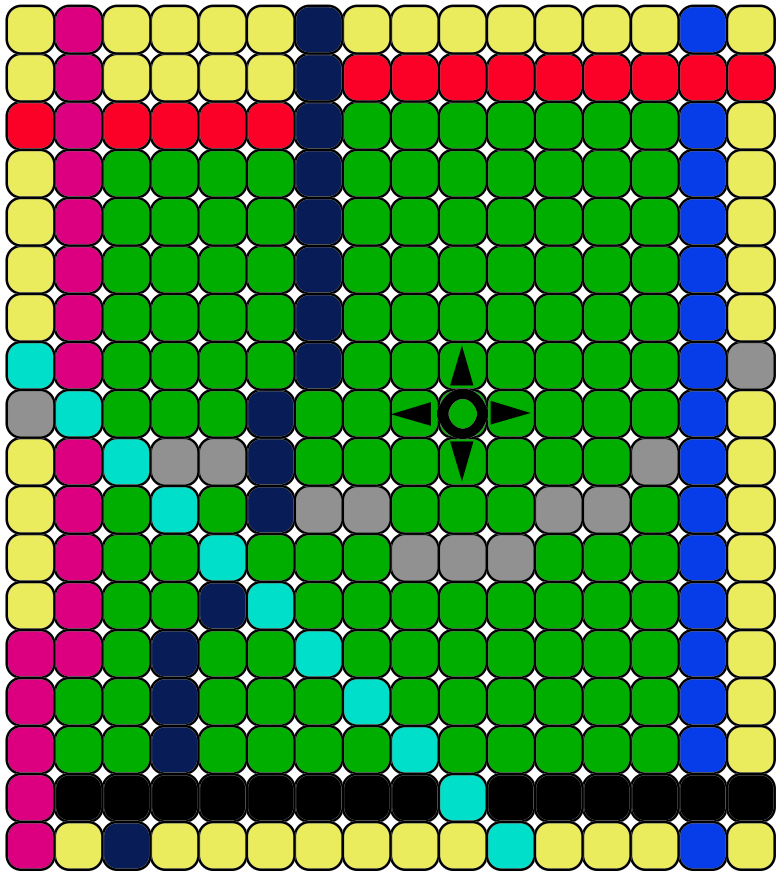




záplavová varianta

8-souvislá oblast





záplavová varianta

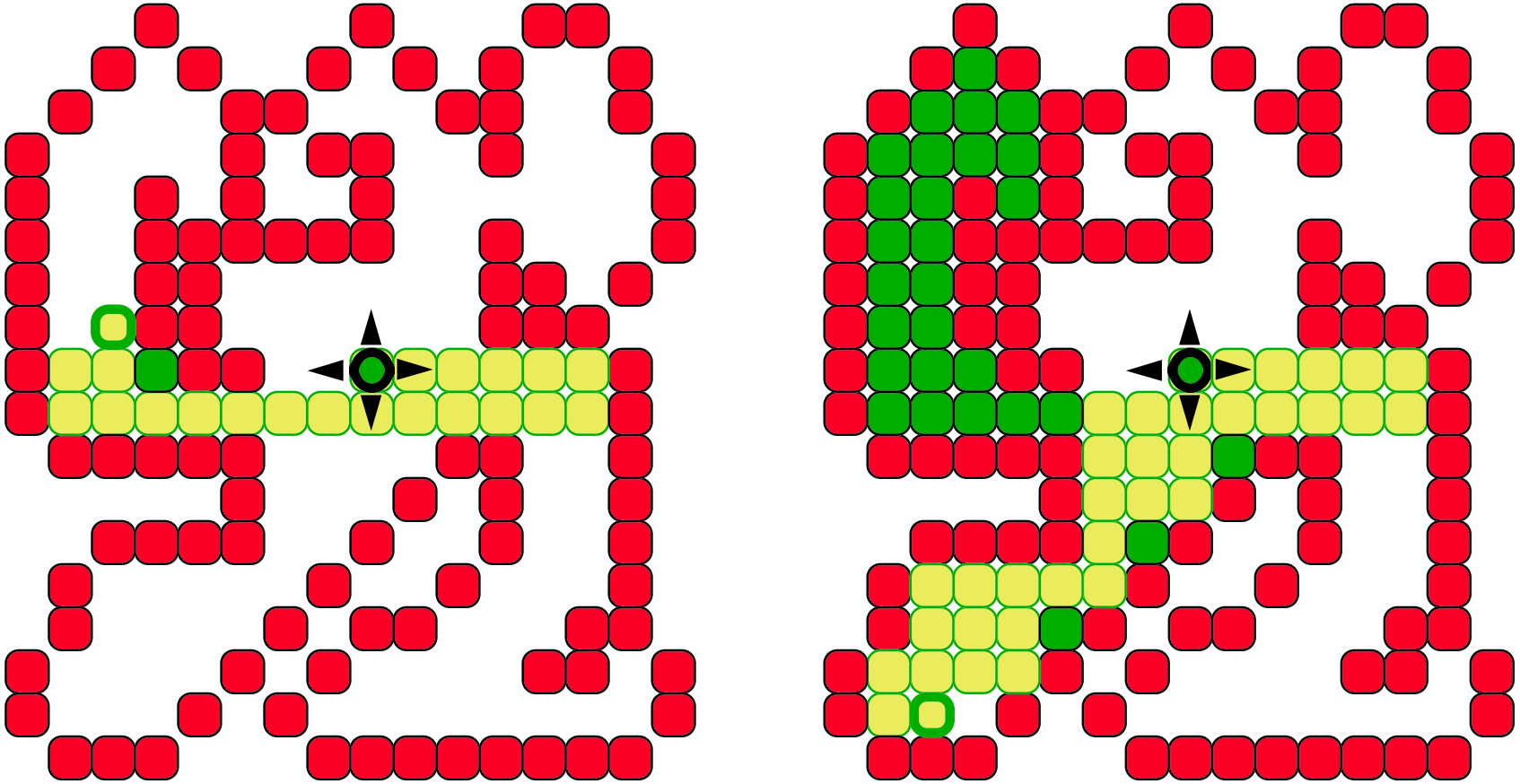
Naivní rekurzivní algoritmus

```
procedure FloodFill4 ( x, y, oldc, newc : integer );
        { záplavová 4-souvislá varianta, oldc <> newc }
begin
  if GetPixel(x,y) = oldc then
    begin           { pixel [x,y] patří do oblasti }
      PutPixel(x,y,newc);
      FloodFill4(x+1,y,oldc,newc);   { čtyři sousedé: }
      FloodFill4(x-1,y,oldc,newc);
      FloodFill4(x,y+1,oldc,newc);
      FloodFill4(x,y-1,oldc,newc);
    end;
end;
```

hraniční varianta:

```
(GetPixel(x,y) <> boundc) and
(GetPixel(x,y) <> newc)
```

Postup vyplňování:



Použití fronty místo zásobníku

```
procedure FloodFill4 ( x, y, oldc, newc : integer );
        { záplavová 4-souvislá varianta, oldc <> newc }
var Q : Queue;
begin
  Q.Init; Q.Put(x,y);
  repeat
    Q.Get(x,y);
    if GetPixel(x,y) = oldc then
      begin          { pixel [x,y] patří do oblasti }
        PutPixel(x,y,newc);
        Q.Put(x+1,y); Q.Put(x-1,y);
        Q.Put(x,y+1); Q.Put(x,y-1);
      end;
  until Q.Empty;
end;
```

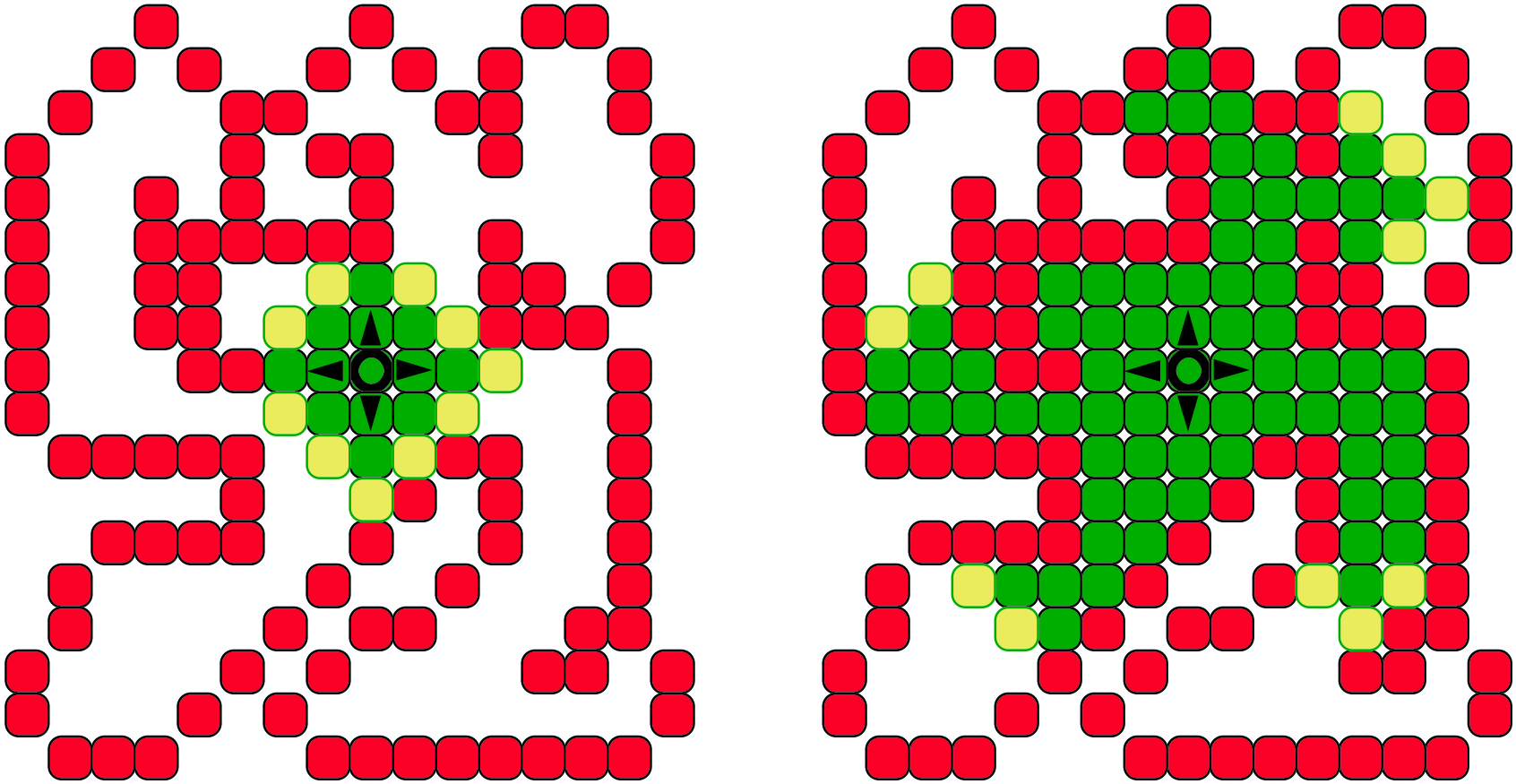
Úspornější varianta

```
procedure FloodFill4 ( x, y, oldc, newc : integer );
        { záplavová 4-souvislá varianta, oldc <> newc }
var Q : Queue;

  procedure NextPixel ( x, y : integer );
  begin      { patří-li pixel do oblasti, uloží ho do fronty }
    if GetPixel(x,y) = oldc then
      begin
        PutPixel(x,y,newc); Q.Put(x,y);
      end;
  end;

begin
  Q.Init; NextPixel(x,y);          { startovní pixel }
  repeat
    Q.Get(x,y);
    NextPixel(x+1,y); NextPixel(x-1,y);   { čtyři sousedé: }
    NextPixel(x,y+1); NextPixel(x,y-1);
  until Q.Empty;
end;
```

Postup vyplňování:



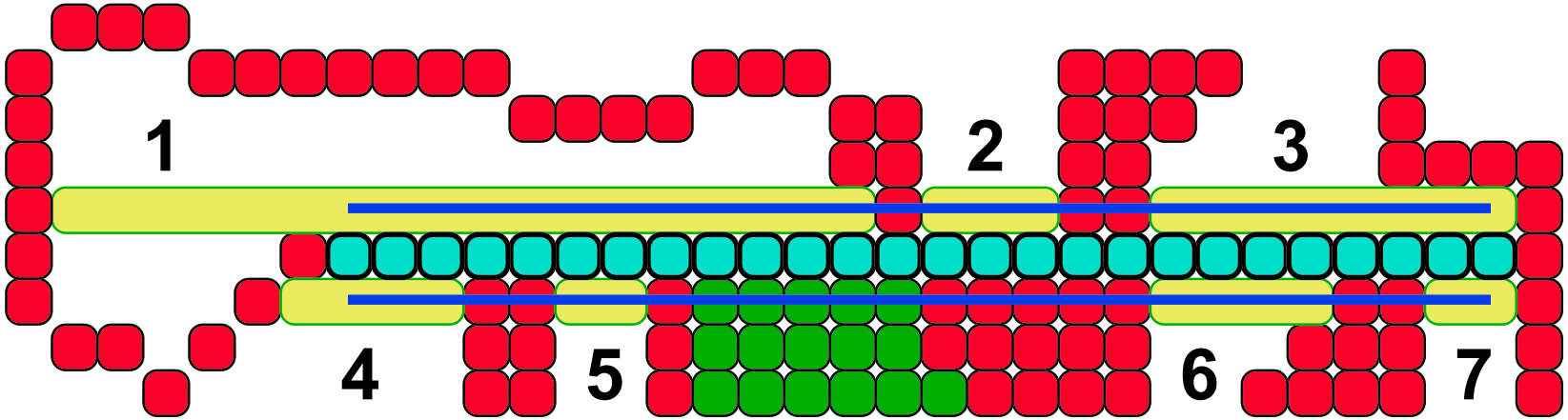
Řádkové vyplňování

```
procedure LineFloodFill4 ( x, y, oldc, newc : integer );
        { záplavová 4-souvislá varianta, oldc <> newc }
var S : Stack;                  { položka: [Xmin,Xmax,y] }
    Xmin, Xmax : integer;       { meze na aktuální řádce }

  procedure Search ( Xmin, Xmax, y : integer );
  var Xm : integer;
  begin        { najde všechna pokračování v daném úseku řádky }
    while GetPixel(Xmin-1,y) = oldc do Dec(Xmin);
    repeat                  { zkouším [Xmin,y] }
      Xm := Xmin;                { hledám pravý konec úseku: }
      while GetPixel(Xm+1,y) = oldc do Inc(Xm);
      S.Push(Xmin,Xm,y);
      Xmin := Xm+2;              { hledám následující úsek: }
      while (Xmin <= Xmax) and (GetPixel(Xmin,y) <> oldc) do
        Inc(Xmin);
    until Xmin > Xmax;
  end;

...
```

Hledání následníků:



hranice

dříve vyplněné pixely

naposledy vyplněné pixely

prohledávané řádky

1-7 nové položky na zásobníku

Řádkové vyplňování

```
...
begin
  S.Init; Search(x,x,y);    { první bod (semínko) }
  repeat
    S.Pop(Xmin,Xmax,y);
    if GetPixel(Xmin,y) = oldc then
      begin                  { úsek ještě nebyl vyplněn }
        Line(Xmin,y,Xmax,y,newc);
        Search(Xmin,Xmax,y-1);
        Search(Xmin,Xmax,y+1);
      end;
  until S.Empty;
end;
```

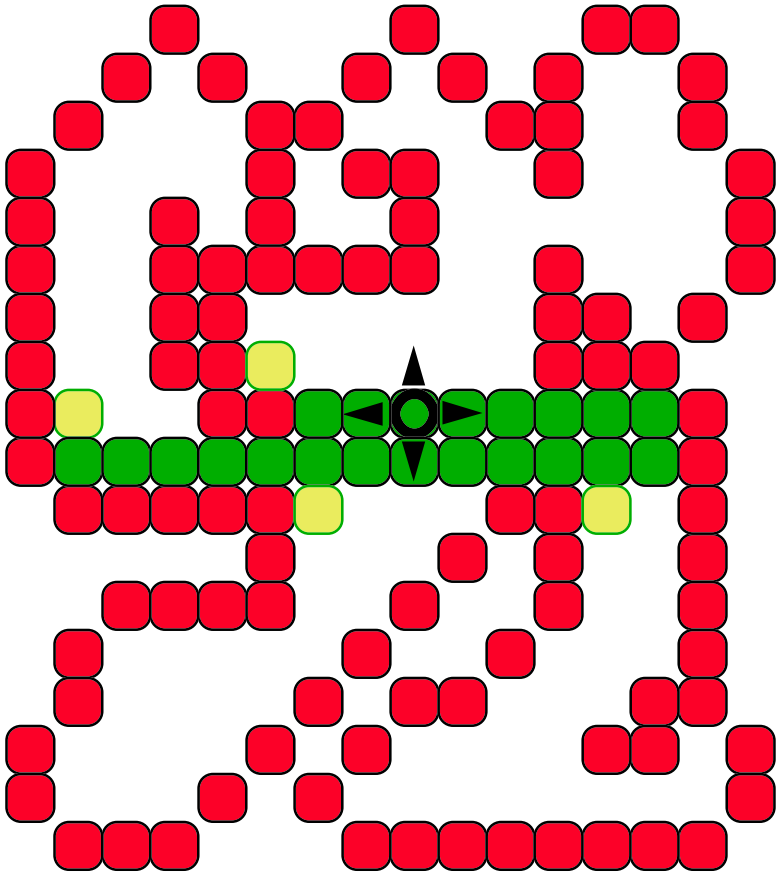
hraniční varianta:

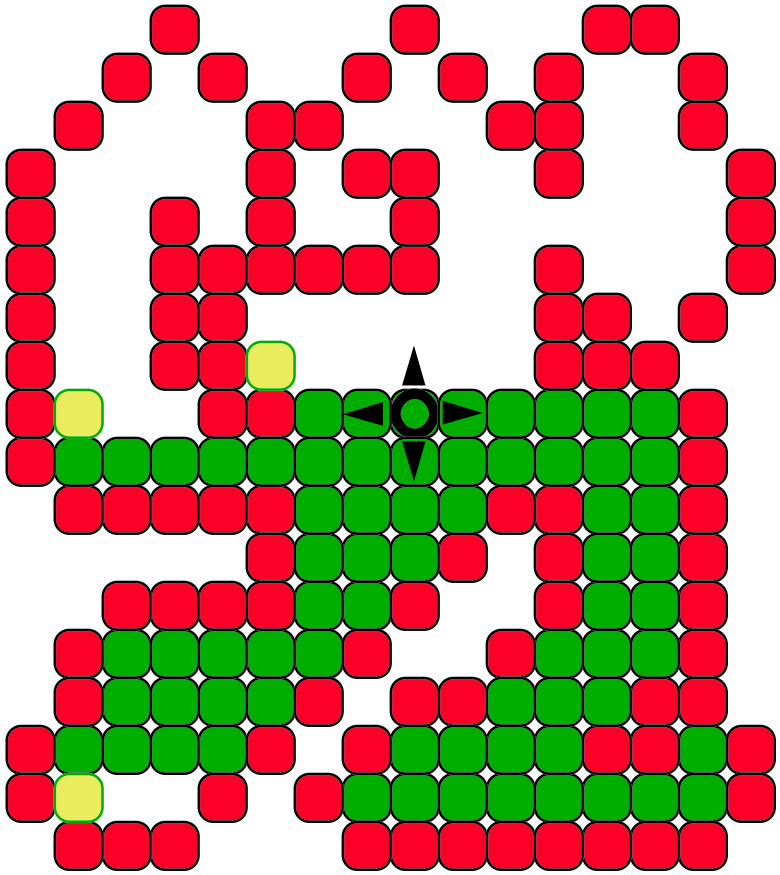
```
(GetPixel(Xmin,y) <> boundc)
and (GetPixel(Xmin,y) <> newc)
```

8-souvislé vyplňování:

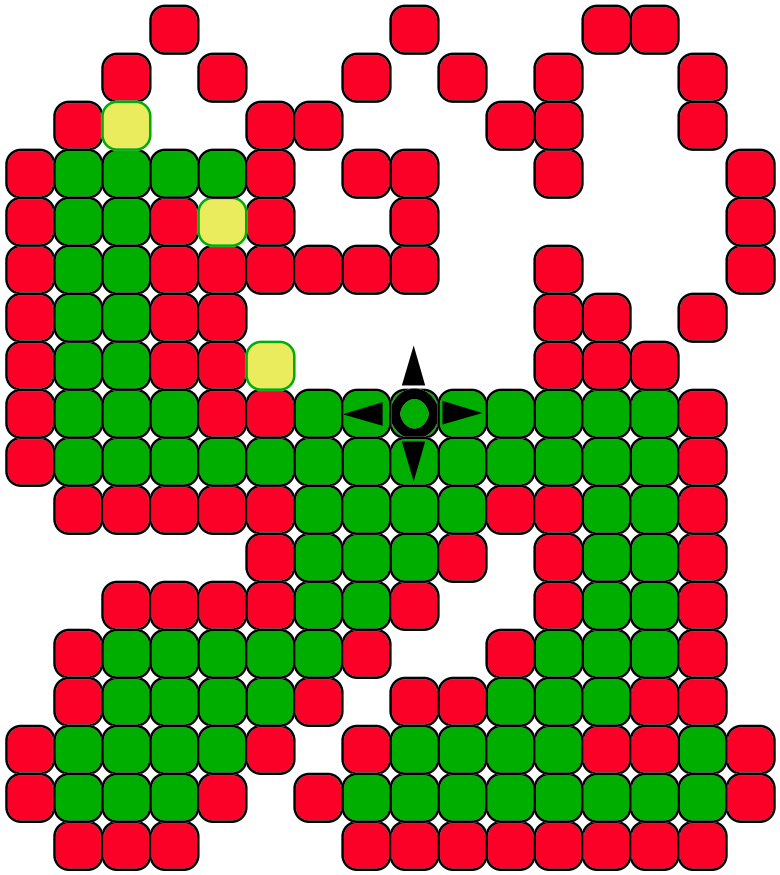
```
Search(Xmin-1,Xmax+1,*)
```

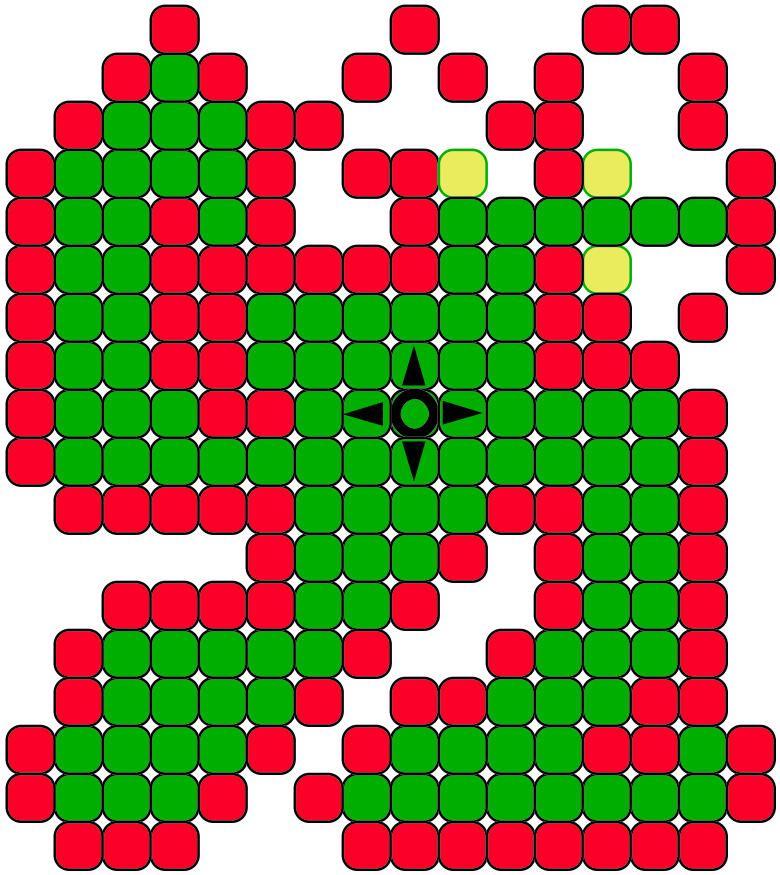
Postup vyplňování:





Postup vyplňování:





Výhody řádkového algoritmu

- **menší spotřeba paměti**
  - zásobník v běžných případech roste jen pomalu
- **větší rychlost**
  - úspornější přístup do VideoRAM po řádkách
- **zásobník** versus **fronta**:
  - při použití zásobníku je postup vyplňování lokální
  - výhodné při přepínání stránek VideoRAM

Konec

Další informace:

- **J. Foley, A. van Dam, S. Feiner, J. Hughes**: ***Computer Graphics, Principles and Practice***, 979-982
- **Jiří Žára a kol.**: ***Počítačová grafika***, principy a algoritmy, 142-147
- **LAN** na Malé Straně:
  - **barbora\usr:\vyuka\pelikan\2**

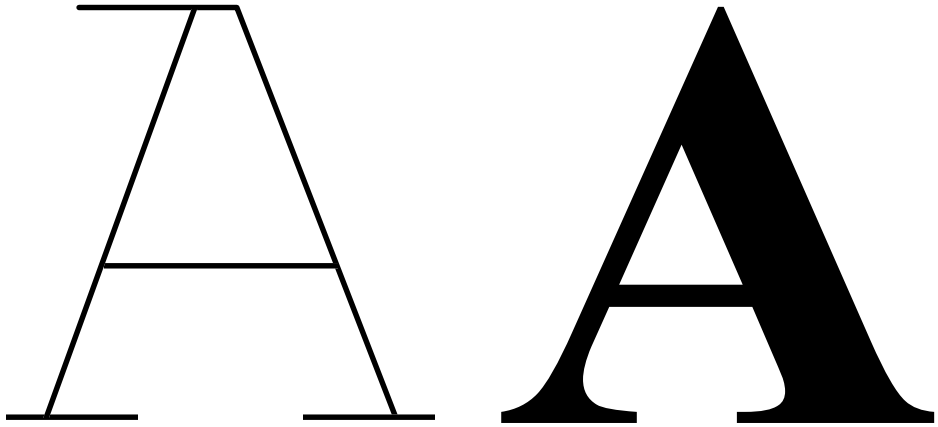
Kreslení písma

KSVI MFF UK Praha

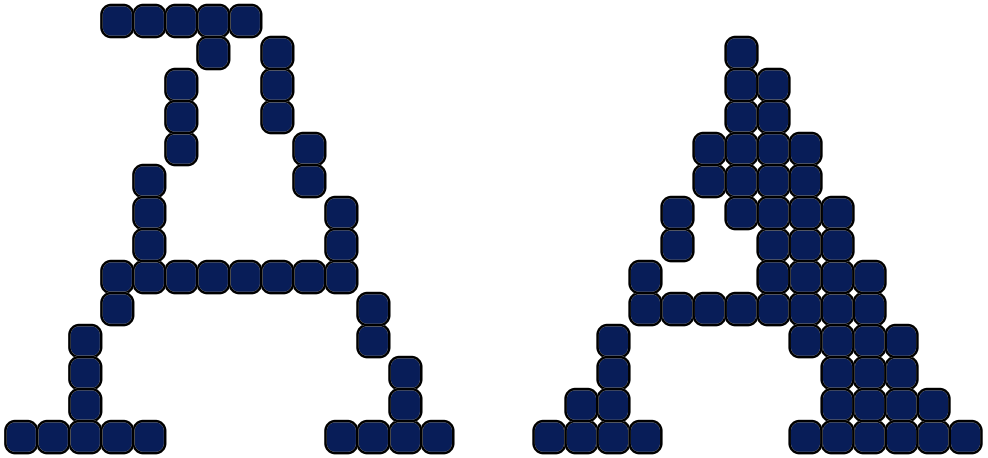
e-mail: Josef.Pelikan@mff.cuni.cz

WWW: http://cgg.ms.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 operace “BitBlt”
- **neúspěch** při hledání ve “font cache”:
  - písmeno musím převést do rastrové podoby, přidám ho do “cache”
  - z “cache” odstraňuji **nejdéle nepoužívané položky**

Konec

Další informace:

- **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

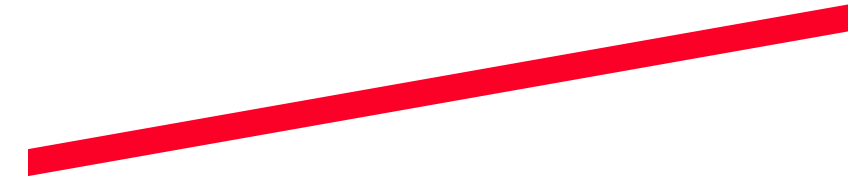
Vyhlazování ("anti-aliasing")

KSVI MFF UK Praha

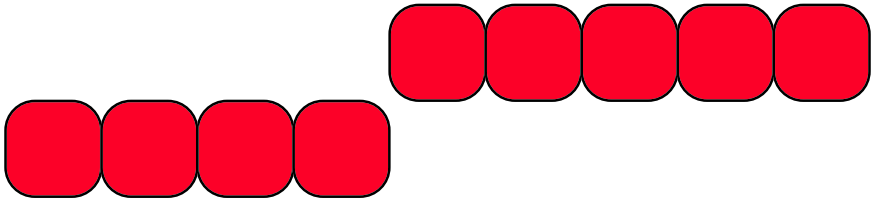
e-mail: Josef.Pelikan@mff.cuni.cz

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

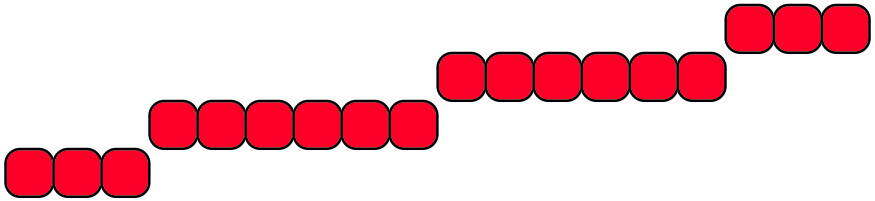
Úsečky na rastrovém zařízení



ideální úsečka

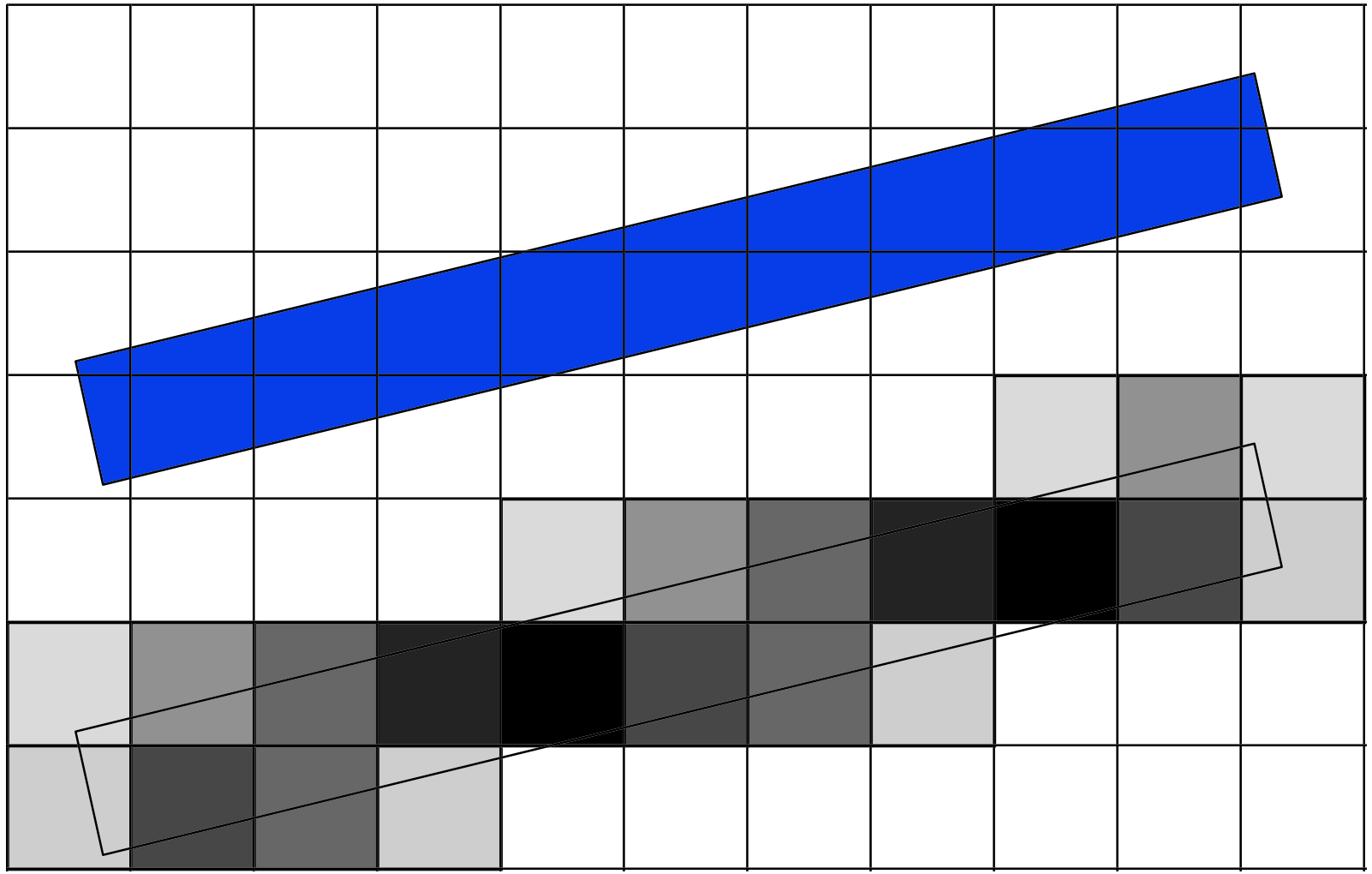


rastrová kresba

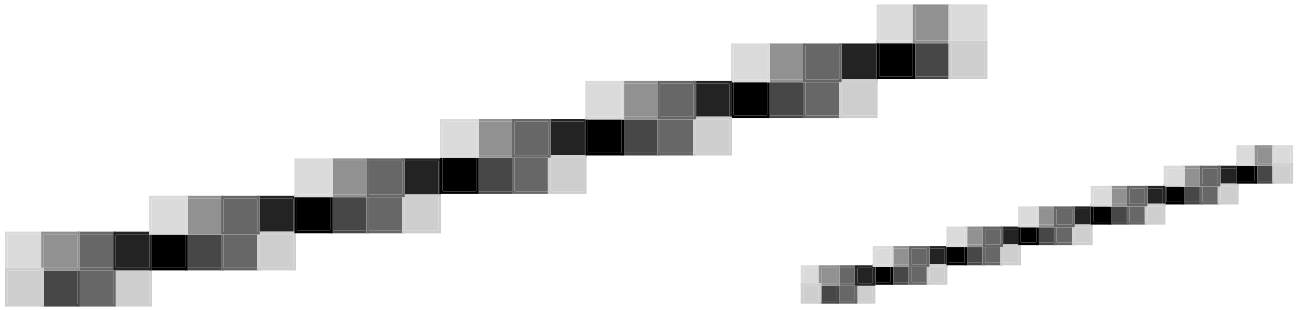


dvakrát větší rozlišení

Pokrytí plochy pixelu

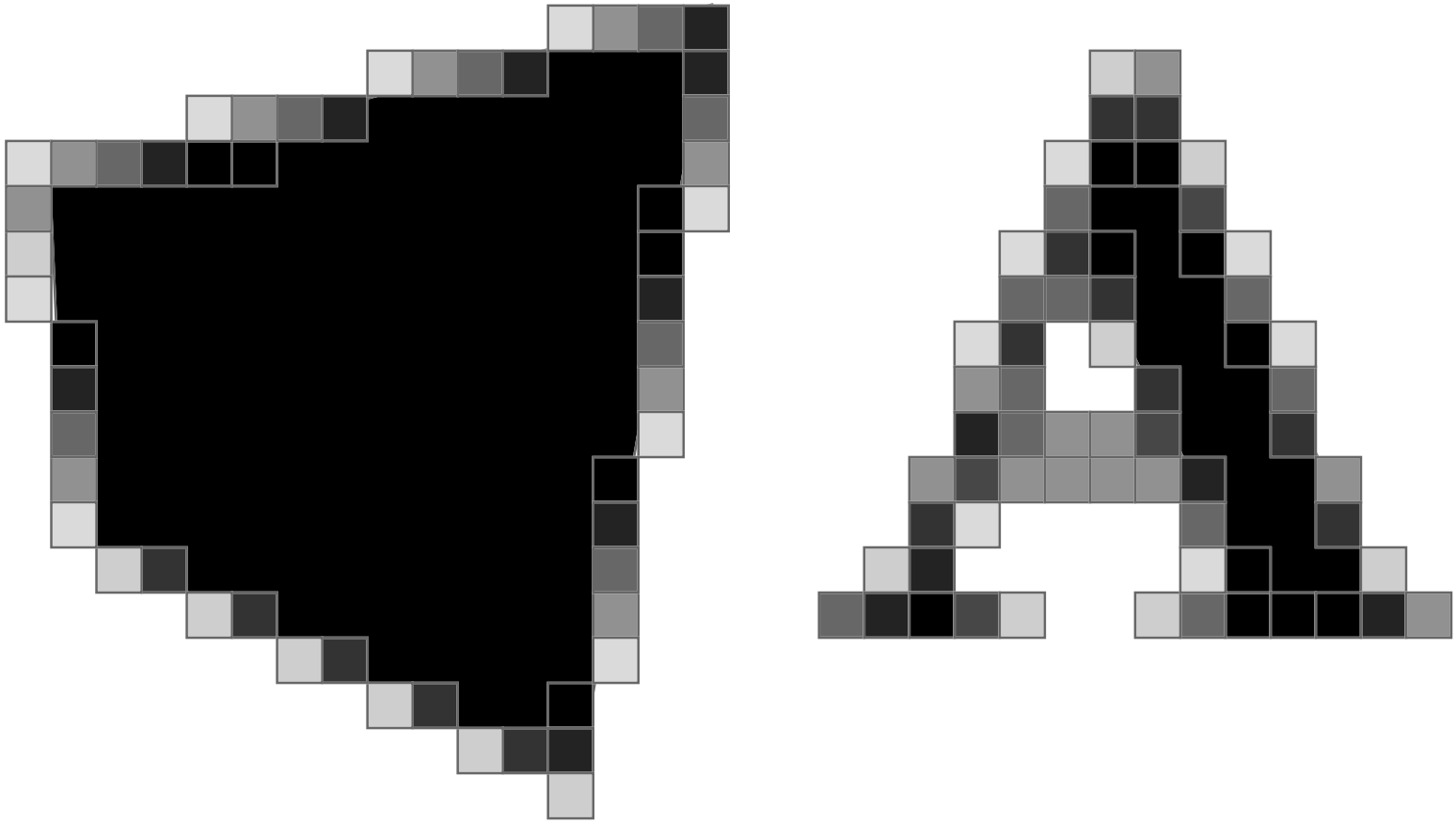


Pokrytí plochy pixelu



- ke kreslení použiji **více odstínů** dané barvy
  - zvětším prostorové rozlišení na úkor barevného
- pixely i kreslené objekty jsou **plošné útvary**
- každý pixel rozsvítím intenzitou úměrnou **ploše jeho pokryté části**

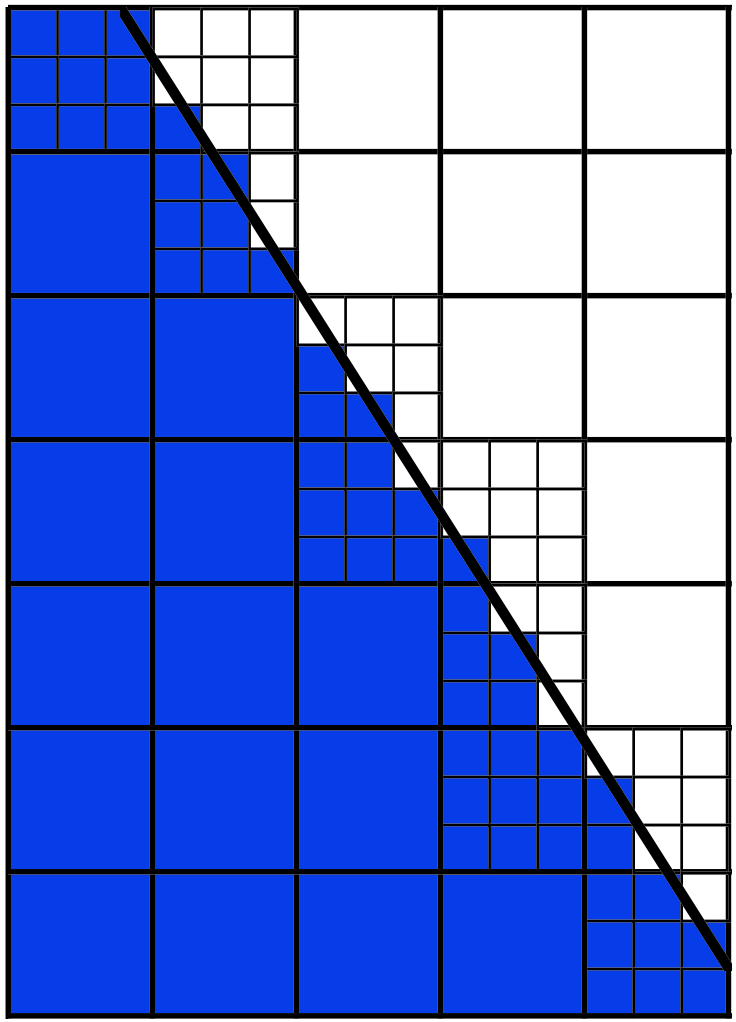
N-úhelníky a text:

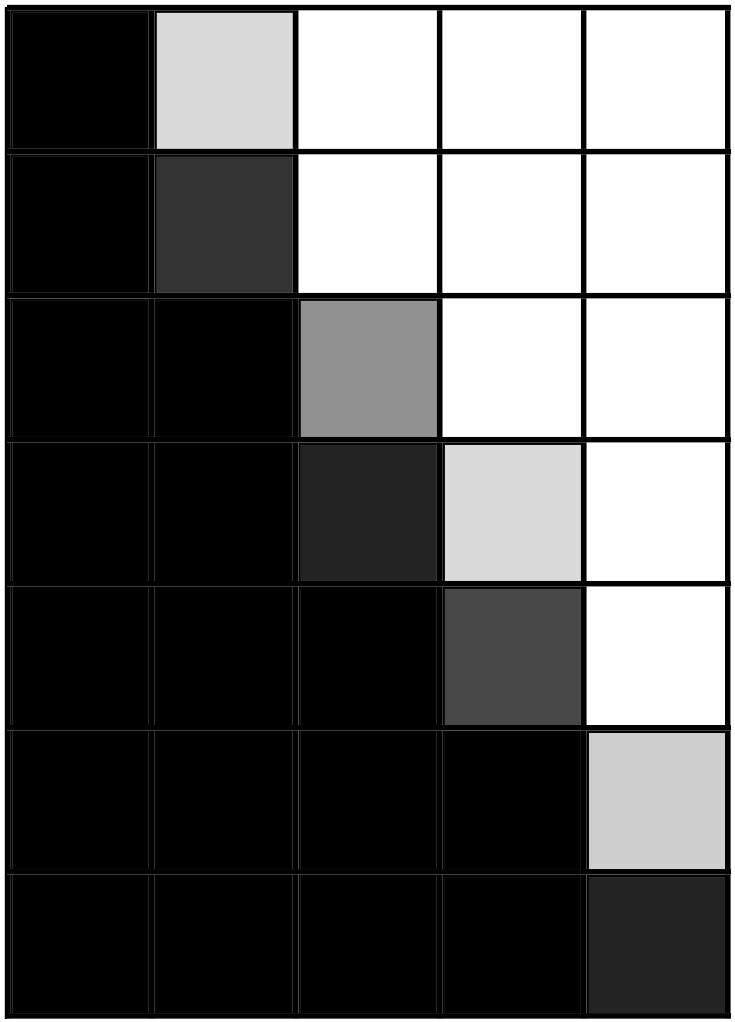


Kreslení s vyhlazováním

- **úsečka**: kreslím vždy oba pixely, mezi kterými úsečka prochází
  - intenzitu určím podle vzdálenosti středu pixelu od úsečky (desetinná část **y** v DDA, člen **D** v Bresenhamově algoritmu)
- **n-úhelník**: kreslím všechny pixely, do jejichž plochy n-úhelník zasahuje
  - intenzitu okrajových pixelů spočítám opět podle vzdálenosti (desetinná část **y**, **D**)

Vícenásobné vzorkování ("supersampling")





Vícenásobné vzorkování

- objekt nakreslím do bufferu **ve větším rozlišení** (při zvětšení např. 2× až 4×)
  - každý pixel se rozloží na “subpixely”
- **barevný odstín** skutečně kresleného pixelu určím jako aritmetický průměr odstínů jeho subpixelů
  - někdy se používá **vážený průměr** (subpixely ležící ve středu pixelu mají větší váhu)

Konec

Další informace:

- **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
- **LAN** na Malé Straně:
  - **barbora\usr:\vyuka\pelikan\2**

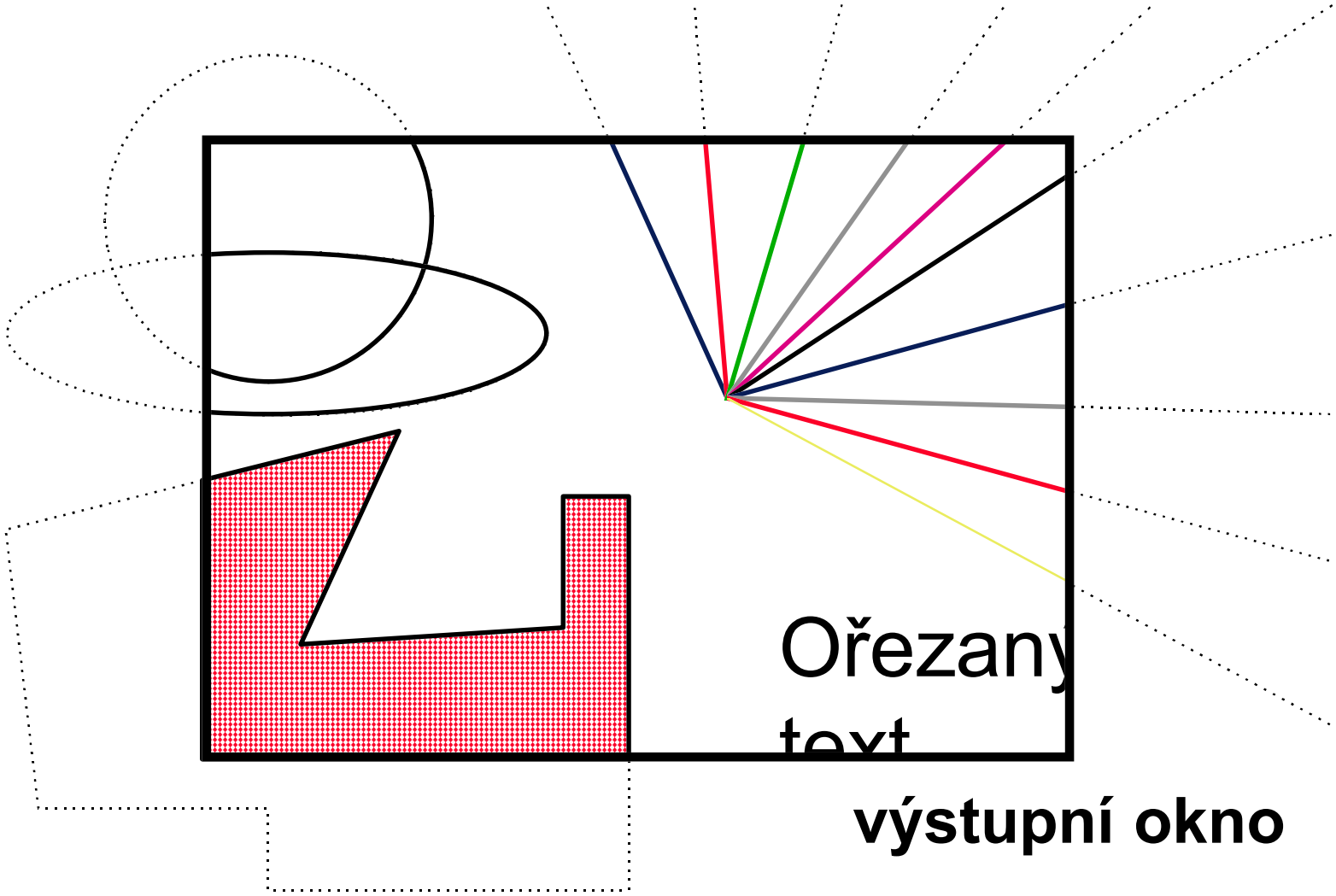
Ořezávání v rovině

KSVI MFF UK Praha

e-mail: Josef.Pelikan@mff.cuni.cz

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

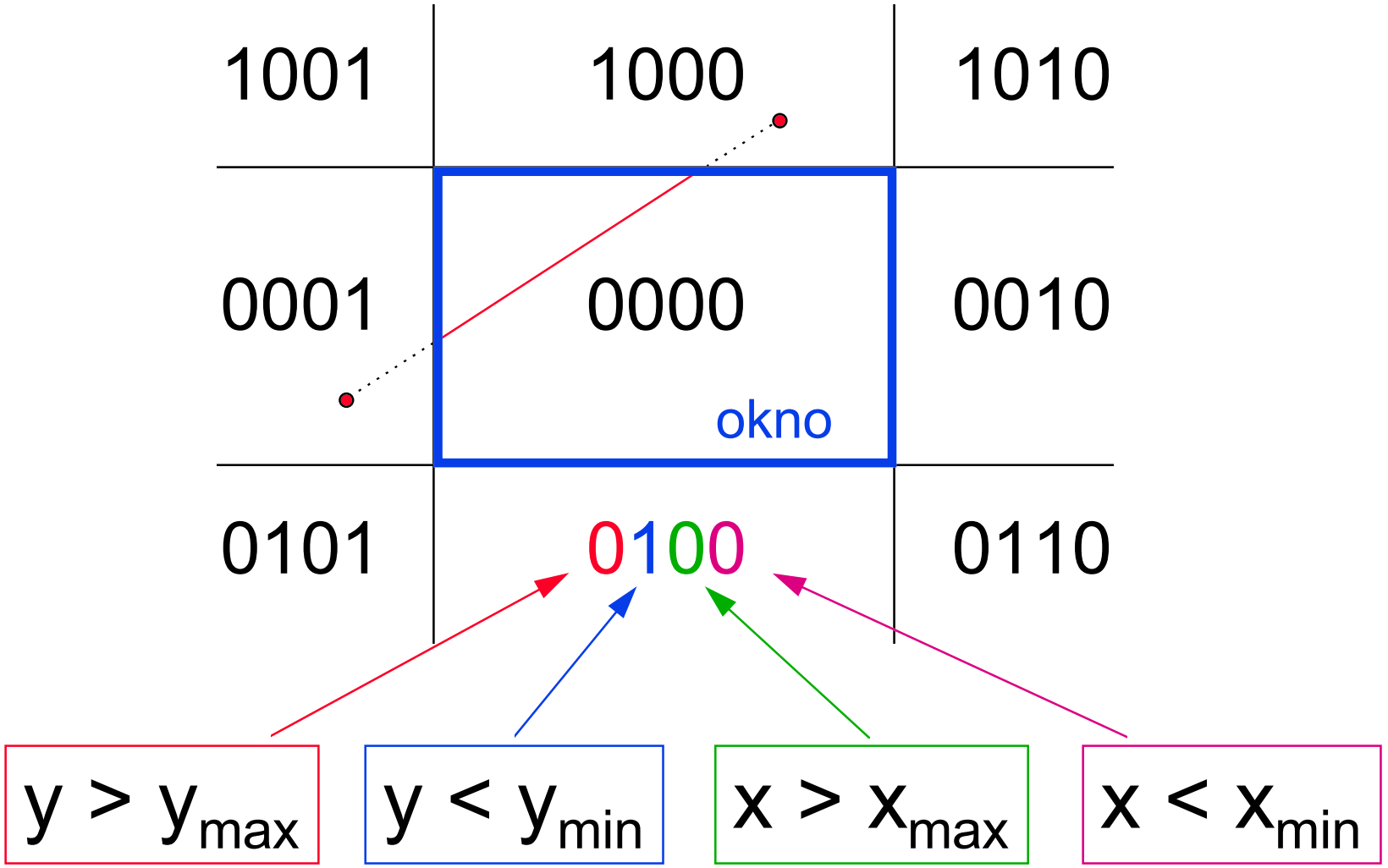
2D ořezávání



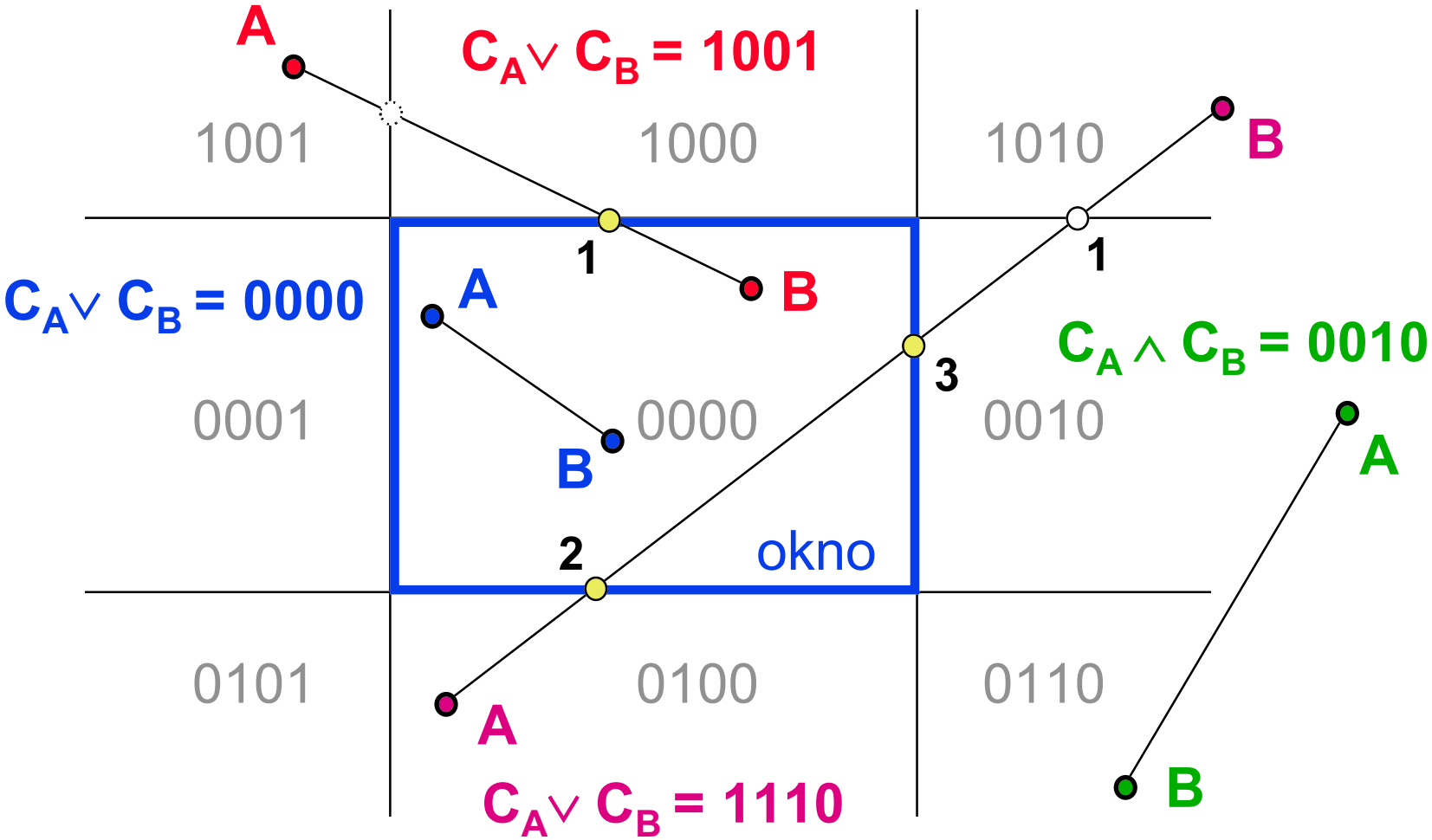
Ořezávání úseček

- přepočítávají se **koncové body** úseček:
  - $[x_1,y_1]$-$[x_2,y_2] \rightarrow [x_A,y_A]$-$[x_B,y_B]$ nebo $\varnothing$
- běžně jsou x_A,y_A,x_B,y_B v **celočíselném** formátu
  - nepřesnosti v kresbě
- **racionální** souřadnice jsou nepraktické
  - algoritmus kreslení úsečky mívá celočíselný vstup
- nejlepší by bylo spočítat **mezivýsledky** algoritmu na kreslení úsečky (Bresenham)

Kódy oblastí (Cohen-Sutherland)



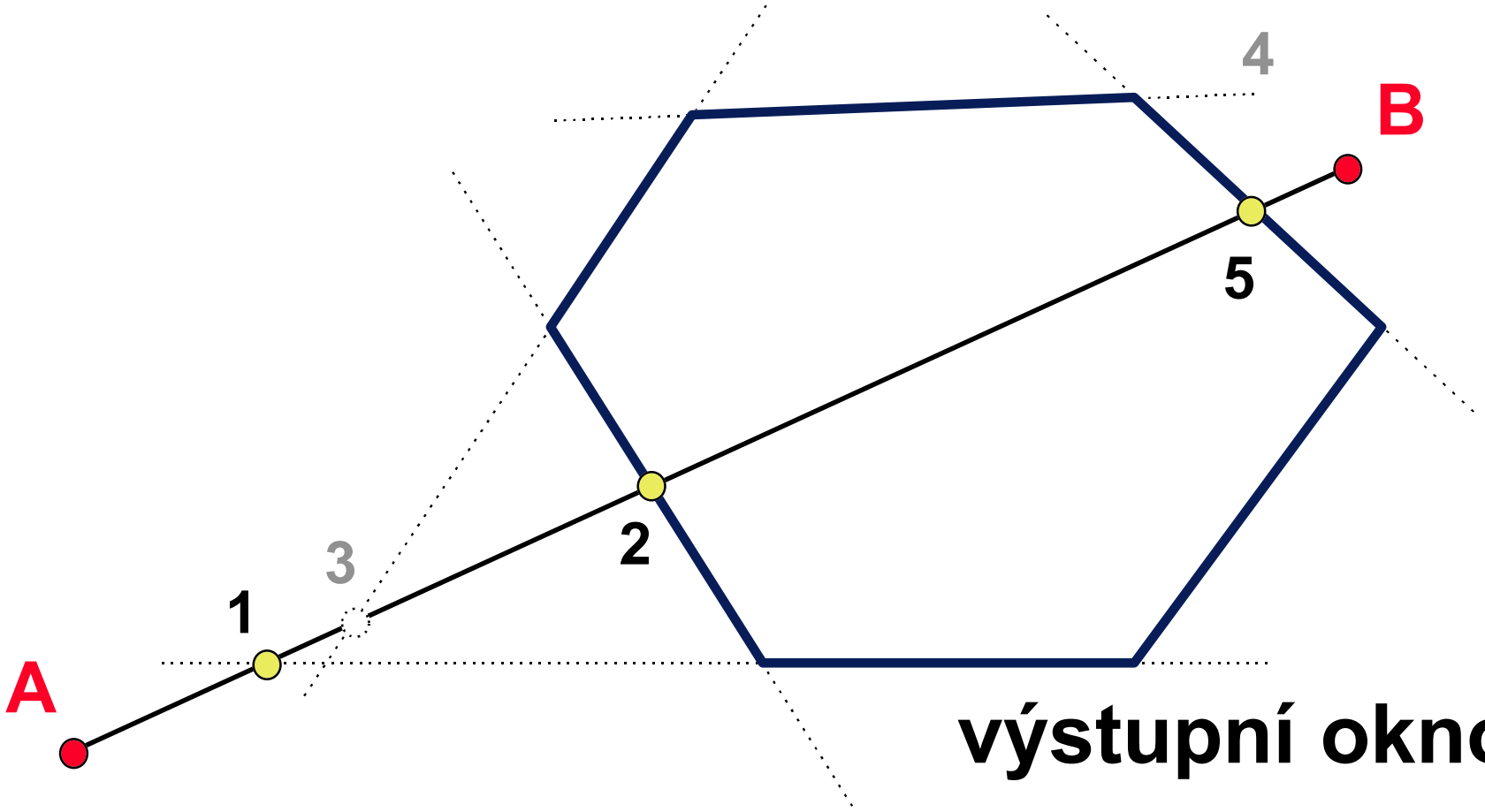
Cohen-Sutherland



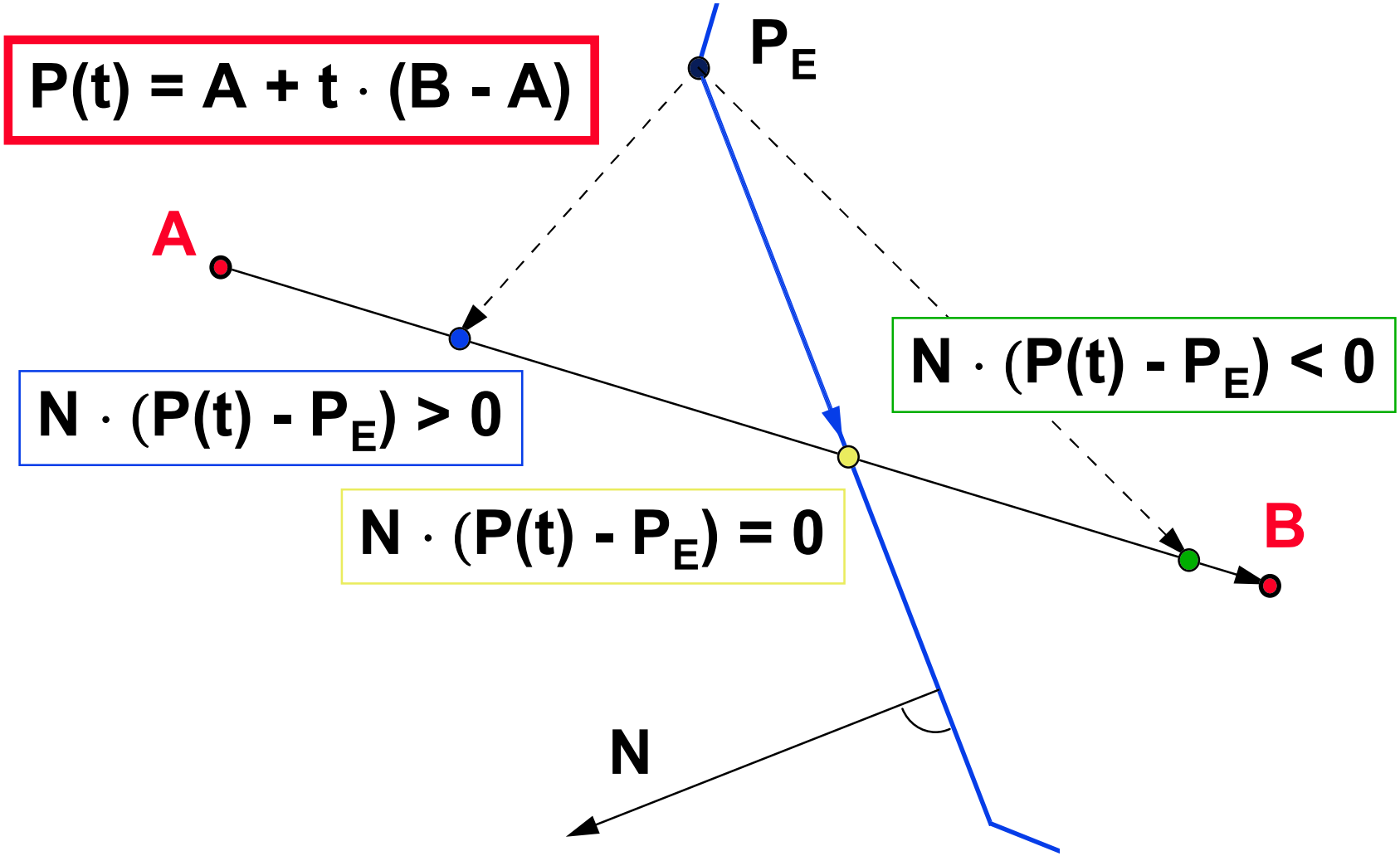
Cohen-Sutherland

1. spočítám **kódy koncových bodů C_A, C_B**
2. je-li **$C_A \vee C_B = 0$**, celá úsečka leží **uvnitř okna**
3. je-li **$C_A \wedge C_B \neq 0$**, celá úsečka leží **mimo okno**
4. pro každou jedničku v **$C_A \vee C_B$** postupně úsečku ořezávám - např. pro **Y_{max}** počítám:
   - **$X := X_A + (X_B - X_A) * (Y_{max} - Y_A) / (Y_B - Y_A)$**
   - **$Y := Y_{max}$**
   - (**A** nebo **B**) **:= [X,Y]**, opravím kód **C_A** nebo **C_B**

Okno = konvexní n-úhelník



Parametrické ořezávání
(Cyrus-Beck)



Výpočet průsečíku:

$$\mathbf{N}\cdot\left(\mathbf{P}(\mathbf{t_0})-\mathbf{P_E}\right)=\mathbf{0}$$

$$\mathbf{N}\cdot\left[\mathbf{A}+\mathbf{t_0}\cdot\left(\mathbf{B}-\mathbf{A}\right)-\mathbf{P_E}\right]=\mathbf{0}$$

$$\mathbf{t_0}=-\frac{\mathbf{N}\cdot\left(\mathbf{A}-\mathbf{P_E}\right)}{\mathbf{N}\cdot\left(\mathbf{B}-\mathbf{A}\right)}$$

$\mathbf{N}\cdot\left(\mathbf{B}-\mathbf{A}\right)=\mathbf{0}\Rightarrow$ úsečka je rovnoběžná s hranicí

$\mathbf{N}\cdot\left(\mathbf{B}-\mathbf{A}\right)<\mathbf{0}\Rightarrow$ úsečka míří dovnitř okna

$\mathbf{N}\cdot\left(\mathbf{B}-\mathbf{A}\right)>\mathbf{0}\Rightarrow$ úsečka míří ven z okna

Cyrus-Beck

❶ $\mathbf{t_{min} := 0.0}$; $\mathbf{t_{max} := 1.0}$

❷ dokud $\mathbf{t_{min} < t_{max}}$, opakuji pro každou hraniční přímku kroky ❸ a ❹ [pak spočtu $\mathbf{P(t_{min})}$ a $\mathbf{P(t_{max})}$]

❸ je-li $\mathbf{N \cdot (B - A) = 0}$, **celá úsečka** leží v jedné polorovině (rozhodnu podle znaménka $\mathbf{N \cdot (A - P_E)}$)

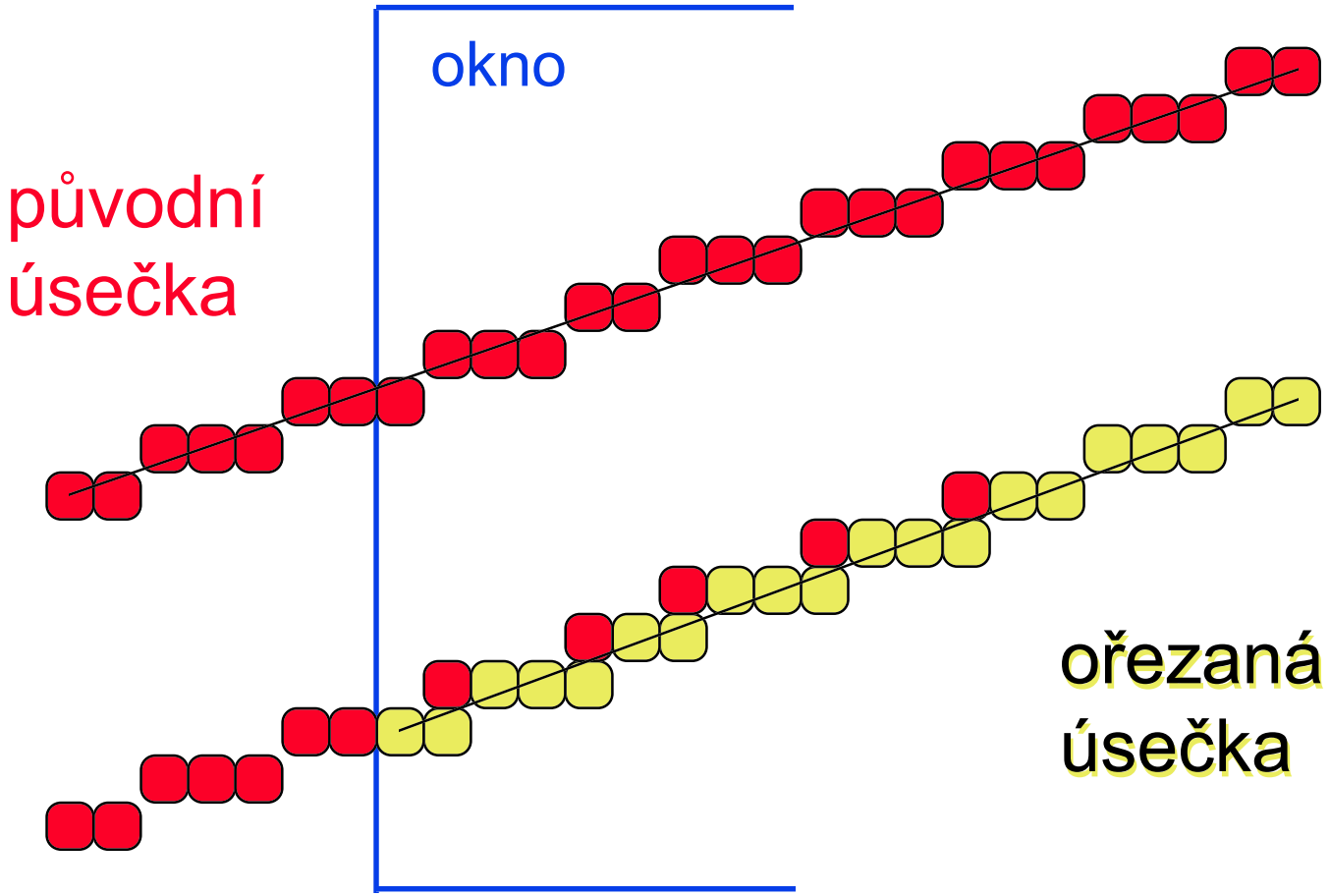
❹ jinak spočítám $\mathbf{t_0}$ a podle znaménka $\mathbf{N \cdot (B - A)}$ opravím $\mathbf{t_{min}}$ nebo $\mathbf{t_{max}}$:

$\mathbf{t_{min} := max\{ t_{min}, t_0 \}}$ nebo $\mathbf{t_{max} := min\{ t_{max}, t_0 \}}$

Liang-Barsky

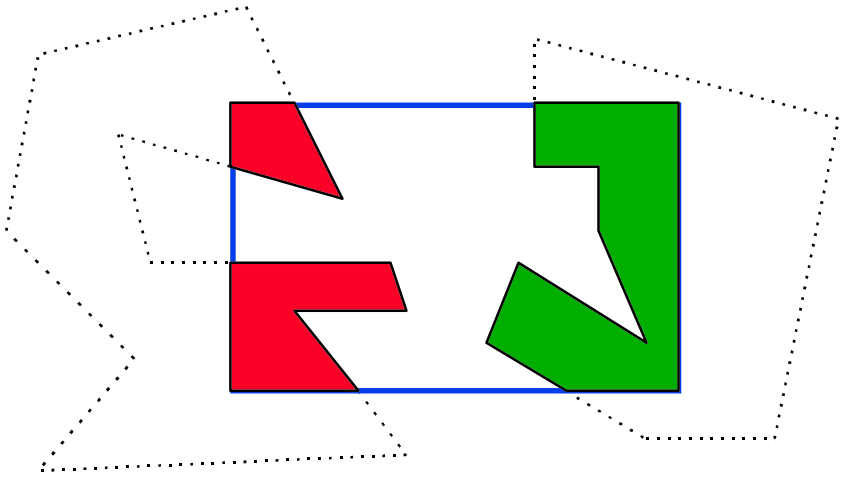
- efektivní úprava parametrického algoritmu pro **obdélníkové okno**
  - původní algoritmus Cyrus-Beck počítá s obecným konvexním n-úhelníkem
  - normálové vektory jsou triviální ([1,0], [0,-1], ...)
  - maximálně: 8 add/sub , 14 cmp , 4 div, 4 mul
- **Cohen-Sutherland**: většina případů je triviálních
- **Liang-Barsky**: většina úseček se ořezává
  - čím více se ořezává, tím je výhodnější než C-S

Zaokrouhlovací chyby

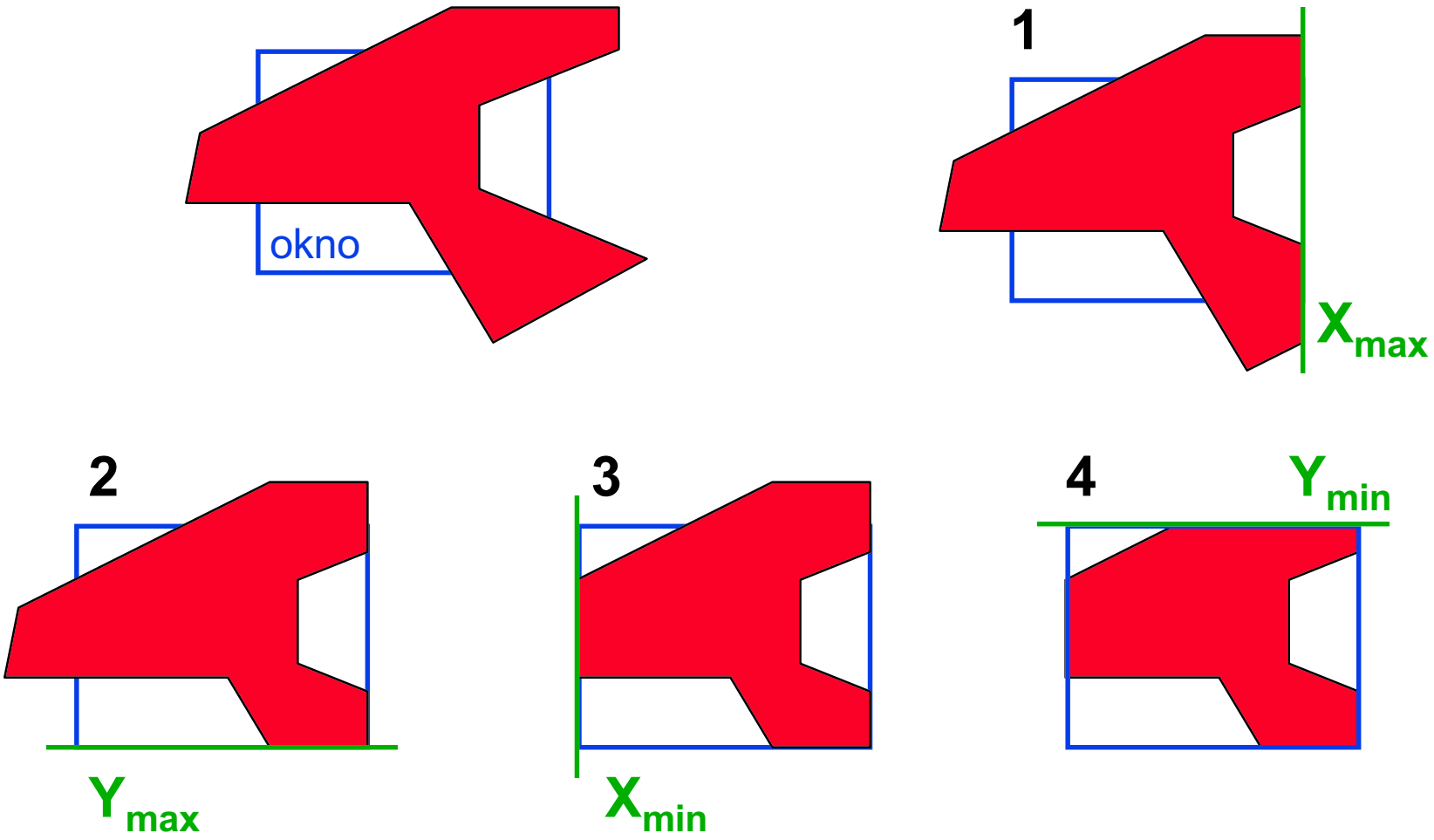


Ořezávání n-úhelníků

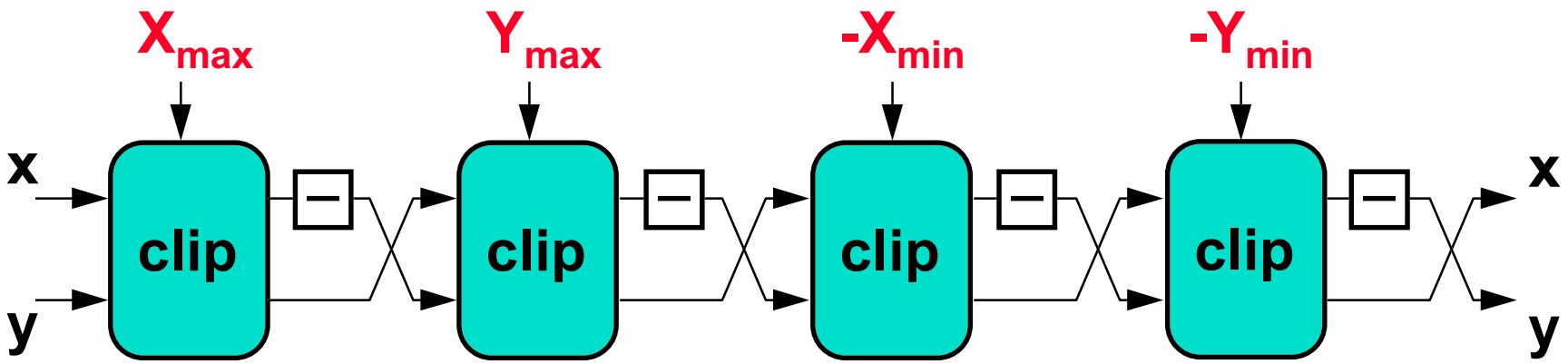
- jestliže chceme kreslit pouze **obrys n-úhelníku**, pak stačí ořezávat hrany samostatně jako úsečky
- chceme-li **vybarvovat vnitřek n-úhelníku**, musíme ho oříznout speciálním algoritmem:



Proudové ořezávání
(Sutherland-Hodgman)



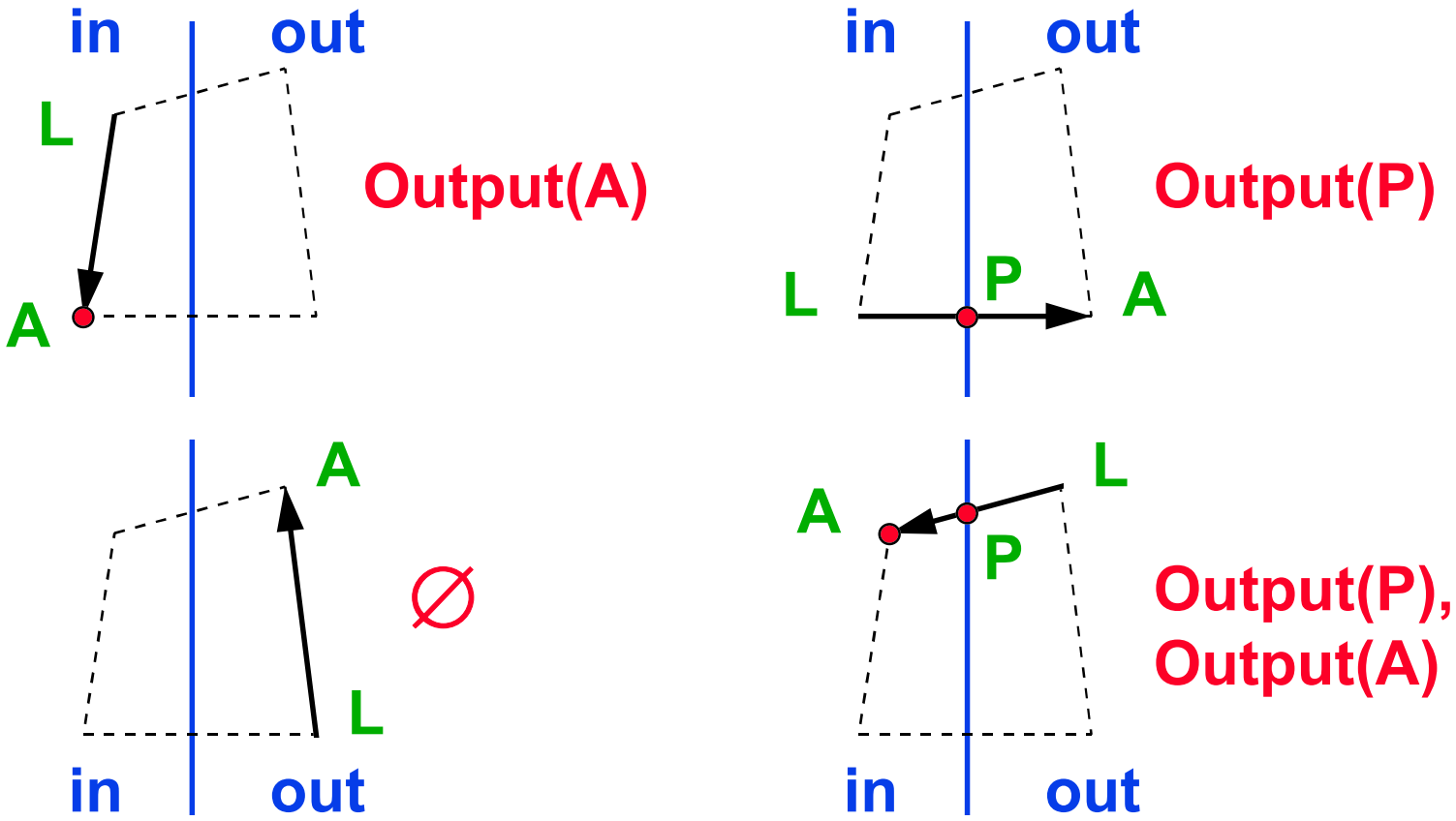
Sutherland-Hodgman (HW)



- modul **“clip”** ořízne n-úhelník podle hranice **$x=X_{max}$**
- mezi jednotlivými moduly se souřadnice **otáčejí o 90°**

Modul “**clip**”:

- pamatuje si poslední dva vrcholy **L** a **A**



Konec

Další informace:

- **J. Foley, A. van Dam, S. Feiner, J. Hughes**: ***Computer Graphics, Principles and Practice***, 111-127
- **Jiří Žára a kol.**: ***Počítačová grafika***, principy a algoritmy, 153-168
- **LAN** na Malé Straně:
  - **barbora\usr:\vyuka\pelikan\3**

Monochromatické zobrazování

KSVI MFF UK Praha

e-mail: Josef.Pelikan@mff.cuni.cz

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

Vnímání šedých odstínů

- šedý odstín má jediný **atribut**
  - **intenzita** (fyzikální smysl, vyzařovaná energie)
  - **jas** (subjektivní vjem člověka)

- vztah mezi intenzitou a jasem **není lineární**
  - člověk vnímá intenzity **relativně**
  - pro rovnoměrně odstupňované jasové odstíny je třeba použít **logaritmickou stupnici** intenzit
  - minimální zobrazitelná intenzita: $\mathbf{I_0 = 0.005 \div 0.025}$, ostatní intenzity $\mathbf{I_j = I_0 * r^j}$ ($\mathbf{r \cong 1.015}$ pro 256 odstínů)

Gamma korekce

- intenzita světla vyzařovaného stínítkem monitoru **nezávisí lineárně** na hodnotě napětí přiváděného do monitoru
  - $\mathbf{I = K * V^{\gamma}}$, kde **I** je **intenzita světla**, **V hodnota pixelu** a **K**, γ konstanty závislé na **typu monitoru** (exponent γ má typickou hodnotu **2.2 ÷ 2.5**)
  - požadovaná hodnota pixelu $\mathbf{V_j = (I_j/K)^{1/\gamma}}$

➡ "**gamma-korekce**" se často provádí už při digitalizaci obrázku (přepočet jen při změně monitoru)

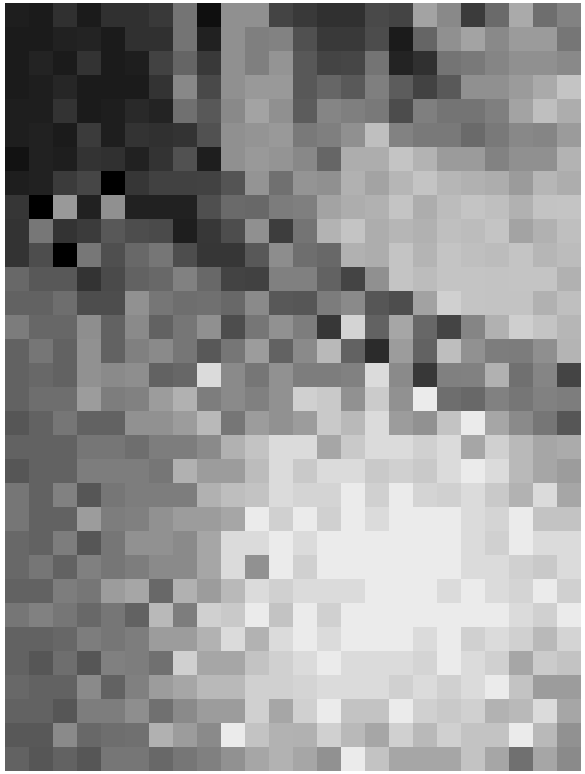
Počet odstínů šedi

- počet potřebných zobrazovacích odstínů **n** závisí na dynamickém rozsahu výstupního zařízení (předpokládáme **r = 1.01**):

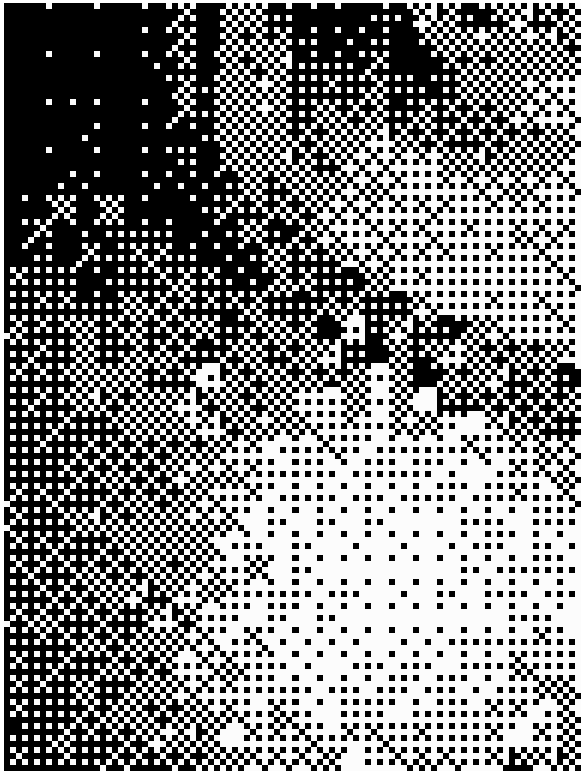
zařízení	**dynamika ($1/I_0$)**	**n**
displej	50-200	400-530
fotografie	100	465
diapozitiv	1000	700
černobílý tisk	100	465
barevný tisk	50	400

- na **displeji** většinou postačí **64 ÷ 256** úrovní šedi

Půltónování a rozptylování



odstíny šedi



černobílé výstupní zařízení

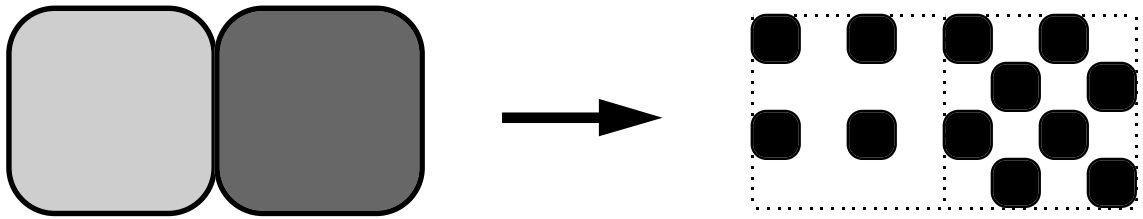
Půltónování a rozptylování

- napodobení vjemu šedých (barevných) odstínů na zařízení s **malým barevným rozlišením**
  - zvětšuji barevné rozlišení na úkor prostorového
  - typické použití: **černobílé tiskárny** nebo displeje
- **půltónování** (“halftoning”): na výstupu mohu zvětšit rastrové rozlišení obrázku (1 : N)
- **rozptylování** (“dithering”): musím zobrazovat bez zvětšování (1 : 1)

Půltónování

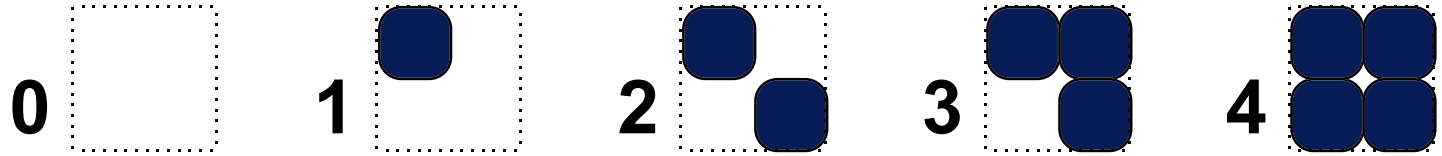
- situace: výstupní zařízení umí zobrazovat pouze **černé body (1)** na **bílém pozadí (0)**
- jeden vstupní pixel (s rozsahem hodnot **0 ÷ N^2**) nakreslím jako **čtverec N×N pixelů** na výstupu
  - výsledný vjem šedého odstínu závisí na počtu černých bodů v rastru **N×N**

odstíny číslo 4 a 8 (ze škály 0÷16)

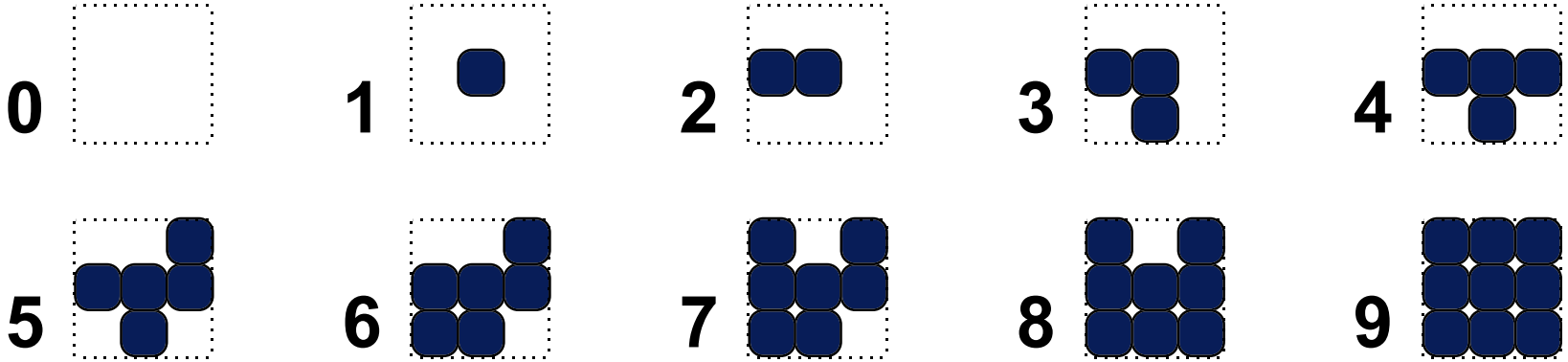


Půltónovací rastry

pravidelný rastr 2×2



rastr 3×3



Inkrementální rastry

- půltónovací rastr je **inkrementální**, jestliže:
  - vzorek odstínu **k** obsahuje právě **k** černých pixelů
  - dva sousední vzorky (**k** a **k+1**) se mezi sebou liší právě v jednom pixelu (**k+1** má o jeden černý pixel více)

- inkrementální rastr lze uložit do **matice** velikosti **N×N** obsahující celá čísla **0 ÷ N^2-1**

  - např. $\mathbf{M} = \begin{matrix} \mathbf{6} & \mathbf{8} & \mathbf{4} \\ \mathbf{1} & \mathbf{0} & \mathbf{3} \\ \mathbf{5} & \mathbf{2} & \mathbf{7} \end{matrix}$

Pravidelný rastr

I) velikost 2×2: $\mathbf{M}^{(2)} = \begin{bmatrix} 0 & 2 \\ 3 & 1 \end{bmatrix}$

II) přechod $\mathbf{N \times N \rightarrow 2N \times 2N}$**:**

$$\mathbf{M}^{(2N)} = \begin{bmatrix} 4\mathbf{M}^{(N)} & 4\mathbf{M}^{(N)} + 2\mathbf{J}^{(N)} \\ 4\mathbf{M}^{(N)} + 3\mathbf{J}^{(N)} & 4\mathbf{M}^{(N)} + \mathbf{J}^{(N)} \end{bmatrix}$$

matice $\mathbf{J}^{(N)}$ je typu $\mathbf{N \times N}$ a obsahuje samé jedničky

Pravidelný rastr

$$\mathbf{M}^{(4)} = \begin{bmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{bmatrix}$$

- body pravidelných vzorků jsou vždy rozmístěny **rovnoměrně**

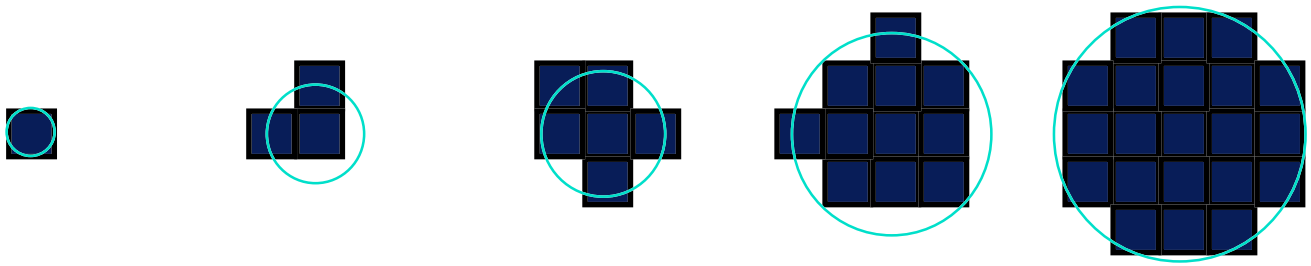
- **pravidelný rastr** je vhodný pro **obrazovku** a některé tiskárny (jehličkové s malým rozlišením)

Pravidelný rastr na tiskárně

odstín 8 na obrazovce a tiskárně s velkým rozlišením

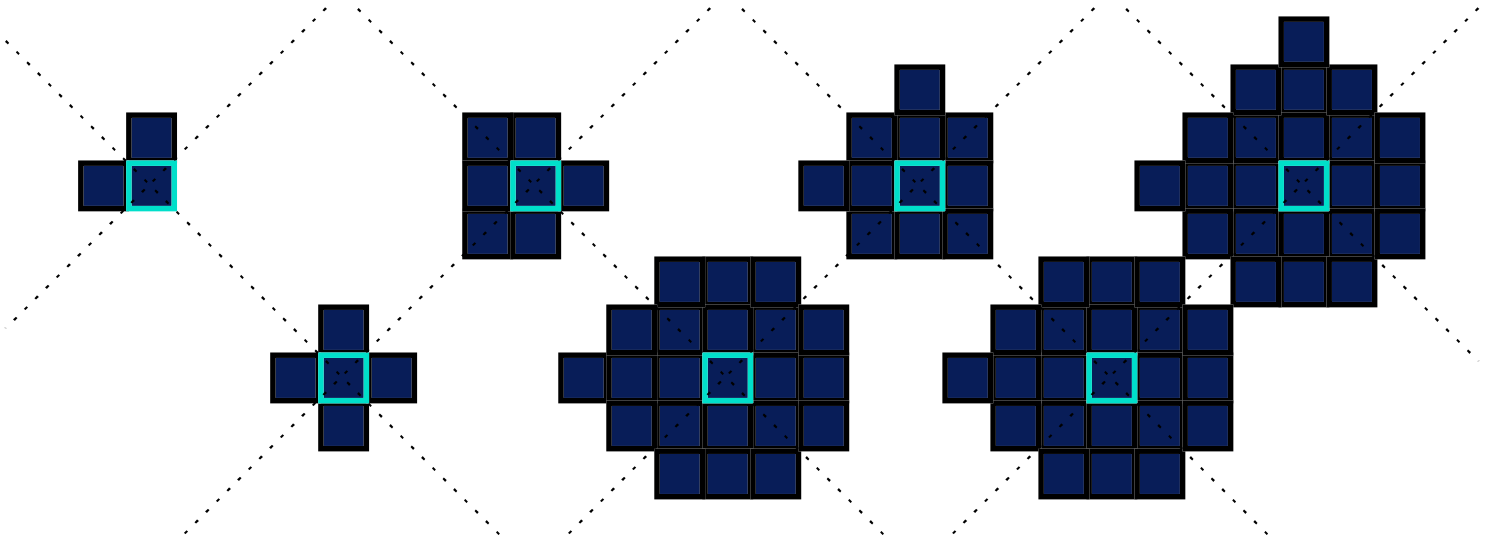
- u vyšších odstínů se sousední kapičky barvy **slévají**
- nižší odstíny obsahují **samostatné tečky**, které se špatně udrží na papíře

Tečkový rastr (“screen”)



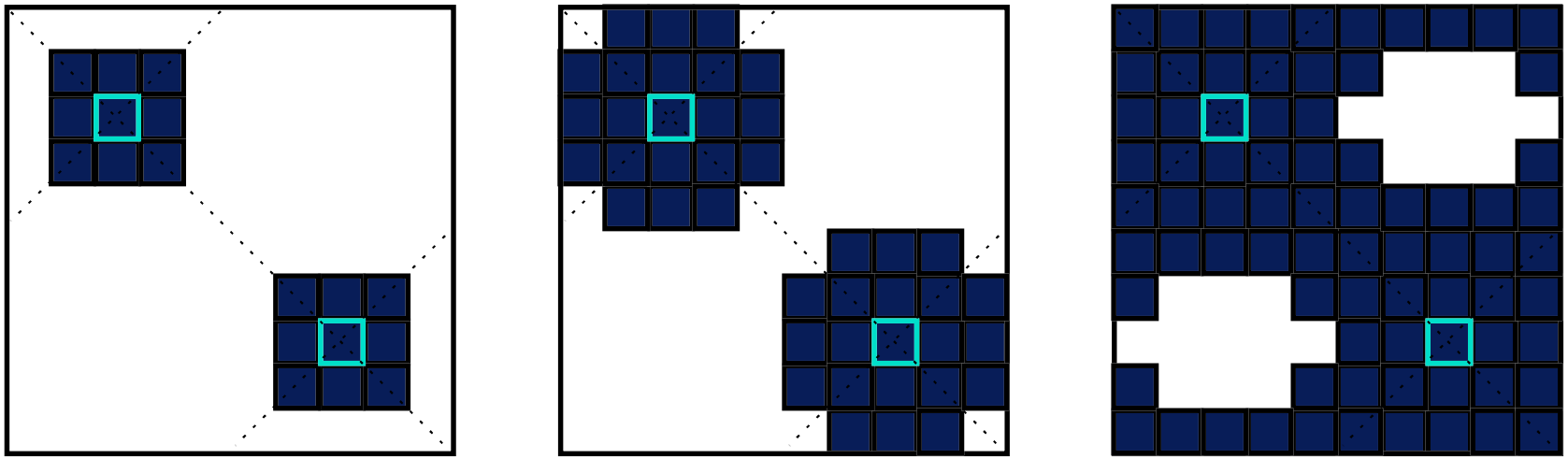
- jednotlivé vzorky jsou tvořeny **tečkami** různých poloměrů
  - netisknou se samostatné kapičky (až na odstín č.1)
  - rozpíjení kapiček způsobí pouze malou změnu poloměru teček

Tečkový rastr - otáčení



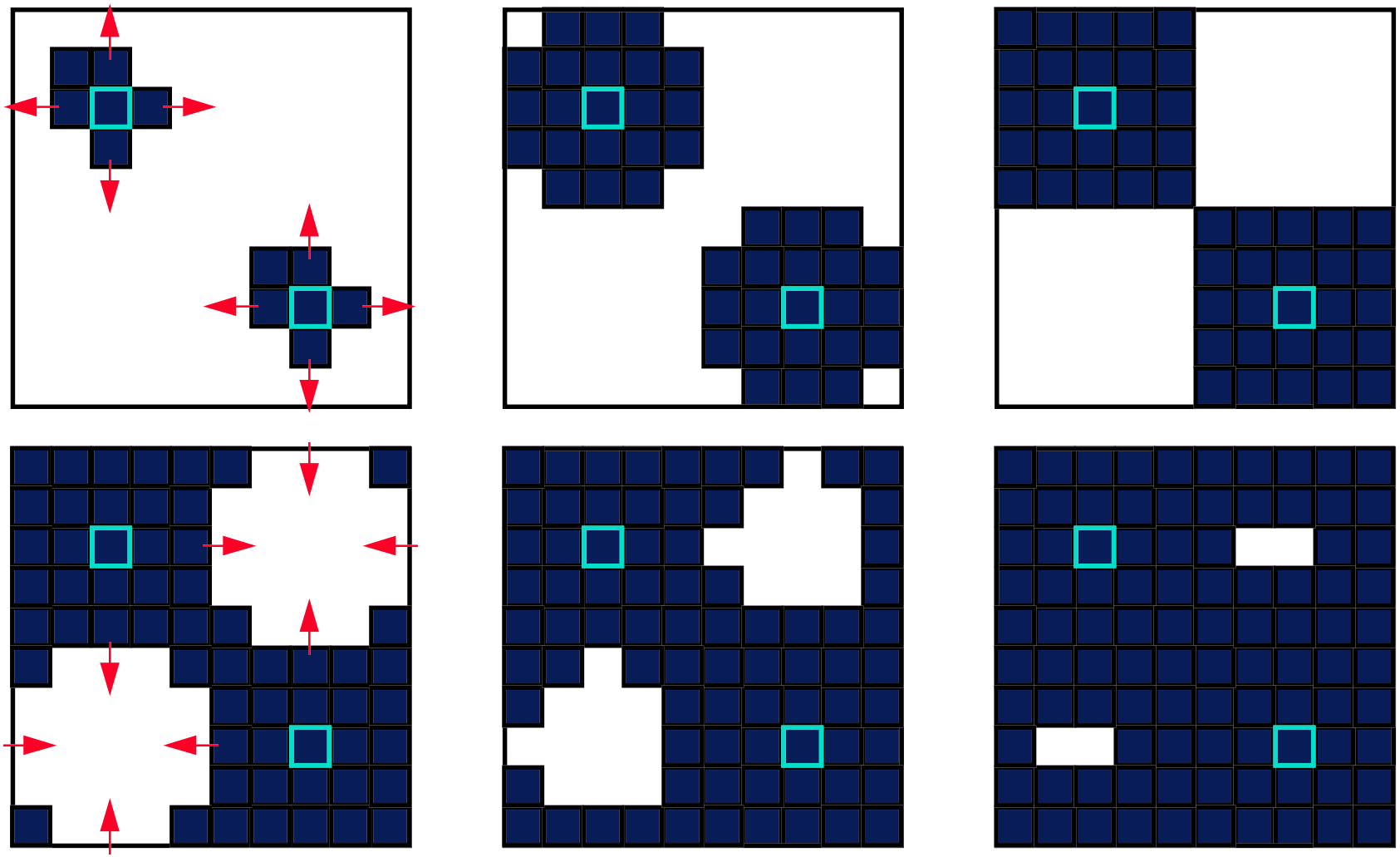
- tečkový rastr se často **otáčí** (o **45°**, 15°, 75°,..)
  - eliminují se svislé a vodorovné linie (nejzřetelnější pro lidské oko)
  - pro racionální směrnice lze tečky ukládat v matici

Varianty tečkového rastru



- **čtvercové tečky** (problémy při jemných přechodech odstínů - “vignettes”)
- **kruhové tečky** (plus různé modifikace)

Konstrukce tečkového rastru

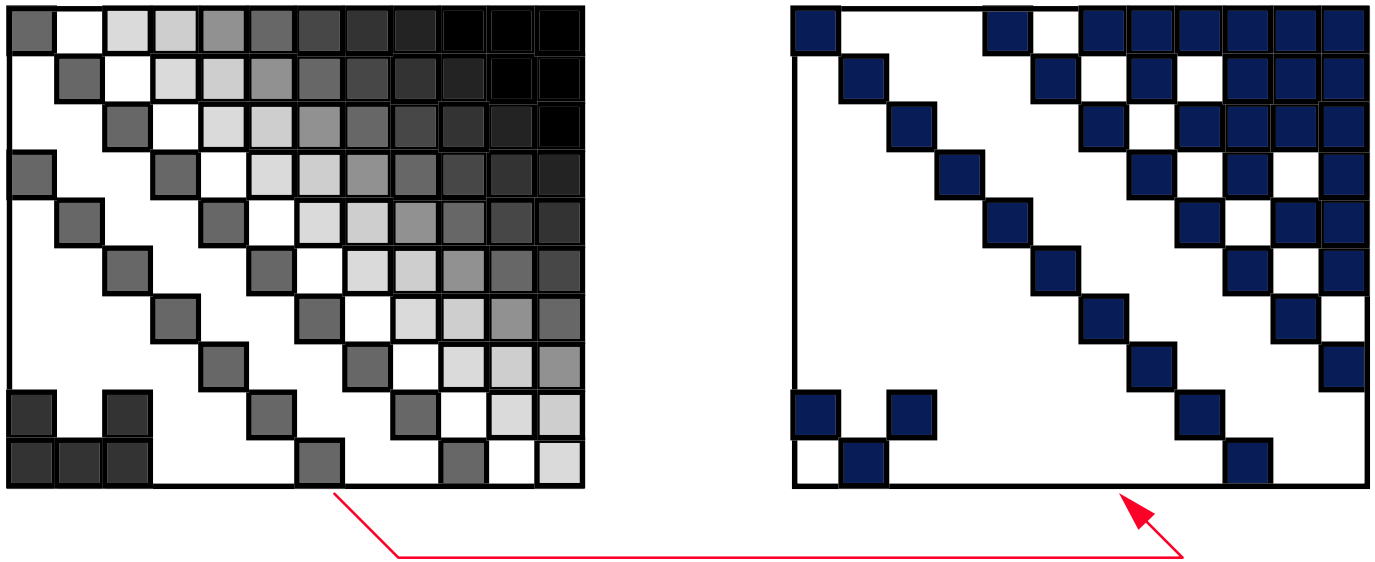


Maticové rozptylování

- zobrazuje se v **měřítku 1:1** (jeden vstupní pixel na jeden výstupní pixel)
- lze použít libovolnou **půltónovací matici**
  – nejčastěji se používá matice pravidelného rastru
- ➡ několik sousedních pixelů sdílí jednu matici:

```
procedure MatrixDither ( x, y, color : integer );
begin
  if M[ y mod N, x mod N ] < color
 then PutPixel(x,y,1)
 else PutPixel(x,y,0);
end;
```

Maticové rozptylování



- **drobné detaily** (čáry) bývají velmi zkreslené
- při použití **neinkrementálního rastru** by mohly být zvýrazněny hranice mezi sousedními odstíny

Náhodné rozptylování

- šum a nahodilost jsou pro lidské oko přirozenější než pravidelný rastr

➡ velmi jednoduchá implementace:

```
procedure RandomDither ( x, y, color : integer );
begin
  if Random(MaxGray) < color
 then PutPixel(x,y,1)
 else PutPixel(x,y,0);
end;
```

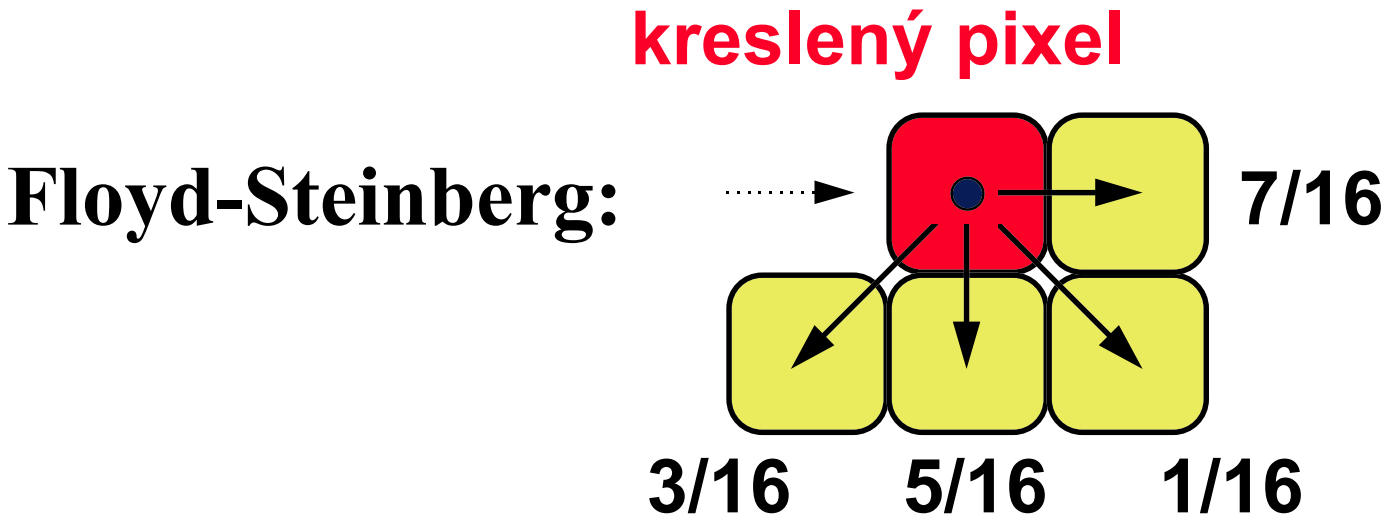
- u **Č/B obrázků** je výstup příliš zašuměný
  – lepší výsledky dává při **více výstupních odstínech**

Metody distribuce chyby

- intenzitu kresleného pixelu **zaokrouhlím** na nejbližší zobrazitelnou hodnotu a nakreslím ji:
  - **0/1** pro Č/B výstupní zařízení
  - **0, 1, .. K** pro víceúrovňové zařízení

- **rozdíl mezi požadovanou a skutečně zobrazenou intenzitou** rozdělím ve vhodném poměru do sousedních pixelů
  - je zachován lokální poměr počtu černých a bílých pixelů (odpovídající vstupnímu odstínu)
  - chyba se předává jen do dosud nenakreslených pixelů

Metoda distribuce chyby



- kreslení musí postupovat **po řádkách**
  – řádky lze procházet střídavě zleva a zprava
- pro **akumulaci chyb** na následující řádce je nutné použít **pomocný buffer**

Jiné distribuční filtry

F. Sierra:

	●	1/2	
1/4	1/4	0	

J. Jarvis,
C. Judice,
W. Ninke:

		●	7	5	
3	5	7	5	3	/ 48
1	3	5	3	1	

Stucki:

		●	8	4	
2	4	8	4	2	/ 42
1	2	4	2	1	

Metody distribuce chyby

- **vysoká kvalita** výstupu na monitoru
  - vzorek je nepravidelný a příjemný pro lidské oko
- **nevýhody:**
  - nutnost kreslit výstup **po řádkách**
  - není možné se **vracet zpět** (proto se nepoužívá ve vyplňovacích rutinách grafických knihoven)
  - je potřeba **pomocný buffer** minimálně na 1 řádku
  - větší **časová náročnost**

Více výstupních odstínů

- na výstupu předpokládáme **K+1 odstínů**
  - **0 ÷ K** (**0** .. bílá, **K** .. černá)
- naše rozptylovací metoda umí zpracovat **M+1 vstupních odstínů** do dvou výstupních barev:
  - vstup: **0 ÷ M**
  - výstup: **0 / 1**
- na vstupu kombinované metody může být **K*M+1** odstínů

Více výstupních odstínů

```
function Dither ( x, y, color : integer ) : integer;
  { vstupní odstín: 0 až M, vrací 0 nebo 1 }
...


procedure MultiDither ( x, y, color : integer );
  { vstupní odstín: 0 až K*M, kreslený odstín: 0 až K }
var base : integer;
begin
  base := color div M;          { 0 <= base <= K }
  PutPixel(x,y, base + Dither(x,y,color mod M) );
end;
```

Literatura

- **J. Foley, A. van Dam, S. Feiner, J. Hughes**: ***Computer Graphics, Principles and Practice***, 563-573
- **V. Skala:** ***Algoritmy počítačové grafiky III***, skriptum ZČU, 1992, 5-22
- **Jiří Žára a kol.**: ***Počítačová grafika***, principy a algoritmy, 329-335

Konec

Další informace:

- **J. Jarvis, C. Judice, W. Ninke**: ***A Survey of Techniques for the Image Display of Continuous Tone Pictures on Bilevel Displays***, CGIP vol.5, #1, March 1976

- **LAN** na Malé Straně:
  - **barbora\usr:\vyuka\pelikan\3**

Barevné vidění

KSVI MFF UK Praha

e-mail: Josef.Pelikan@mff.cuni.cz

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

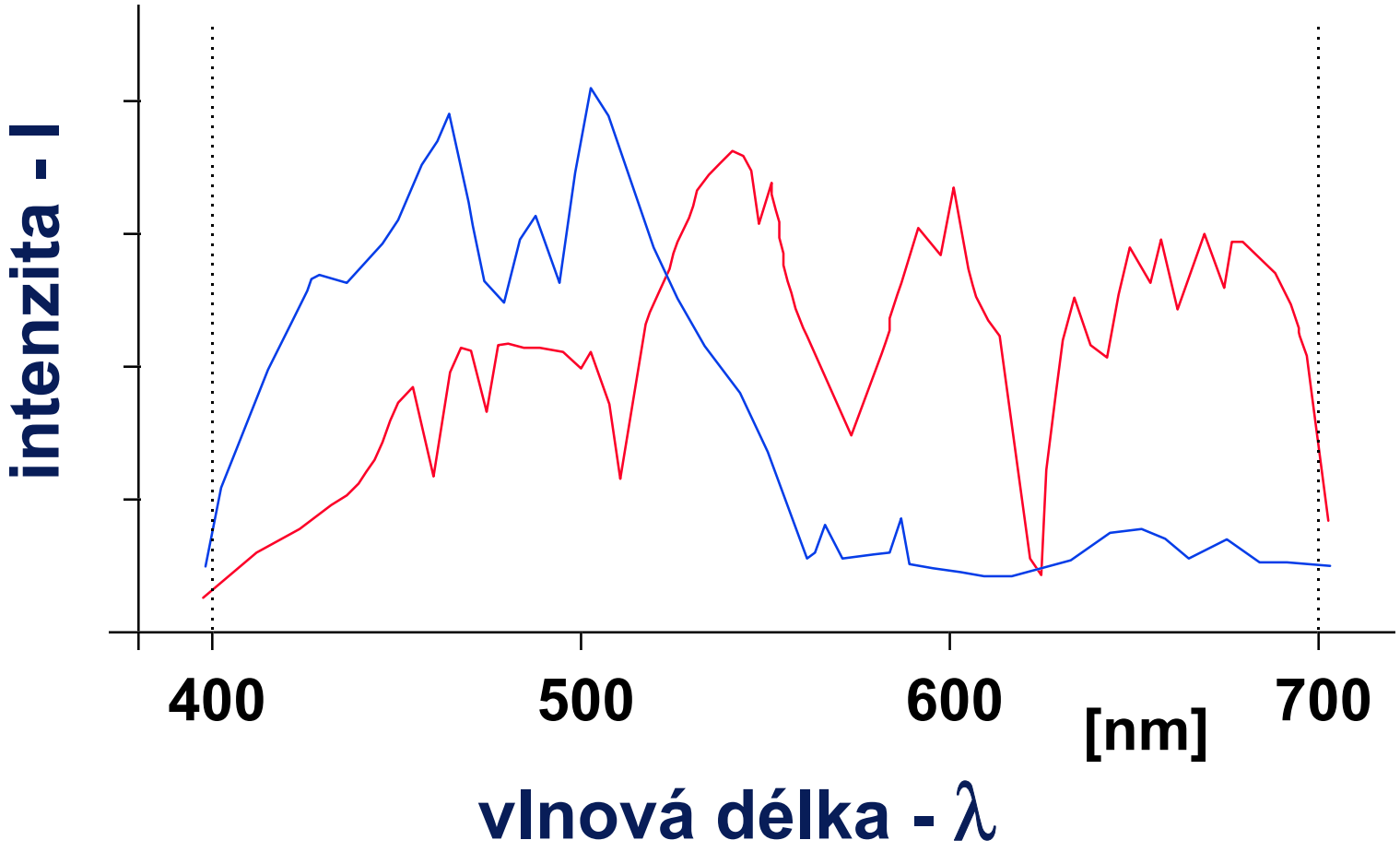
Co je světlo?

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



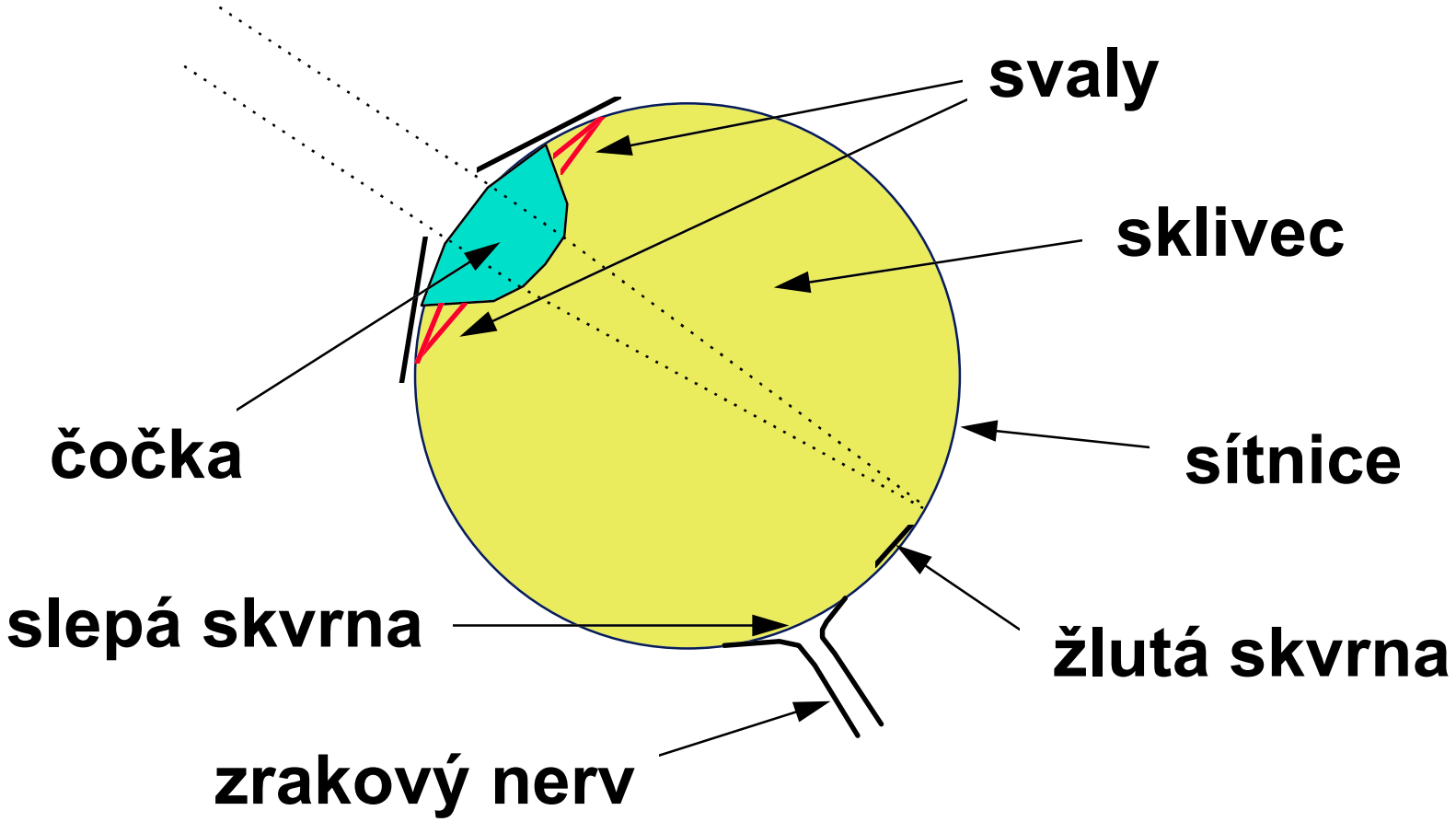
Barevný vjem

- prostor všech spekter má **nekonečnou dimenzi**
  - systém lidského vidění je však nedokáže všechny rozeznat (“metamery”)

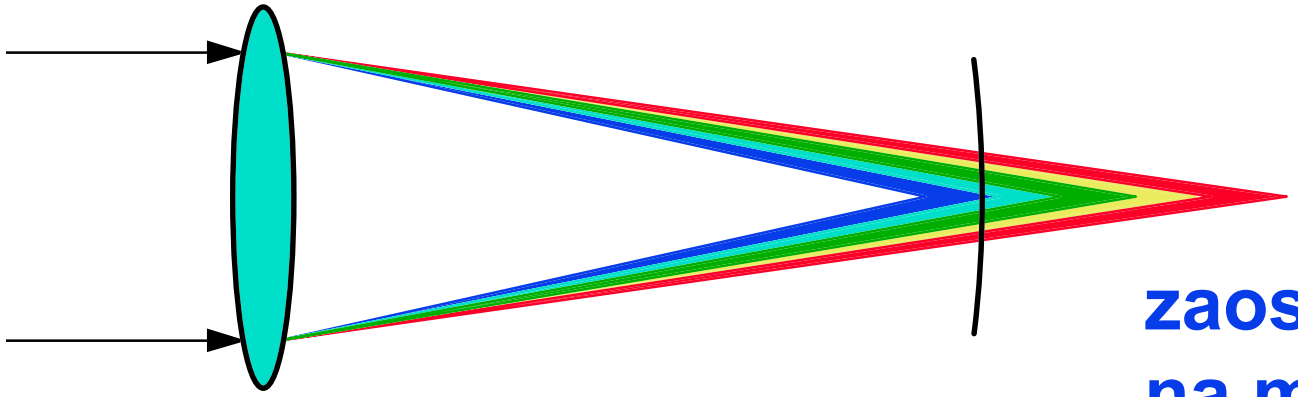
- **Grassmanovy zákony** (1854) - lidské oko vnímá:
  - **dominantní vlnovou délku** (odstín, “hue”)
  - **čistotu barvy** (sytost, “saturation”)
  - **intenzitu** (jas, “brightness”)

  barvy lze aditivně skládat (**A=B**, **C=D** $\Rightarrow$ **A+C=B+D**)

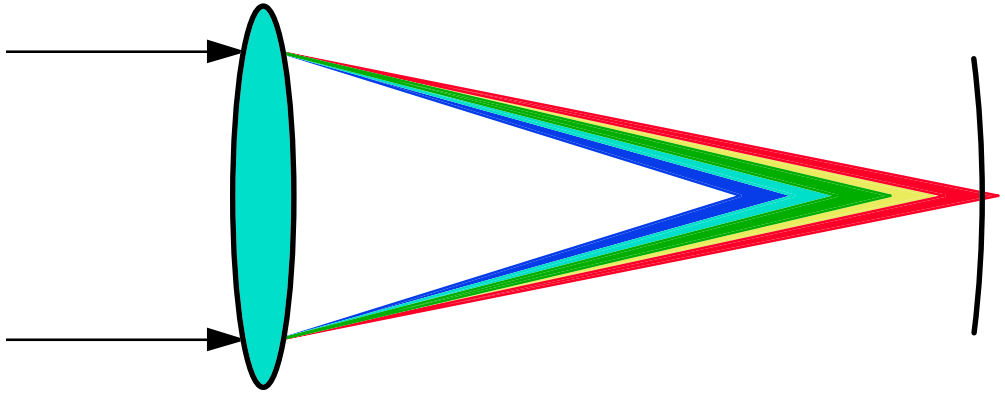
Lidské oko


svaly
sklivec
čočka
sítnice
slepá skvrna
žlutá skvrna
zrakový nerv

Barevná aberace

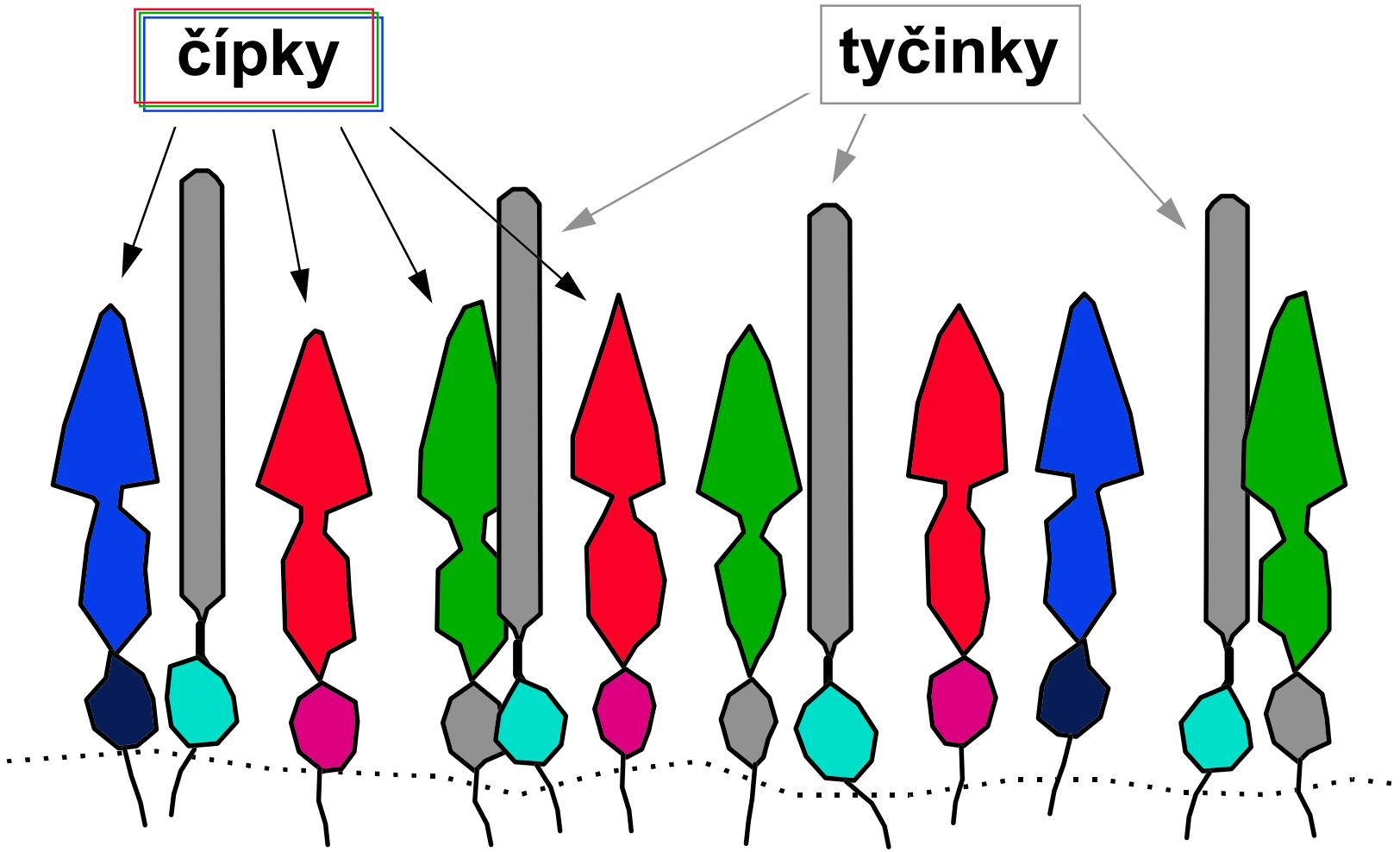


zaostřeno
na modrou

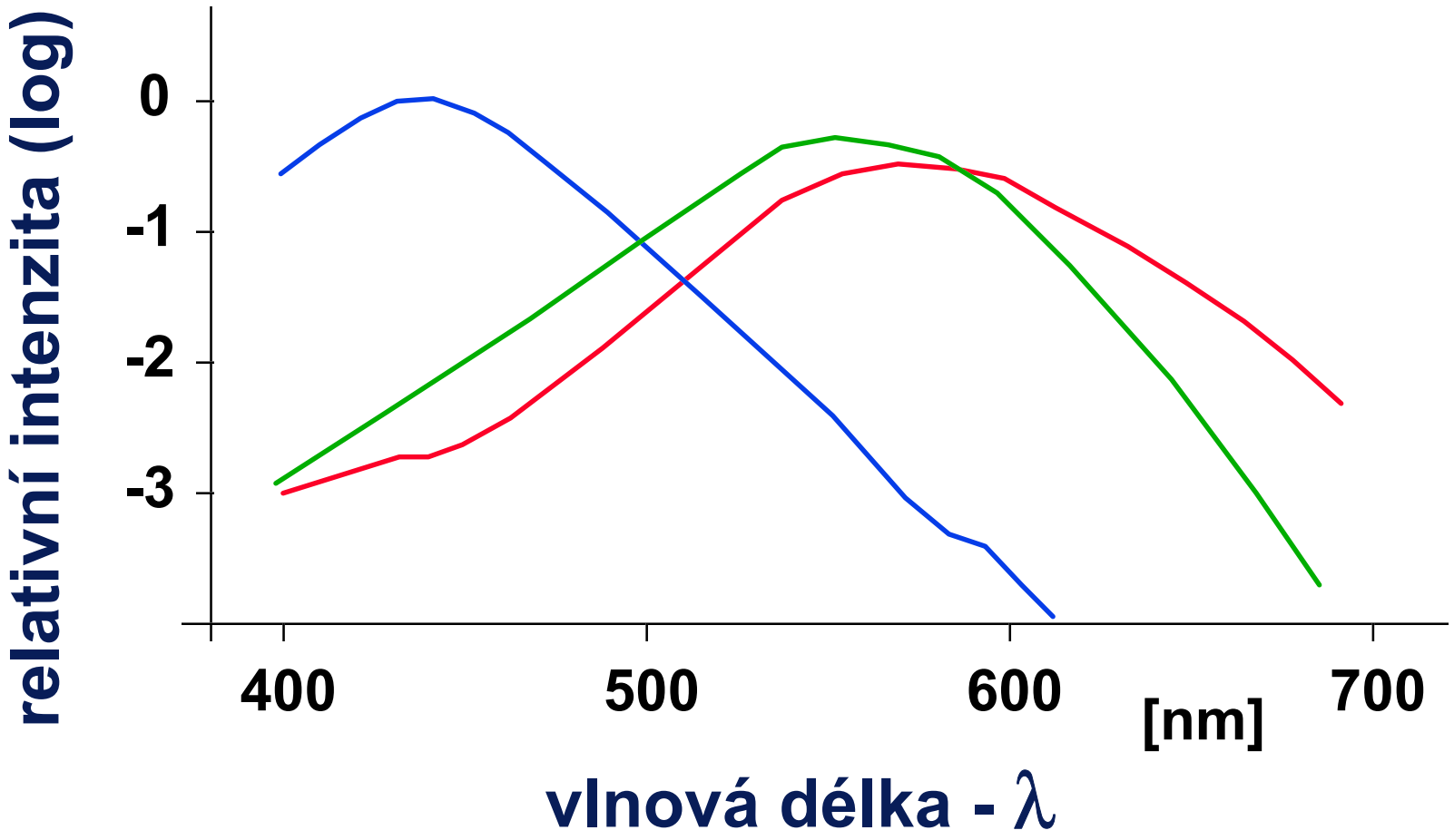


zaostřeno
na červenou

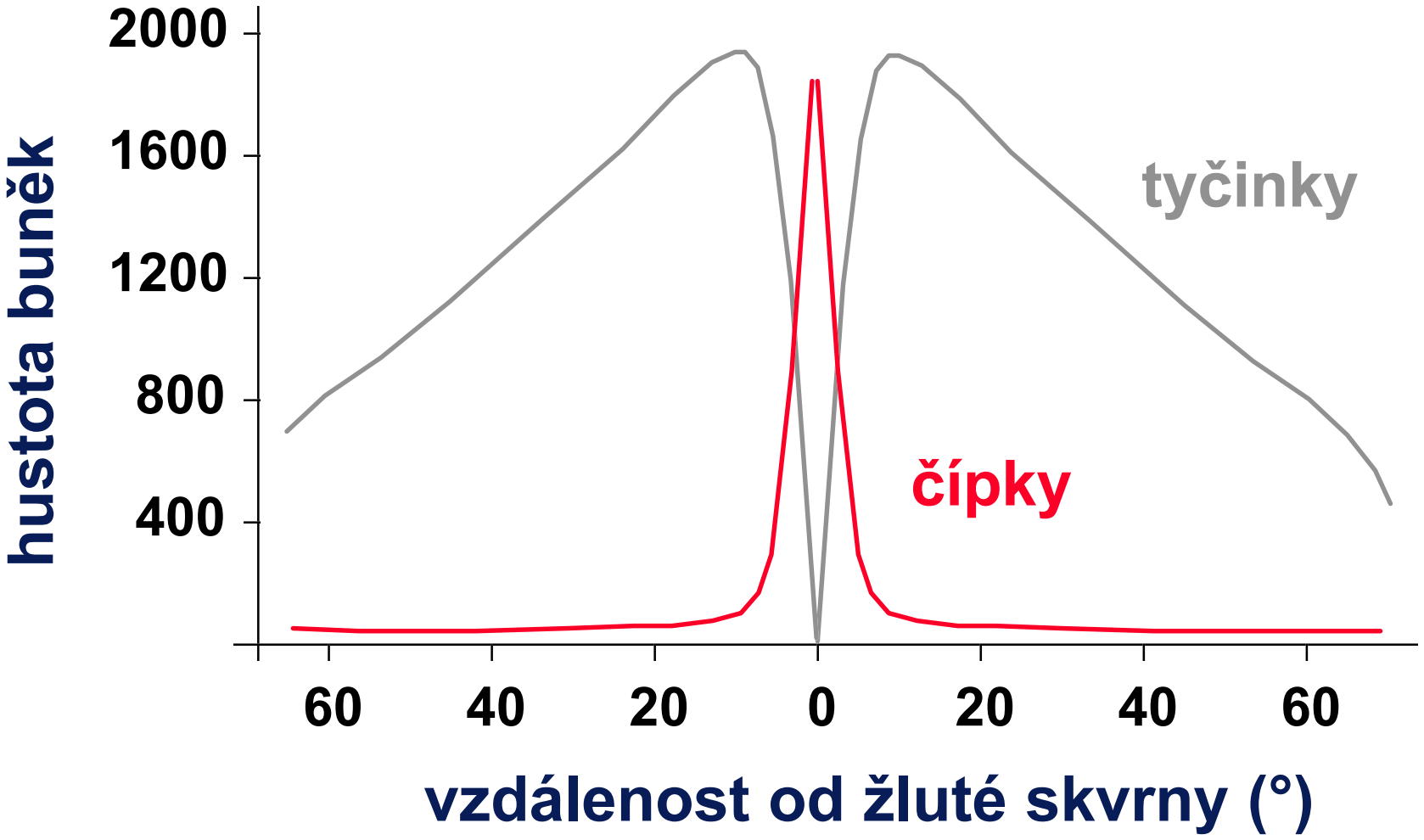
Sítnice



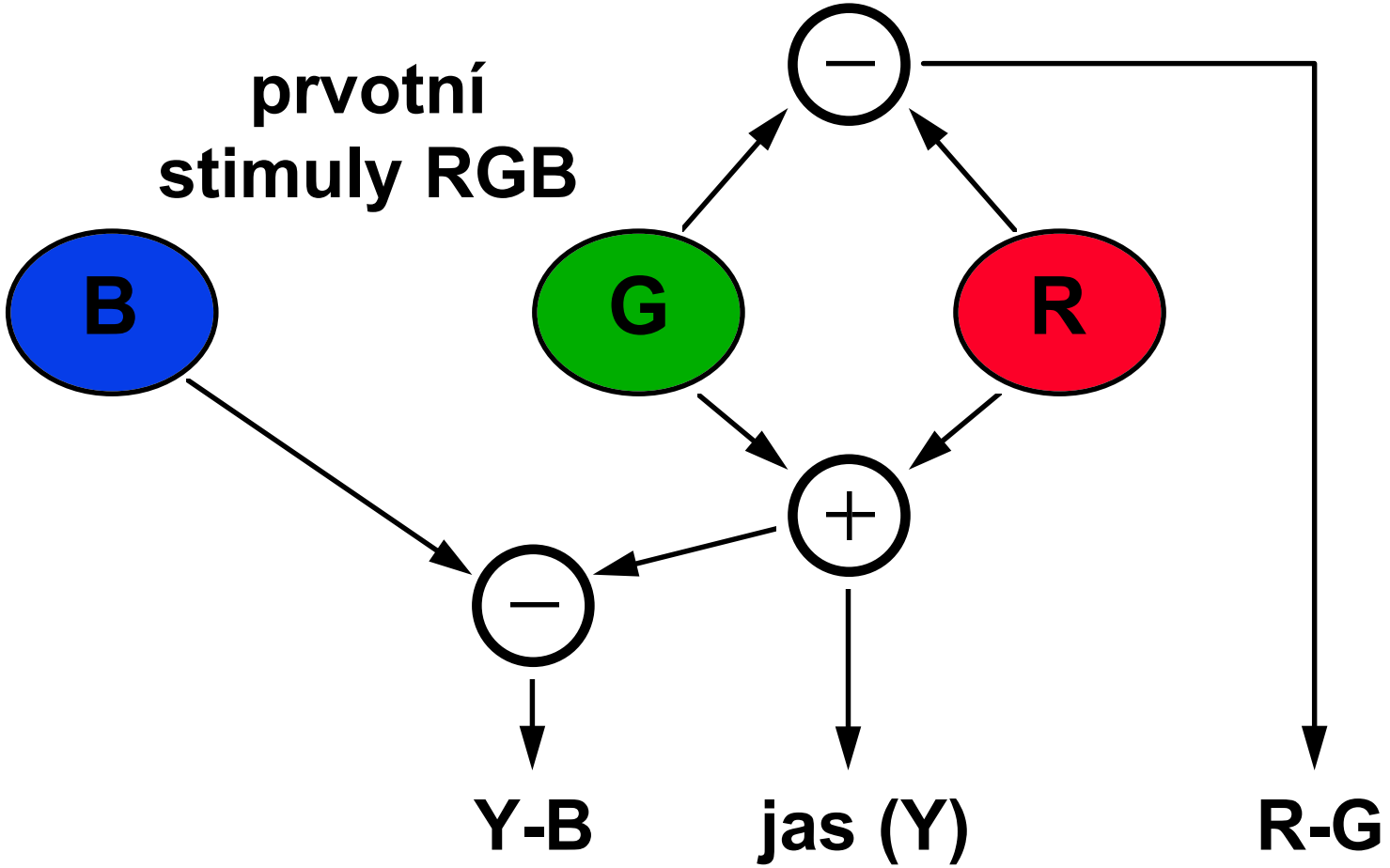
Citlivost tří druhů fotopigmentu



Rozložení fotoreceptorů



Předzpracování barev

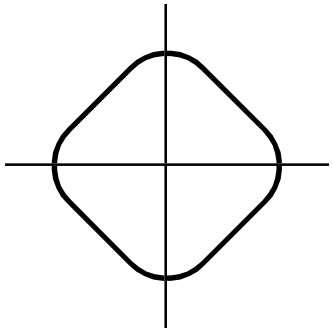


Vlastnosti systému vidění

- 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

Vlastnosti systému vidění

- **větší rozlišovací schopnost** ve svislém a vodorovném směru
  - v šikmých směrech asi o 30% menší



- **přeostřování** na barvy vzdálené ve spektru
- **setrvačnost** (“afterimage”)
  - laterální inhibice nervových buněk
- **očekávání** (“expectation”)
  - psycho-fyziologická vlastnost

Vlastnosti systému vidění

- **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 malé objekty a tenké čáry modře**
  – málo “modrého” pigmentu ve středu žluté skvrny
- **na pozadí nepoužívat červenou a zelenou**
  – modrá i žlutá vyhovují
- **nekreslit vedle sebe syté barvy vzdálené 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

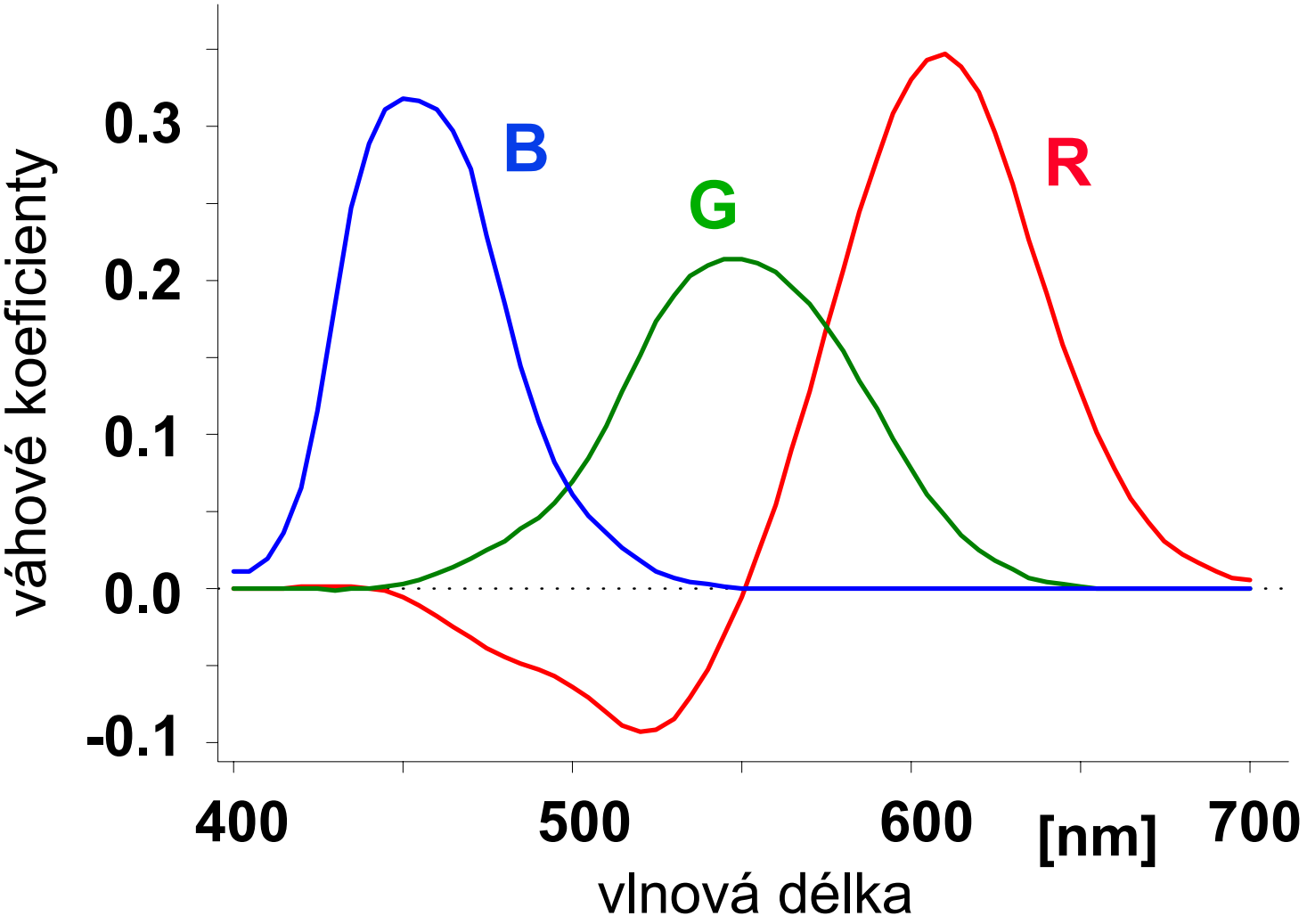
Barevné systémy

KSVI MFF UK Praha

e-mail: Josef.Pelikan@mff.cuni.cz

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

Rozklad spektrálních barev

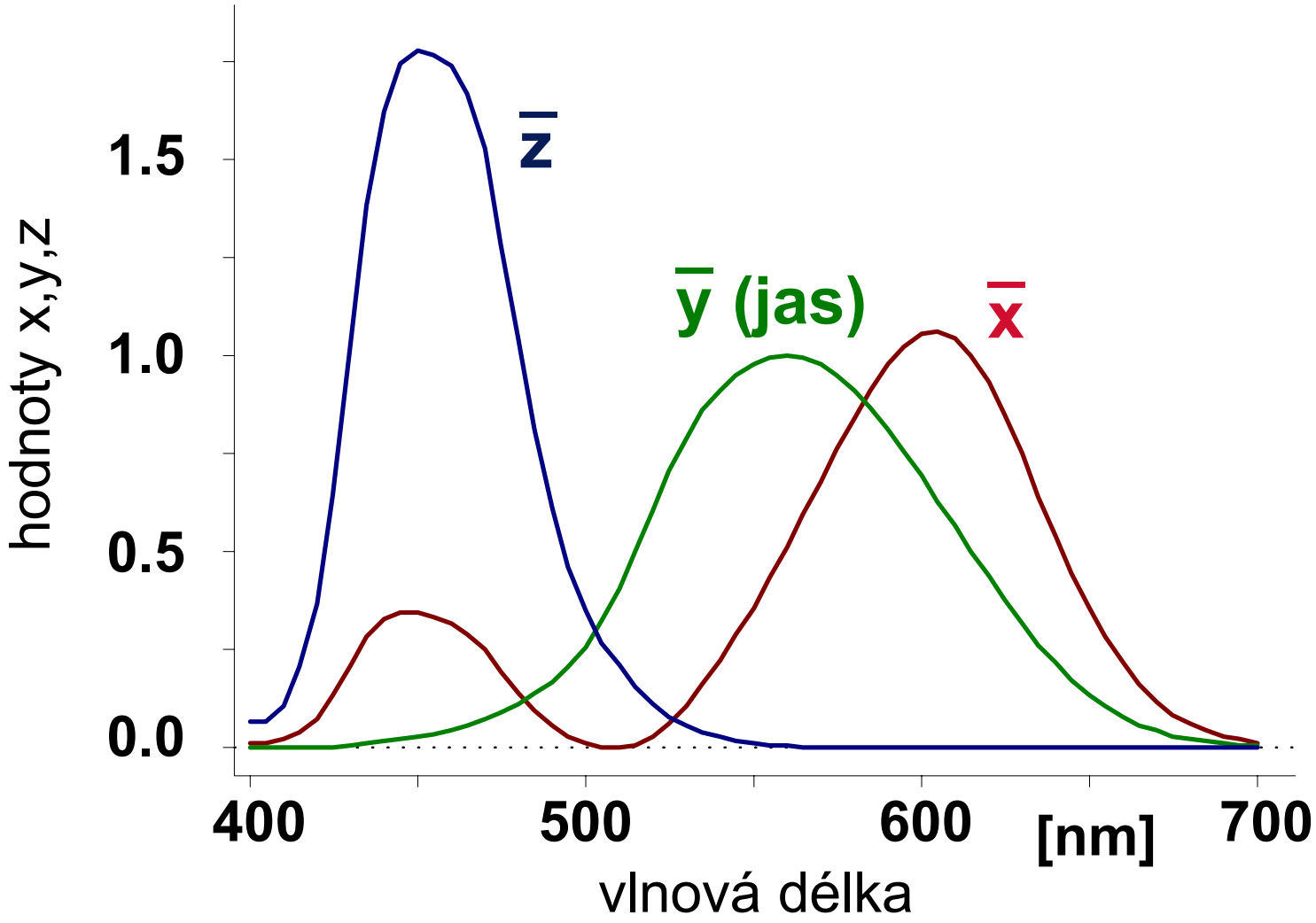


Virtuální barevná primitiva X,Y,Z

- ***Commision Internationale de l' Éclairge*** **(CIE)** v roce 1931 definovala tři virtuální barvy **X, Y, Z**, jejichž **konvexní kombinací** již vytvoříme libovolnou viditelnou barvu
  - X, Y, Z jsou definovány pomocí svých spektrálních charakteristik **x, y, z** (tabelovaných po 1nm)

- závislost mezi složkami R,G,B a X,Y,Z je **lineární**
  - převodní matice 3×3

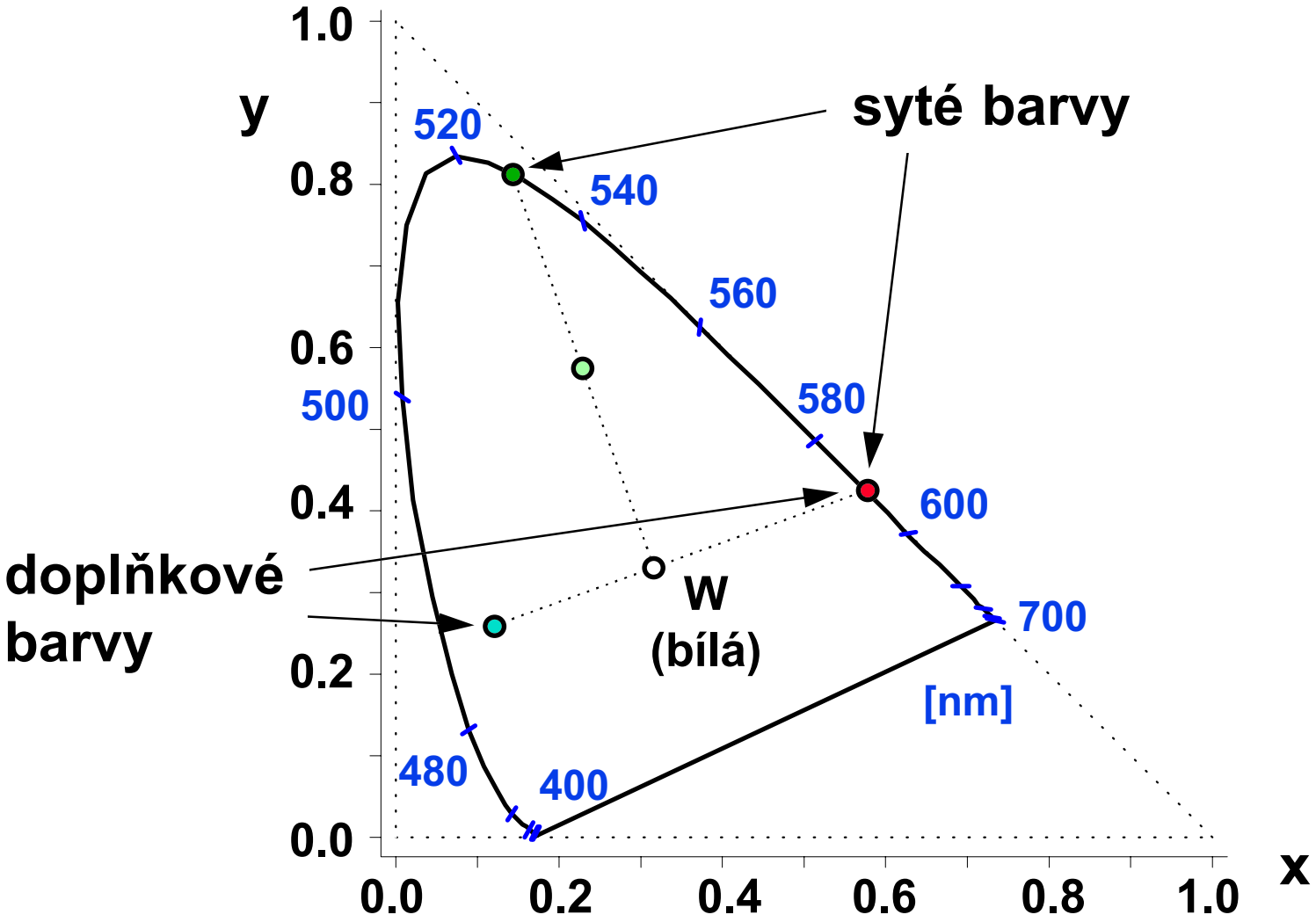
Srovnávací funkce CIE



Barevný prostor CIE-xy

- **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 pouze informace o odstínu a sytosti, jas chybí
- barevný diagram **CIE-xy** nepoužívá složku **z**
  - je závislá na předchozích dvou (**z = 1 - x - y**)
- systém **CIE-xy** nezohledňuje **subjektivní citlivost** na barevné rozdíly (uniformní **CIE-uv**)

Barevný diagram CIE-xy

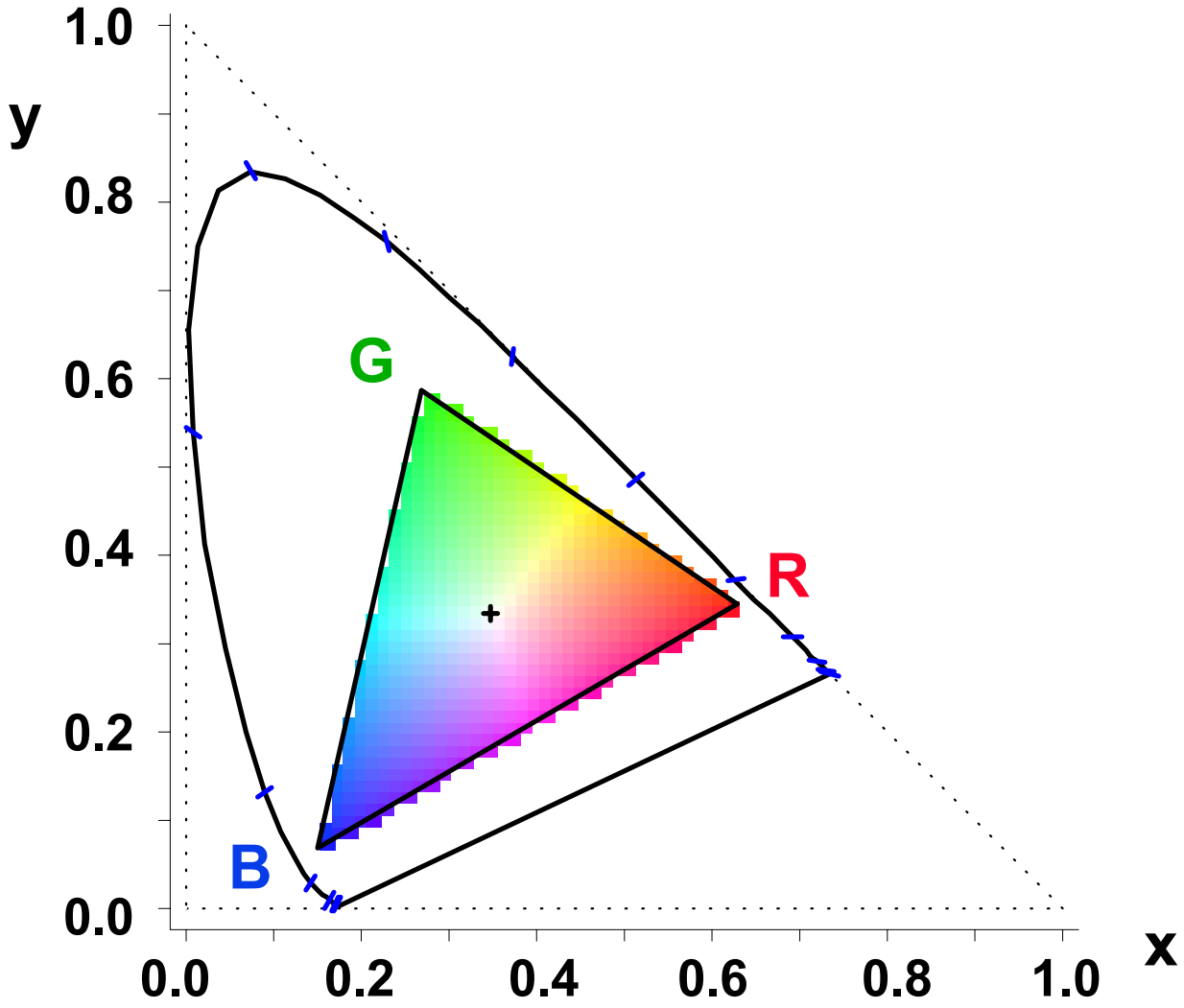


Barevná primitiva RGB

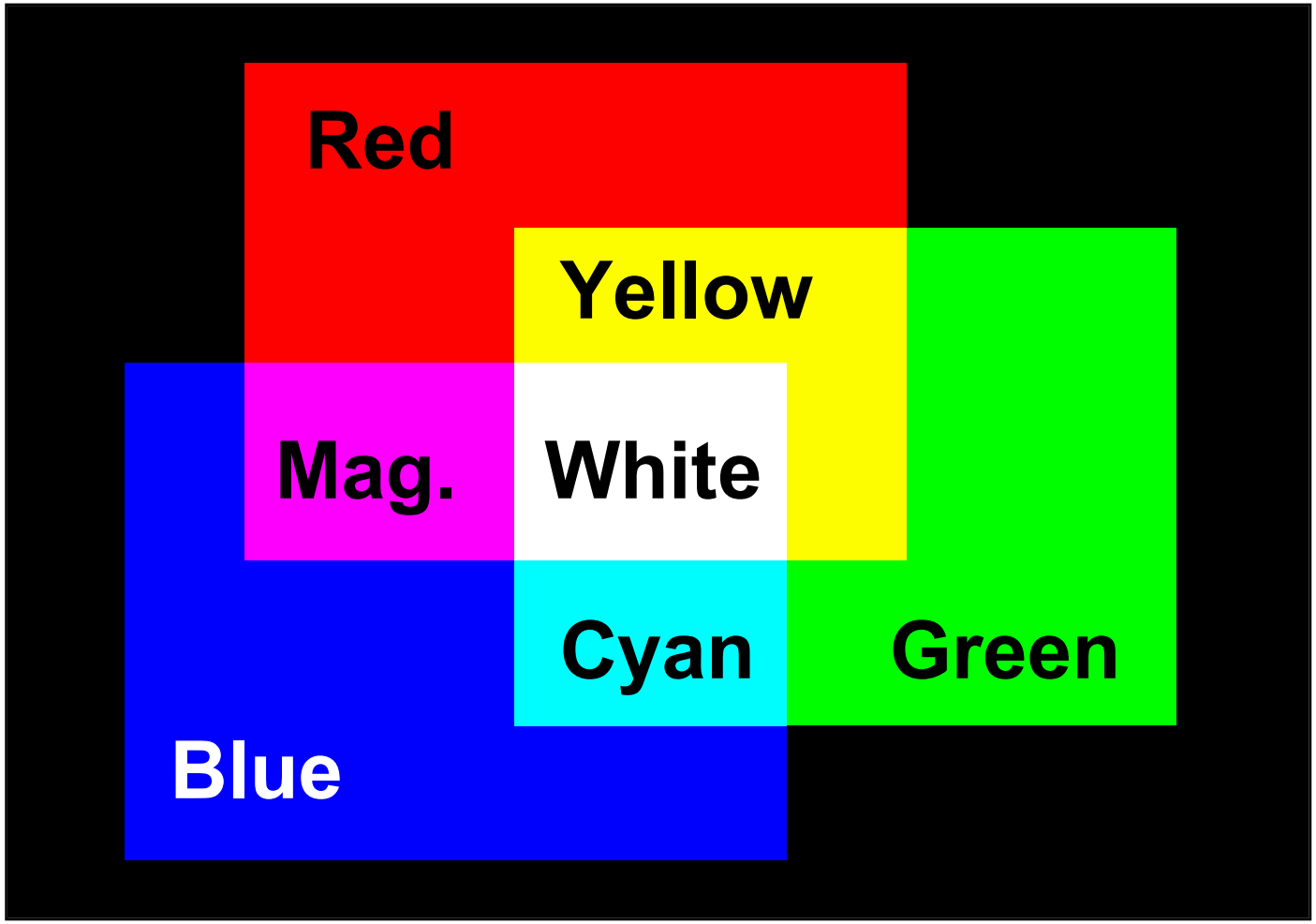
- odpovídají poloze **tří typů barevných luminoforů**:
  - **R = [0.628,0.346], G = [0.268,0.588], B = [0.150,0.070]**
  - bílá **W(D_{6500}) = [0.313,0.329]**

- **izoenergetická bílá W** má souřadnice **[1/3,1/3]**, bílá **R** podle televizní **NTSC** normy **[0.31,0.316]**

Gamut monitoru v CIE-xy



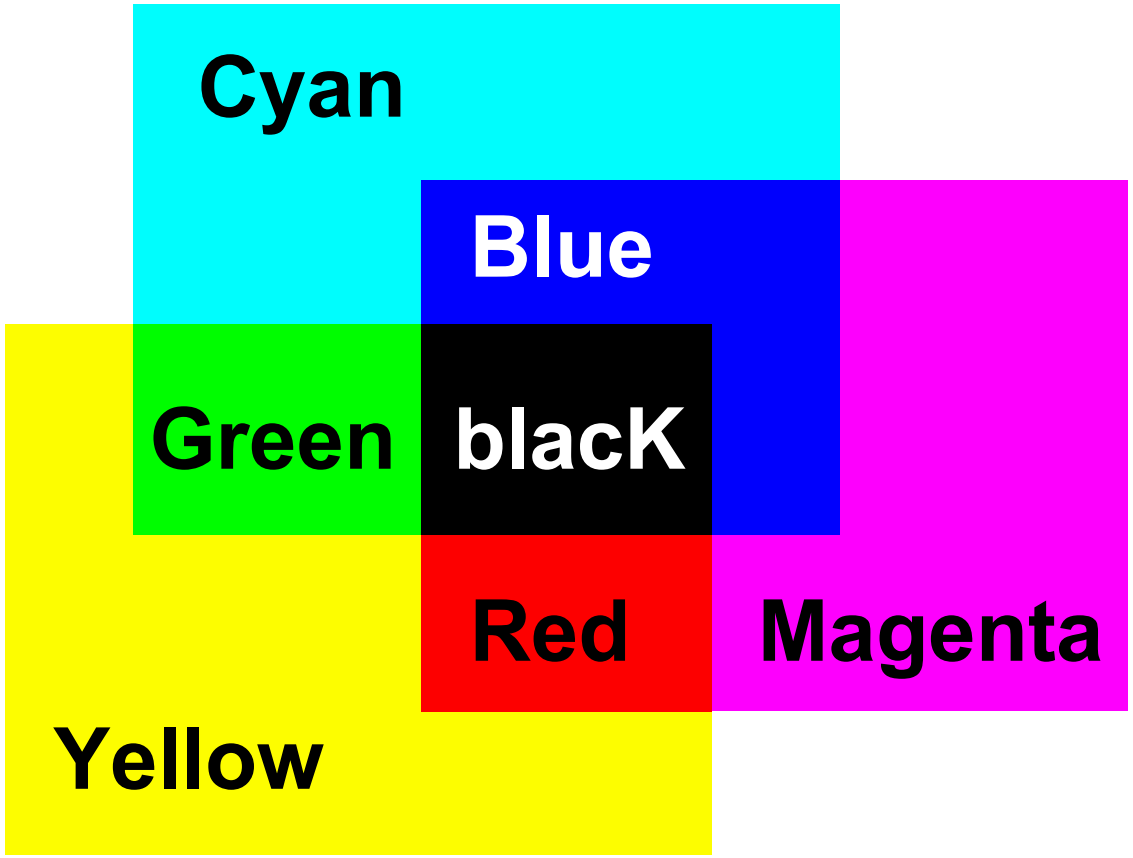
Aditivní skládání barev (RGB)



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
  - **K' = min(C,M,Y), C' = C-K, M' = M-K, Y' = 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**

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);  { zkratka za 3 ":=" }
          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

Konec

Další informace:

- **Jiří Žára a kol.**: ***Počítačová grafika***, principy a algoritmy, 316-328
- **LAN** na Malé Straně:
  - **barbora\usr:\vyuka\pelikan\4**

Zobrazování barev

KSVI MFF UK Praha

e-mail: Josef.Pelikan@mff.cuni.cz

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

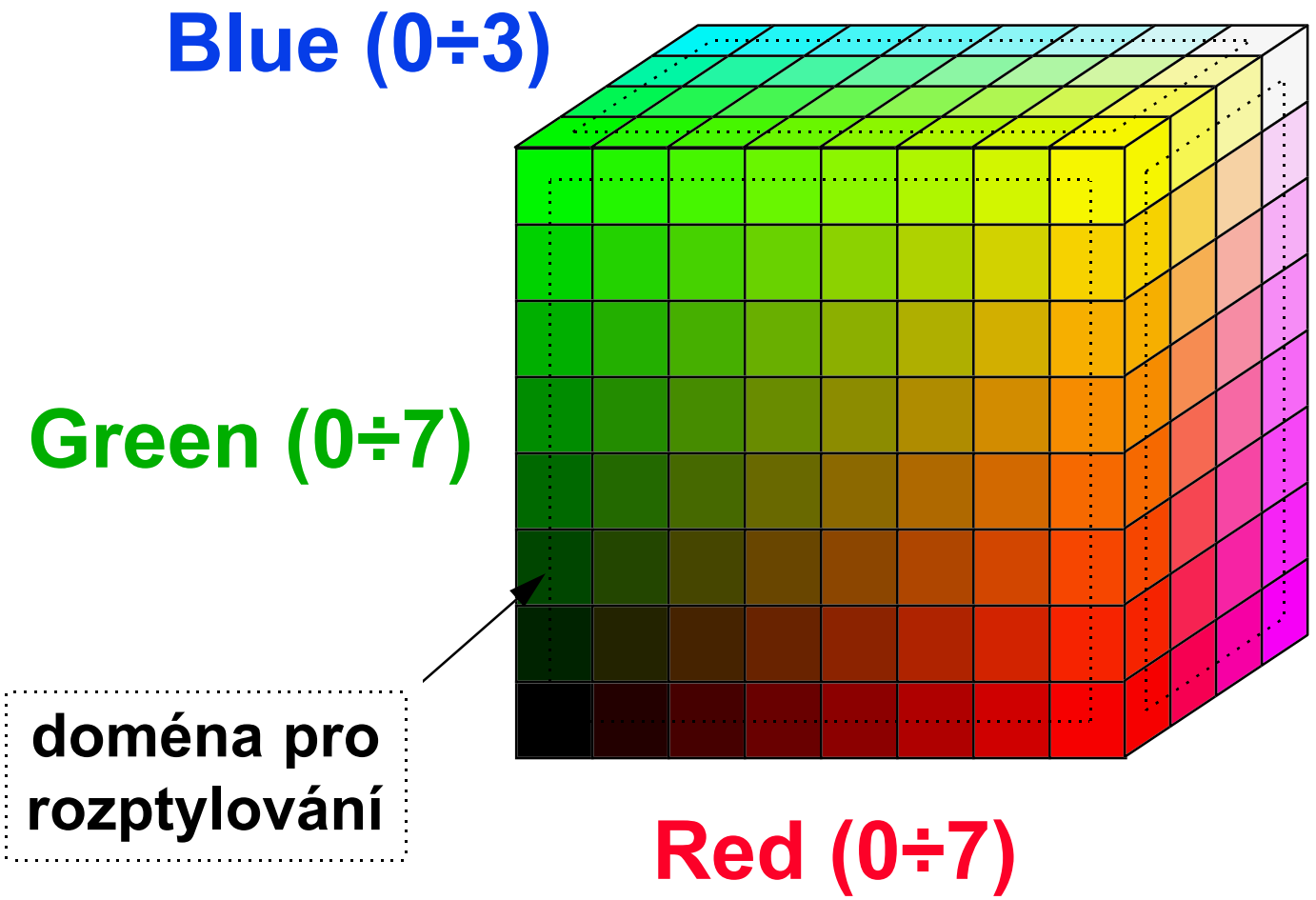
Barevné schopnosti HW

- "**True-color**" nebo "**pseudo true-color**"
  - přímý výstup barevných složek: **RGB**, **CMY(K)**
  - alespoň 5 bitů na složku a pixel (typicky 8)
  - displeje: **15**, **16** (5-6-5), **24-bitová barva**
  - zvětšení barevného rozsahu: rozptylování
- zařízení s **barevnou paletou** ("colormap")
  - pevná nebo nahrávaná paleta
  - počet barev: **16 ÷ 4096** (nejčastěji **256**)
  - **redukce počtu barev** ("color quantization")

Zobrazení barev pomocí palety

- převod barev na **odstíny šedi**
  - složka **Y** (**0.2989*R + 0.5866*G + 0.1144*B**)
- **univerzální barevná paleta** + rozptylování
  - např. **3-3-2 paleta** (256 barev), 6-7-6 (252 barev)
  - maticové, náhodné rozptylování, distribuce chyby
- **adaptovaná barevná paleta** (+ rozptylování)
  - paleta optimalizovaná pro jeden konkrétní obrázek
  - metody konstrukce palety **“shora-dolů”** (Heckbert) a **“zdola-nahoru”** (shluková analýza)

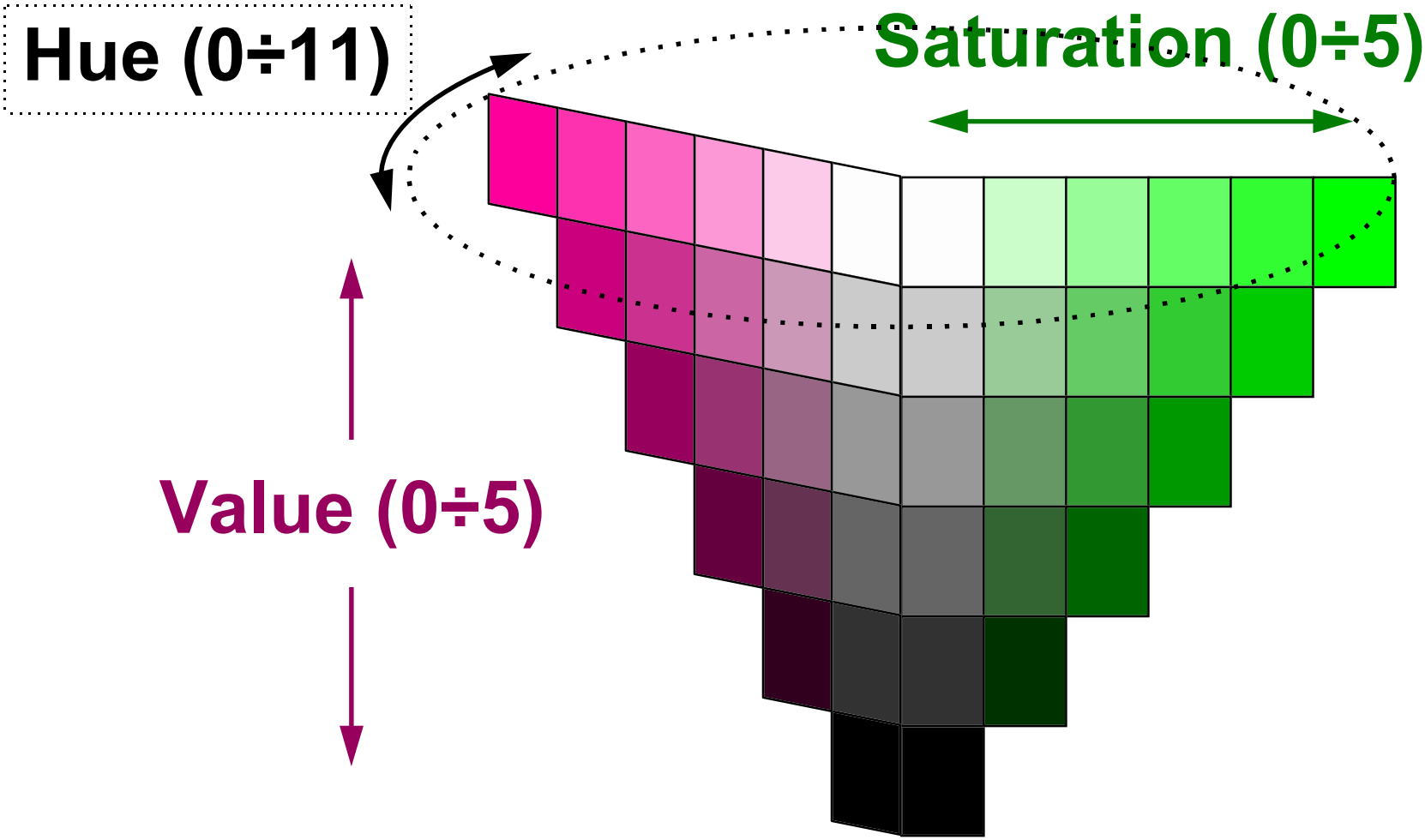
Univerzální “3-3-2 paleta”



Univerzální palety

- **paleta “3-3-2”**: 8×8×4 barvy (256 barev)
  - snadné převody (bez operace násobení)
- **paleta “6×7×6”**: 6×7×6 barev (252 barev)
  - rovnoměrné rozdělení RGB prostoru
- **paleta “7×12×3”**: 7×12×3 barvy (252 barev)
  - zohledňuje různou citlivost oka na barevné složky
- palety pro **jiné barevné systémy**
  - např. 12×(1+2+3+4+5+6) pro **HSV** (186 barev)

Univerzální paleta pro **HSV**



Konstrukce adaptované palety

- speciální paleta přizpůsobená pro zobrazení **jednoho konkrétního obrázku**
  - její výpočet může být značně časově náročný
- konstrukce metodou **“shora-dolů”**
  - dělím množinu použitých barev tak dlouho, až dostanu žádaný počet skupin (např. 256)
- konstrukce metodou **“zdola-nahoru”**
  - sdružuji příbuzné barvy do skupin, dokud nemám požadovaný počet skupin (shluková analýza)

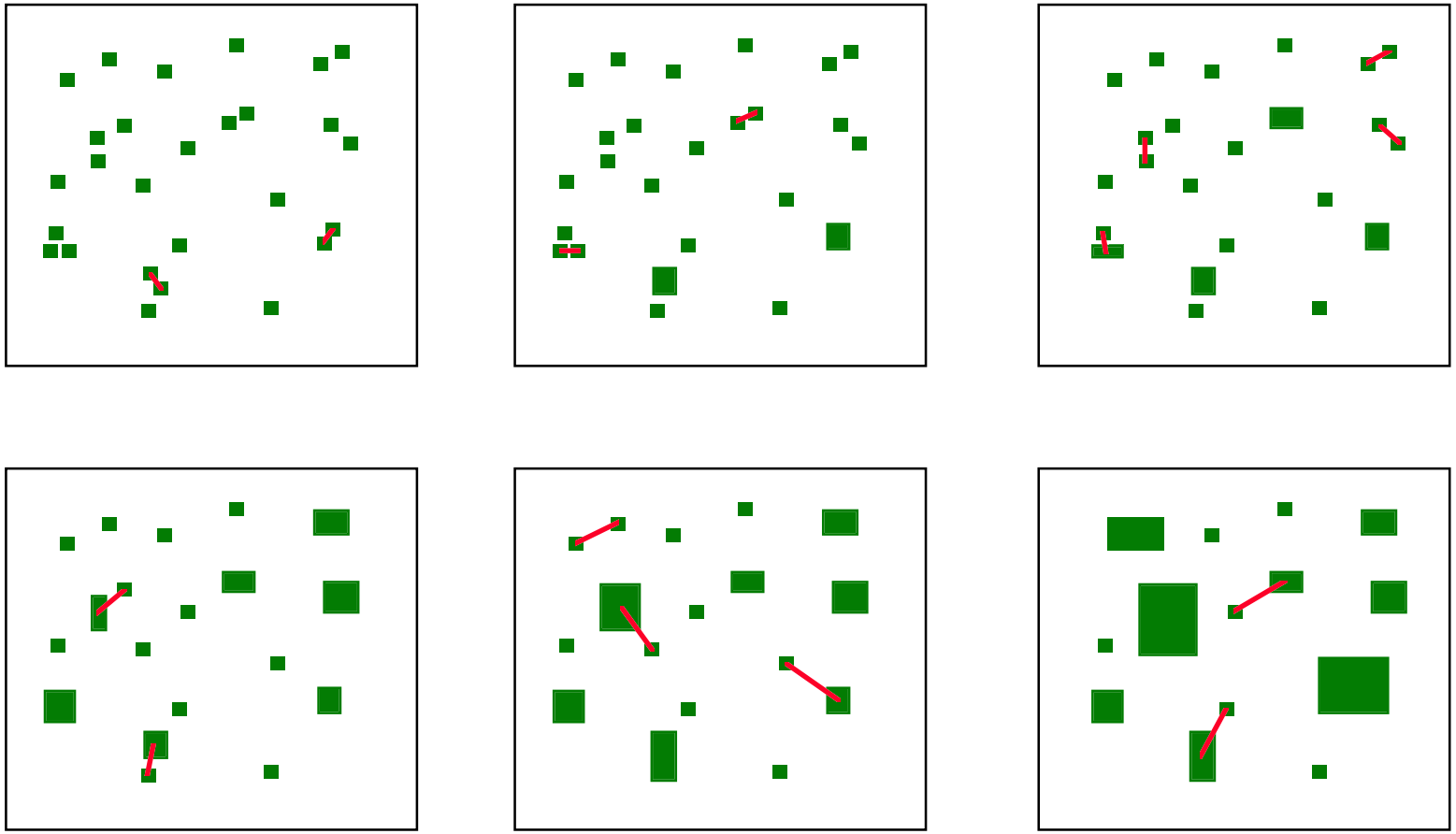
Metoda shlukové analýzy

❶ vytvořím **barevný histogram** obrázku
- výčet všech použitých barev včetně jejich četností
- každá barva tvoří na začátku samostatnou skupinu

❷ najdu dvě **nejbližší** skupiny barev a spojím je
- kritéria podobnosti: **vzdálenost** (**min{$|C_i-C_j|$}**), **poloměr** (**max{$|C_i-C_j|$}/2**), **rozptyl** ($\Sigma(C_i-\varnothing C)^2/n$)

❸ krok ❷ opakuji tak dlouho, dokud nedostanu požadovaný počet skupin **N** (např. 256)
- slučovacích kroků je třeba udělat velmi mnoho!

Postup výpočtu



Algoritmus “octree”

- šetří **paměť** i **čas výpočtu**
  - rychlejší hledání nejbližších skupin barev

1. z **prvních N různých barev** vytvořím skupiny
2. načítám zbytek obrázku a pro každý pixel s dosud se nevyskytující barvou provedu:
3. z **N+1 skupin** vyberu dvě nejbližší a sloučím je
   - algoritmus není symetrický (záleží na vstupním pořadí barev)

Heckbertův algoritmus ("median cut")

❶ vytvořím **barevný histogram** obrázku

– všechny barvy tvoří na začátku jednu skupinu (tvaru kvádru)

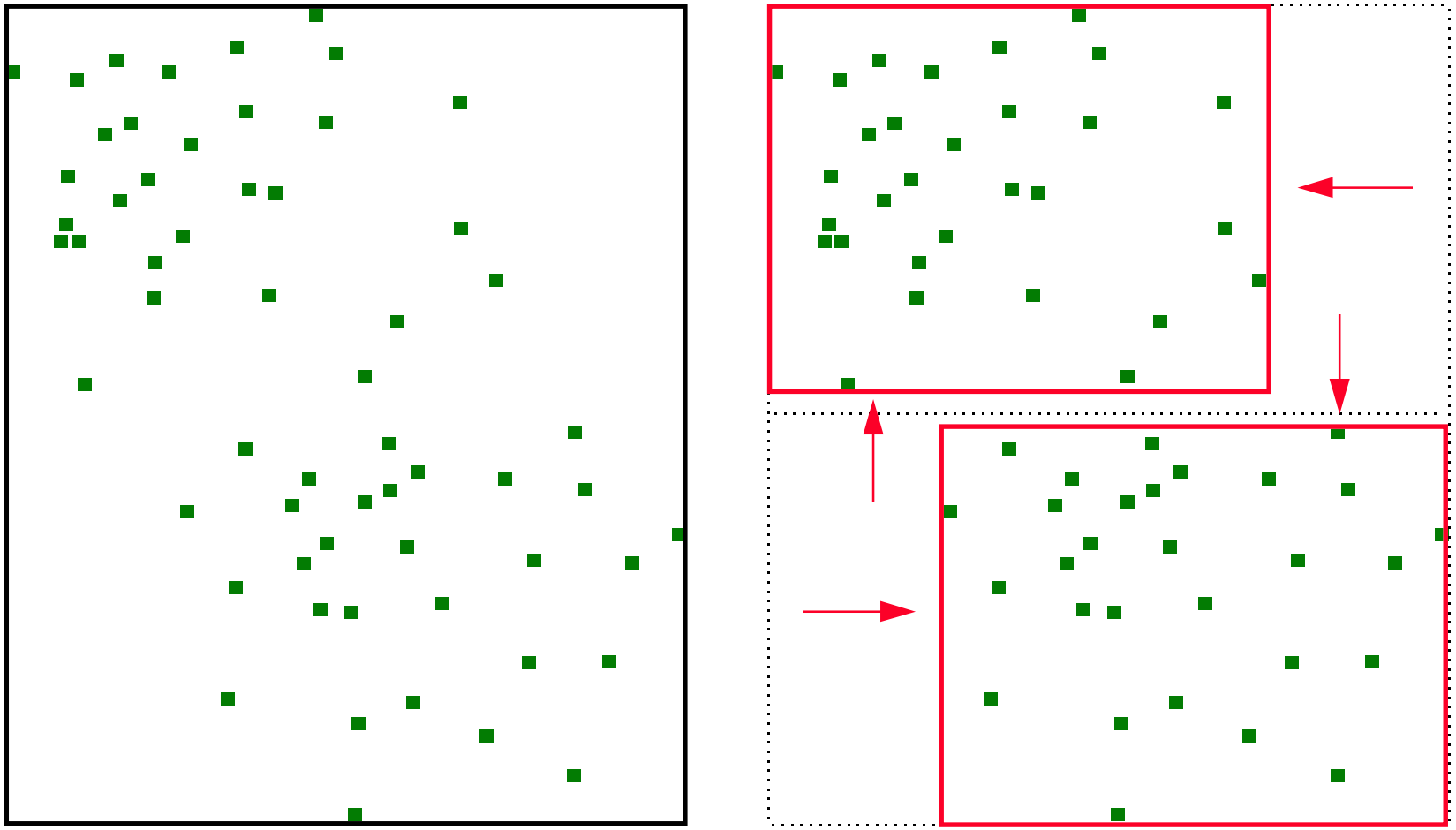
❷ vyberu **"největší" skupinu** barev a rozdělím ji na dvě

– různé metody pro výběr i dělení skupiny barev

❸ krok ❷ opakuji tak dlouho, dokud nedostanu požadovaný počet skupin **N** (např. 256)

– při mapování barev se může použít rozptylování

Dělení skupiny

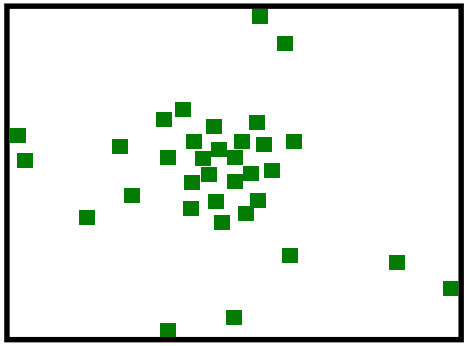


Kritéria dělení skupin barev

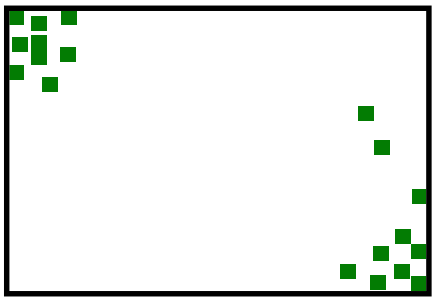
- **velikost kvádru** (délka nejdelší hrany)
  – nejdelší hranu kvádru rozdělím v polovině
- **subjektivní velikost kvádru**
  – jednotlivé složky jsou váženy citlivostí lidského oka
- **počet barev** (počet vstupních pixelů)
  – rozdělím nejdelší hranu kvádru v místě mediánu
- **rozptyl barev** (vážený počtem pixelů)
  – rozdělím nejdelší hranu kvádru v průměrné hodnotě

Kritéria dělení - příklady

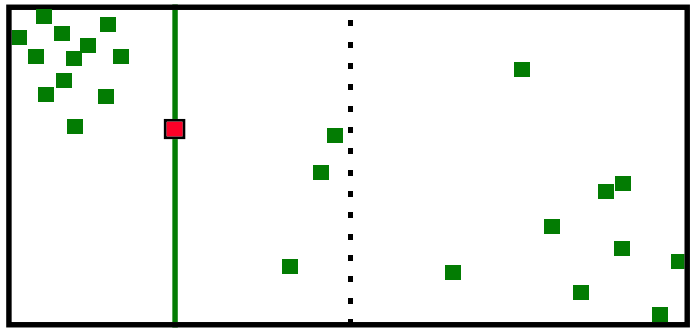
málo barev



více barev, malý rozptyl



méně barev, velký rozptyl



dělení podle mediánu

Implementace

- **vytváření histogramu** je velmi náročné na čas i paměť
  - řídké uložení histogramu (šetří paměť)
  - datová struktura s rychlým vyhledáváním (hašování)
- **přemapování barev**
  - **zaokrouhlení** (vyhledání příslušné skupiny - boxu)
  - **rozptylování** mezi nejbližšími barvami palety (distribuce chyby: hledání nejbližší barvy, vstupní barvy musí ležet v konvexním obalu barev palety)

Barevný tisk

- **malý počet základních barev** (2-6)
  - velké základní - pixelové - rozlišení (tisíce dpi)
  - univerzální čtyřbarevný tisk: **CMYK**
- každá základní barva se **půltónuje**
  - jednotlivé půltónovací rastry (“screens”) mívají rozlišení **60 ÷ 480 lpi**
  - používají se rastry s čtvercovými, kruhovými, eliptickými tečkami, kombinované a speciální rastry (“Monet”, náhodný rastr, ..)

Soutisk rastrů

- rastry se navzájem **otáčejí**
  - zabraňuje se tak vzniku rušivých interferencí
  - klasická sada úhlů pro čtyřbarevný tisk **CMYK**: **0°**, **15°**, **45°**, **75°** (“Offset angles”)
  - jiná sada úhlů: **7.5°**, **22.5°**, **37.5°**, **52.5°**, **67.5°**, **82.5°** (“Flexo angles”)
  - úhly s racionální směrnicí jsou výhodnější pro implementaci

Konec

Další informace:

- **Jiří Žára a kol.**: ***Počítačová grafika***, principy a algoritmy, 335-342
- **V. Skala:** ***Algoritmy počítačové grafiky III***, skriptum ZČU, 1992, 60-61
- **LAN** na Malé Straně:
  - **barbora\usr:\vyuka\pelikan\4**

Redukce barev - praktické výsledky

KSVI MFF UK Praha

e-mail: Josef.Pelikan@mff.cuni.cz

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

Podmínky měření

- **dva testovací obrázky:**
  - **“Lena”** 512×480×24 bitů (145.126 různých barev)
  - **“cube”** 440×330×24 bitů (4.839 různých barev)
- byla měřena **střední kvadratická odchylka** výsledných obrázků od originálu (“RMS error”):
  - barevné složky odchylek byly násobeny váhovými koeficienty **0.299** (R), **0.587** (G), **0.114** (B)
  - pro napodobení vlivu rozptylovacích metod byly odchylky měřeny i přes **obdélníkové filtry** velikosti 2×2 a 3×3

Detaily implementace

- **adaptivní algoritmy obecně**:
  - základní kriterium slučování/rozdělování skupin: **Euklidovský průměr** (barevné složky jsou váženy)
  - **“variance”**: kritériem je **rozptyl** barev ve skupině
  - **“error”**: kritériem je **celková kvadratická chyba** barev ve skupině
  - pro **rozptylování** je v paletě zafixováno osm barev (rohy RGB krychle) - vybírá se jen **N-8** barev
  - výběr **reprezentanta do palety**: **těžiště** skupiny barev

Detaily implementace

- **Heckbertův algoritmus**:
  - dělí se **nejdelší** (vážená) **hrana kvádru** v místě **těžiště**
  - ve variantách **“variance”** a **“error”** se dělí hrana s **největším** (váženým) **rozptylem hodnot** v místě **těžiště**
  + **“number”**: kritériem je počet pixelů ve skupině (pokud skupina obsahuje alespoň dvě různé barvy)

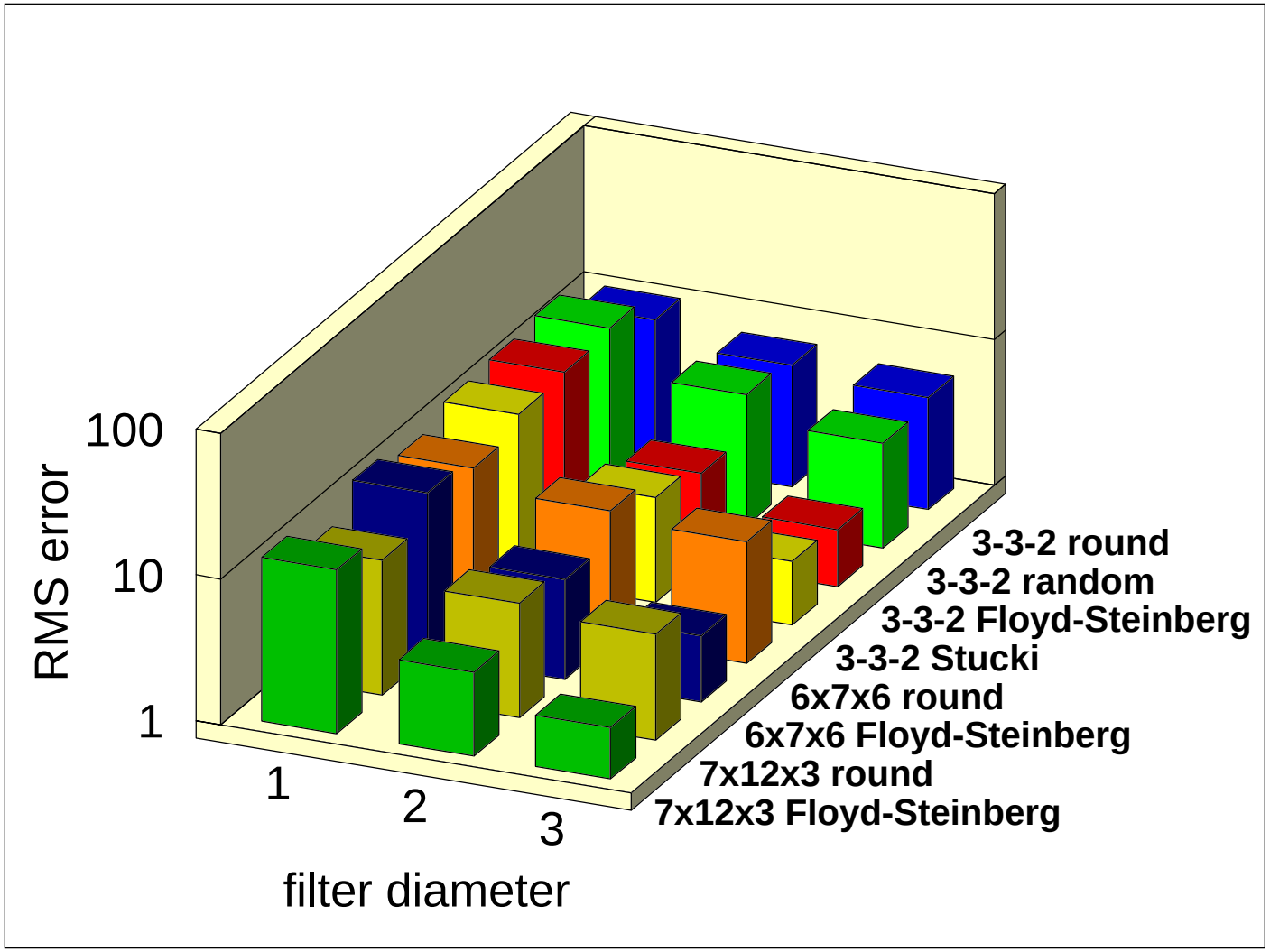
Výsledky - univerzální palety

	"Lena"			"cube"		
průměr filtru:	**1**	**2**	**3**	**1**	**2**	**3**
3-3-2 round	**9.80**	6.83	5.88	13.88	13.57	13.33
3-3-2 random	15.81	7.90	5.26	10.46	5.26	3.53
3-3-2 Floyd-Steinb.	14.46	**4.19**	**2.46**	**10.22**	**3.07**	**1.87**
3-3-2 Stucki	13.62	5.24	**2.73**	**9.99**	3.59	**1.81**
6x7x6 round	**10.75**	7.81	6.84	15.76	15.41	15.15
6x7x6 Floyd-Steinb.	13.29	**4.83**	2.85	11.52	**3.54**	2.07
7x12x3 round	**8.45**	6.10	5.38	11.47	11.19	10.97
7x12x3 Floyd-Steinb.	13.29	**3.76**	**2.25**	**9.69**	**2.83**	**1.66**

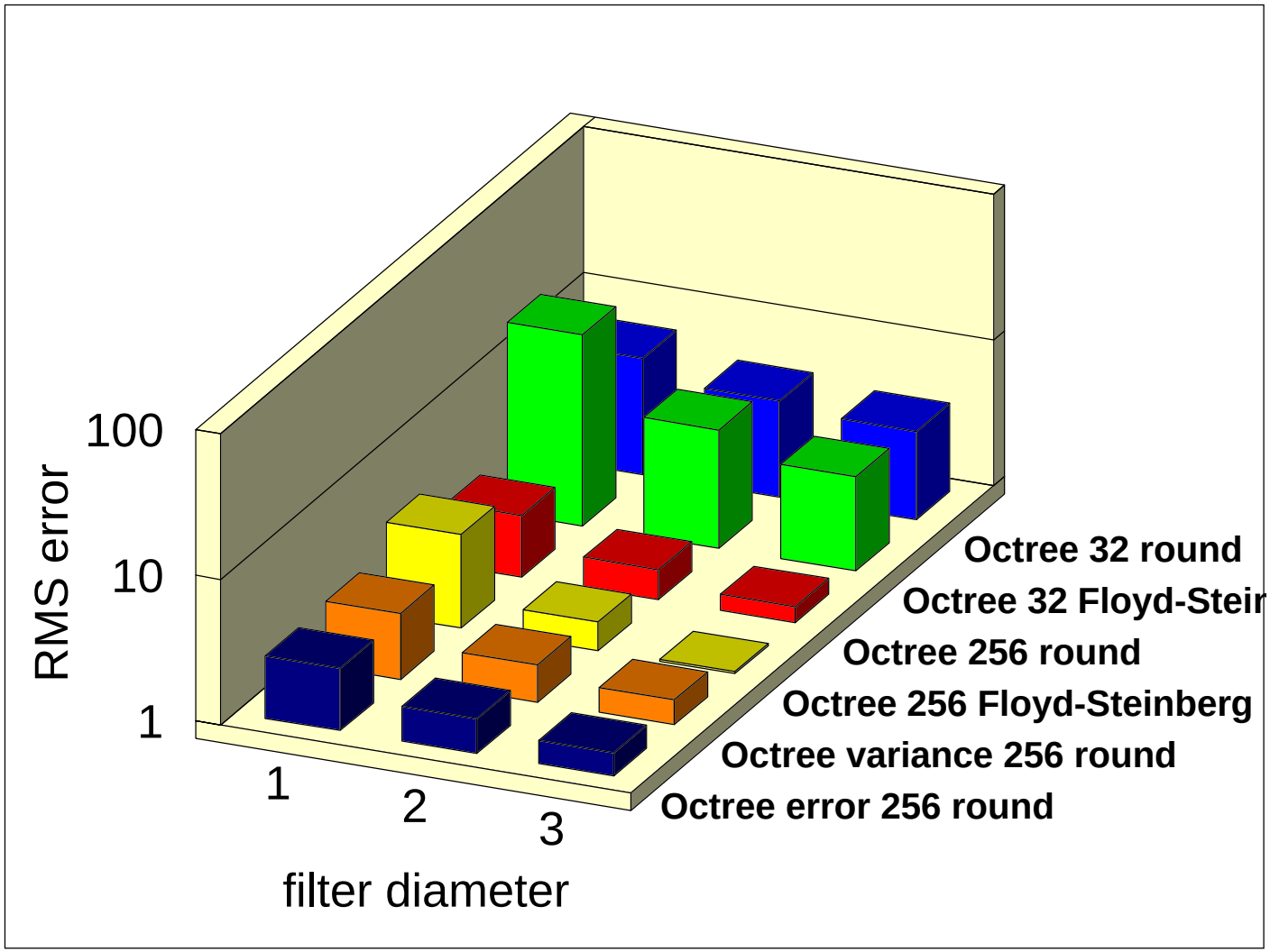
Výsledky - adaptované palety

	"Lena"			"cube"		
Octree 32 round	6.32	4.60	4.06	11.15	10.77	10.51
Octree 32 F.-S.	20.52	6.52	4.45	29.96	7.62	6.15
Octree 256 round	**2.66**	1.61	1.28	3.72	3.35	3.10
Octree 256 F.-S.	4.40	**1.58**	**1.03**	6.30	**2.05**	**1.49**
Octree variance 256	2.85	1.80	1.48	17.48	16.91	16.48
Octree error 256	2.69	1.73	1.44	5.46	5.12	4.89
Heckbert 32 round	5.71	3.93	3.39	10.96	10.65	10.44
Heckbert 32 F.-S.	16.74	5.55	3.75	32.66	7.78	6.32
Heckbert 256 round	2.79	1.62	1.26	**3.54**	3.20	2.96
Heckbert 256 F.-S.	3.97	**1.50**	**0.94**	5.96	**2.31**	**1.79**
Heckbert number 256	**2.59**	1.61	1.31	**3.59**	3.18	2.89
Heckbert variance 256	2.69	1.65	1.33	3.95	3.64	3.42
Heckbert error 256	**2.34**	**1.49**	**1.24**	**3.15**	**2.83**	**2.60**

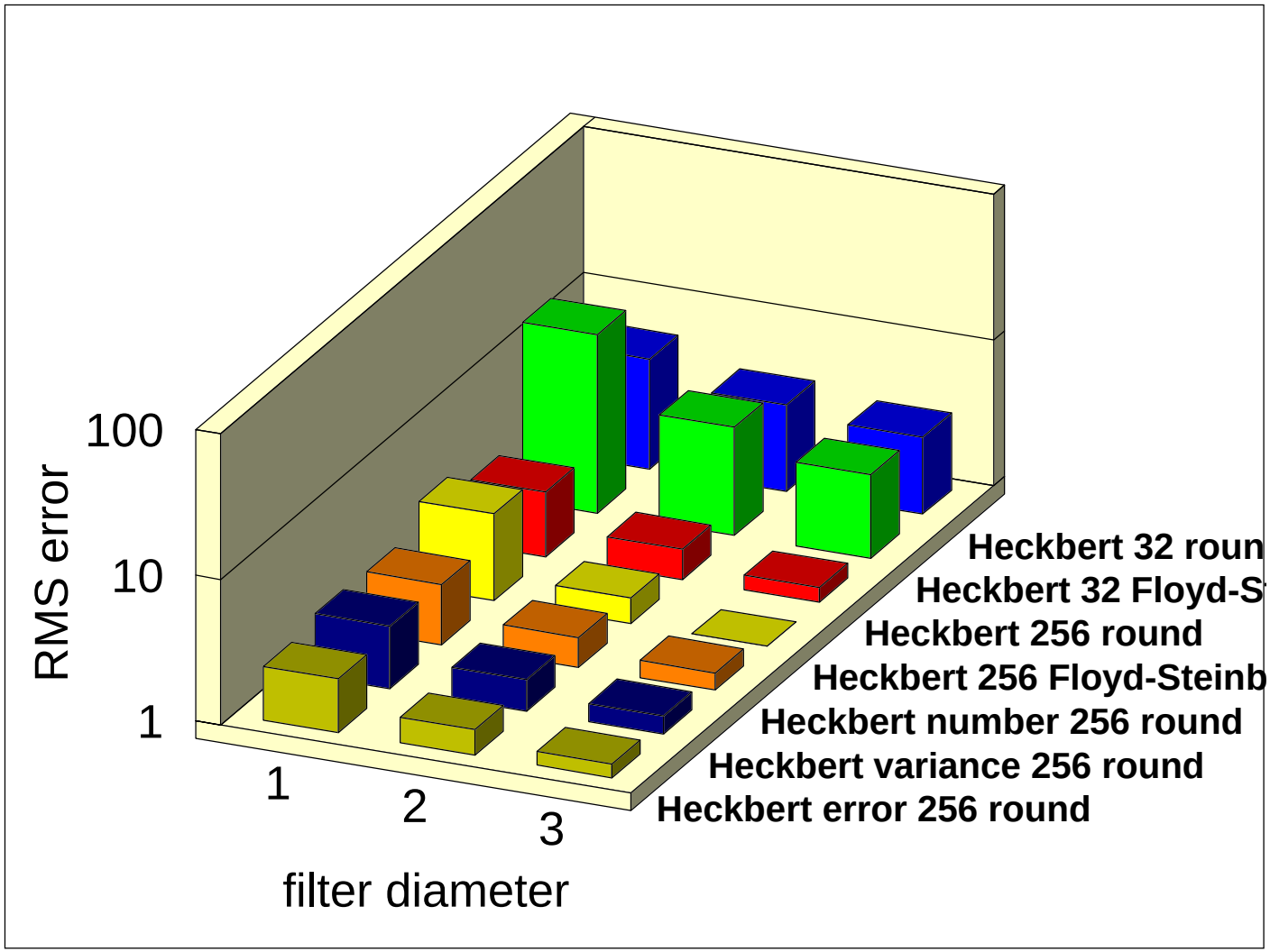
“Lena” - univerzální palety



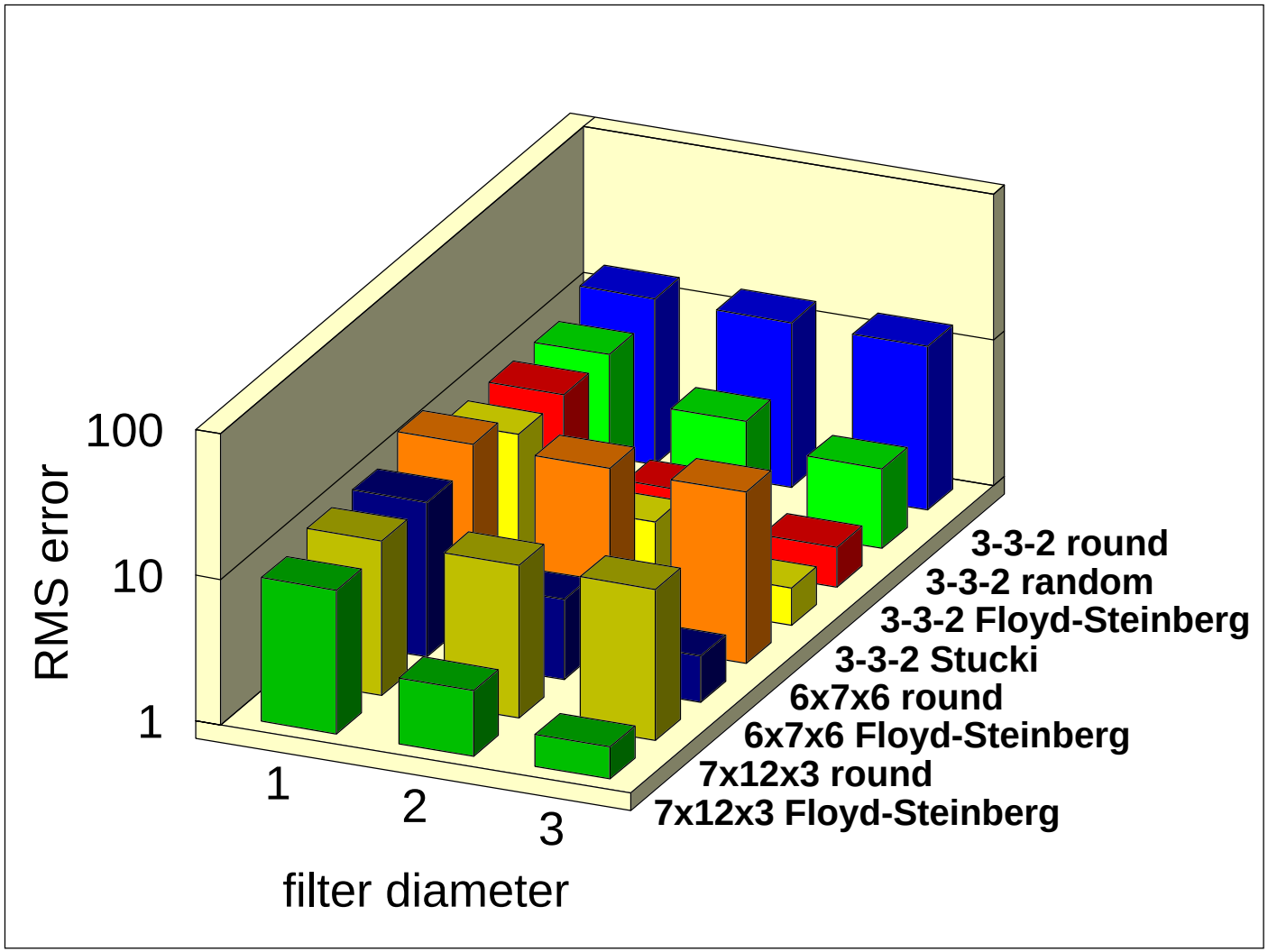
“Lena” - algoritmus “octree”



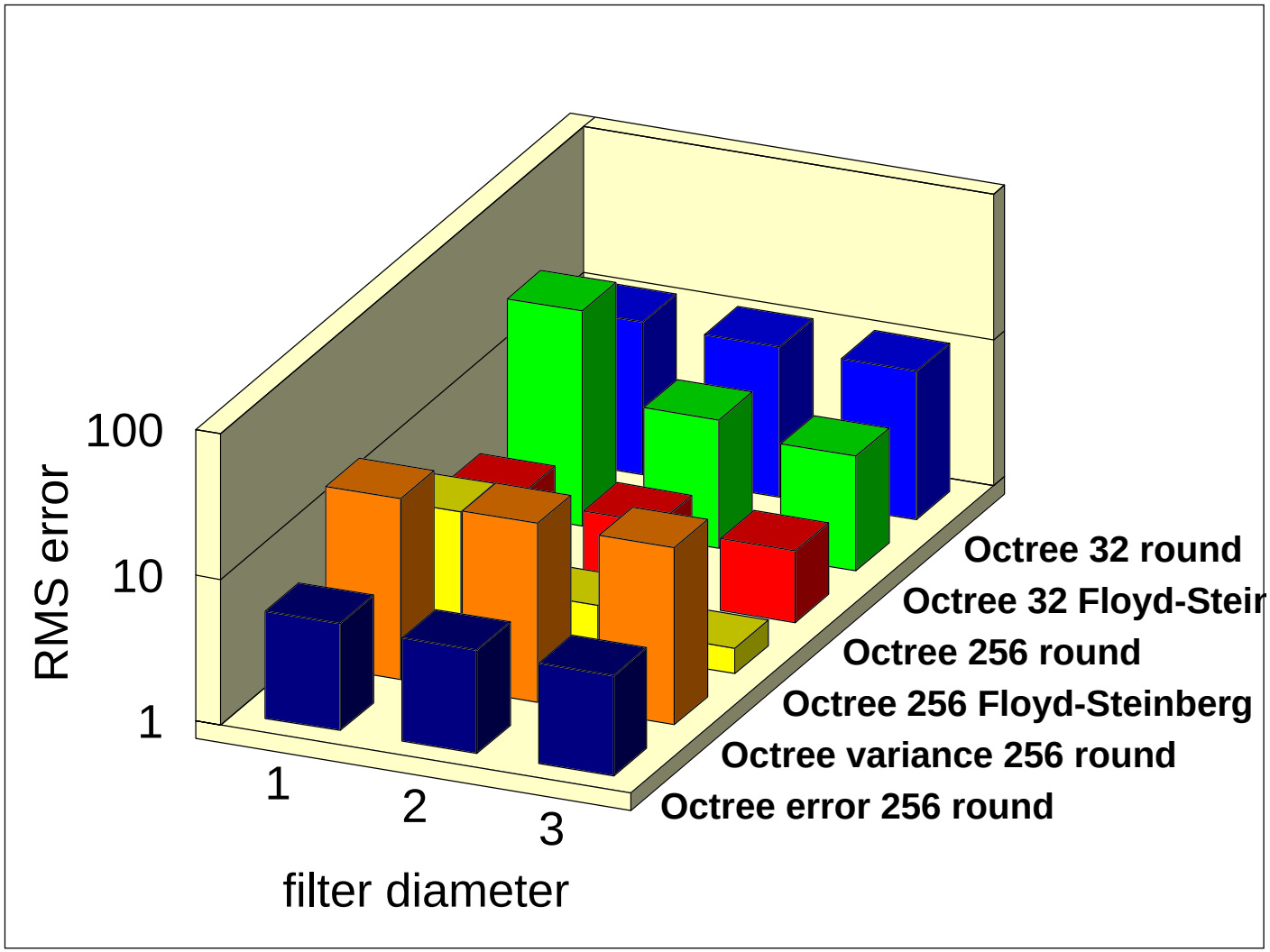
“Lena” - Heckbertův algoritmus



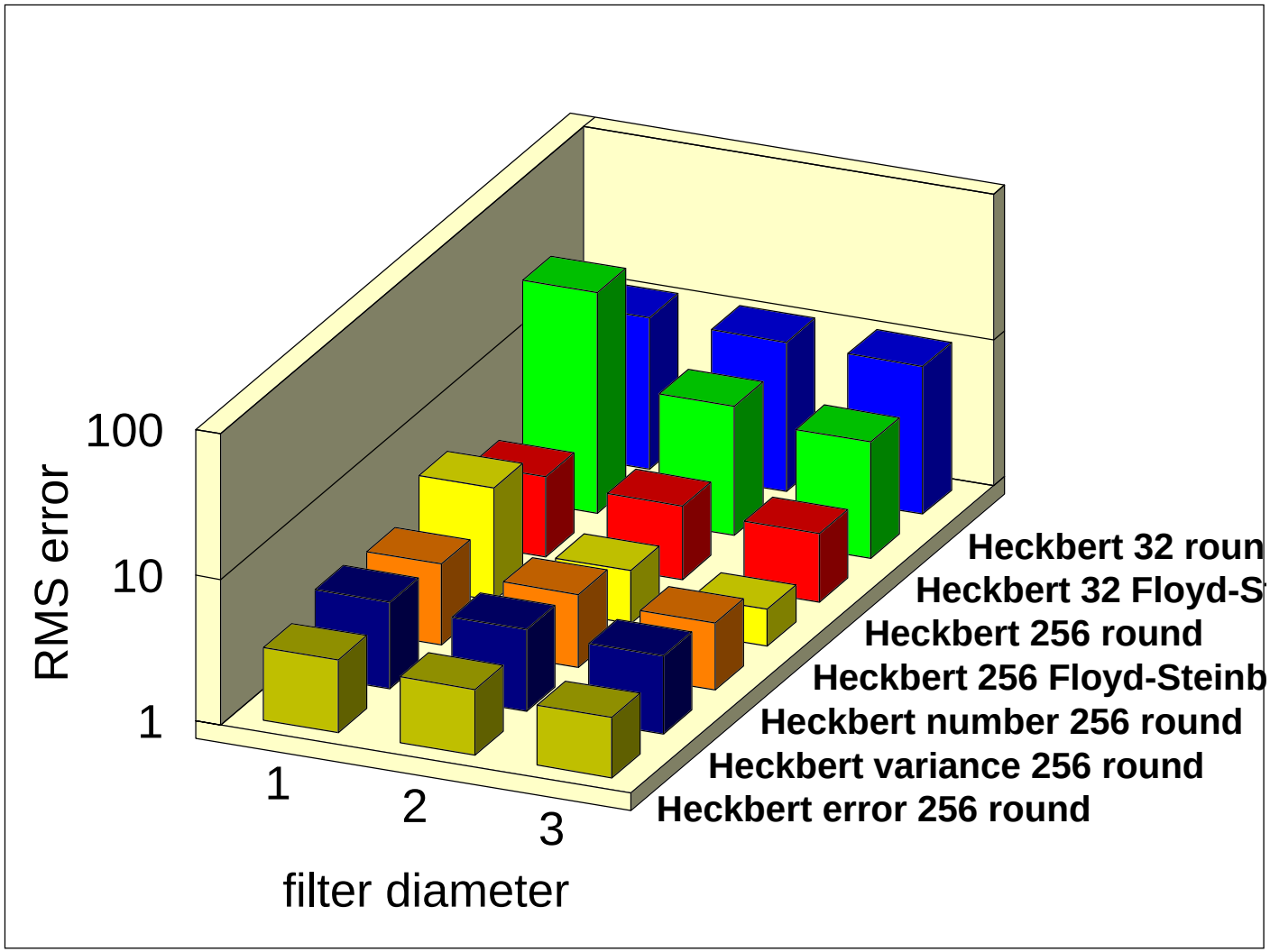
“cube” - univerzální palety



“cube” - algoritmus “octree”



“cube” - Heckbertův algoritmus



Barevné palety

KSVI MFF UK Praha

e-mail: Josef.Pelikan@mff.cuni.cz

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

Barevné palety

- **diskrétní**
  - neexistuje souvislost mezi barvou a jejím umístěním v paletě
  - kreslicí programy (malířská paleta)
- **topologické**
  - barvy jsou v paletě uspořádány podle určitého systému
  - vyhlazování (“anti-aliasing”), univerzální barevné zobrazování, rozptylování

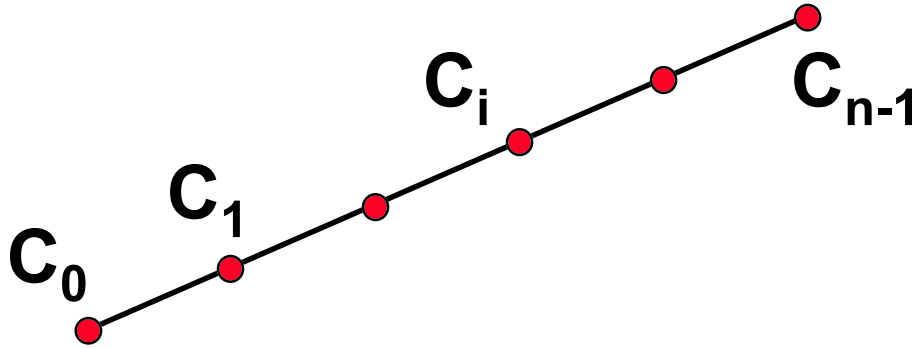
Univerzální zobrazování barev

- snadný výpočet čísla barvy v paletě
  - jednoduchá implementace rozptylovacích metod
- palety pro systém **RGB**
  - paleta **“3-3-2”**: 8×8×4 barvy
  - paleta **“6×7×6”**: 6×7×6 barev
  - paleta **“7×12×3”**: 7×12×3 barvy
- palety pro **jiné barevné systémy**
  - 12×(1+2+3+4+5+6) pro **HSV**

Vyhlazování - směs dvou barev

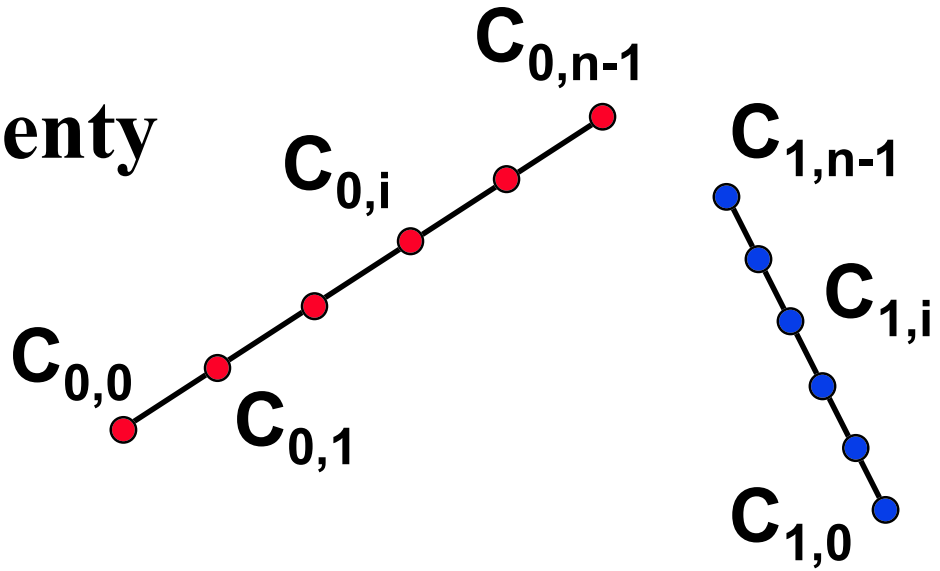
I) jeden segment

i



II) disjunktní segmenty

s	i

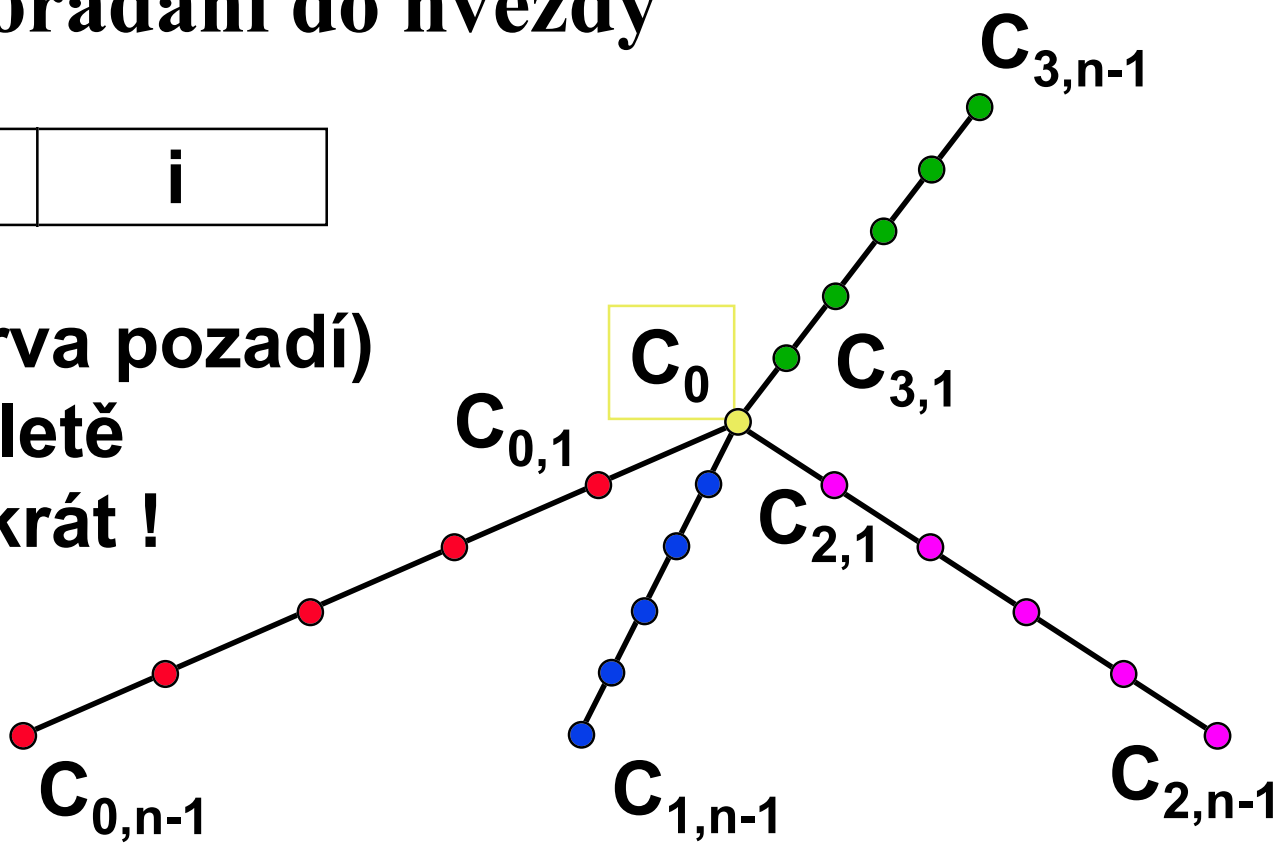


Vyhlazování - směs dvou barev

III) uspořádání do hvězdy

s	i

C_0 (barva pozadí) je v paletě několikrát !



Vyhlazování - směs dvou barev

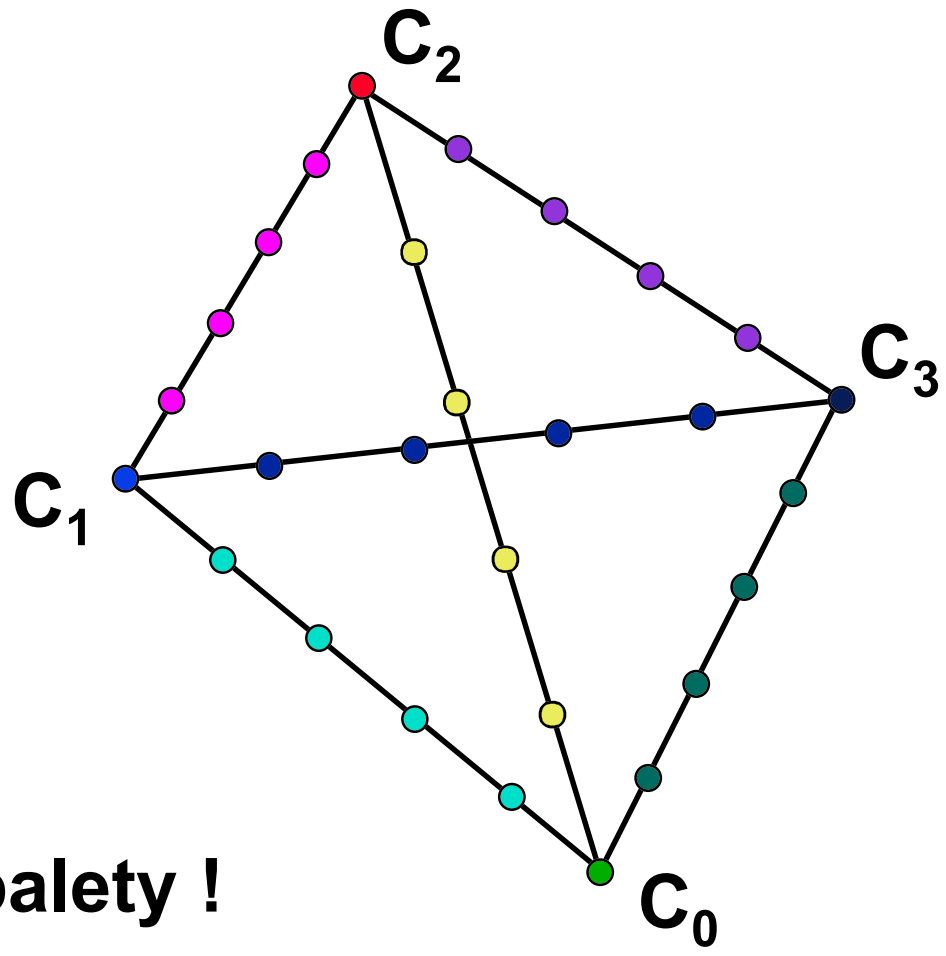
IV) barevná síť

b_1	b_2	i

$b_1 > b_2$

$[b_1,b_2]$:
[1,0], [2,0], [2,1],
[3,0], [3,1], [3,2]

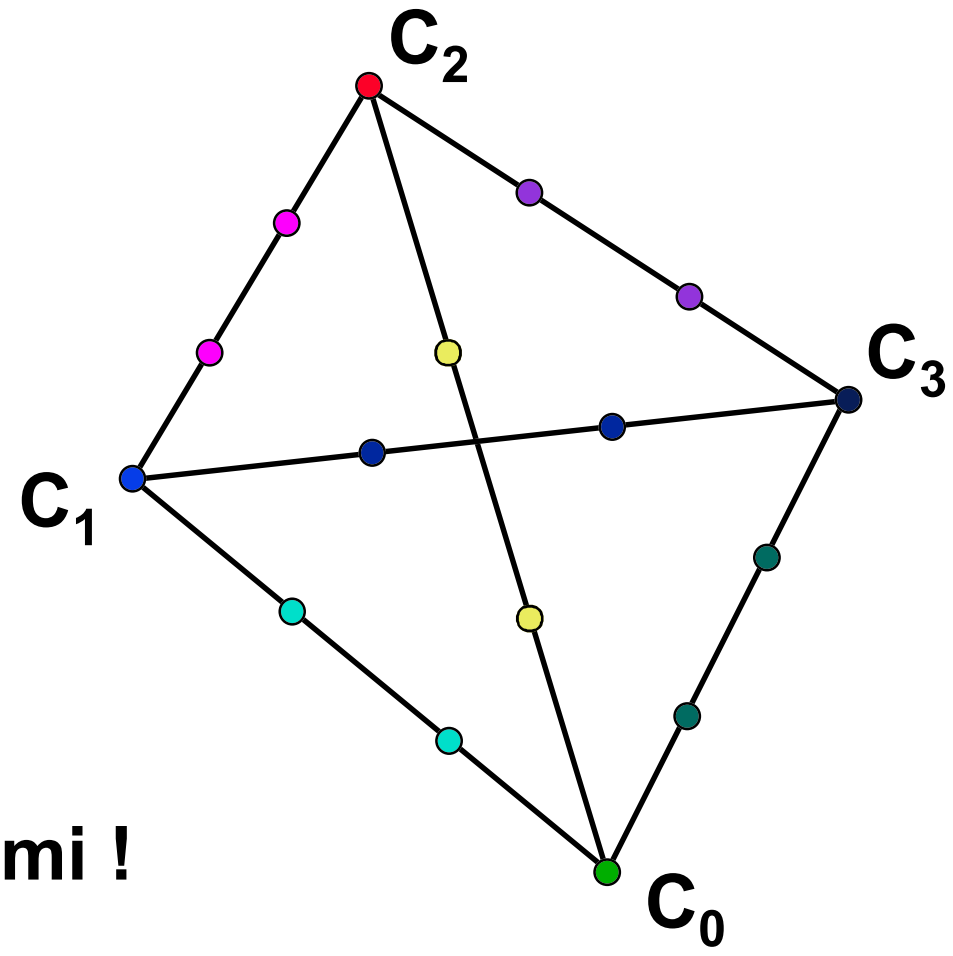
Nesouvislý úsek palety !



Vyhlazování - směs dvou barev

V) “major-minor”

maj	min



$C_{maj}*2/3 + C_{min}*1/3$

Neplýtvá se barvami !

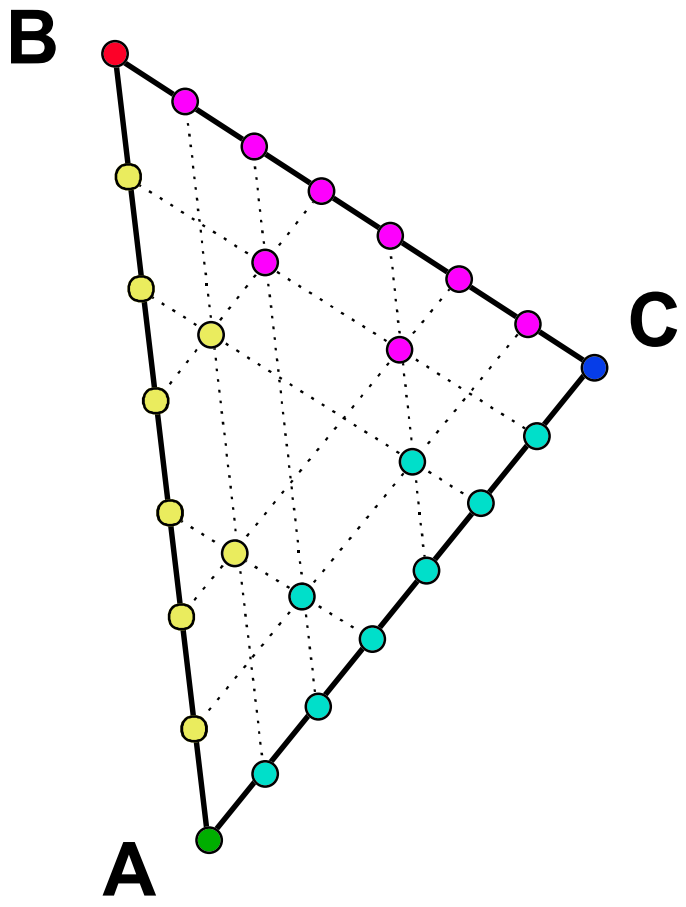
Vyhlazování - směs tří barev

VI) tři komponenty

b_1	b_2	b_3

$b_1*4/7 + b_2*2/7 + b_3*1/7$

Neplýtvá se barvami !



Vyhlazování - směs čtyř barev

VII) čtyři komponenty

b_1	b_2	b_3	b_4

$$b_1*8/15 + b_2*4/15 + b_3*2/15 + b_4*1/15$$

mezi dvěma základními barvami: **14 odstínů**
v rovině tří základních barev: **6 dalších odstínů**
uvnitř každého čtyřstěnu: **24 dalších odstínů**

Další informace uložené v paletě

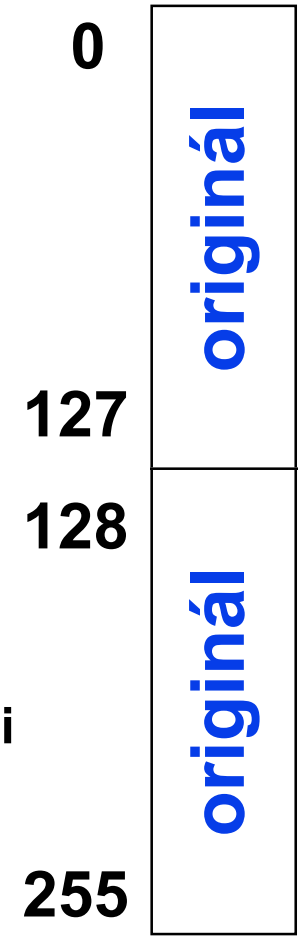
- **jednobitová maska** pro zvýraznění označených pixelů (označená oblast)
  - programy pro zpracování obrazu
- uložení **více obrázků** do jedné Video-RAM
  - pro 256-barevný režim: dva 16-barevné obrázky, čtyři čtyřbarevné, osm dvoubarevných
  - mohu zobrazit jejich libovolnou kombinaci
  - plány měst, GIS (geografické informační systémy)

Další informace uložené v paletě

- **prolínání dvou obrázků** (“fade-in”, “fade-out”)
  - dva 16-barevné obrázky v 8-bitové Video-RAM
- **animace pomocí palety**
  - uložení více snímků do jedné Video-RAM
  - rychlé přepínání snímků pouze změnou palety
  - možnoust prolínání sousedních snímků

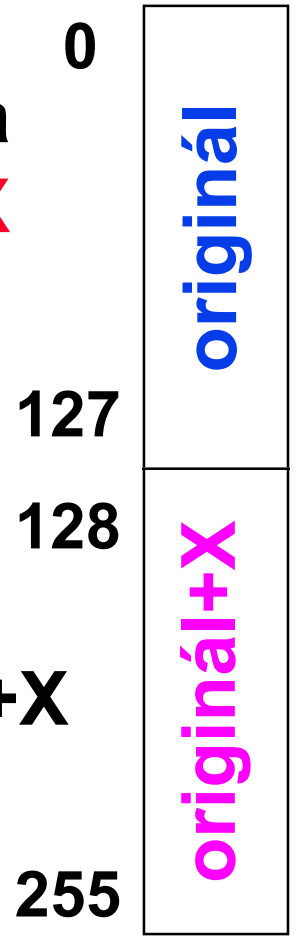
Jednobitová maska

1) maska není vidět



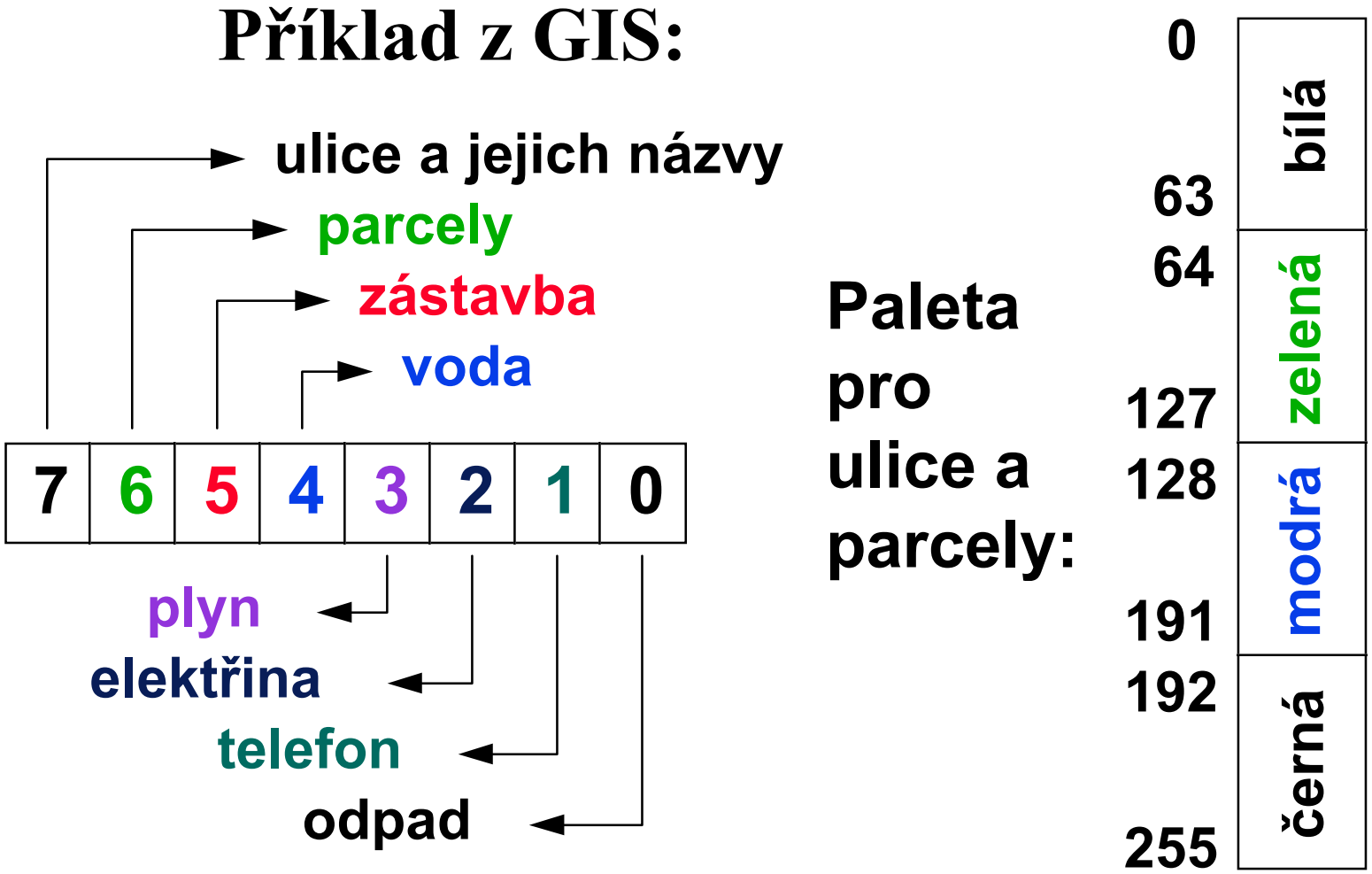
$C_{i+128} = C_i$

2) maska je zvýrazněna odstínem X



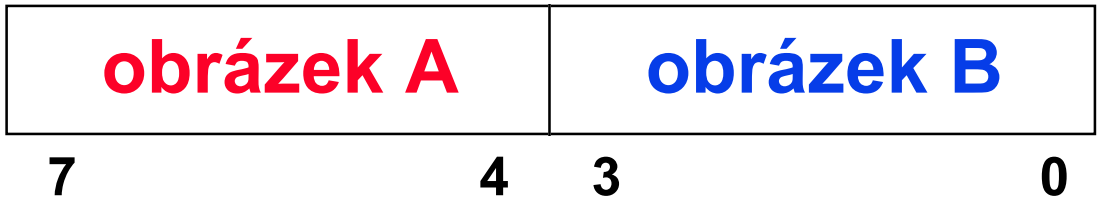
$C_{i+128} = C_i + X$

Více obrázků ve Video-RAM



Prolínání dvou obrázků

- dva **16-barevné** obrázky v 8-bitovém bufferu
  - vstupní barvy: $\mathbf{A_0}$-$\mathbf{A_{15}}$, $\mathbf{B_0}$-$\mathbf{B_{15}}$



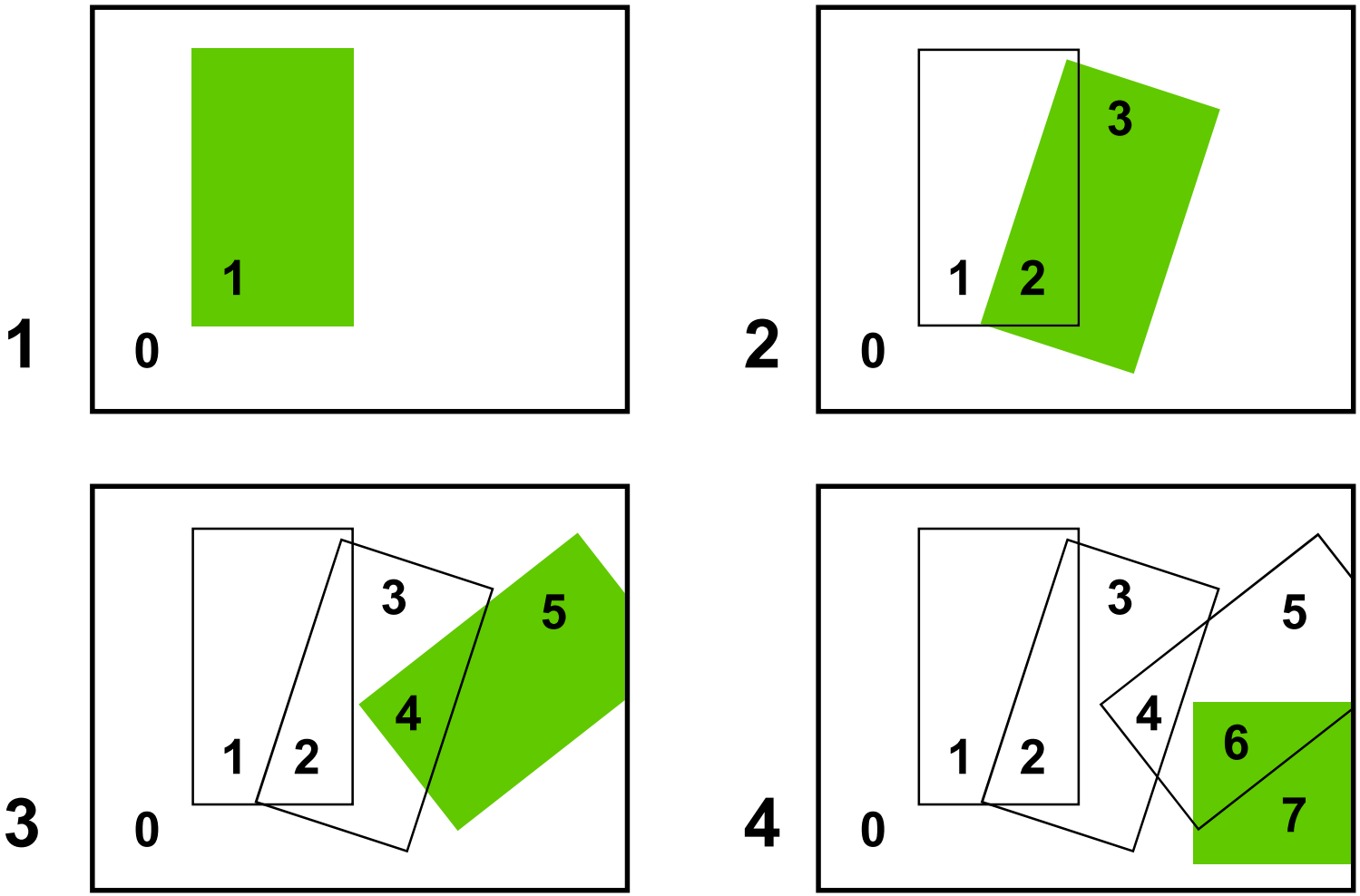
- **palety pro přechod** jednoho obrázku do druhého:
  - $\mathbf{C_i = (1\text{-}t) * A_{i\ div\ 16} + t * B_{i\ mod\ 16}}$
  - **t** se pohybuje od **0** do **1**

Animace pomocí palety

- časově náročné **kreslení do Video-RAM** se dělá jen na začátku
  - snímky se přepínají pouze změnou palety

- **univerzální konstruktivní algoritmus:**
  - hledám rozklad všech pixelů na třídy **ekvivalence**: **[x,y] ~ [u,v]**, jestliže jsou v každém snímku **i** barvy $C_i[x,y] = C_i[u,v]$
  - tříd ekvivalence může být maximálně tolik, kolik barev v paletě mám k dispozici
  - inkrementální konstrukce relace **“~”**

Konstrukce paletové animace



Konec

Další informace:

- **LAN** na Malé Straně:
  - **barbora\usr:\vyuka\pelikan\5**

Kódování rastrových obrázků

KSVI MFF UK Praha

e-mail: Josef.Pelikan@mff.cuni.cz

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

Použití

- **úsporné uložení** rastrového obrázku
  - proti běžným textovým algoritmům mohu využít dvojrozměrné povahy dat
- efektivnější operace s jednoduchými obrázky a **bitovými maskami**
  - množinové operace s bitovými maskami
  - superpozice obrázků

RLE kódování ("run-length enc.")

- využívá se **koherence** ve vodorovném směru:
  - sousední pixely mají často stejnu hodnotu
  - nejvýhodnější u málo barevných obrázků

- **speciální (pří)znak** pro uložení **"běhu"**:

  ESC {počet} {pixel} (PCX)

- dva typy paketů - **"kopírovací"** a **"opakovací"**:

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

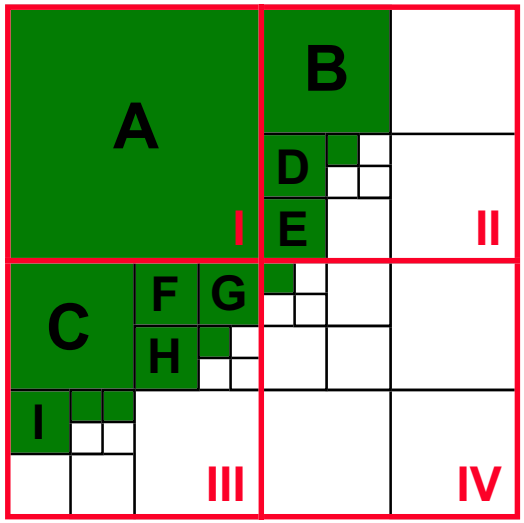
  FILL {počet} {pixel}

Kvadrantový strom (“quadtree”)

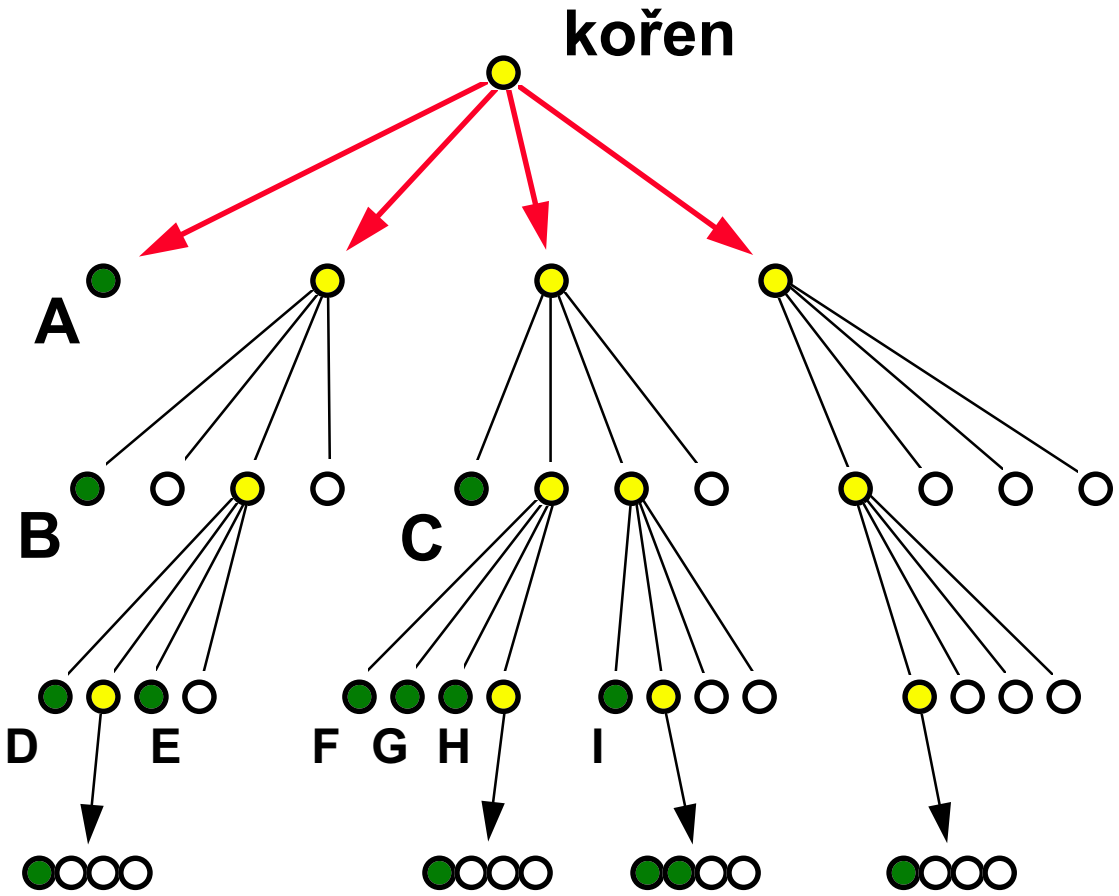
- využívá se **koherence** ve vodorovném i svislém směru
  - úsporně se kódují větší souvislé plochy jedné barvy
  - **adaptivní princip** (postupné dělení “zajímavých”= členitých oblastí)

- **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)

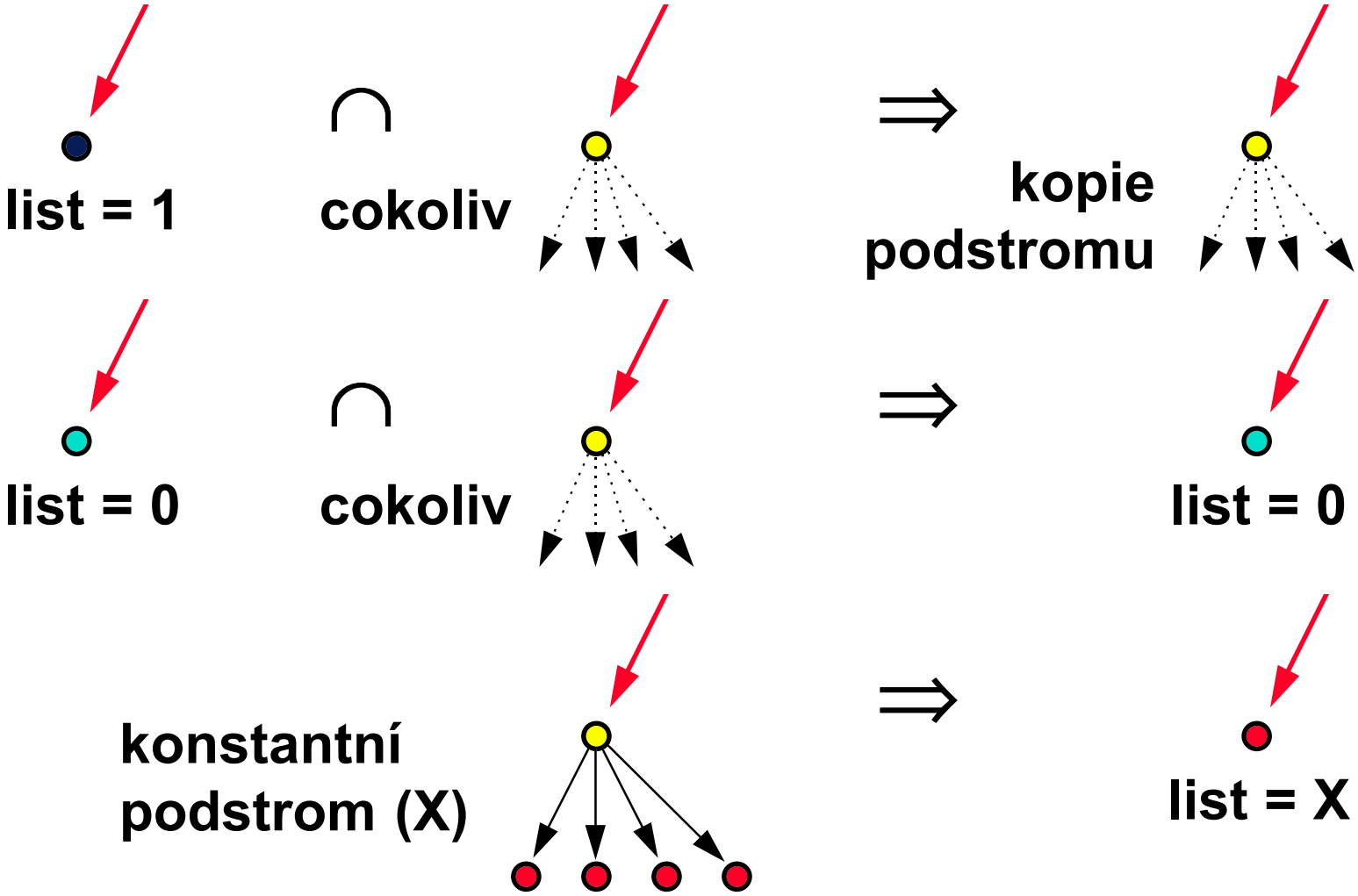
Kódování kvadrantového stromu

- **podle definice** (metoda “shora-dolů”):
  - daný čtverec zkontroluji ⇒ je-li vícebarevný, rozdělím ho na čtyři části, atd. (rekurze, “pre-order”)
  - hodnoty některých pixelů čtu **několikanásobně**
- metoda **“zdola-nahoru”**:
  - začínám od **čtverečků 2×2**, jednobarevné oblasti spojuji do větších uzlů grafu, atd., ... nakonec vytvořím **kořen stromu** (rekurze, “post-order”)
  - každý pixel čtu pouze **jedenkrát**

Množinové operace

- kvadrantové stromy reprezentují **jednobitovou informaci** (množinu, masku)
  - množinové operace (sjednocení, průnik, rozdíl, ..)
  - předpokládám shodný definiční obor operandů
- procházím paralelně všechny **vstupní stromy** a současně konstruuji **výsledný strom**:
  - všechny vstupní uzly jsou vnitřní ⇒ “rozděl a panuj”
  - jeden vstupní uzel je listem ⇒ podle typu množinové operace zpracuji ostatní vstupní podstromy

Pravidla pro operaci “průnik”:



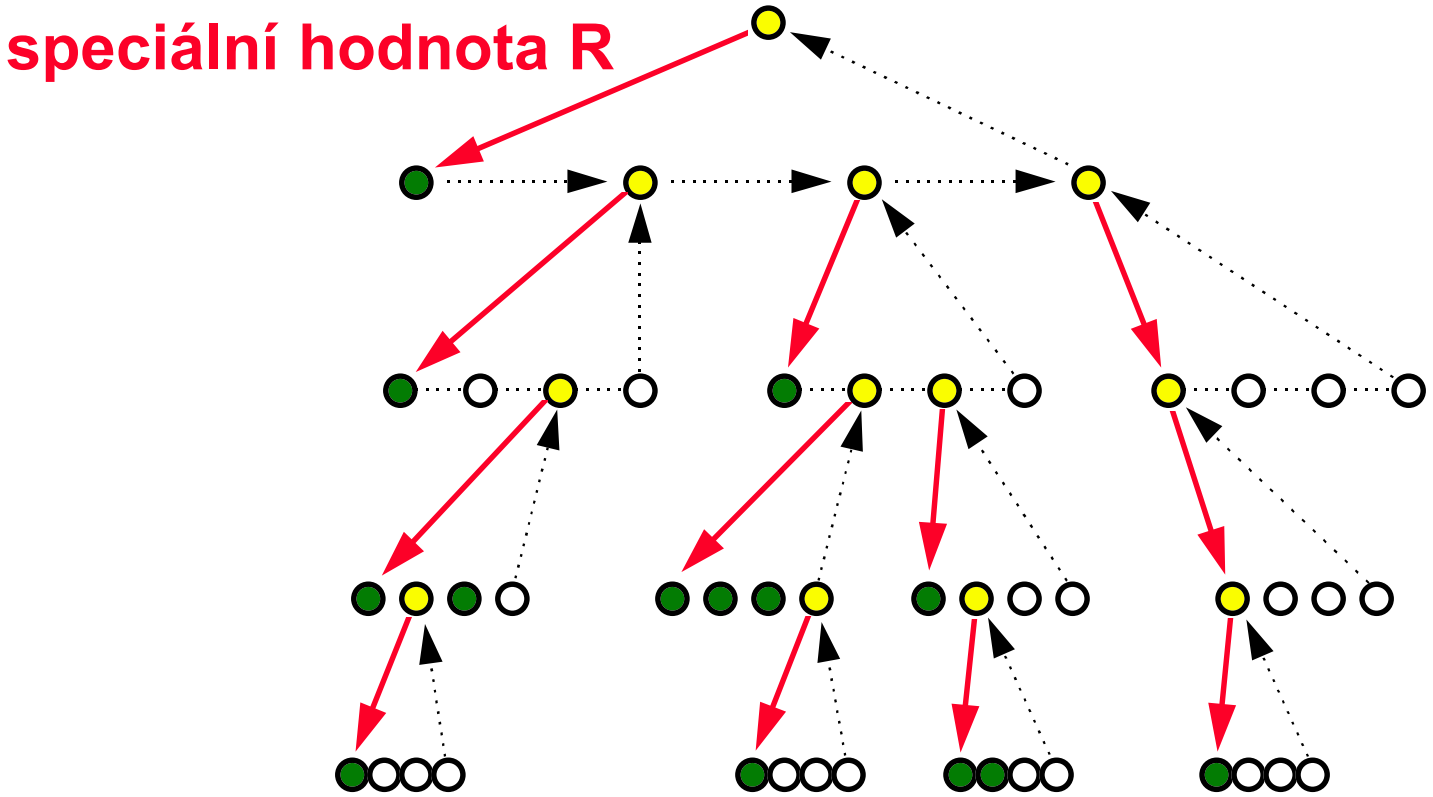
Implementační poznámky

- kódování **obecné oblasti**:
  - zakóduji nejmenší čtverec rozměru $\mathbf{2^n \times 2^n}$, který danou oblast obsahuje
  - pixely ležící **mimo oblast** zakóduji speciální hodnotou (“outside”)
  - jiná varianta: vnějším pixelům přiřadím hodnotu **okrajových pixelů** (“don’t care”) - největší úspora
- úsporné **hybridní kódování** obrázku:
  - je-li podstrom větší než bitmapa, ukládám **bitmapu**

Implementační poznámky

- **společné větve** kvadrantového stromu:
  - **opakuje-li** se ve stromu nějaká větev (podstrom) několikrát, uložím ji pouze jednou a odkazuji se na ni z více míst
  - ze stromu se stává hierarchický **graf** (ADG - acyklický orientovaný graf)
  - společná větev může být použita v různých úrovních
- **lineární uložení** kvadrantového stromu (disk)
  - **průchod stromem** zleva-doprava (“pre-order”)

Lineární uložení kvadr. stromu



R1R10R1R1000100R1R111R1000R1R1100000
RRR1000000000 **... 49 položek**

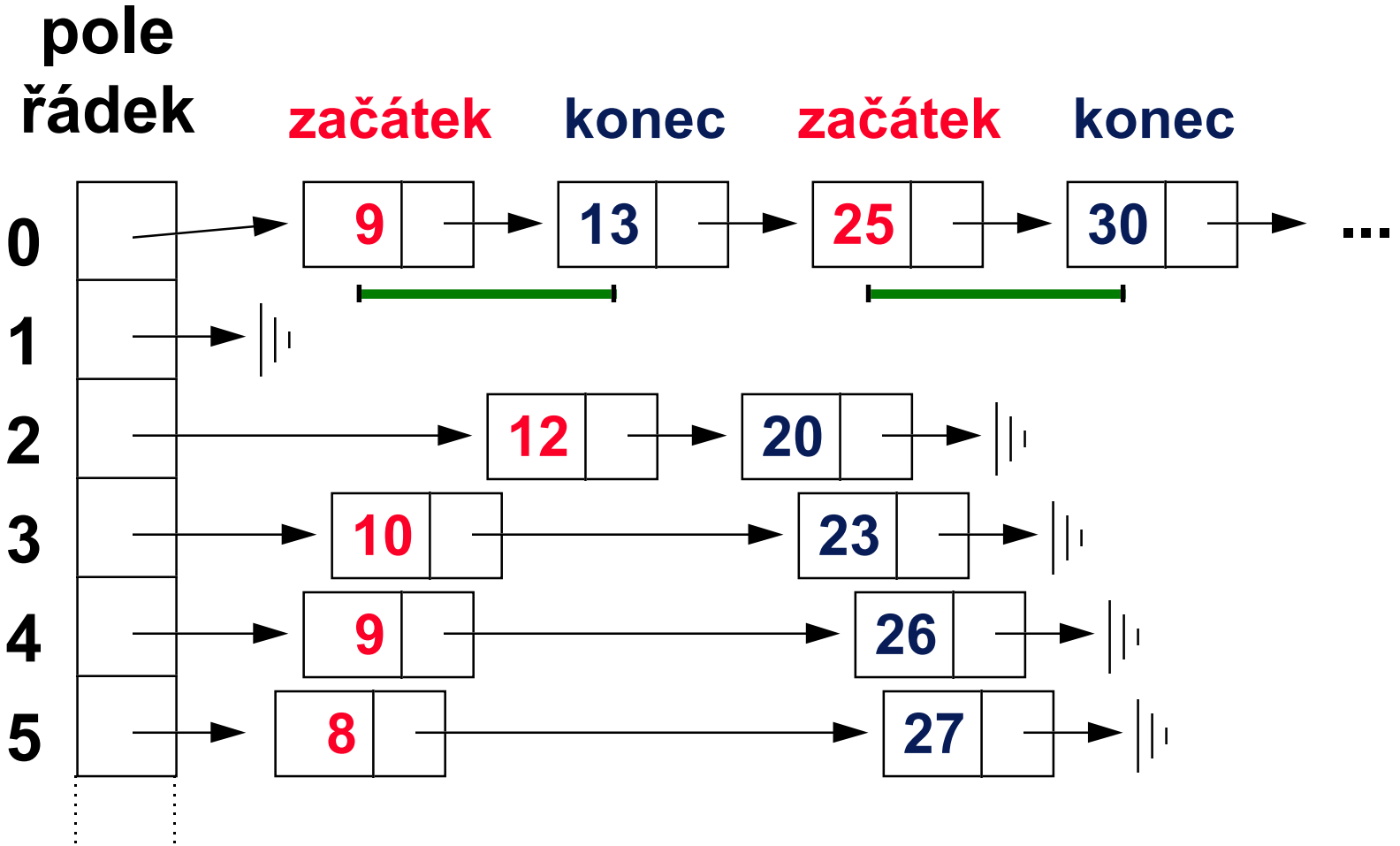
Řádkový seznam změn
("X-transition list")

- 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 ukládám 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 přidám/odstraním prvek **[0]**

- binární operace - **slévám seznamy změn** příslušných vstupních řádek:
  - nonekvivalence (**XOR**) je nejsnazší - dělám jenom slévání (a odstraňuji duplicitní záznamy)
  - u jiných operací zařazuji na výstup jen některé záznamy (stavové pomocné proměnné)

Množinová operace na 1 řádce

```
procedure MergeLists ( var L1, L2, Result : list );
var state, newstate, in1, in2 : boolean;
    x : integer;
begin
  Result.Init; state := false;        { výstupní seznam }
  in1 := false; in2 := false;         { vstupní stavy }
  while (not L1.Empty) and (not L2.Empty) do begin
      x := minimum(L1.First,L2.First);
      if x = L1.First then begin      { odebírám z L1 }
          in1 := not in1; L1.Get; end;
      if x = L2.First then begin      { odebírám z L2 }
          in2 := not in2; L2.Get; end;
      newstate := BooleanOperation(in1,in2);
      if newstate <> state then Result.Put(x);
      state := newstate;
    end;
  { dočtení zbytku vstupního seznamu }
end;
```

Konec

Další informace:

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

- **LAN** na Malé Straně:
  - **barbora\usr:\vyuka\pelikan\5**

Rastrové grafické formáty

KSVI MFF UK Praha

e-mail: Josef.Pelikan@mff.cuni.cz

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

Grafické formáty

- **rastrové**
  - obdélníková **matice pixelů** (“bitmapa”)
  - MS-Windows Bitmap (BMP), CompuServe (GIF), Interchange File Format (IFF), JFIF (JPG), PBM/PGM/PPM, Paintbrush (PCX), Macintosh (PICT), Sun Raster Format, Targa (TGA), Tagged Image File Format (TIFF), ...
- **vektorové**
  - posloupnost **objektů** nebo **příkazů** (škálování)
  - CorelDraw!™ (CDR), Computer Graphics Metafile (CGM), AutoCAD™ (DXF), HPGL, (Encapsulated) PostScript™, MS-Windows Metafile (WMF), DrawPerfect™ (WPG), ...

Rastrové grafické formáty

- **formát uložení barev**
  – barevná paleta, šedá škála, “true-color”, kanál “α”
- **komprese**
  – **bezeztrátová / ztrátová**
  – **RLE**: PCX, TGA; **LZW**: GIF; **JPEG**: JFIF, TIFF; ..
- **rozklad obrázku**
  – prokládané režimy (GIF, TGA, JPEG, ..)
- **negrafické informace** (TIFF, GIF)
- **závislost na HW**, přenositelnost (PCX, TGA)

PCX (ZSoft Corporation 1988)

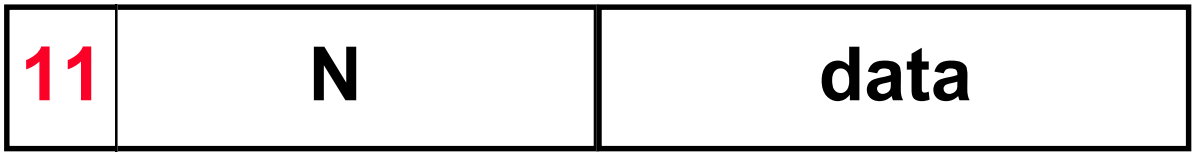
- **rastrový formát** (program PC Paintbrush)
- **omezený počet barev**
  - původně **2 ÷ 16** barev (EGA, VGA)
  - rozšíření: **256** barev, **“true-color”**
- **HW orientovaný** (IBM CGA, EGA, VGA)
  - prokládané bitové roviny (“color planes”): **RGBI**
- jednoduchá **RLE komprese**
  - bytově-orientovaná

Struktura PCX souboru

Hlavička souboru:

- **PCX verze** (2.5, 2.8 bez palety, 2.8 s paletou, 3.0)
- kódování: **RLE**
- počet bitových rovin, počet bitů/pixel/rovinu
- rozměr originální obrazovky
- okno obrázku: $[X_1,Y_1]$ - $[X_2,Y_2]$
- paleta: **16×3** byty (RGB z $[0,255]^3$), formát palety: barvy nebo šedé odstíny

RLE komprese v PCX



N × opakuj ‘data’

xx ≠ 11 (data<192)

obyčejný datový byte

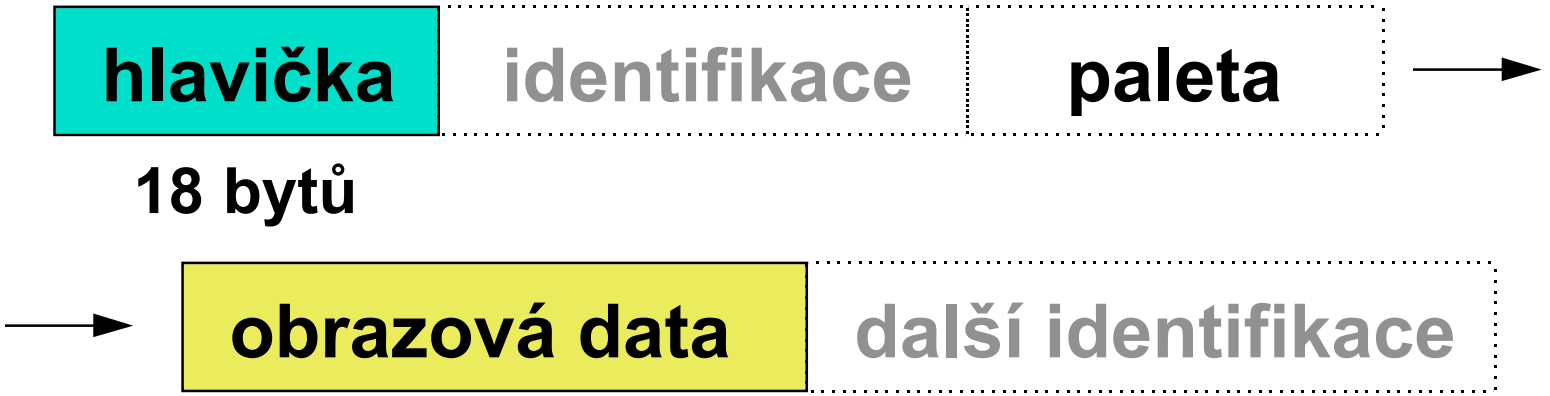
některé soubory se mohou dost prodloužit

– datový byte > 191 se musí kódovat pomocí “běhu”

Targa formát (Truevision Inc.)

- **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

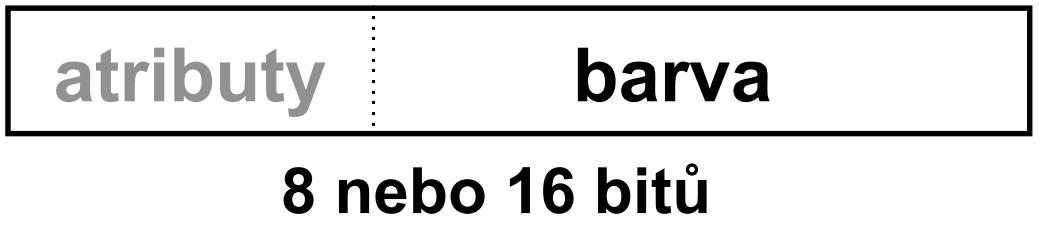


Hlavička souboru:

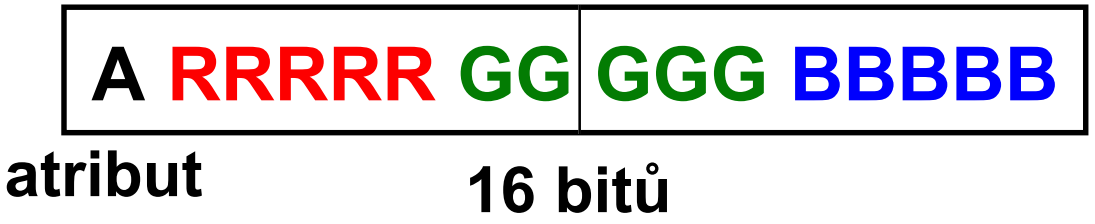
- barevný formát (paleta, RGB, RGBα, šedý obrázek)
- délka identifikace (ASCII text, maximálně 256 znaků)
- typ komprese: bez, RLE, Huffman, delta-modulace
- velikost obrázku: $[X_0,Y_0]$, šířka, výška
- orientace (shora, zdola), typ prokládání (1, 2, 4 fáze)

Formáty pixelu v TGA

paleta,
šedý obrázek

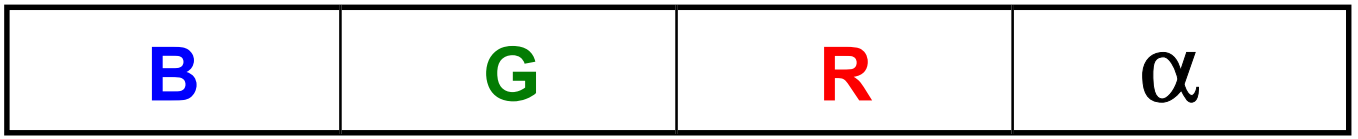


RGB 16

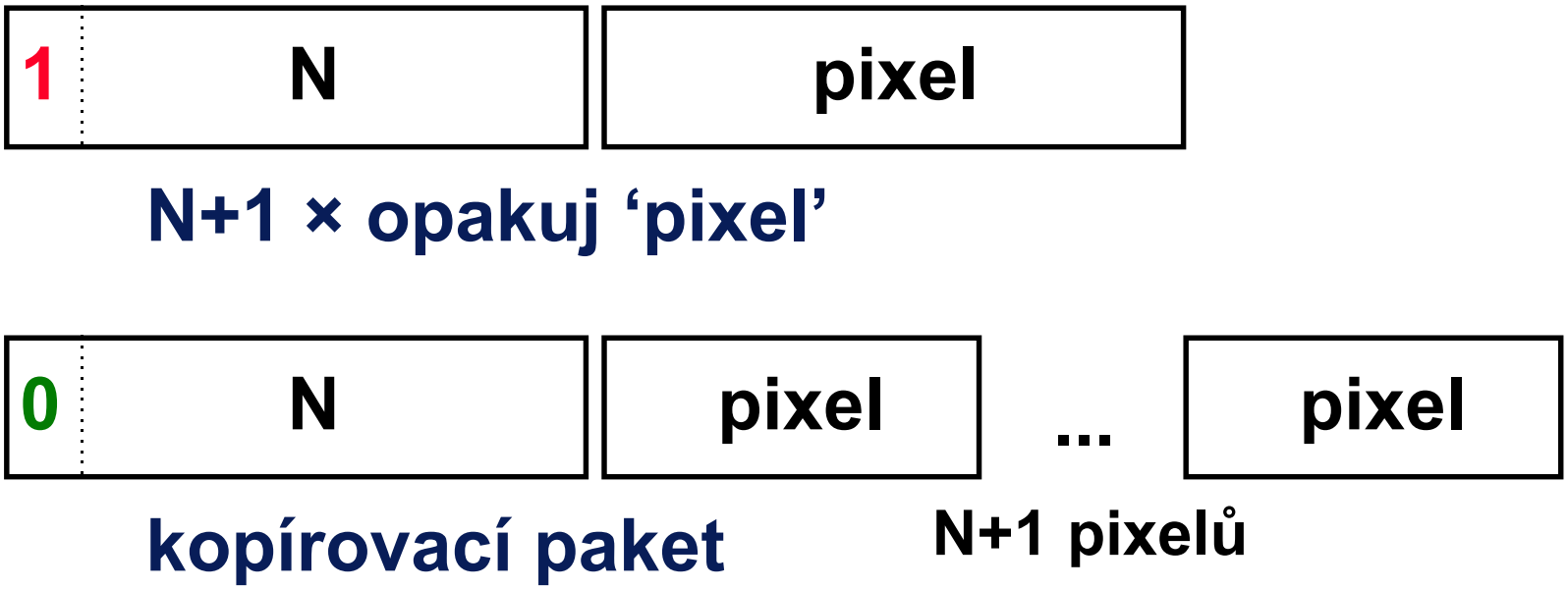


RGB 24

RGB 32



RLE komprese v TGA



maximální délka paketu je 128 pixelů

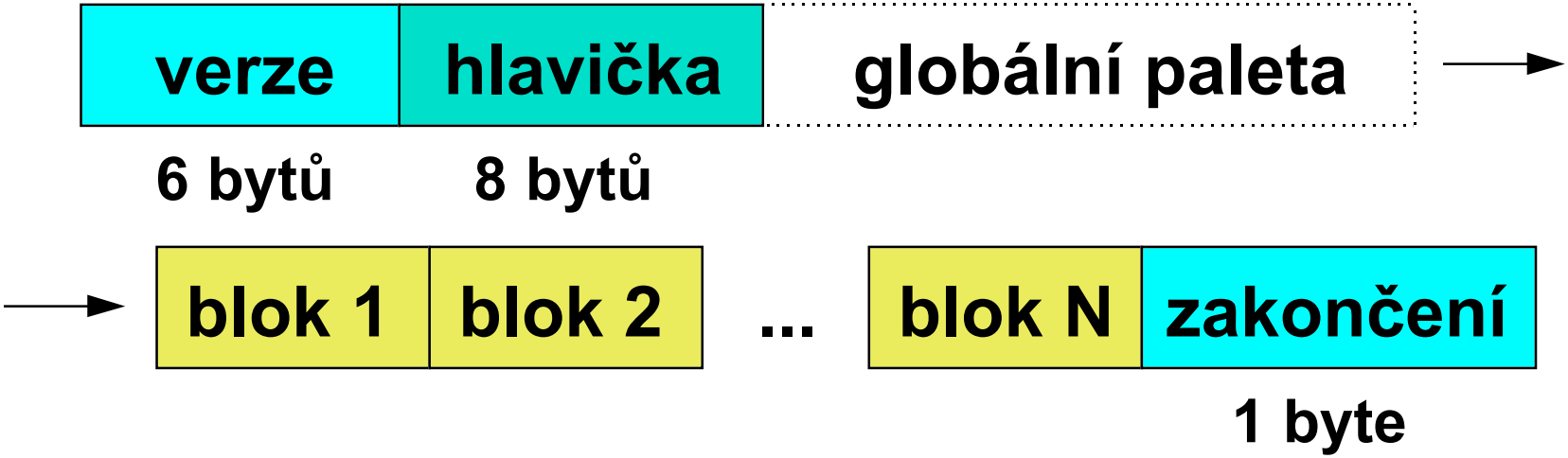
– prodloužení je v nejhorším případě 0.8 % délky souboru

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
  – patent UniSys Inc. (licenční poplatky od roku 1995)
- volitelné 8-fázové **prokládání** (přenos po síti)
- **další rozšíření**: více obdélníkových obrázků v jednom souboru, definice “průsvitné barvy”, interakce uživatele, výpis textu, aplikační neobrazové informace

Struktura GIF souboru



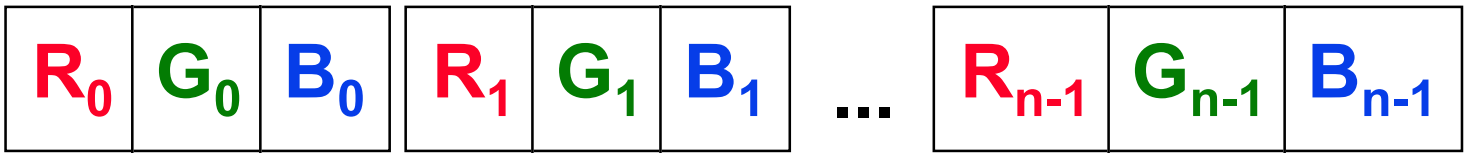
Verze: **‘GIF87a’** nebo **‘GIF89a’**

Globální hlavička:

- výška a šířka **virtuální obrazovky**, počet bitů na pixel, barva pozadí, “pixel aspect ratio” (4/1 až 1/4)
- **globální paleta**: délka, setřídění (významné barvy jsou na začátku)

Struktura GIF souboru

Paleta:



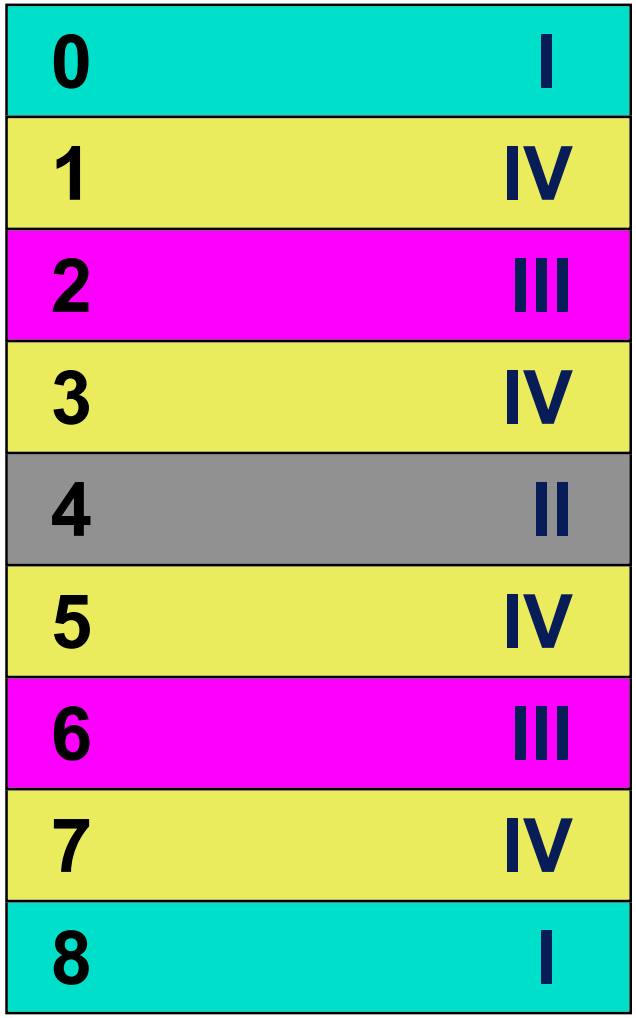
Bloky:

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

Obrazový blok

- **rozměry výřezu**
  - $[X_0, Y_0]$, šířka, výška
- nepovinná **lokální paleta**
  - počet barev, setřídění (podle důležitosti)?
- volba - **prokládání**
  - 8 fází kreslení obrázku (viz dále)
- **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**
- definice **transparentní barvy**

Blok komentáře (jakýkoliv text - pro uživatele)

Blok textu:

- výpis textu na obrazovku (neproporcionální font)

Aplikačně závislý blok:

- libovolná binární data (viz Fractint)

LZW komprese (Lempel-Ziv-Welch)

- **slovníková kompresní metoda**
  - **slovník**: obsahuje přiřazení **“fráze → kód”**
  - **fráze**: posloupnost pixelů
  - **kód**: **n**-bitové číslo (**3 ≤ n ≤ 12**)

- v průběhu kódování se mění
  - **slovník** (adaptivní přizpůsobení kódovaným datům)
  - **délka kódového slova “n”** se zvětšuje po jedné až do **12**

Schema kódovacího algoritmu

1. **inicializace**
   - do slovníku se uloží všechny jednopixelové fráze
   - **Act** := ‘‘ (prázdný řetězec)
2. přečti další pixel ze vstupu do **K**
3. je fráze **Act** + **K** uložena ve slovníku?
   - **Ano**: **Act := Act + K**
   - **Ne**: zapiš na výstup kód fráze **Act**
     přidej **Act** + **K** do slovníku
     Act := K
4. pokud neskončí vstup, opakuj kroky 2 a 3
5. zapiš na výstup kód fráze **Act**

Přidávání fráze do slovníku

- **počáteční nastavení slovníku**:
  - kódy **0** ÷ **2^p-1**: jednopixelové 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 **2^p**, zvětším **n** o 1
  - maximální hodnota **n** je **12** (**4094** fráze)
  - při přeplnění zakonzervuji slovník (méně často) nebo pošlu **"reset" kód** (reinicializace slovníku)

Konec

Další informace:

- **Kay D. C., Levine J. R.:** ***Graphics file formats***, MGWH, 1994
- **WWW:** http://www.cica.indiana.edu/graphics/
- **LAN** na Malé Straně:
  - **barbora\usr:\vyuka\pelikan\zdroj**
  - **..\formaty\gformats.arj**

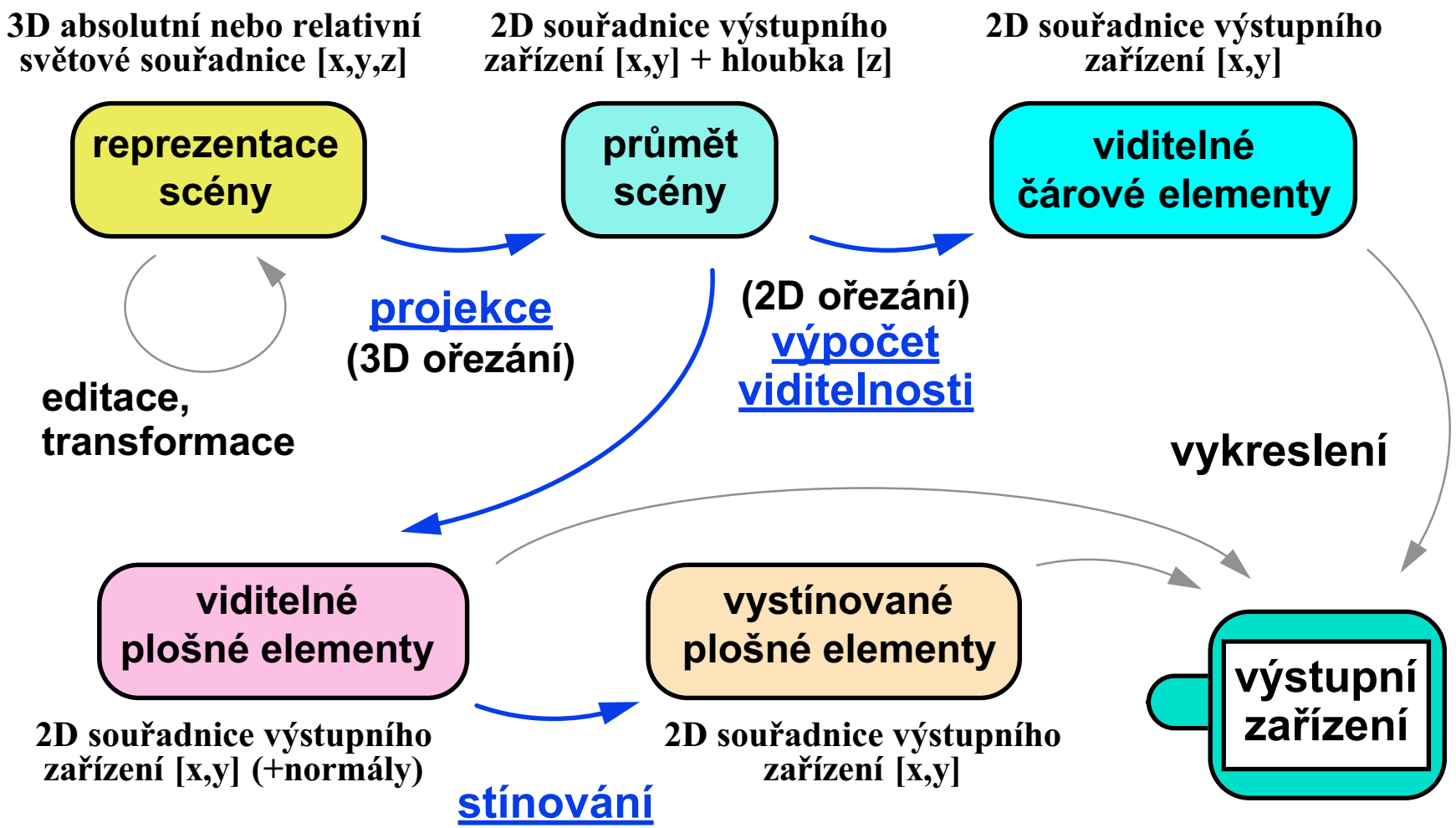
Úvod do 3D grafiky

KSVI MFF UK Praha

e-mail: Josef.Pelikan@mff.cuni.cz

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

3D grafický systém



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” pro výpočet viditelnosti)
  - různé úhly pohledu, perspektiva

Fáze zpracování scény

- **2D ořezání**
  - odstranění objektů ležících mimo kreslený výřez
- **<u>výpočet viditelnosti</u>**
  - odstranění zakrytých objektů nebo jejich částí
  - **čárová kresba** (kreslím jen obrysy ploch)
  - **plošná kresba** (vybarvuji vnitřek ploch)
- **<u>stínování</u>**
  - zlepšení prostorového vjemu napodobením osvětlení scény (někdy i vržené stíny)

Lineární transformace

KSVI MFF UK Praha

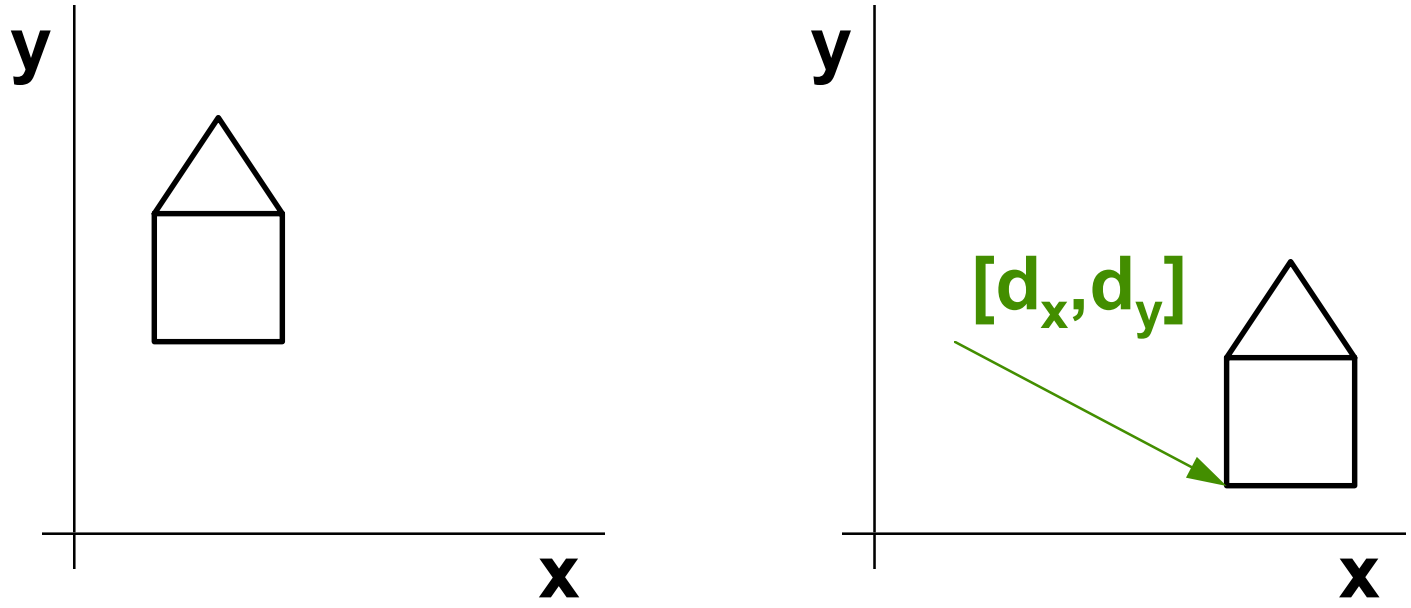
e-mail: Josef.Pelikan@mff.cuni.cz

WWW: http://cgg.ms.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 10^6 transformací najednou)
- **zvláštní operace**:
  - zřetězení jednoduchých transformací, výpočet inverzní transformace, ...

Posunutí v rovině



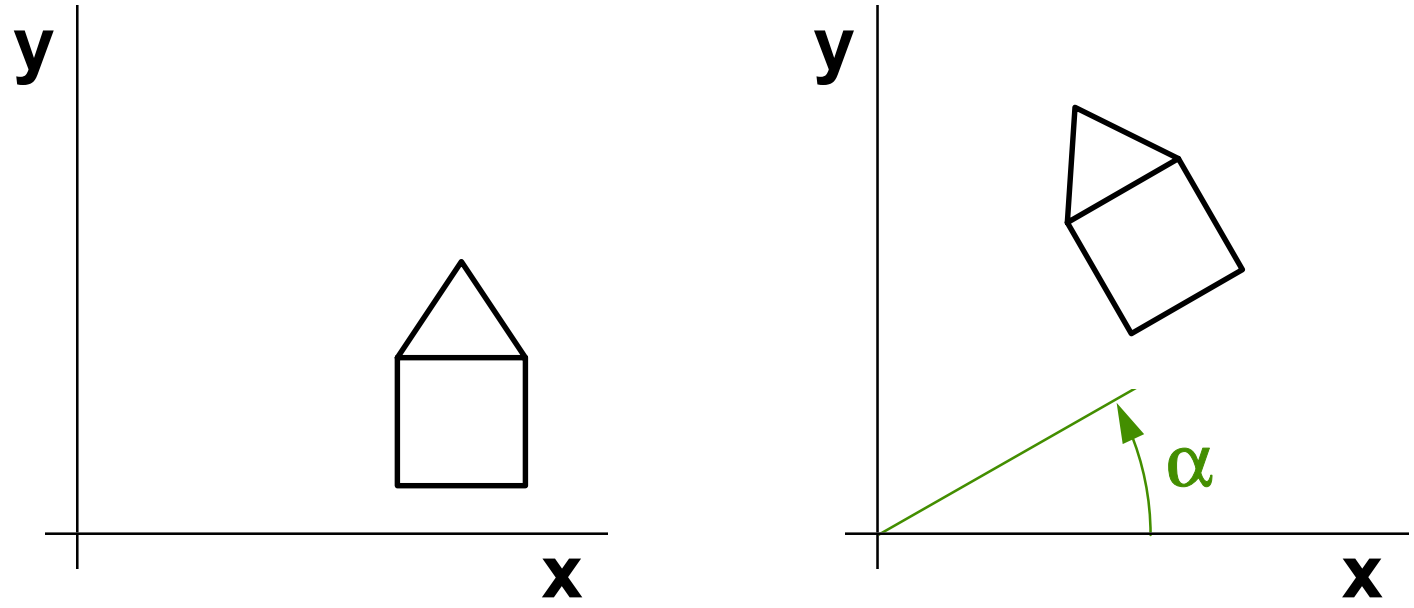
$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} + \begin{bmatrix} d_x & d_y \end{bmatrix}$$

Maticové transformace

- násobení vektoru souřadnic **maticí zprava**
  - kartézské souřadnice bodu **[x,y]** tvoří **řádkový vektor**
  - **transformační matice** je čtvercová (v rovině má rozměr 2×2)

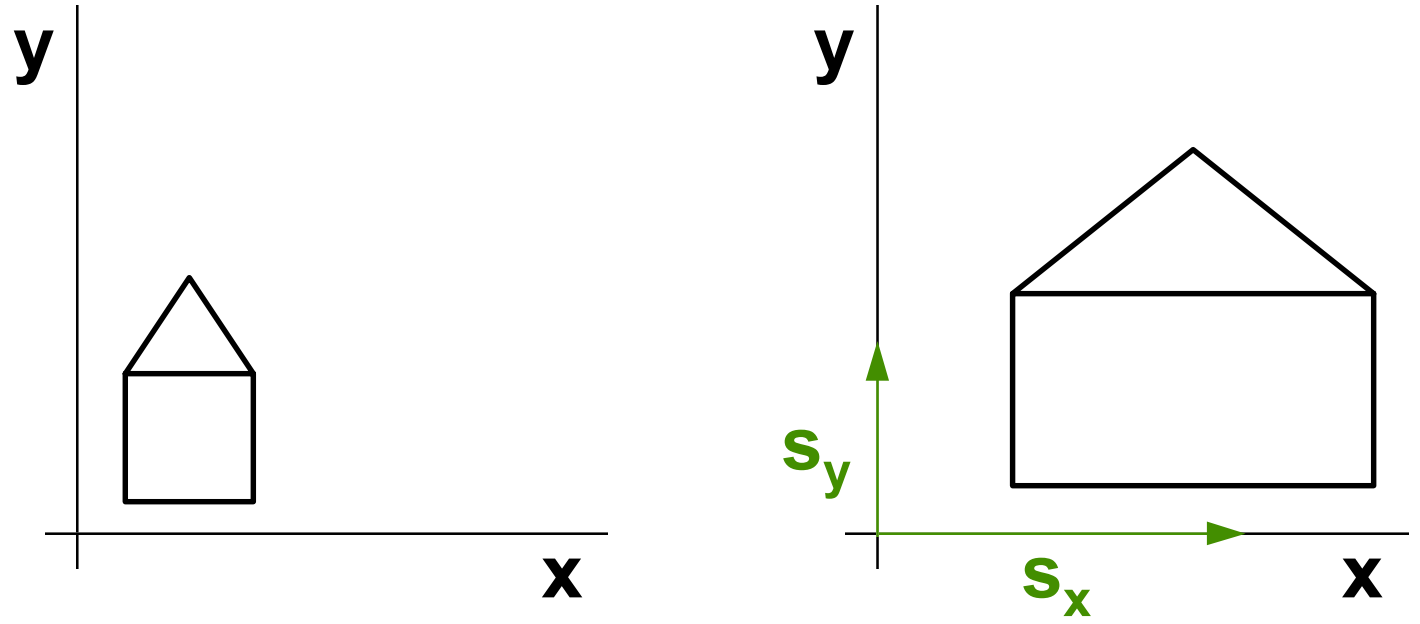
$$[x' \quad y'] = [x \quad y] \cdot \begin{bmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{bmatrix}$$

Otočení v rovině (kolem počátku)



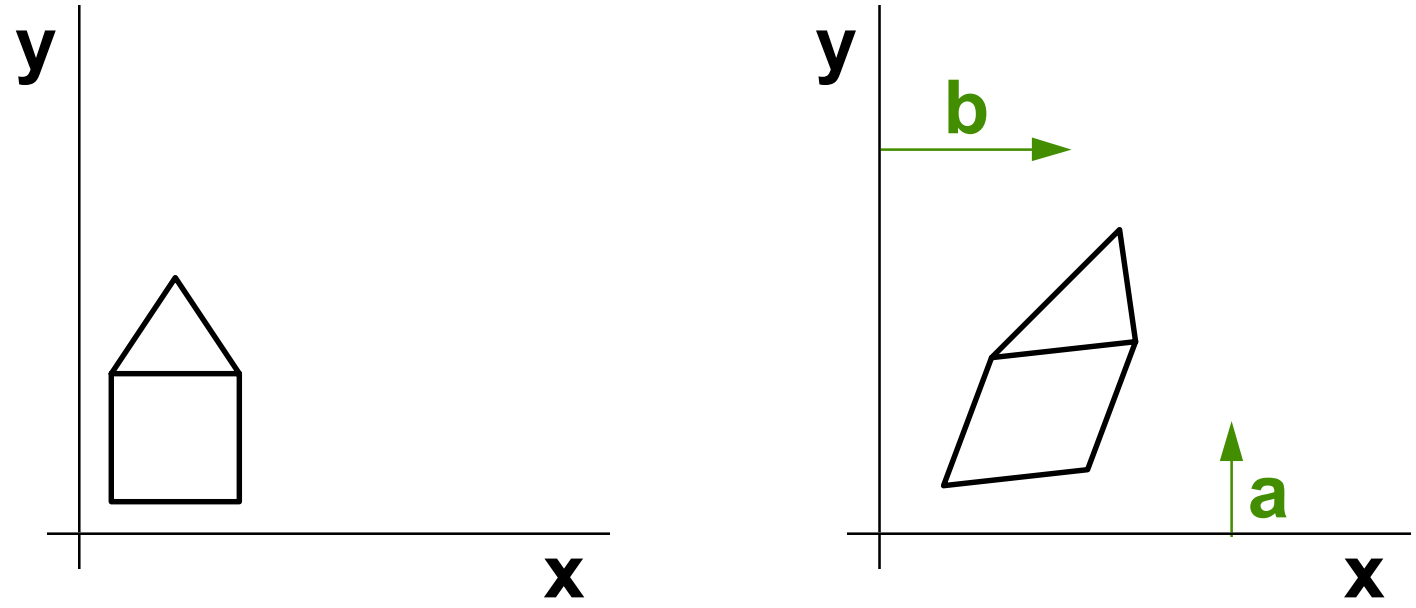
$$\mathbf{R}(\alpha) = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix}$$

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

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

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

soustava **(2)**

Soustava (1) má vždy (nekonečně mnoho) řešení,
soustava (2) má řešení jen pokud není $a_1 \cdot b_2 = a_2 \cdot b_1$

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:

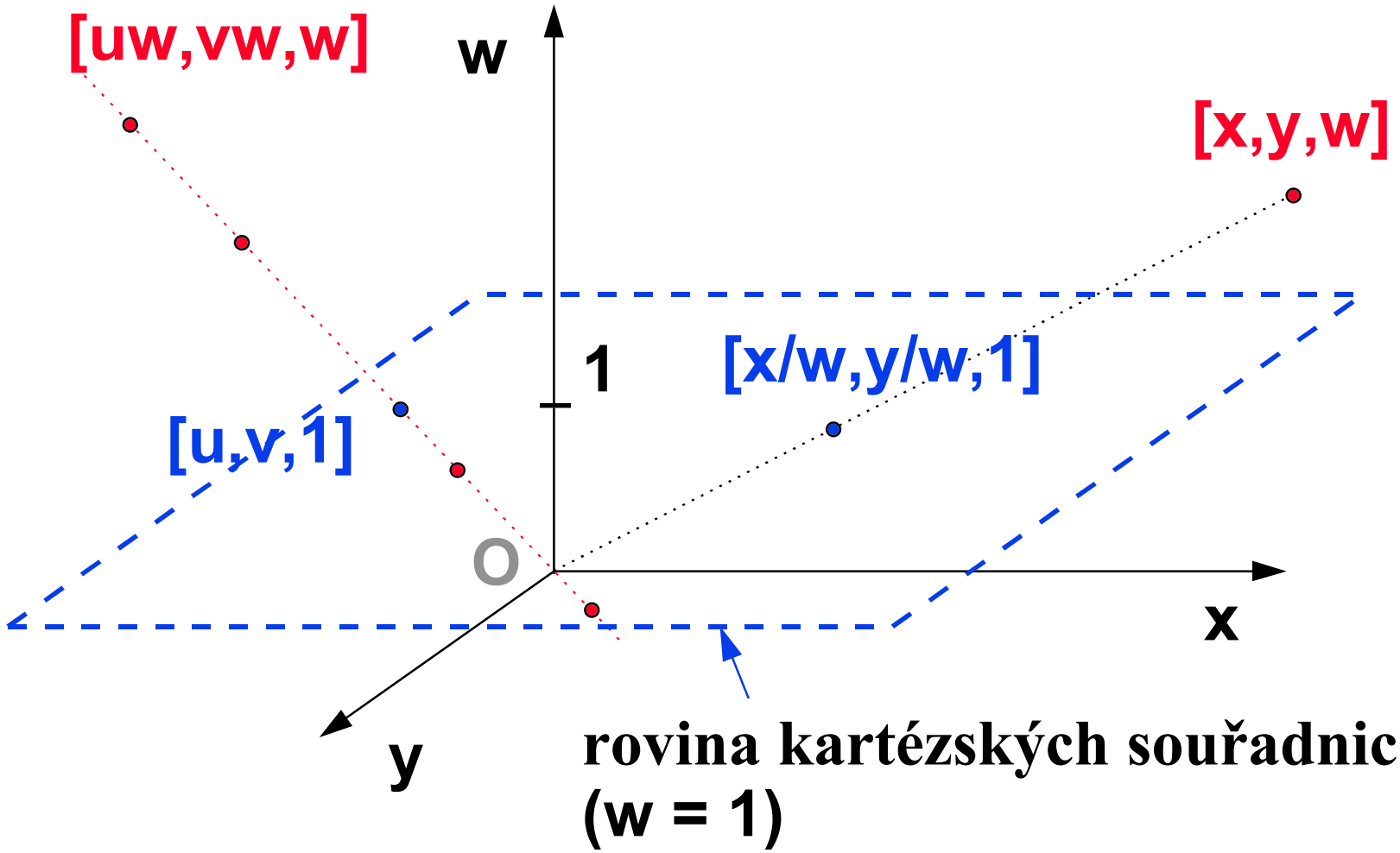
$$[\,\mathbf{x} \quad \mathbf{y}] \rightarrow [\,\mathbf{x} \quad \mathbf{y} \quad \mathbf{1}]$$

Homogenní na kartézské (jen vlastní body):

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

$$\mathbf{w} \neq \mathbf{0}$$

Geometrická představa



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

Zkosení
("shear")

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

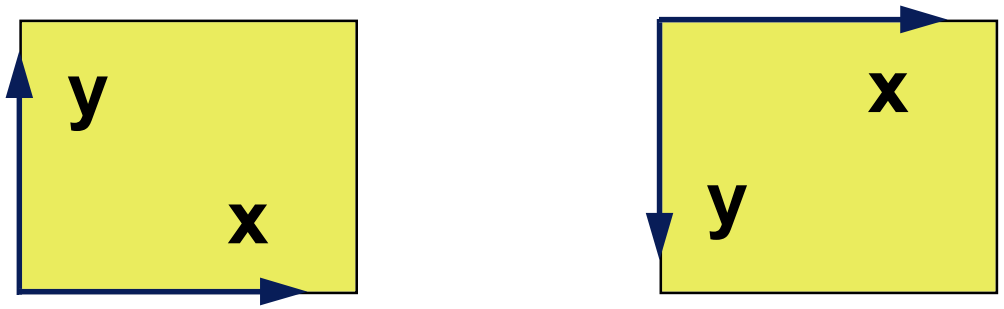
Složené transformace:

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

Otočení o úhel α kolem bodu **[x,y]**:

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

Transformace v průmětně



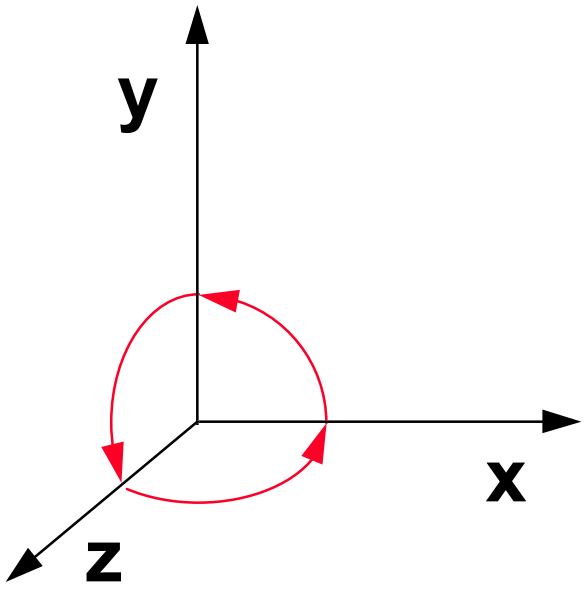
souřadné systémy na obrazovce

Převod reálných souřadnic do **souřadnic zobrazovaného okna:**

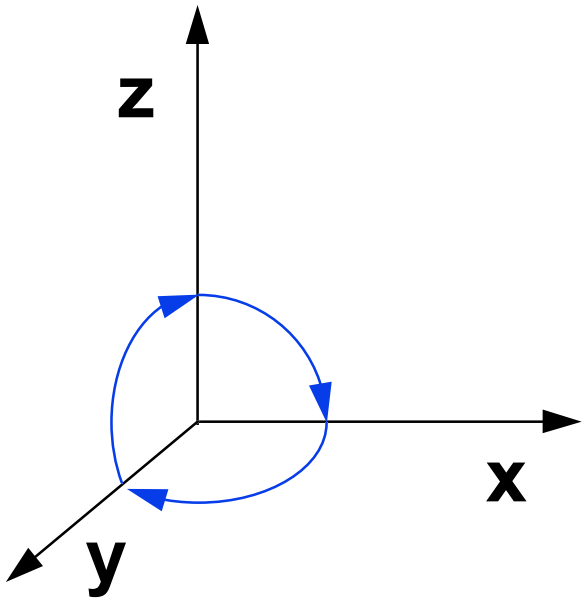
$$X_{int} = \text{round} (D_x + S_x * X_{re})$$

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

Prostorové souřadnice



levotočivý systém
(“right-handed”)



pravotočivý systém
(“left-handed”)

Homogenní souřadnice

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

$$[\,x \quad y \quad z \quad w\,] \rightarrow \left[\,\frac{x}{w} \quad \frac{y}{w} \quad \frac{z}{w}\,\right] \qquad (w \neq 0)$$

Maticová transformace:

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

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

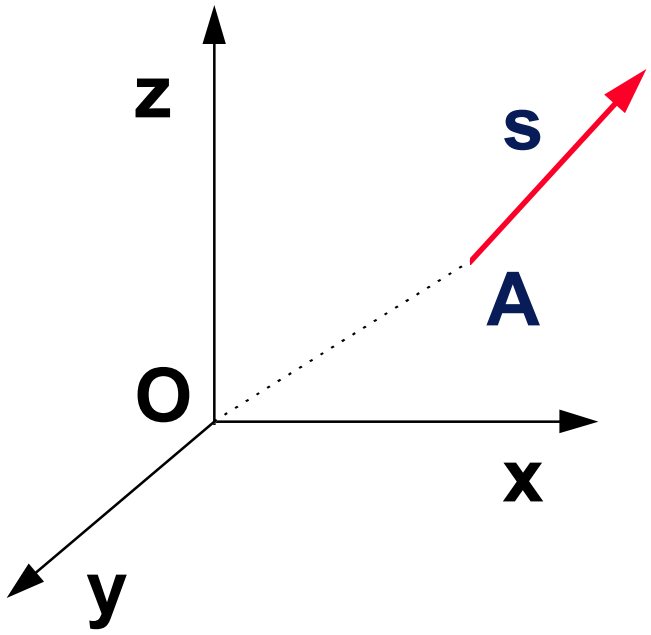
Otočení
kolem osy **y**

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

Otočení
kolem osy **z**

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

Přenos polopřímky do osy **z**



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

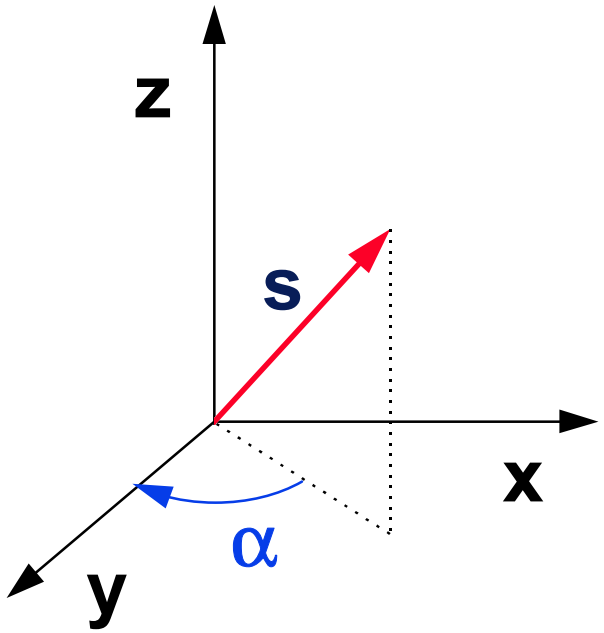
$$\mathbf{M} = \mathbf{T}(-\mathbf{A})$$

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

1. krok:

přenesení bodu **A** do počátku

Přenos polopřímky do osy **z**



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

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

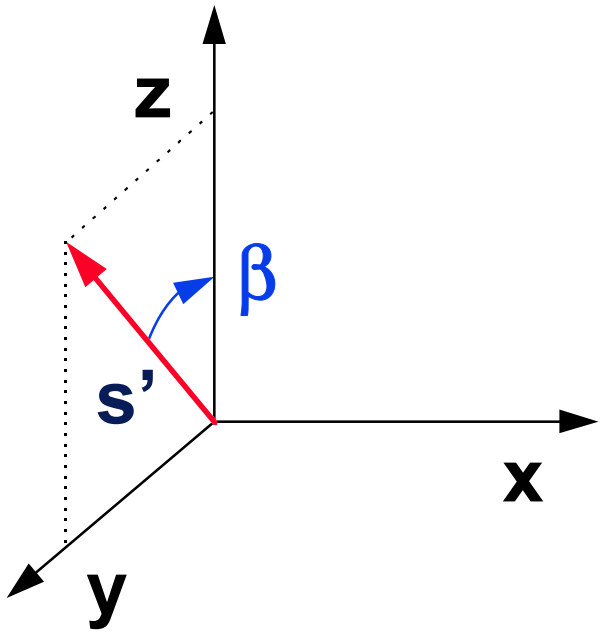
$$\cos\alpha = \frac{s_y}{\sqrt{s_x^2 + s_y^2}}$$

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

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**

$$\mathbf{M}(\mathbf{A},\mathbf{s}) = \mathbf{T}(-\mathbf{A}) \cdot \mathbf{R}_{\mathbf{z}}(\alpha) \cdot \mathbf{R}_{\mathbf{x}}(\beta)$$

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

Otočení kolem dané osy:

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

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:

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

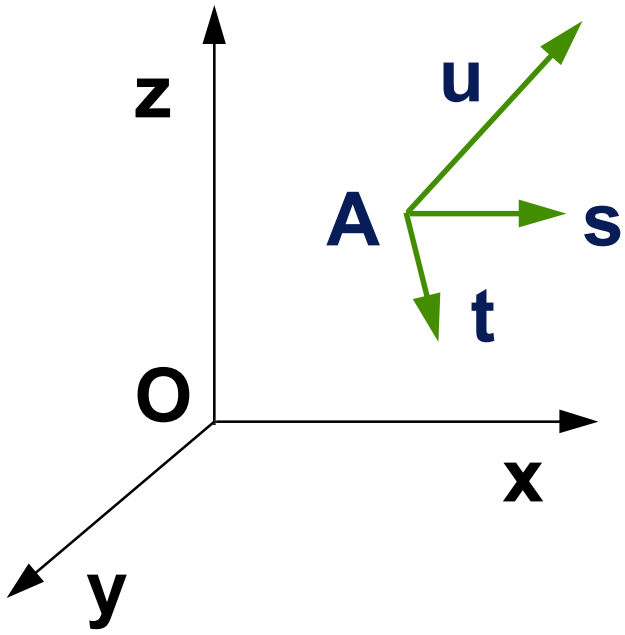
$$\mathbf{M}^{-1} = \boxed{\mathbf{C}^{-1}} \cdot \mathbf{B}^{-1} \cdot \mathbf{A}^{-1}$$

3. transpozice (ortonormální matice):

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

(ortonormální jsou např. všechny **rotační matice**)

Převod mezi souřadnými systémy



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

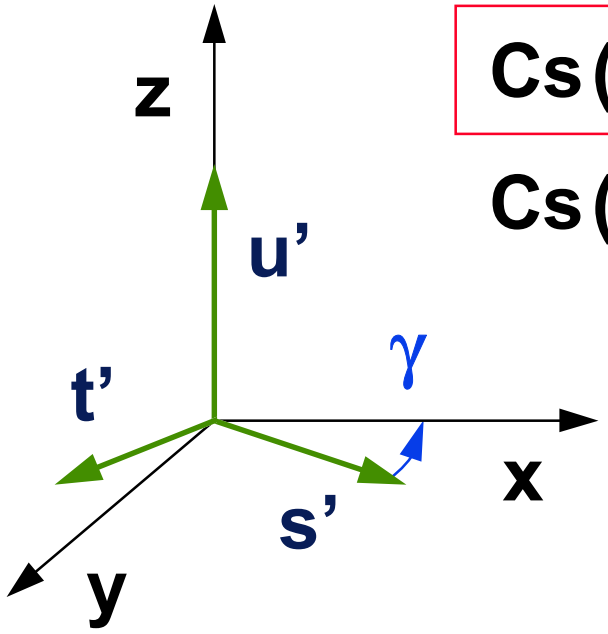
$$\mathbf{Cs} = \mathbf{M(A, u)}$$

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

1. krok:

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

Převod mezi souřadnými systémy



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

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

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

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

2. krok:

ztotožnění os **s’**$\rightarrow$**x** a **t’**$\rightarrow$**y** (otočením kolem **z=u’**)

Konec

Další informace:

- **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
- **LAN** na Malé Straně:
  - **barbora\usr:\vyuka\pelikan\6**

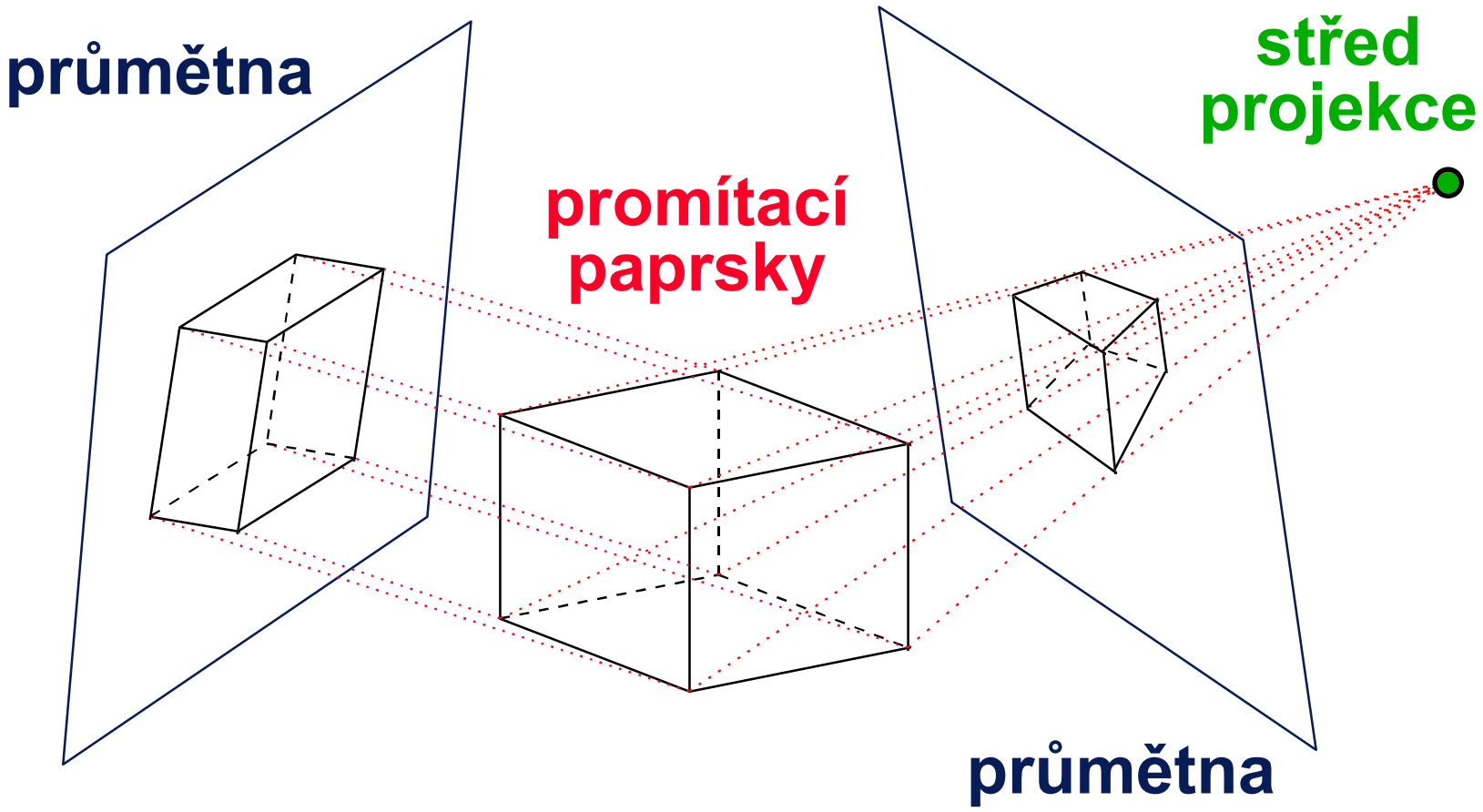
Promítání

KSVI MFF UK Praha

e-mail: Josef.Pelikan@mff.cuni.cz

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

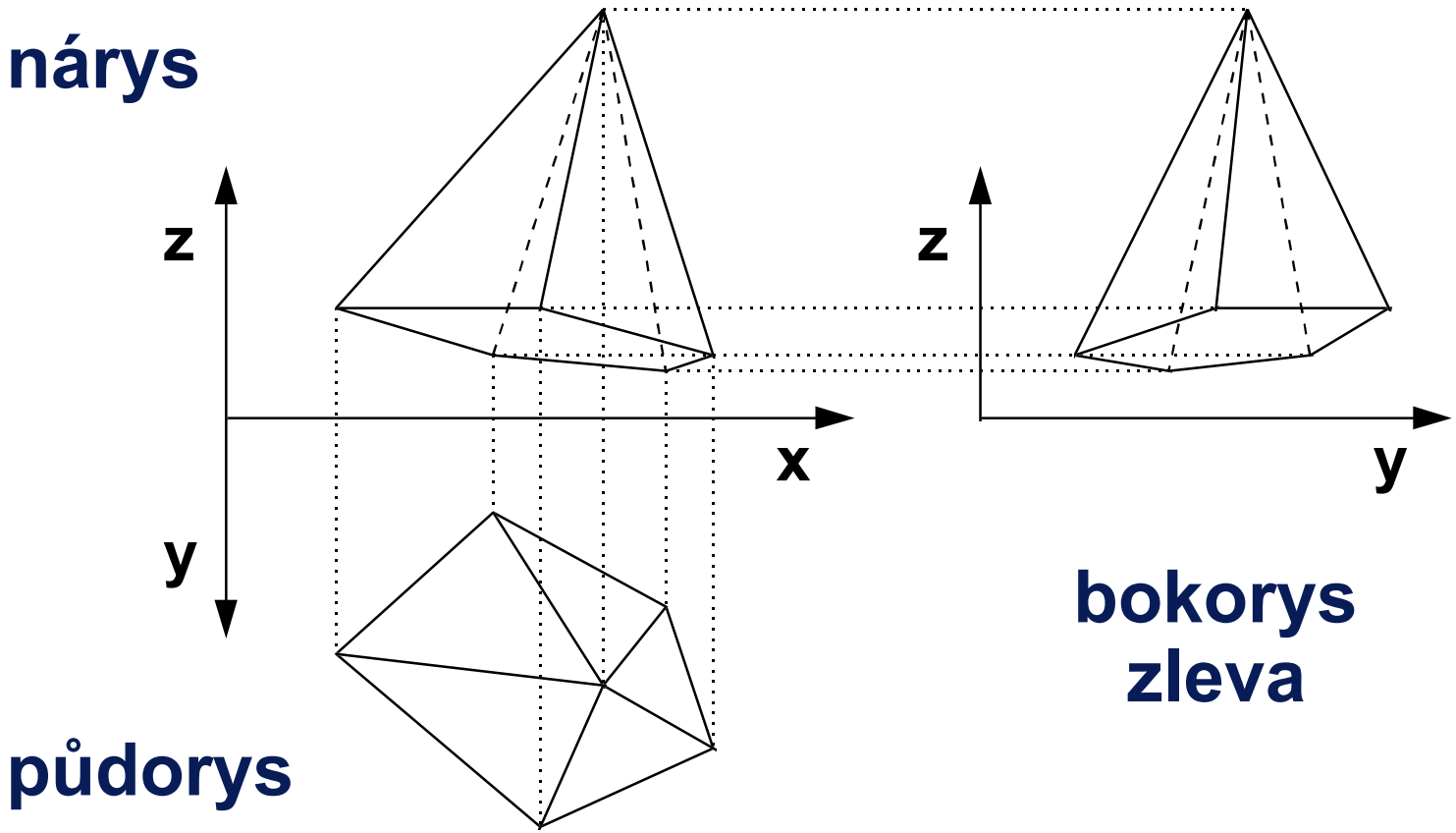
Základní pojmy



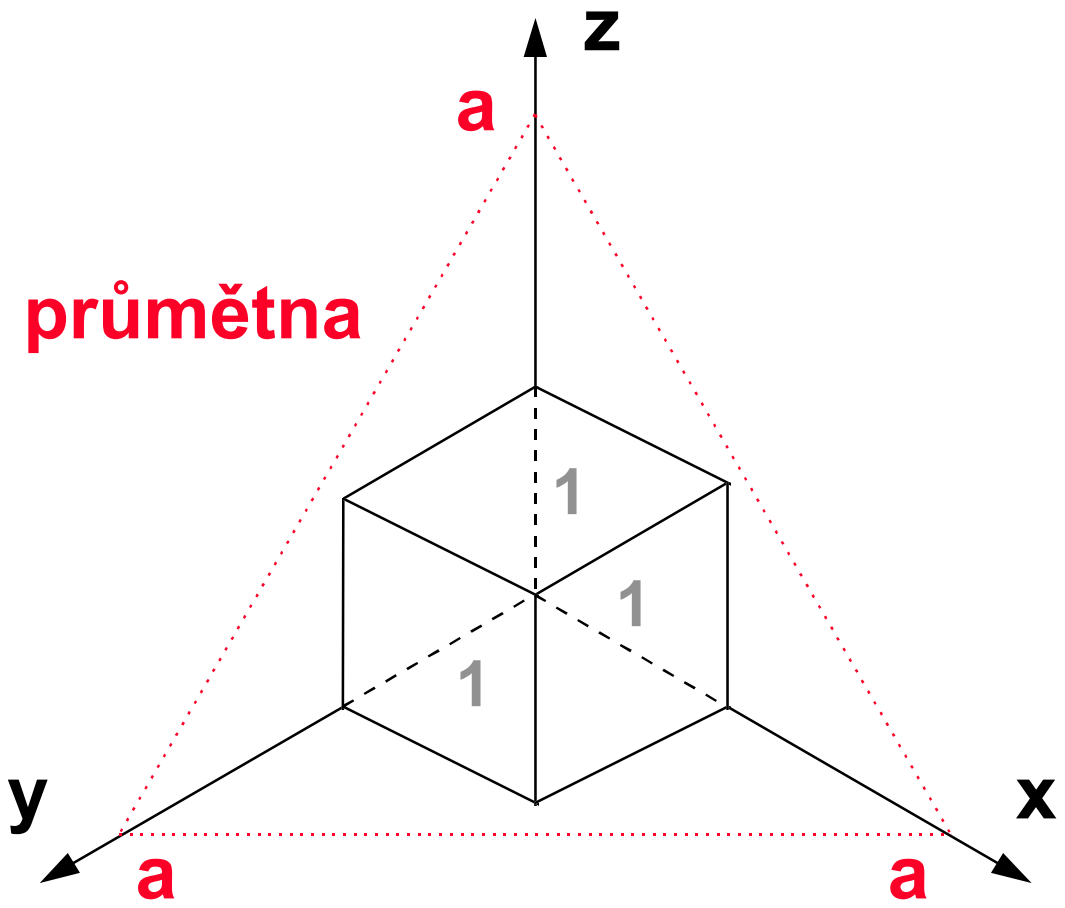
Klasifikace lineárních projekcí

- **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)

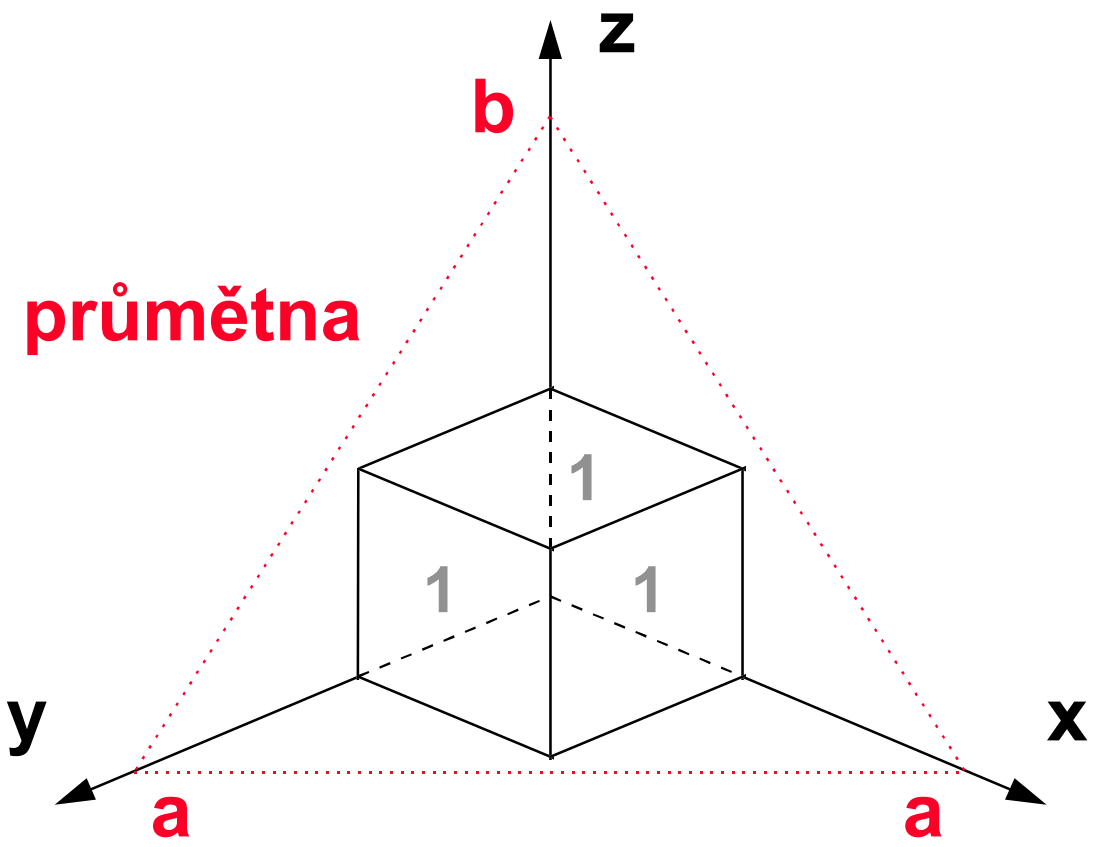
Mongeova projekce



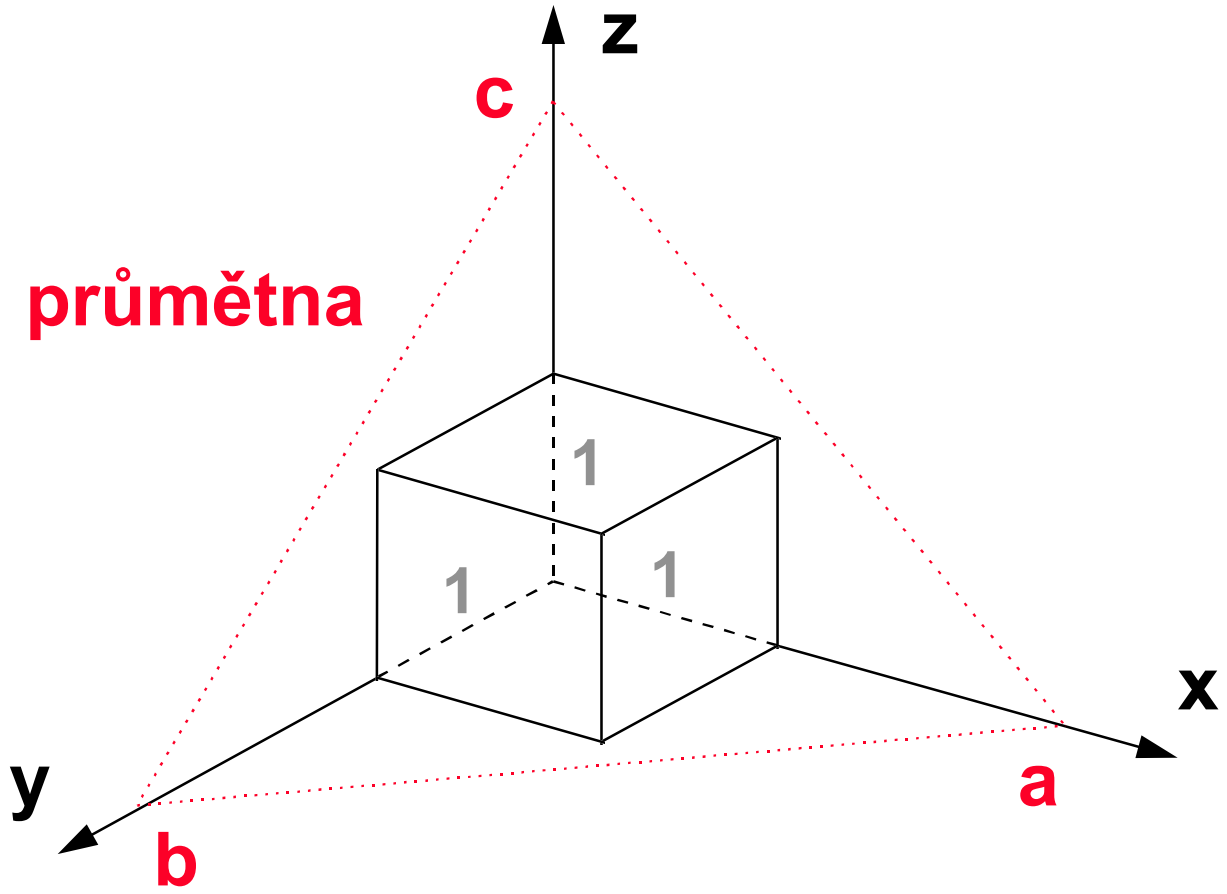
Axonometrie - isometrie



Axonometrie - dimetrie

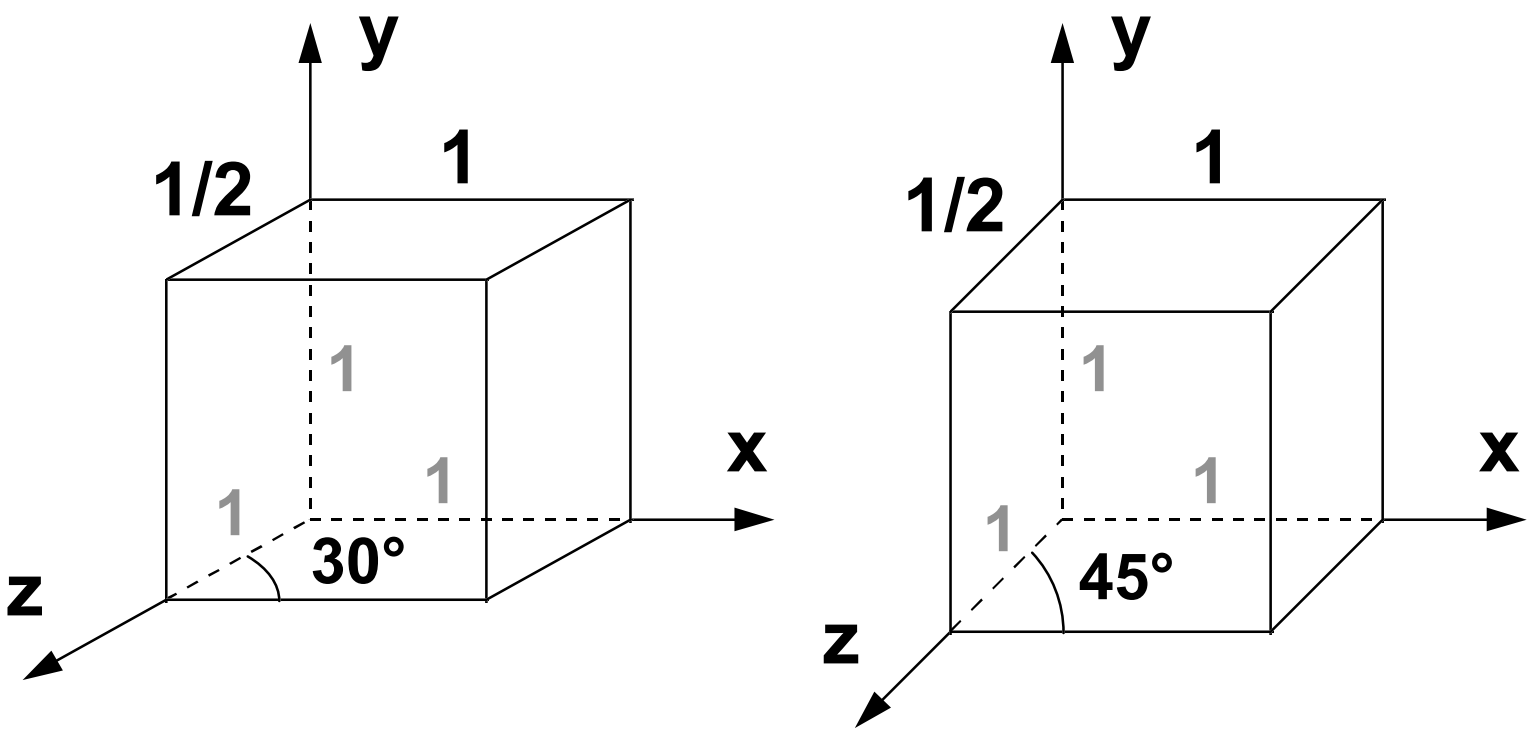


Axonometrie - trimetrie



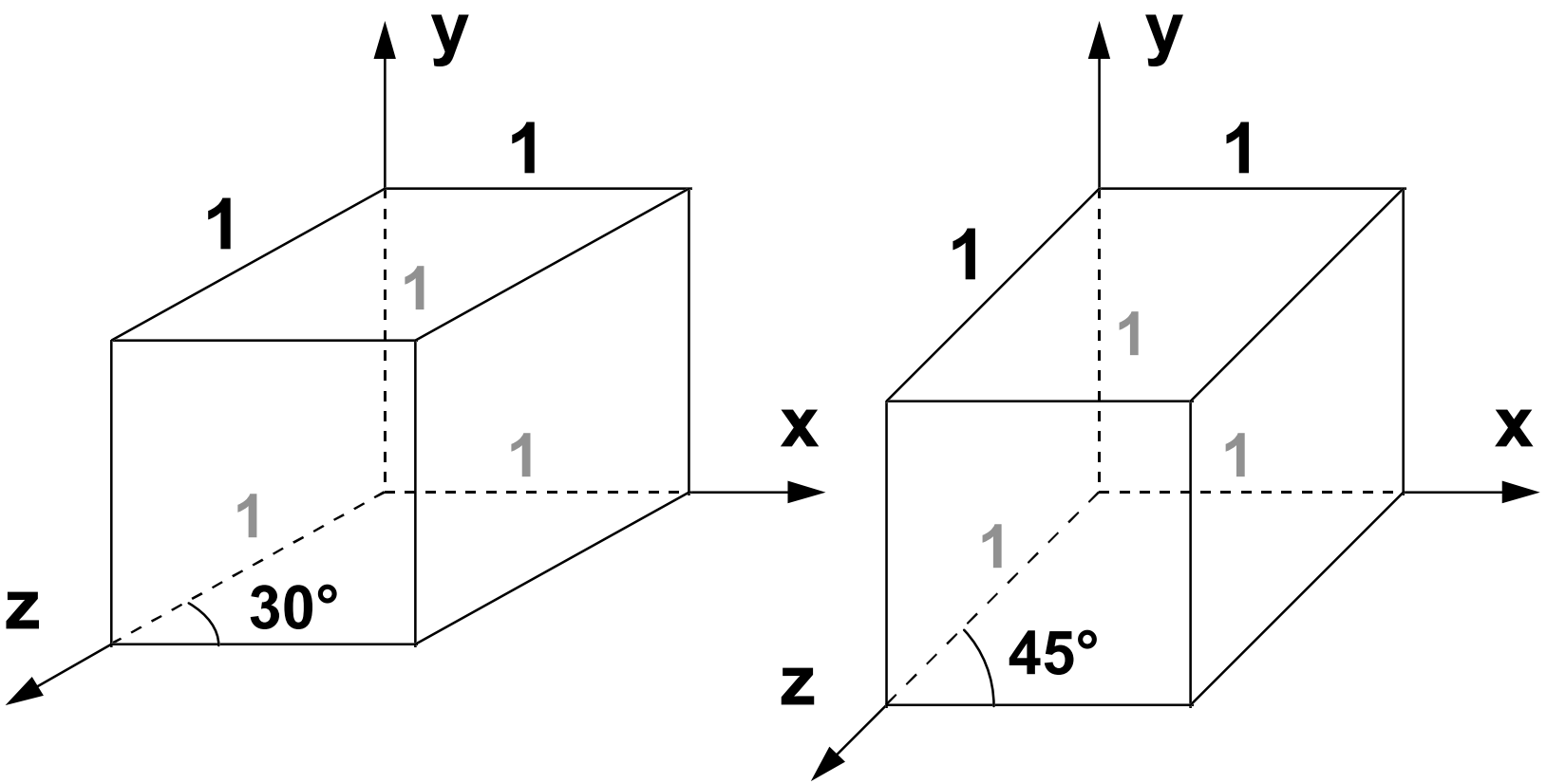
Kabinetní projekce

průmětna = xy



Kavalírní projekce

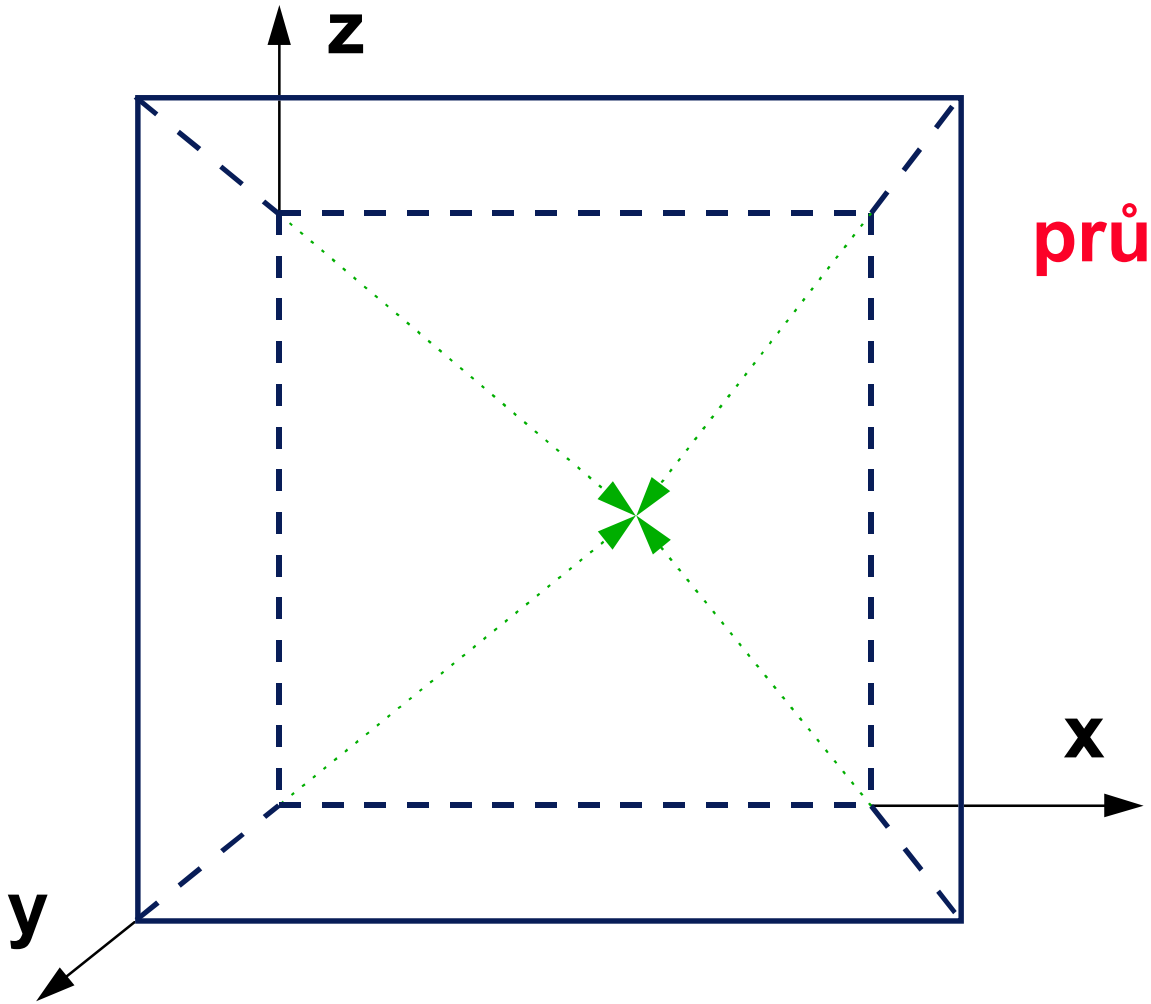
průmětna = xy



Klasifikace lineárních projekcí

- **perspektivní (středové) projekce**
  - 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

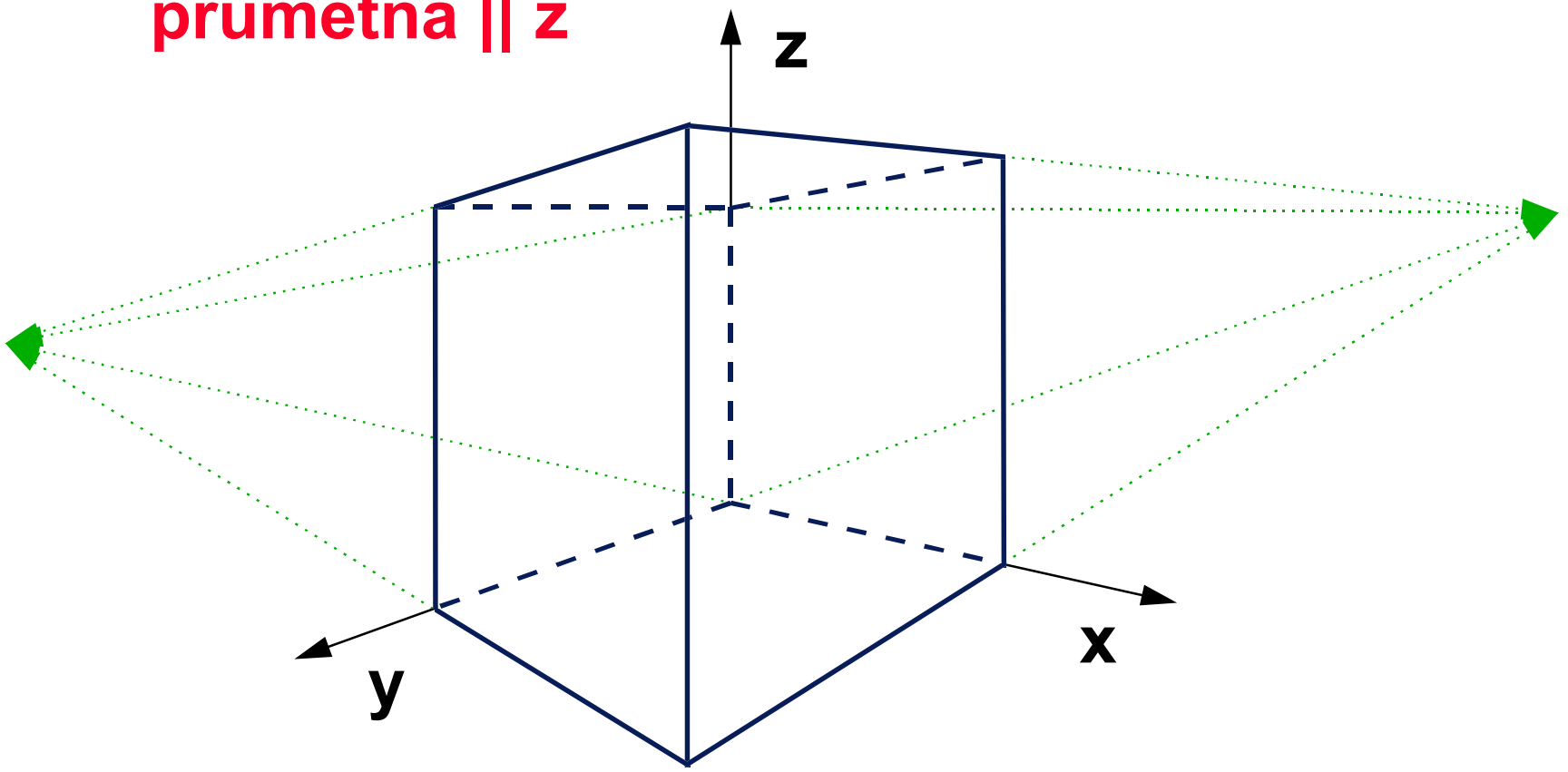


průmětna || xz

Klasifikace lineárních projekcí

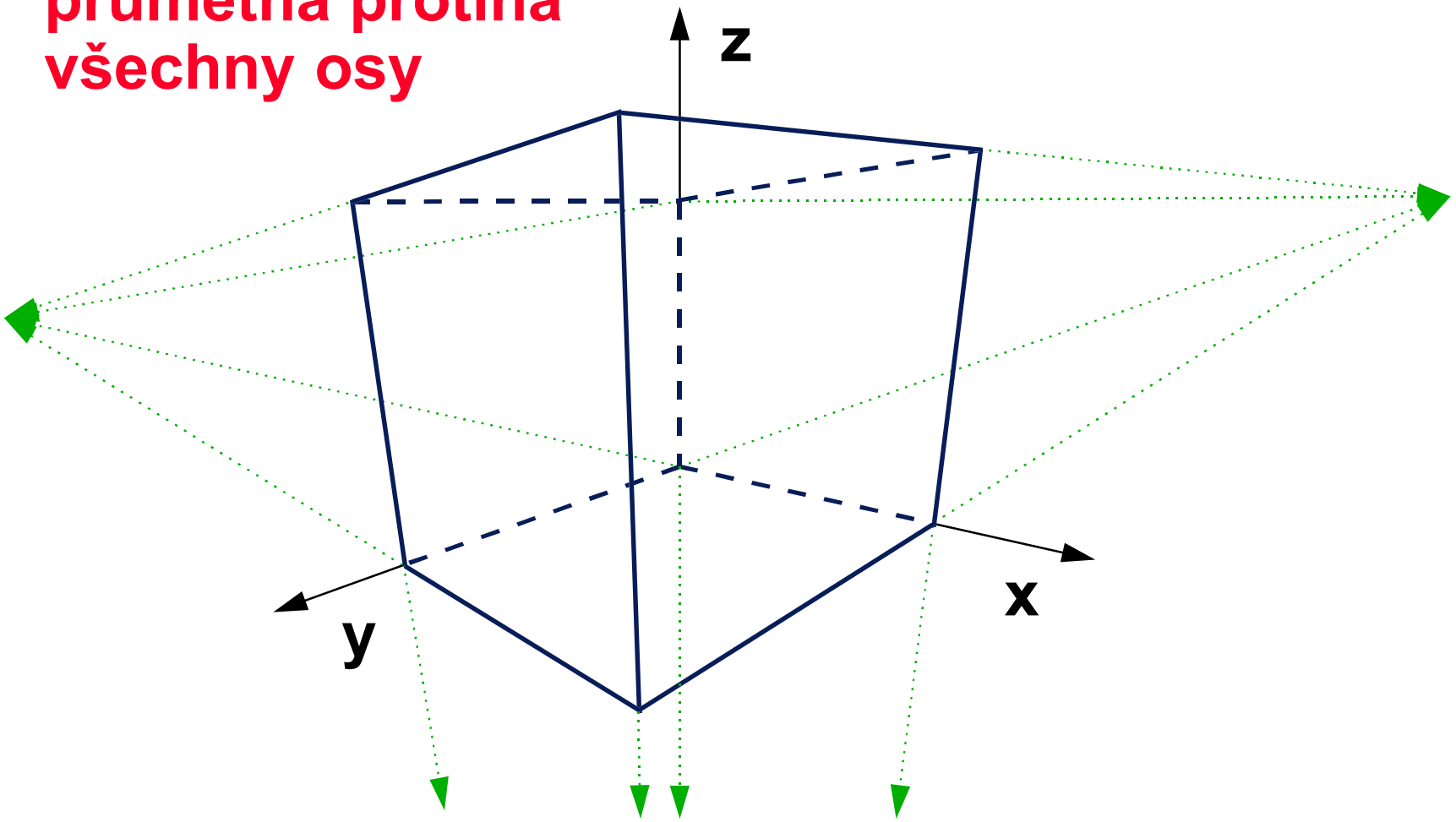
- **dvoubodová perspektiva**
  - průmětna je rovnoběžná s jednou souřadnou osou
  - rovnoběžky s ostatními osami se protínají ve dvou hlavních úběžnících
- **tříbodová perspektiva**
  - průmětna má zcela obecnou polohu
  - rovnoběžky se souřadnými osami se protínají ve třech hlavních úběžnících

Dvoubodová perspektiva



Tříbodová perspektiva

průmětna protíná všechny osy



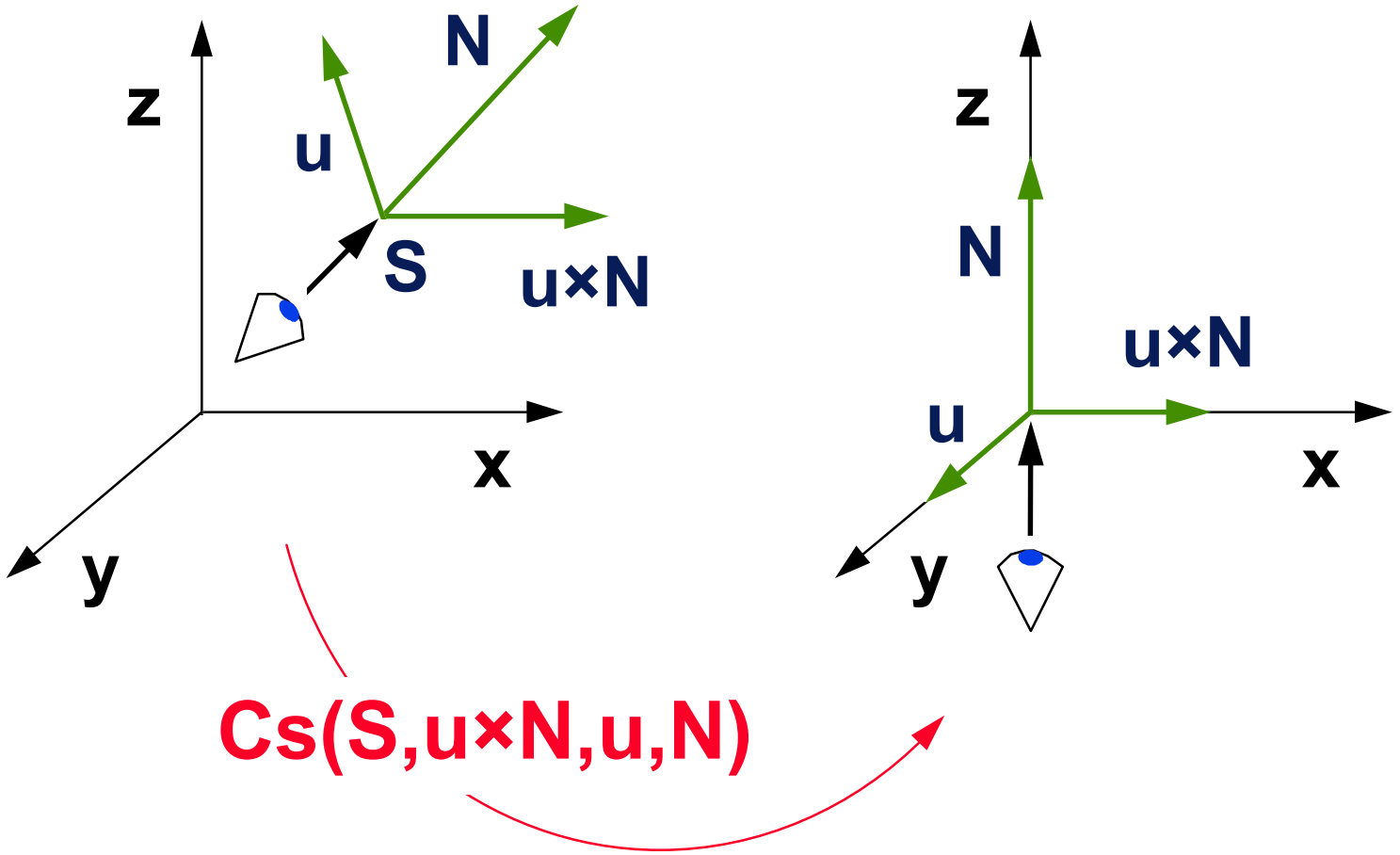
Implementace kolmé projekce

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

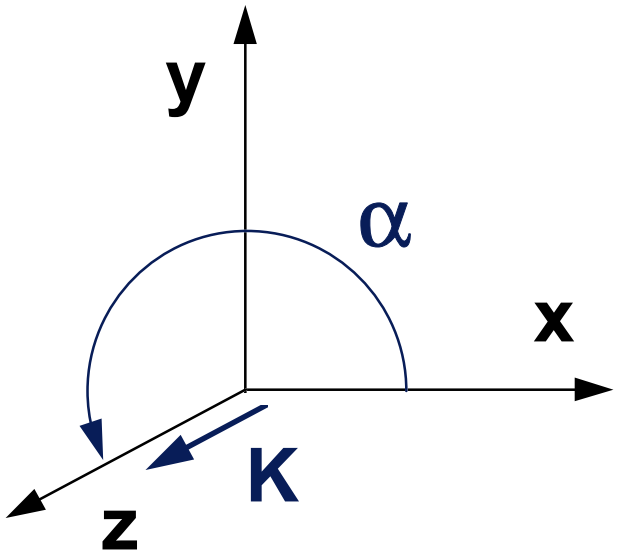
- **základní pohledy** (půdorys, nárys, bokorys)
  - pouze permutace složek **x**, **y** a **z** (s příp. změnou znaménka)

- **obecná kolmá projekce** (axonometrie)
  - **směr pohledu** (normálový vektor průmětny): **N**
  - **svislý vektor**: **u**
  - převedení do základního pohledu: **Cs(S,u×N,u,N)**

Obecná kolmá projekce



Implementace kosoúhlé projekce



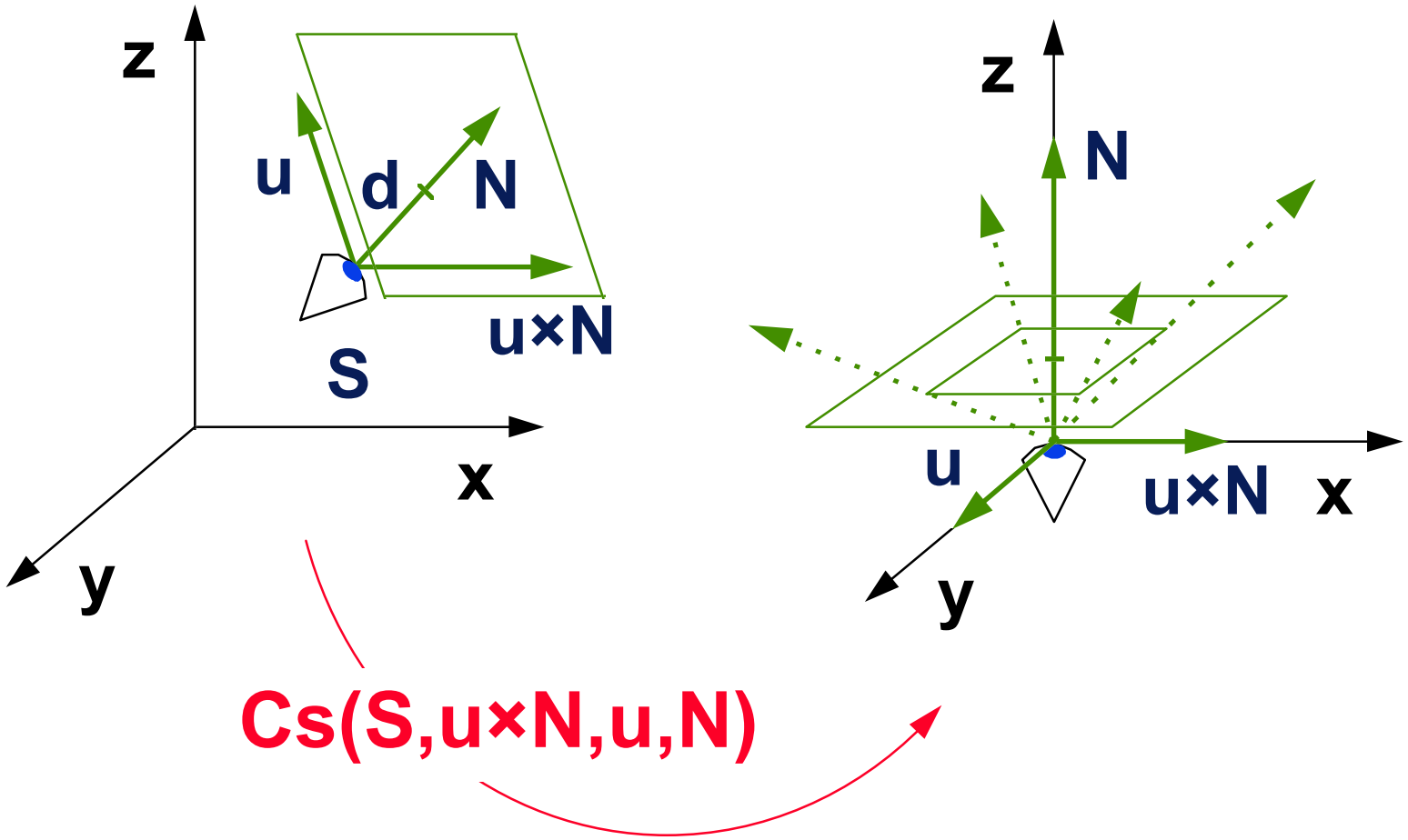
průmětna: xy
koeficient zkrácení: K
úhel průmětu osy z: α

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

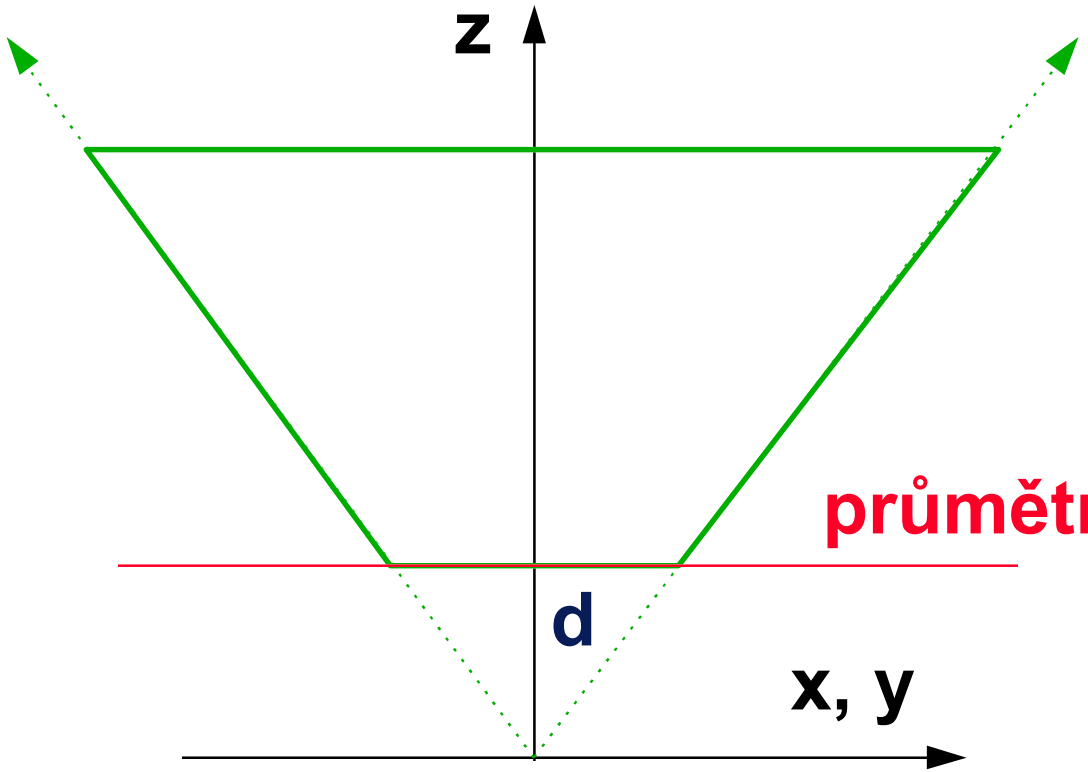
Implementace středové projekce

- **obecná perspektivní projekce**:
  - **střed projekce**: **S**
  - **směr pohledu** (normálový vektor průmětny): **N**
  - **vzdálenost průmětny** od středu projekce: **d**
  - **svislý vektor**: **u**
- **promítací transformace**:
  - převedení do **základní polohy** (střed projekce do počátku, směr pohledu do osy **z**): **Cs(S,u×N,u,N)**
  - **perspektivní projekce**: např. **[x· d/z, y· d/z, z]**

Převedení do základní polohy



Perspektivní transformace



Nezachovává linearitu útvarů!

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

Transformace lineárních útvarů

- **perspektivní transformace úsečky Per**:
  - je zřejmé, že **neplatí** rovnost

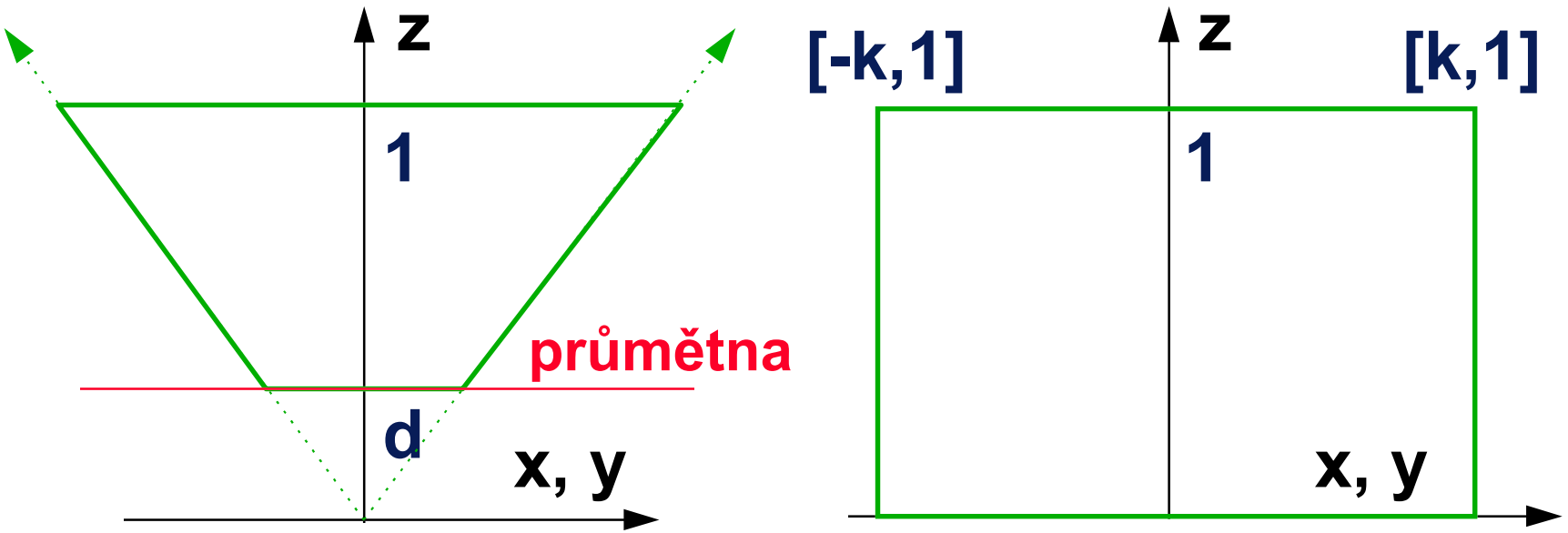
    Per(A + t· [B - A]) = Per(A) + t· [Per(B) - Per(A)]

- 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:

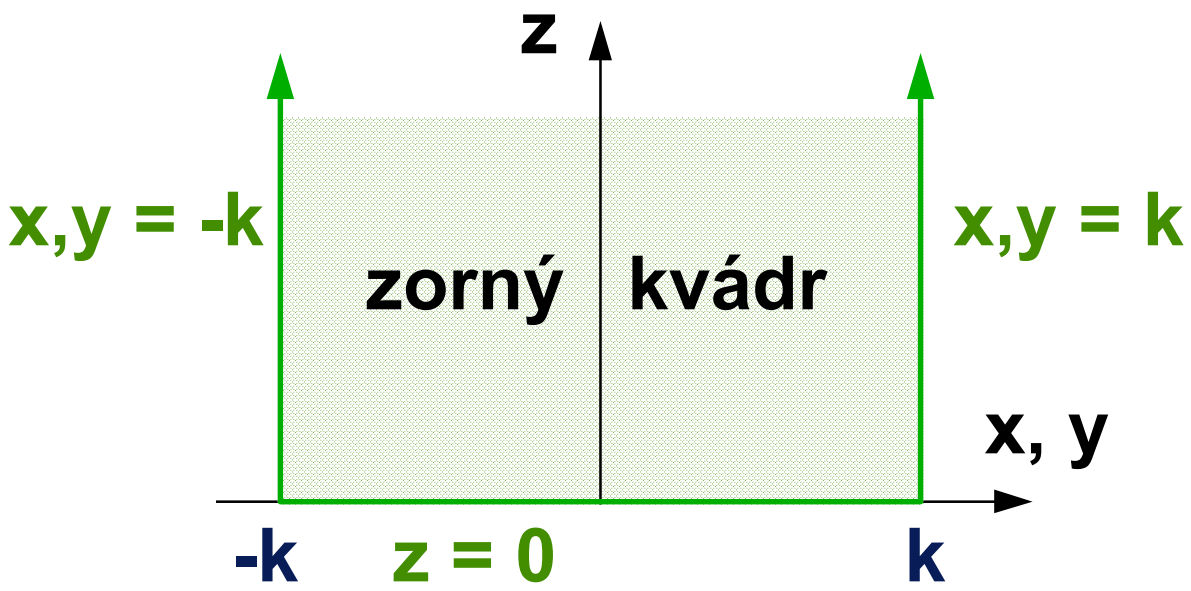
    $\mathbf{C(u)_{z} = Per(A)_{z} + u\cdot [Per(B)_{z} - Per(A)_{z}]}$

Zachování linearity



$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{1-d} & 1 \\ 0 & 0 & \frac{-d}{1-d} & 0 \end{bmatrix}$$

4D ořezávání



hraniční nadroviny:

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

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

Konec

Další informace:

- **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
- **LAN** na Malé Straně:
  - **barbora\usr:\vyuka\pelikan\6**

Reprezentace 3D scény

KSVI MFF UK Praha

e-mail: Josef.Pelikan@mff.cuni.cz

WWW: http://cgg.ms.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**

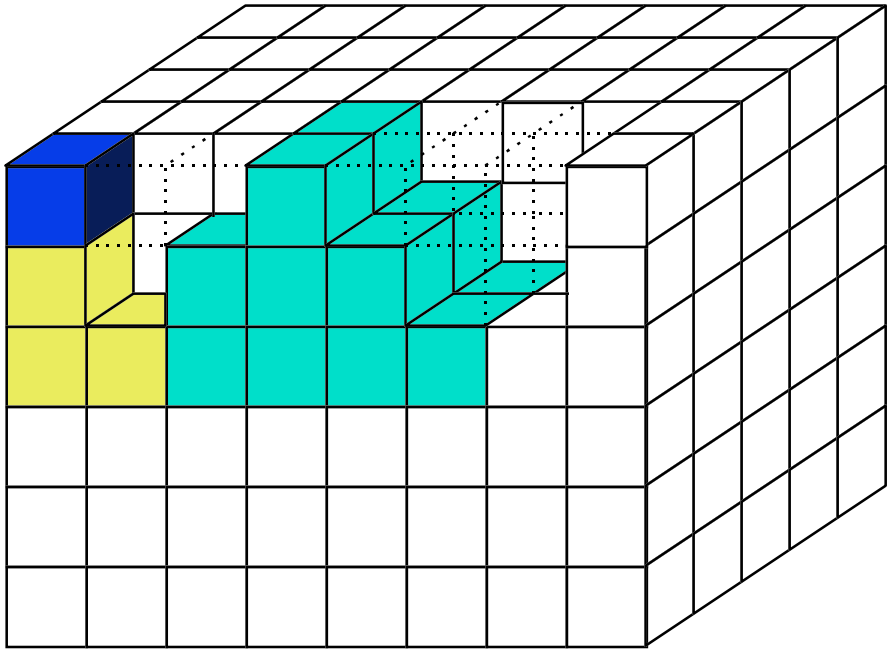
Objemové reprezentace

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

Buněčný model

voxel

“**vol**ume **el**ement”



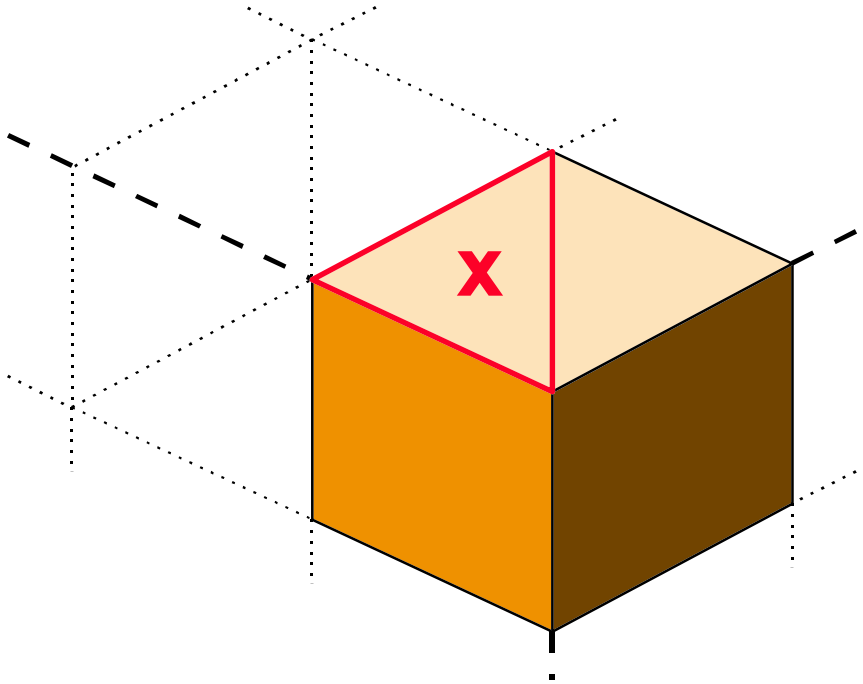
pole k×l×m voxelů

jednobitová varianta: 0 - nic, 1 - těleso
vícebitová varianta: 0 - nic, n > 0 - těleso číslo **n**

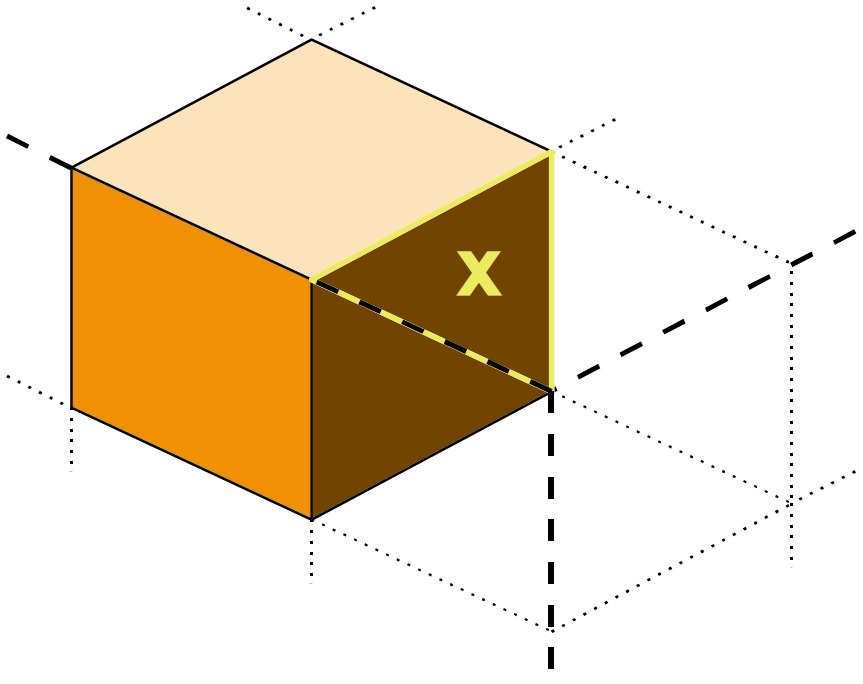
Zobrazování buněčného modelu

- **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

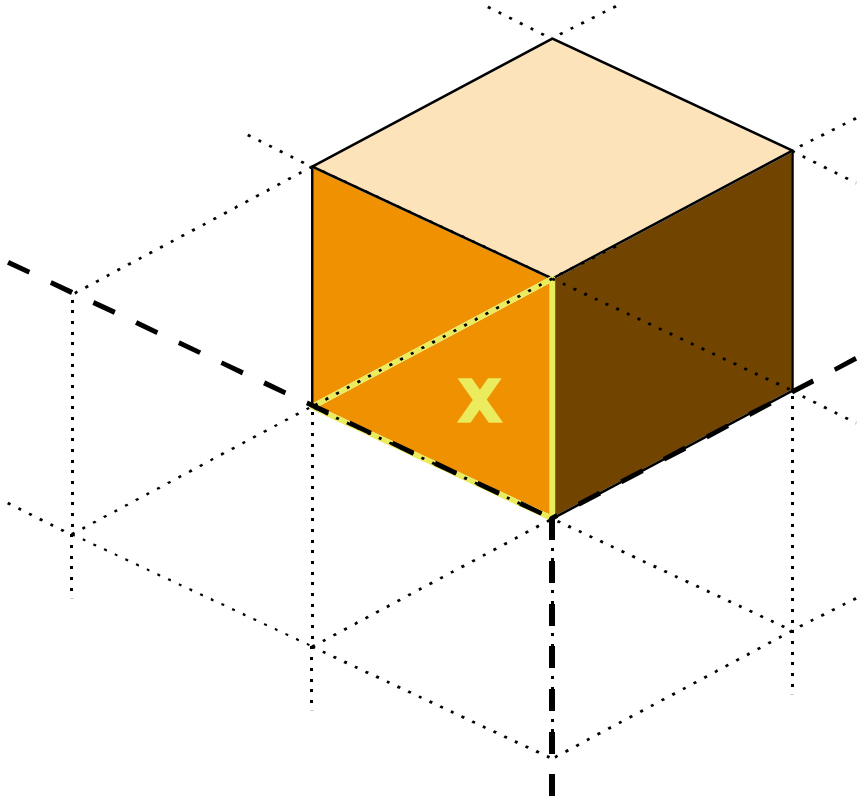


1. horní stěna
[0,0,0]

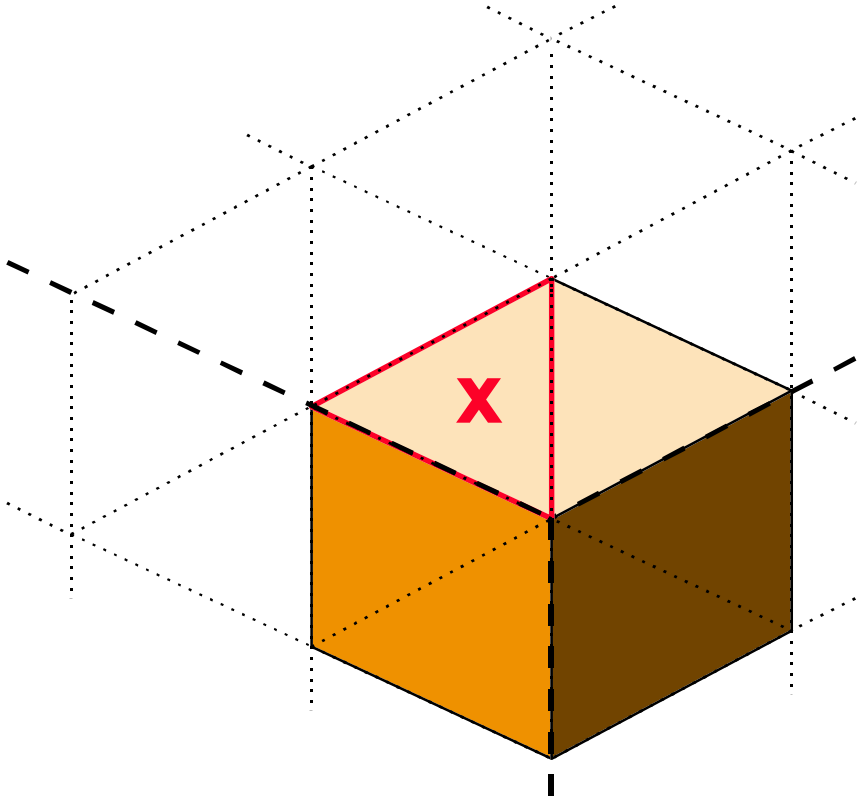


2. pravá stěna
[0,1,0]

Speciální promítání

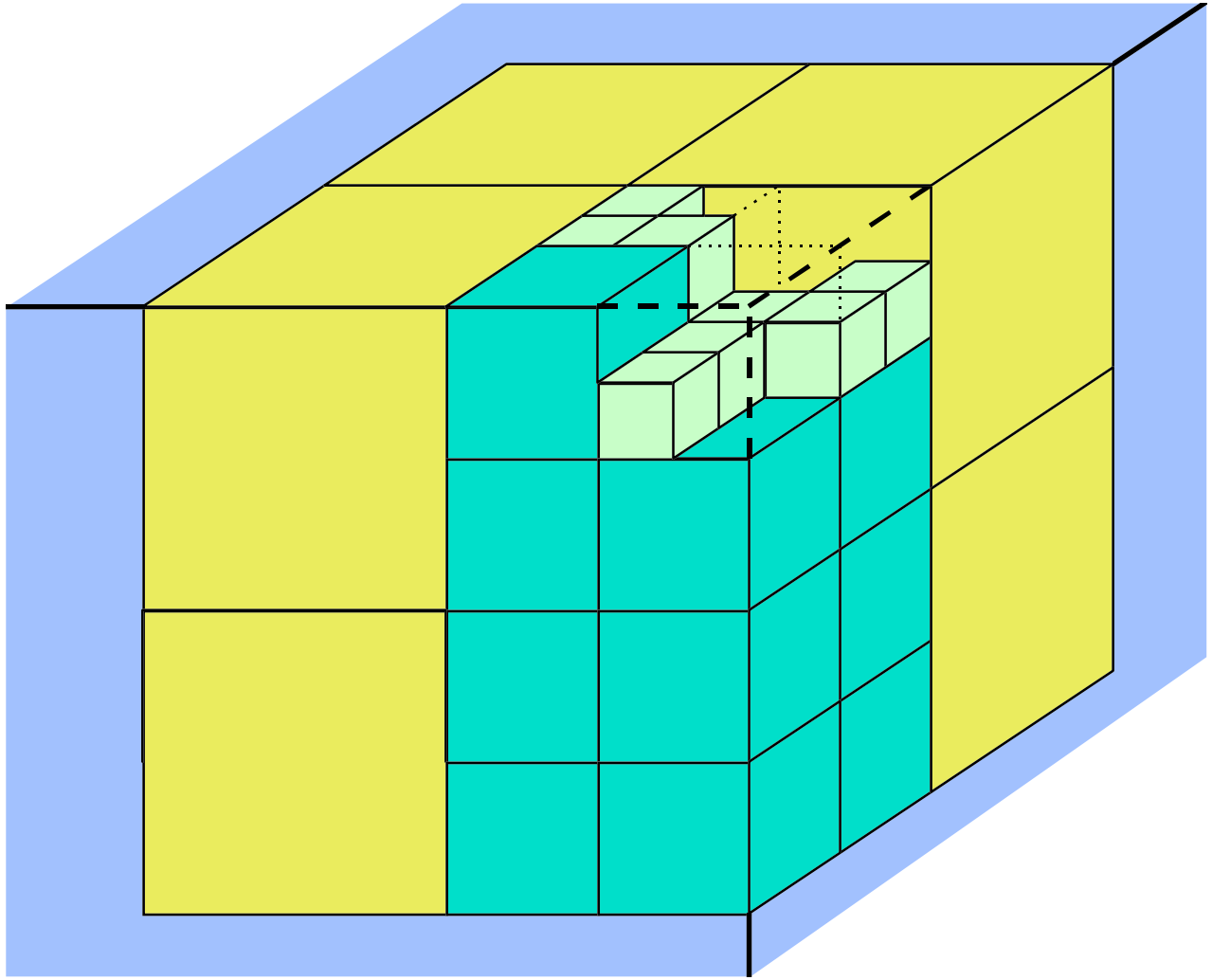


3. levá stěna
[1,1,0]



4. horní stěna
[1,1,1]

Oktantový strom (“octree”)

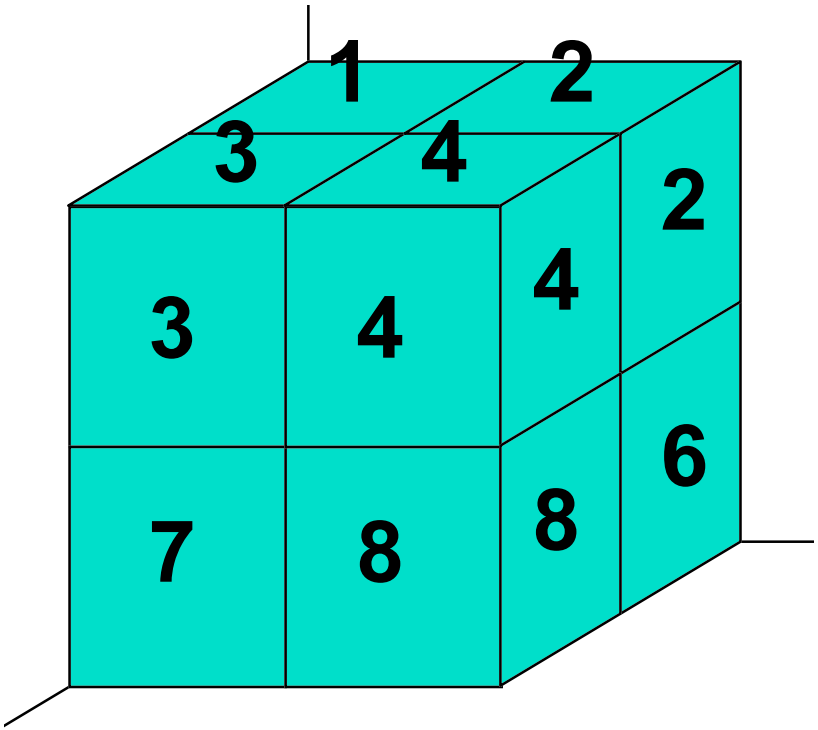


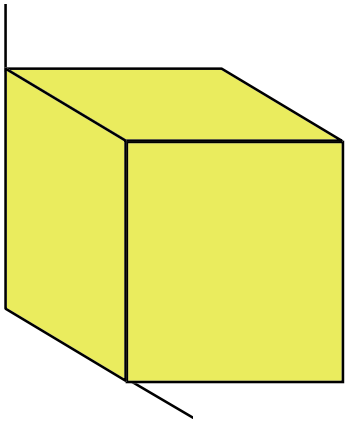
Oktantový strom

- **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

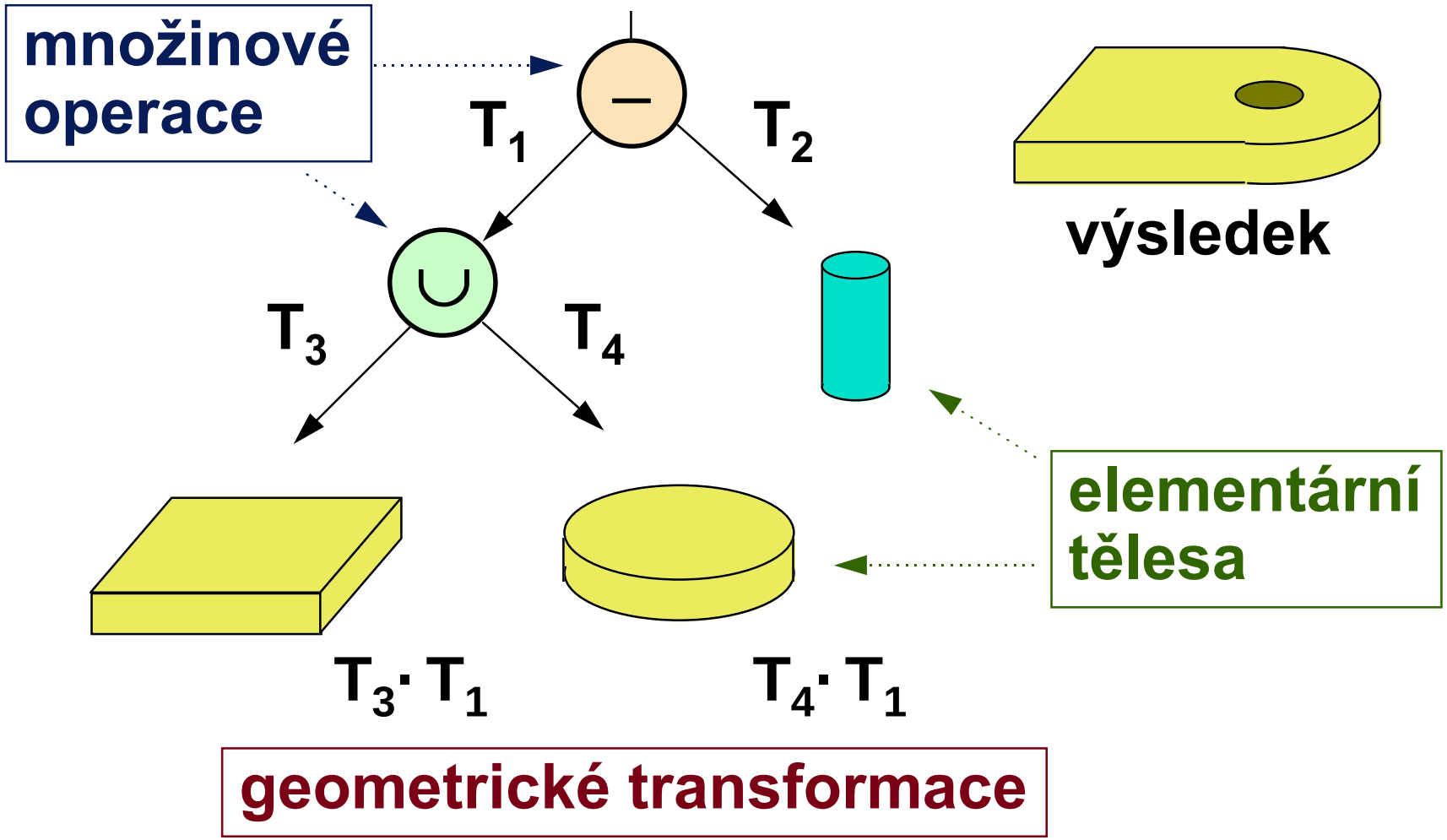
pořadí:

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)

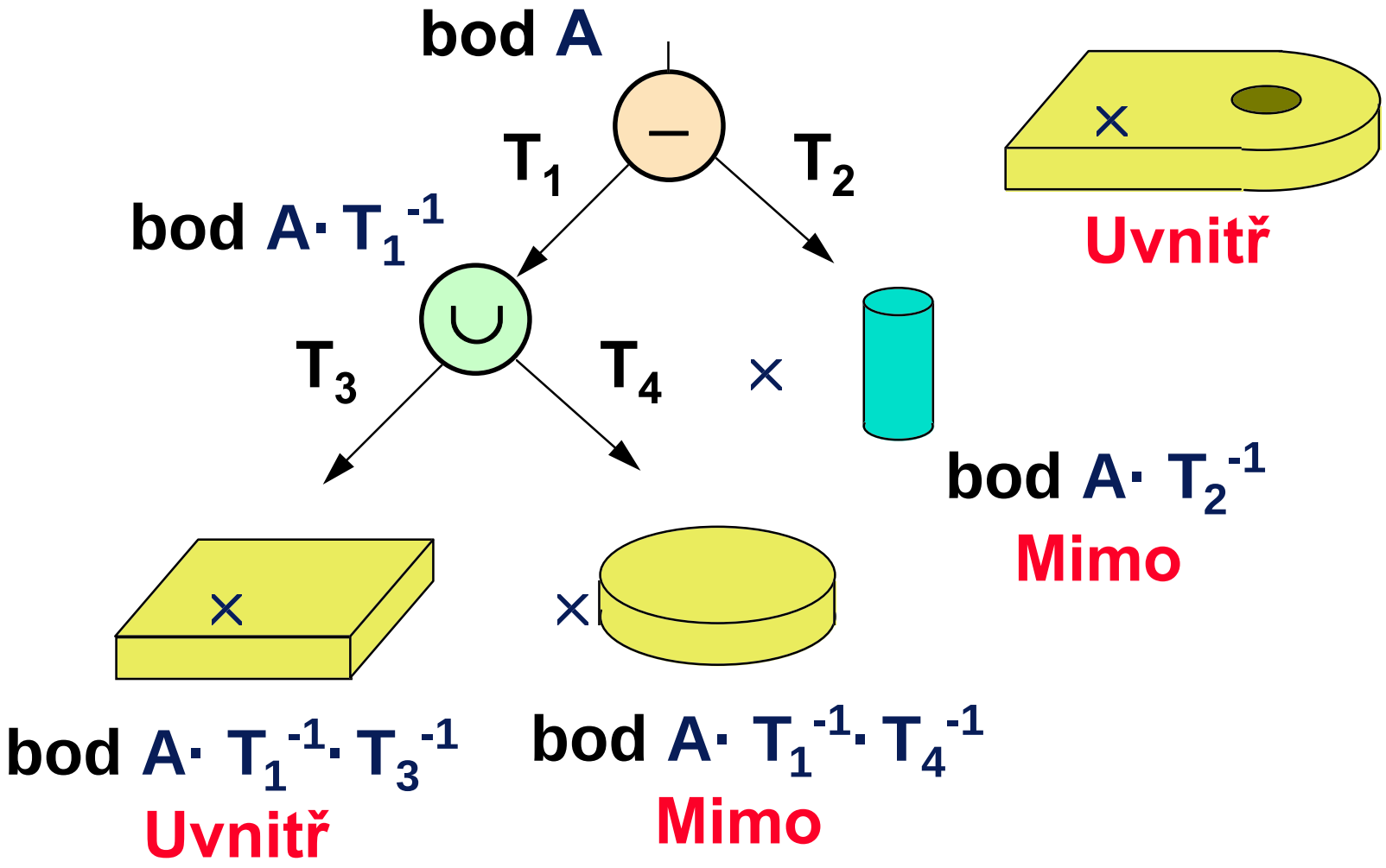
Transformace v CSG stromu

- **uložení transformací jen v listech**
  - **kumulované součiny** (např. $T_3 \cdot T_2 \cdot T_1$ nebo inverzní $T_1^{-1} \cdot T_2^{-1} \cdot T_3^{-1}$)
  - 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 **booleovské ekvivalenty** ($\vee$ místo $\cup$, $\wedge$ místo $\cap$, ...)

Test “bod×CSG strom”



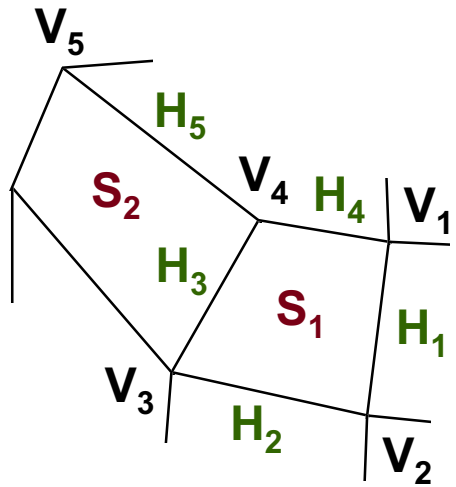
Zobrazování CSG reprezentace

- **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

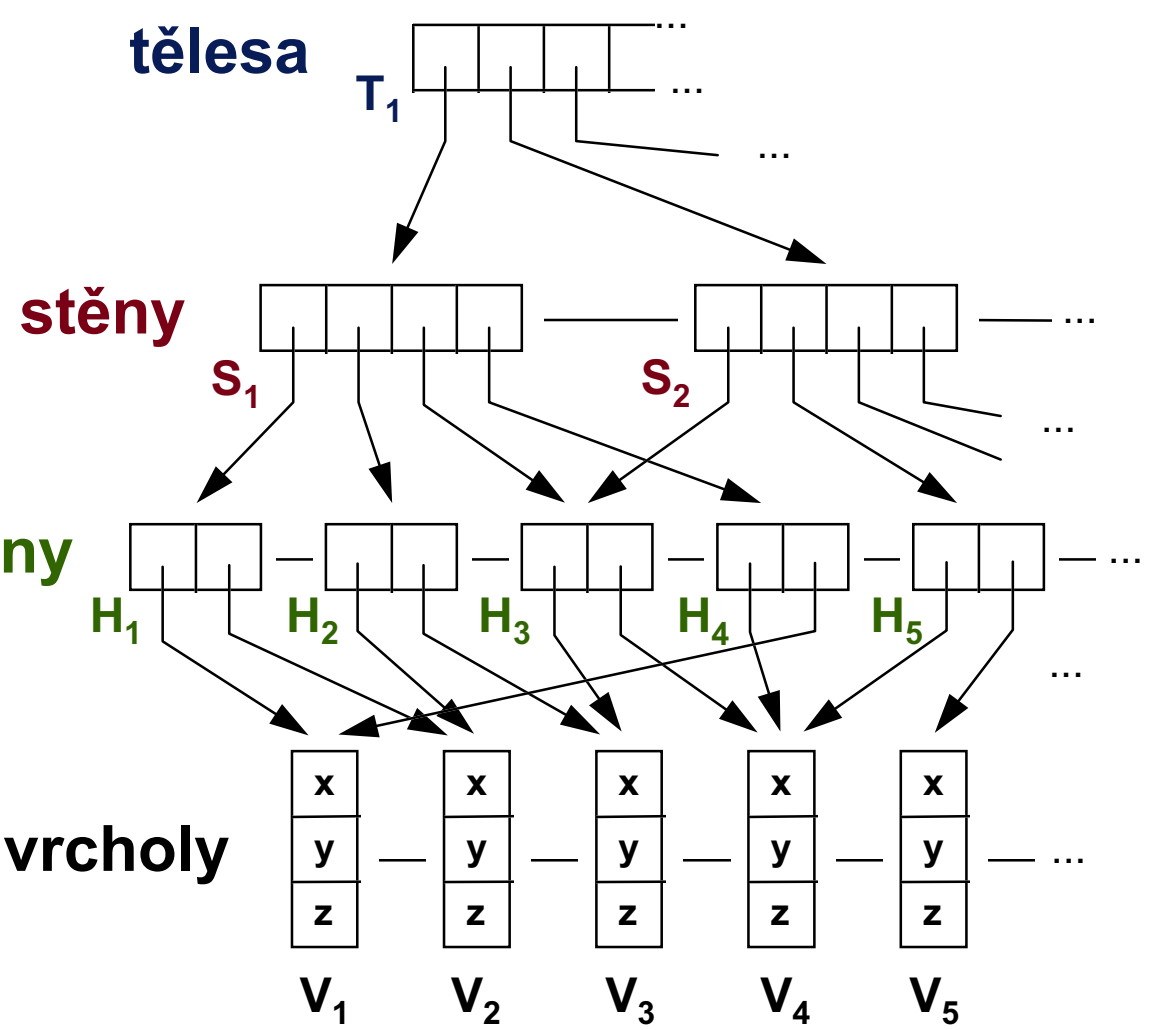
Povrchové reprezentace

- ✓ **“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, ..)

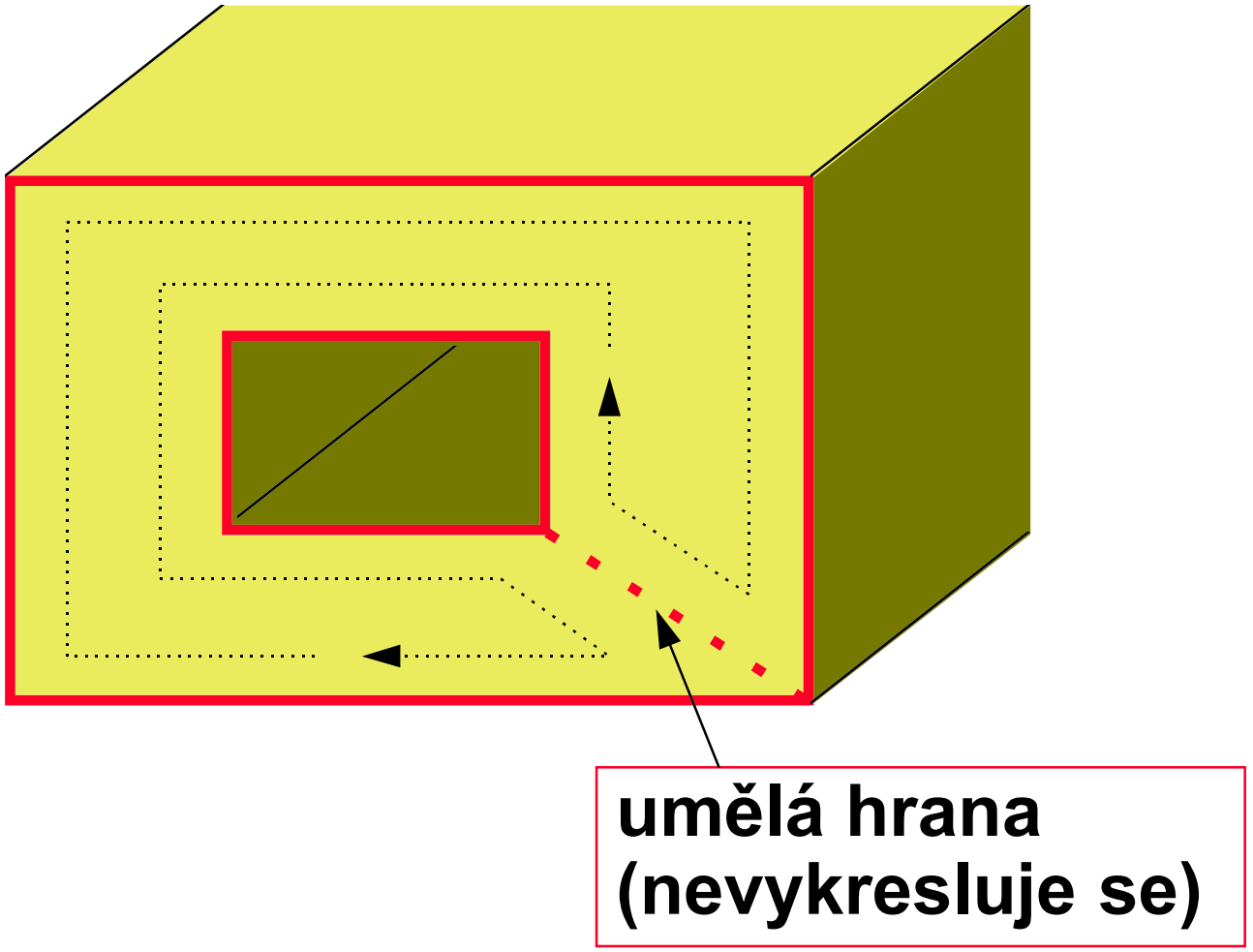
Povrchová reprezentace VHST



těleso T_1

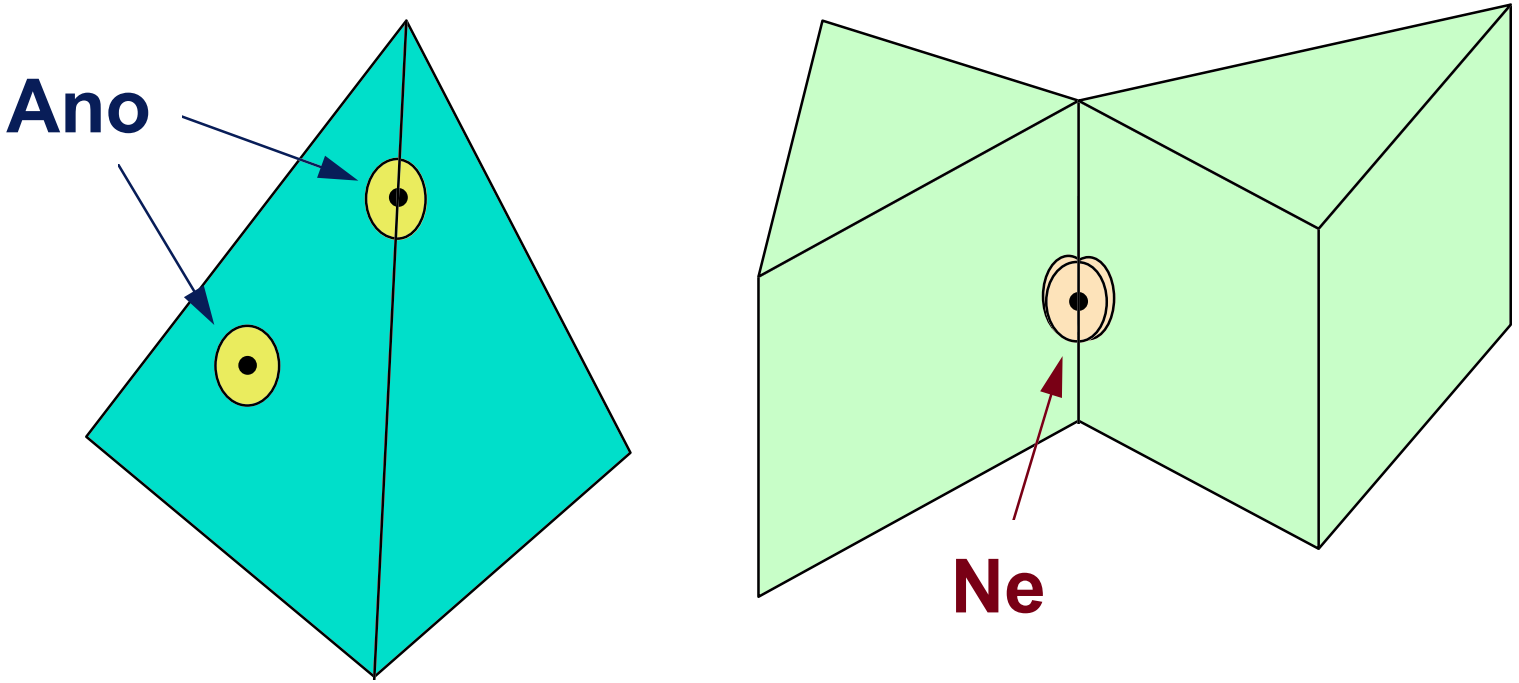


“Děravá” stěna

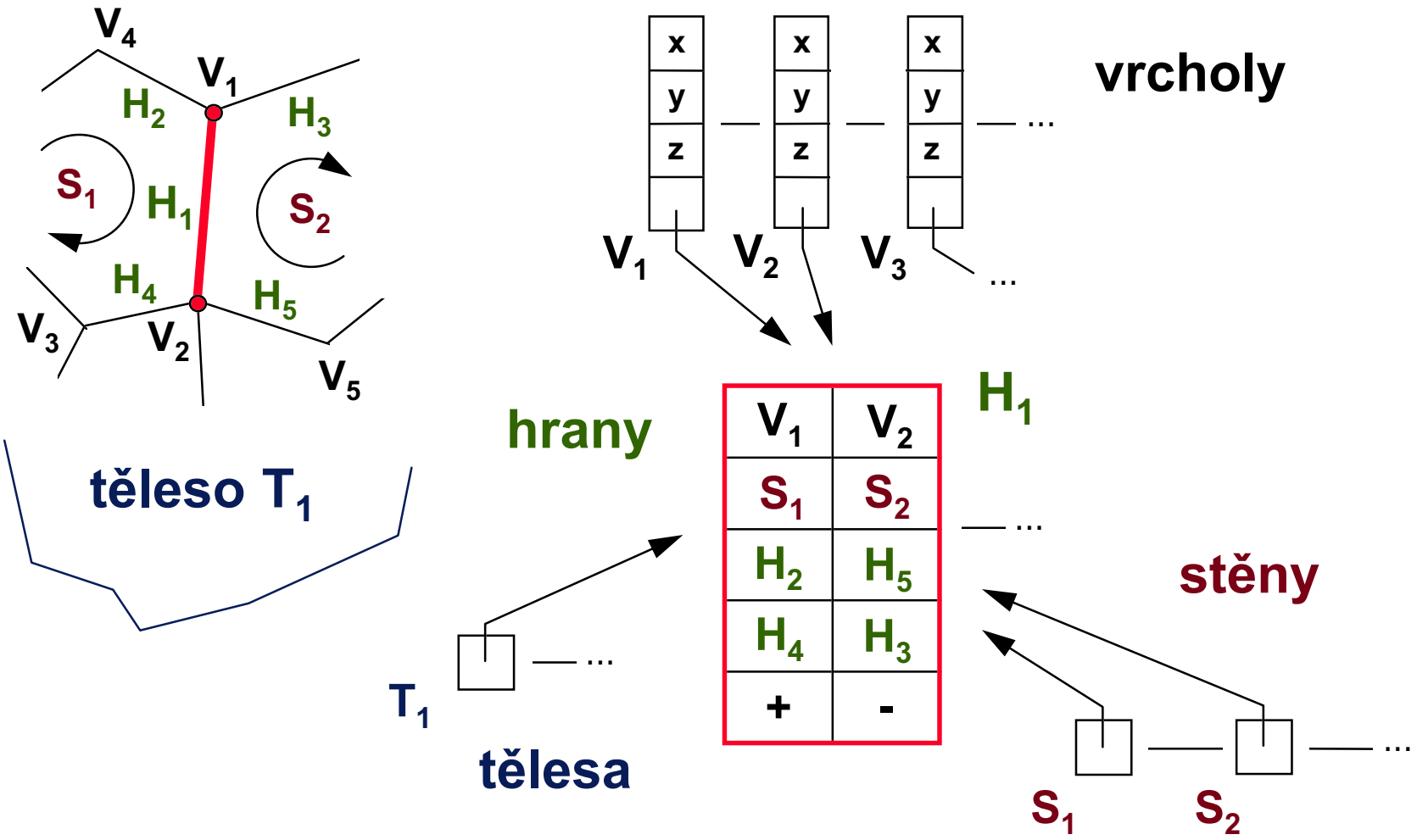


“2-manifold” (manifold)

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



Okřídlená hrana (“winged-edge”)



Další informace v databázi

- **vrchol:**
  - (normálový vektor pro spojité stínování)
- **hrana:**
  - příznak umělé hrany: pro reprezentaci děravých stěn nebo aproximaci křivých ploch sítí polygonů
- **stěna:**
  - barva
  - normálový vektor (stínování, přivrácená/odvrácená)
- **těleso:**
  - barva

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

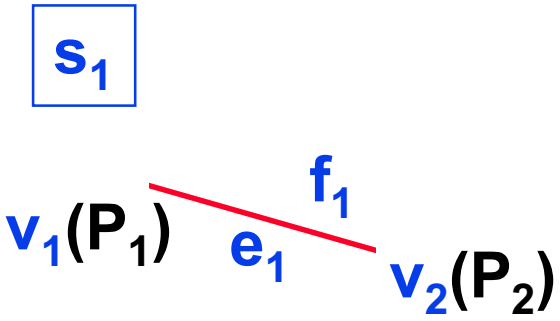
- **konstrukce 2-manifoldu po krocích**
  - v každém kroku je zajištěna platnost Eulerových vzorců (těleso je topologicky korektní)
  - ke každému operátoru existuje inverzní (snadná implementace příkazu “undo”)

- příklady **Eulerových operátorů**:

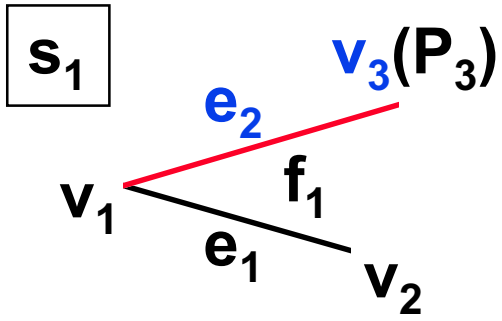
  Msfevv(P_1,P_2): “make solid, face, edge, vertex, vert.”,
  Mev(f_1,v_1,P_2): “make edge, vertex”,
  Mef(f_1,v_1,v_2): “make edge, face”,
  Kef(e): “kill edge, face”, ...

Konstrukce čtyřstěnu

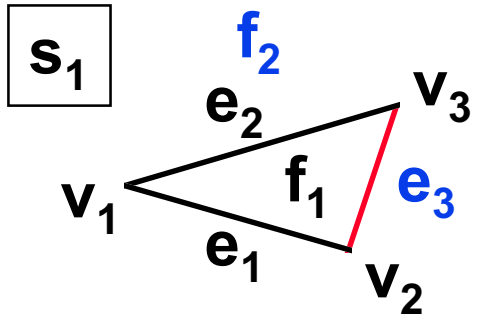
1. Msfevv(P_1,P_2)



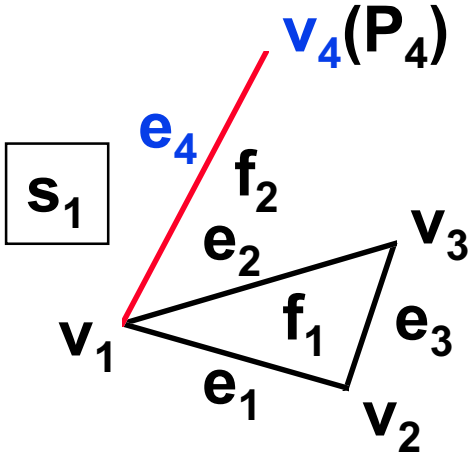
2. Mev(f_1,v_1,P_3)



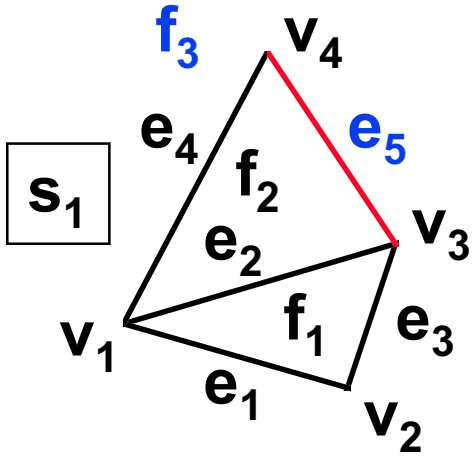
3. Mef(f_1,v_2,v_3)



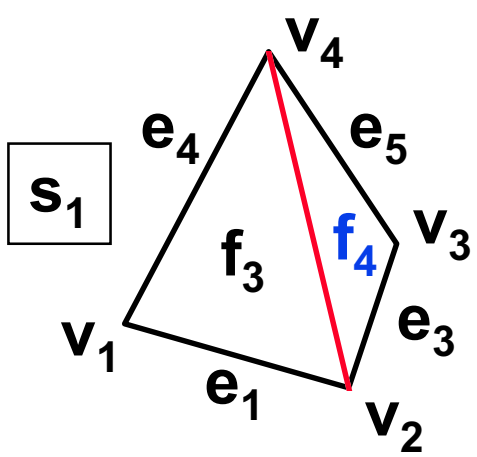
4. Mev(f_2,v_1,P_4)



5. Mef(f_2,v_3,v_4)



6. Mef(f_3,v_2,v_4)



Konec

Další informace:

- **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
- **LAN** na Malé Straně:
  - **barbora\usr:\vyuka\pelikan\6**

Hierarchický model

KSVI MFF UK Praha

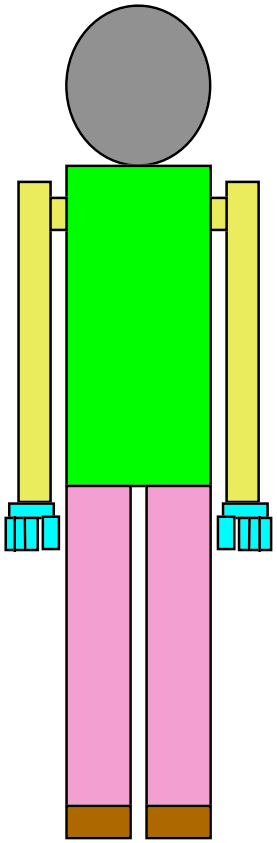
e-mail: Josef.Pelikan@mff.cuni.cz

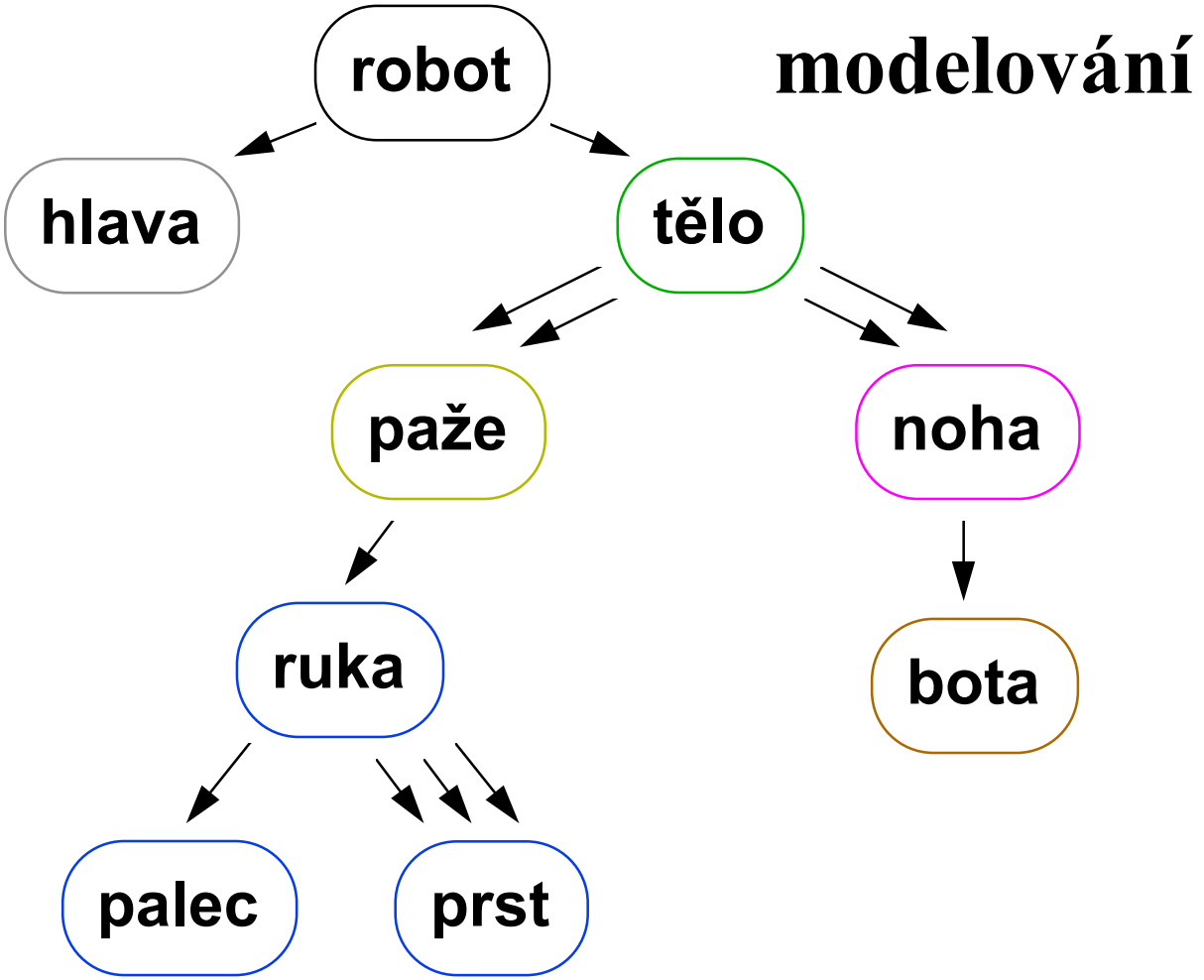
WWW: http://cgg.ms.mff.cuni.cz/~pepca/

Hierarchie v 3D modelování

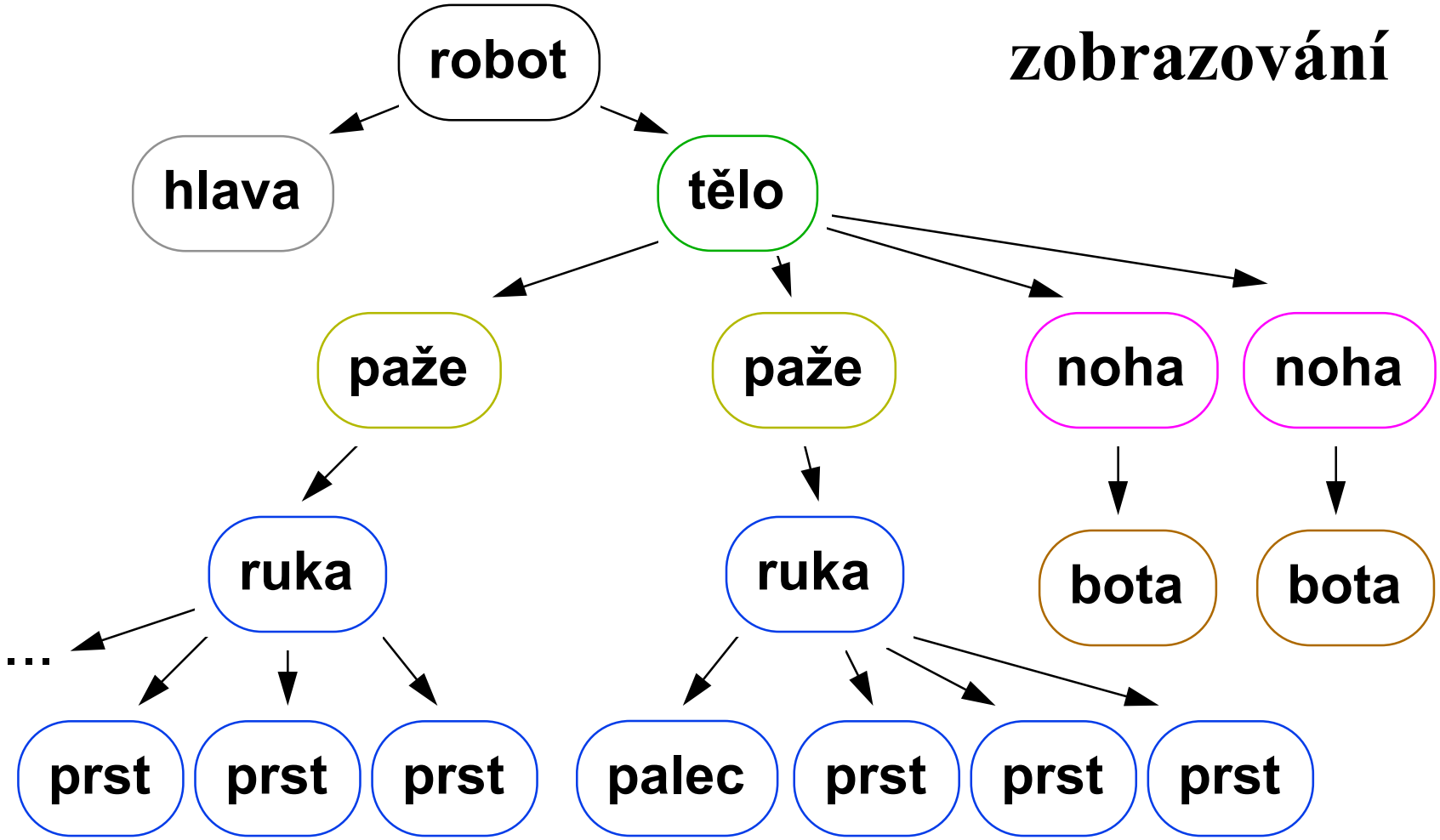
- **kompozice “zdola-nahoru”**
  - složitější objekty se sestavují z jednodušších
  - při modelování se často několikanásobně opakují některé části objektů (stavební prvky, součástky)
- **databáze 3D objektů**
  - ve strojírenství a stavebnictví se často používají standardní (normalizované) prvky
- **parametrické modelování**
  - jednotlivé instance objektu se mohou mírně lišit

Hierarchický model robota





Strom modelu robota



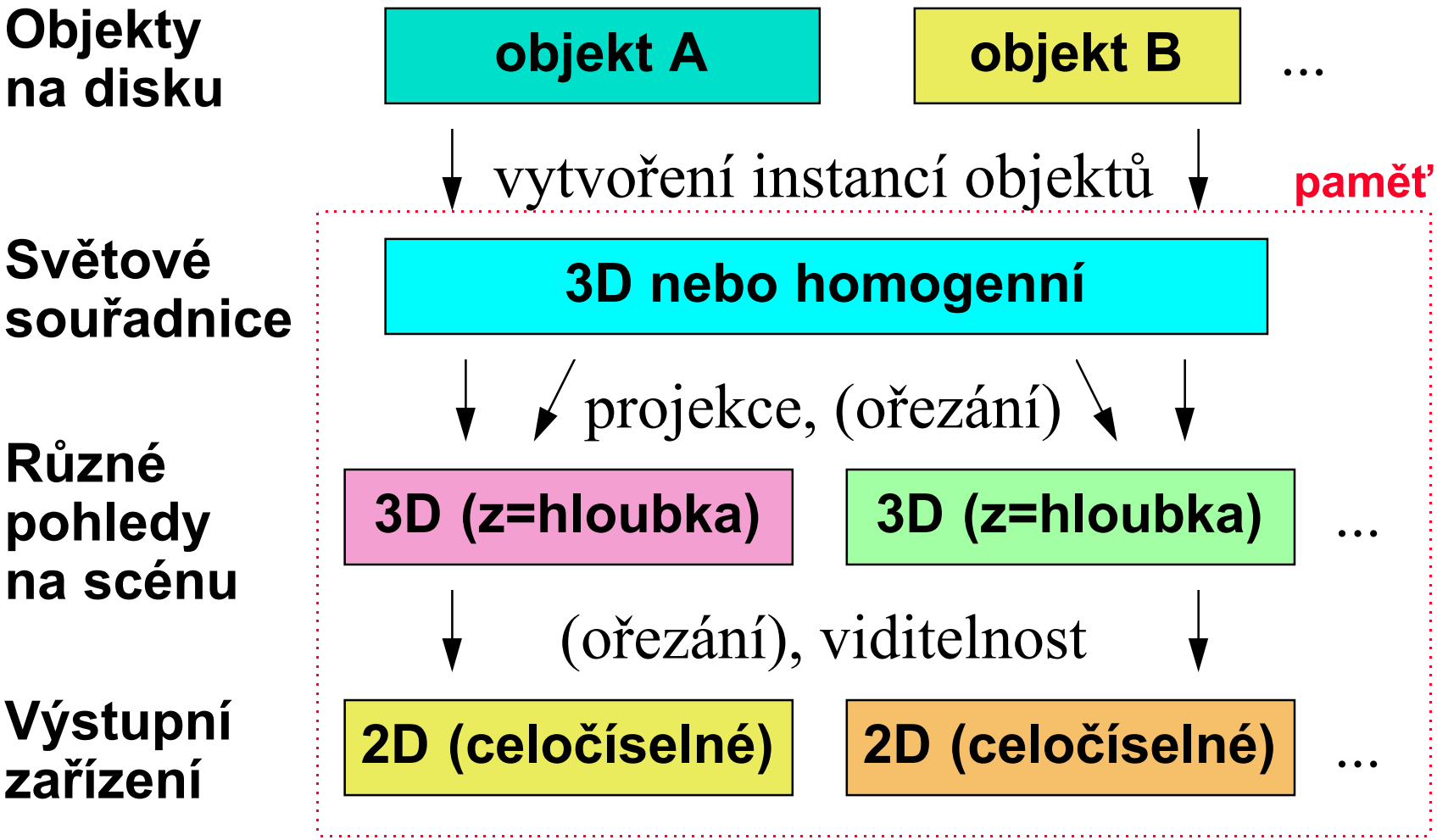
Uložení objektu v databázi

- **globální** (implicitní) **atributy a parametry**
  - barva, materiál, přesnost aproximace křivek, ...
- **vlastní 3D prvky**
  - tělesa, stěny, plochy, ... (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)

Pole souřadnic (vrcholů, uzlů, ..)



Hierarchické 3D formáty

- **PHIGS(+)** (ANSI, ISO)
  - "Programmer's Hierarchical Interactive Graphics System"
- **OpenInventor** (Silicon Graphics Inc.)
  - objektová nadstavba OpenGL systému
- **VRML** ("Virtual Reality Modeling Language")
  - WebSpace (World-Wide Web)
- **vstupní formáty** zobrazovacích programů
  - PoV Ray, RayShade, Radiance, ...

Konec

Další informace:

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

- **LAN** na Malé Straně:
  - **barbora\usr:\vyuka\pelikan\7**

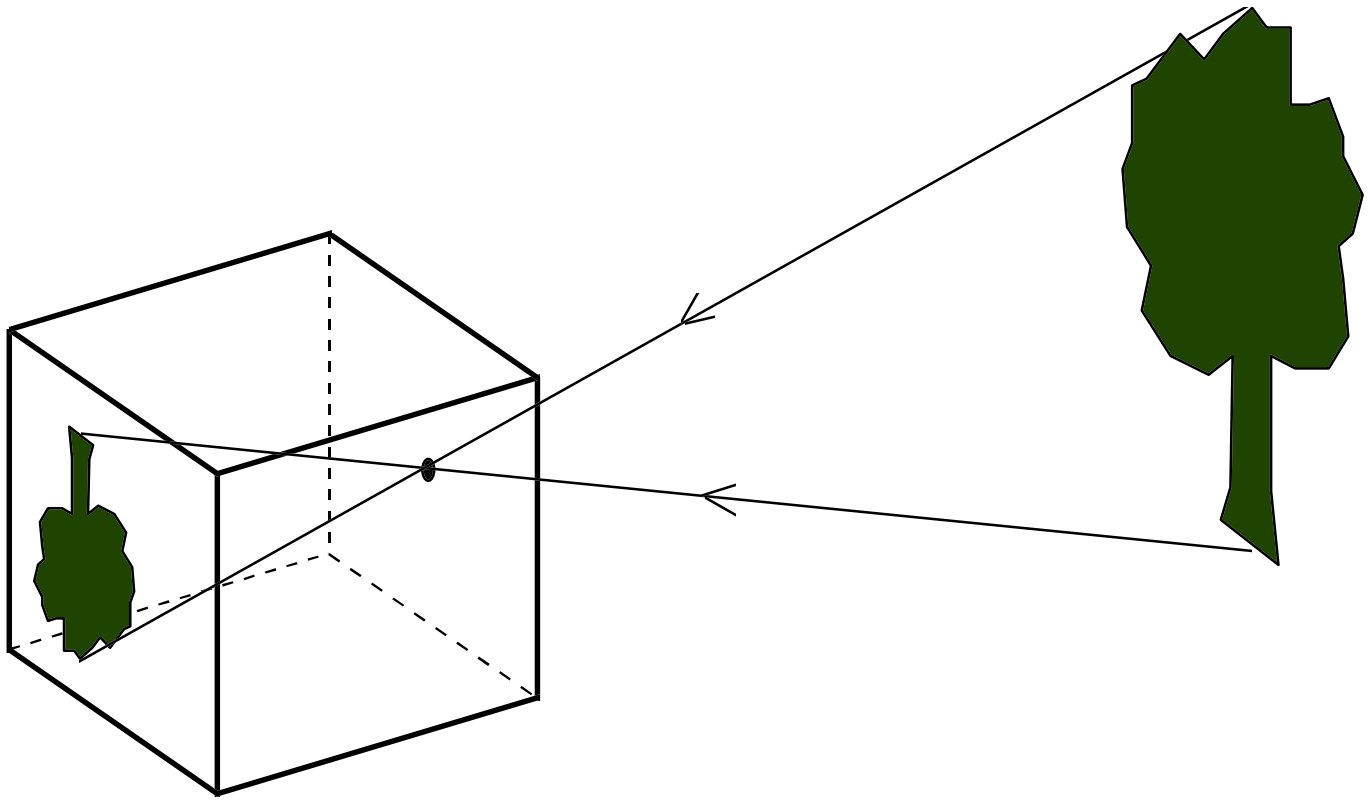
Vrhání paprsku (CSG)

KSVI MFF UK Praha

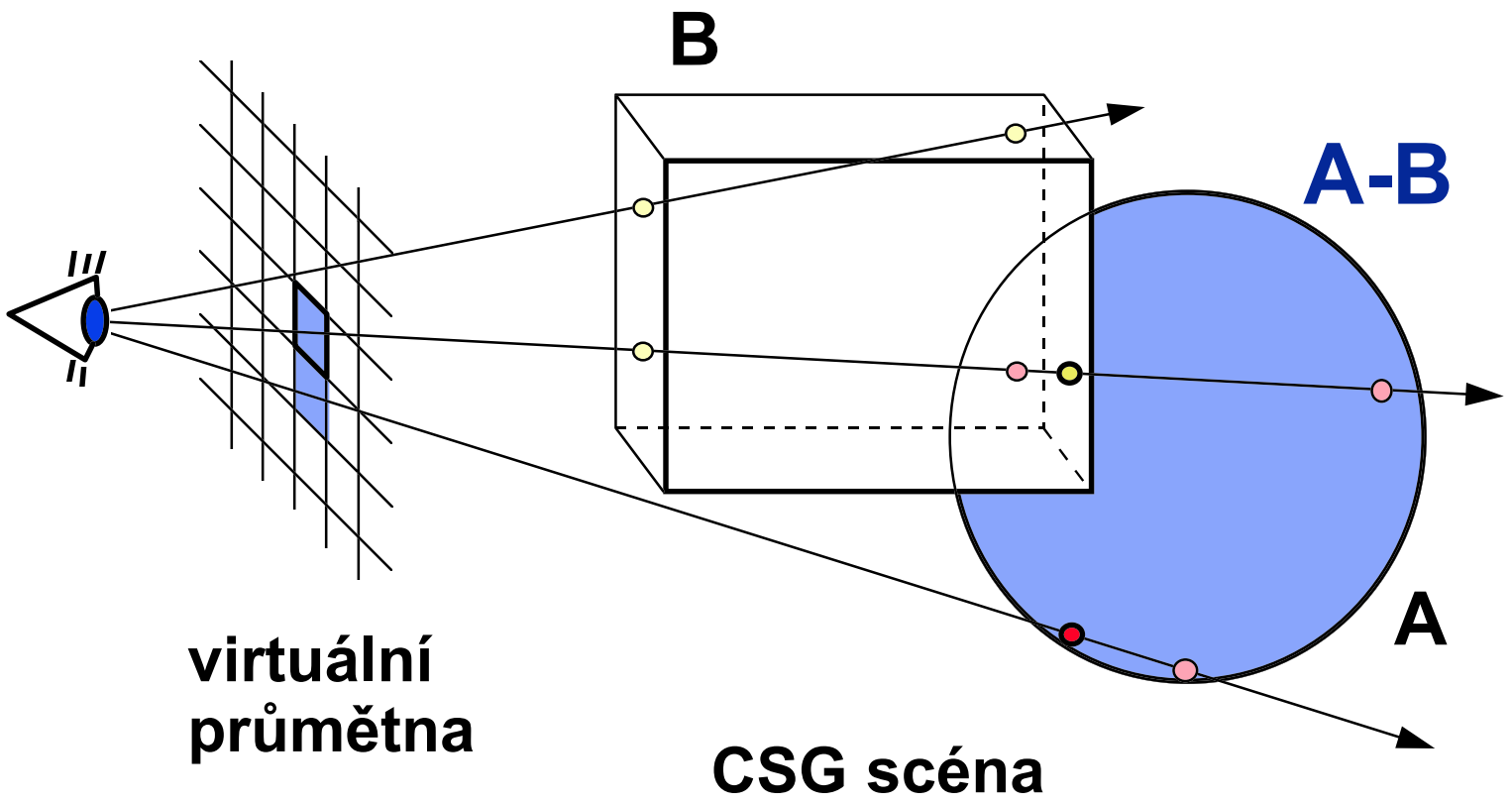
e-mail: Josef.Pelikan@mff.cuni.cz

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

Model dírkové kamery



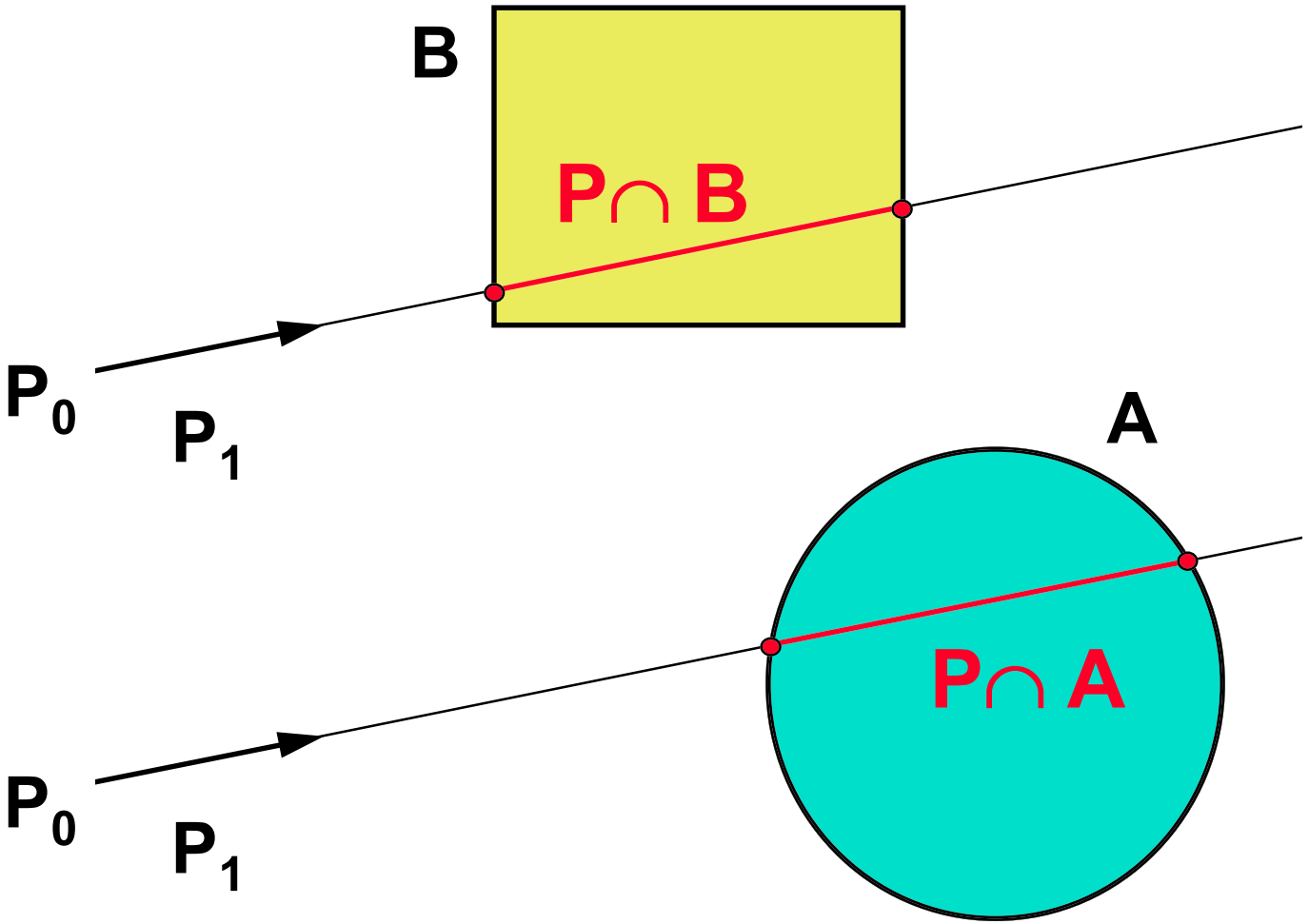
Zobrazování vrháním paprsku



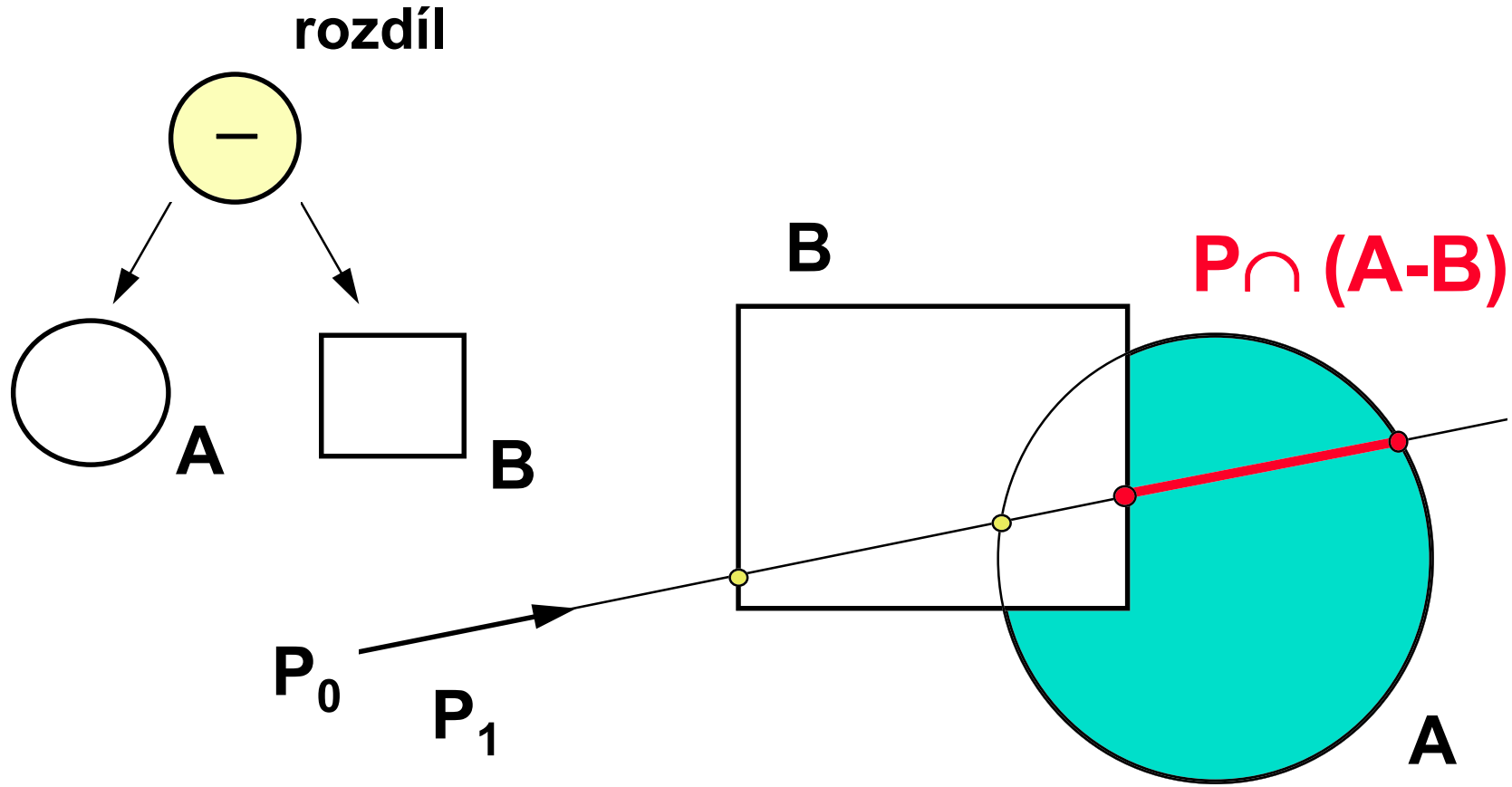
Průsečík paprsku s CSG scénou

- 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∩A,P∩B:



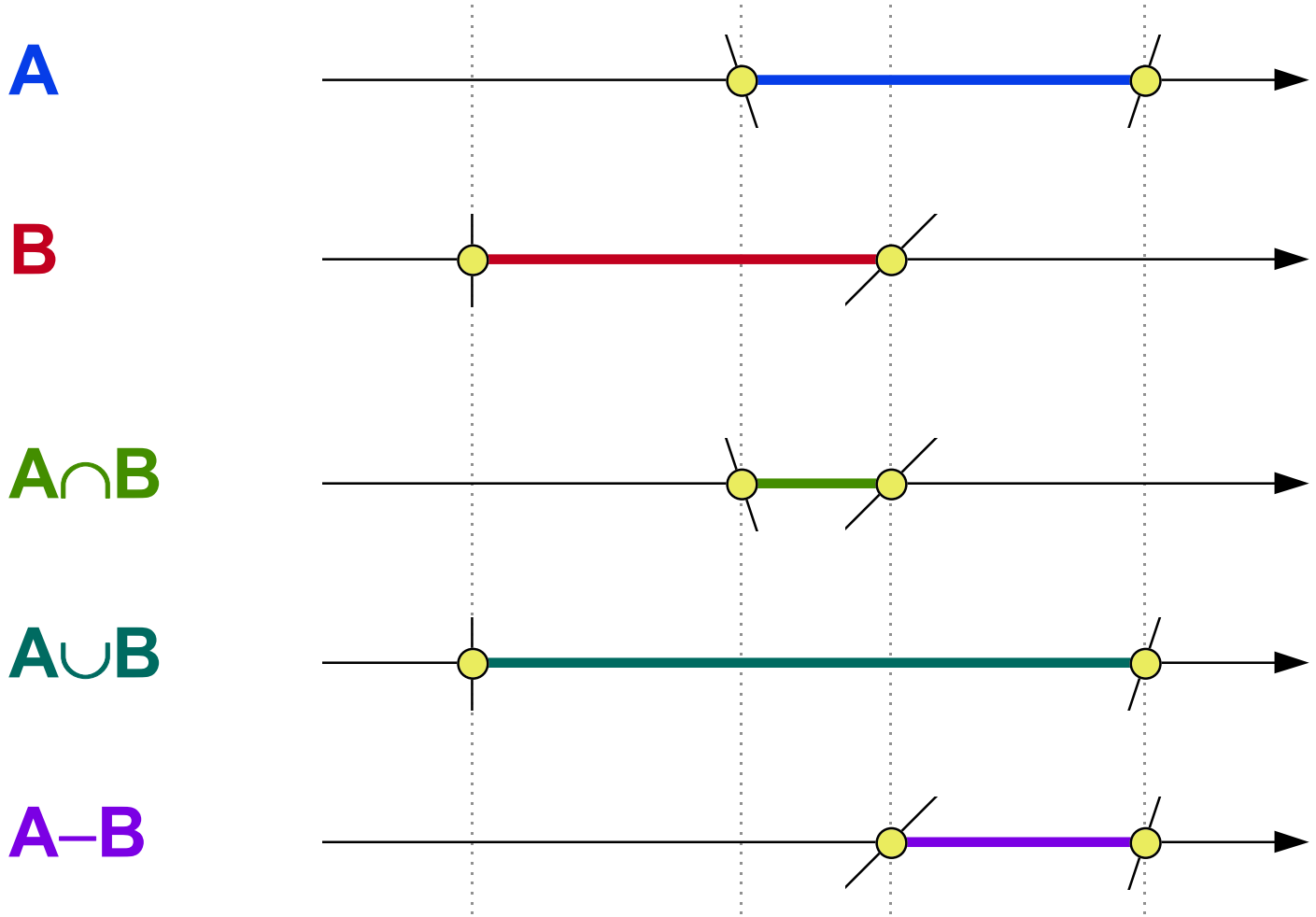
Průsečík P∩(A-B):



Implementace

- **paprsek**:
  - počáteční bod $\mathbf{P_0}$ a směrový vektor $\mathbf{P_1}$
  - transformuje se inverzními maticemi $\mathbf{T_i^{-1}}$
- **průnik paprsku se scénou** (částí scény):
  - uspořádaný seznam hodnot parametru **t**: [$\mathbf{t_1}$, $\mathbf{t_2}$, $\mathbf{t_3}$, ..]
- **množinové operace**:
  - zobecněné slévání vstupních seznamů - např. [$\mathbf{t_1}$, $\mathbf{t_2}$, $\mathbf{t_3}$, ..] a [$\mathbf{u_1}$, $\mathbf{u_2}$, $\mathbf{u_3}$, ..]
  - 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 - $\mathbf{t_1}$)
- obarvení podle **typu množinové operace**:
  - složitější pravidla přenášení barev při výpočtu množinových operací
  - např. speciální barva pro odečtenou část tělesa

Konec

Další informace:

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

- **LAN** na Malé Straně:
  - **barbora\usr:\vyuka\pelikan\7**

Algoritmus plovoucího horizontu

KSVI MFF UK Praha

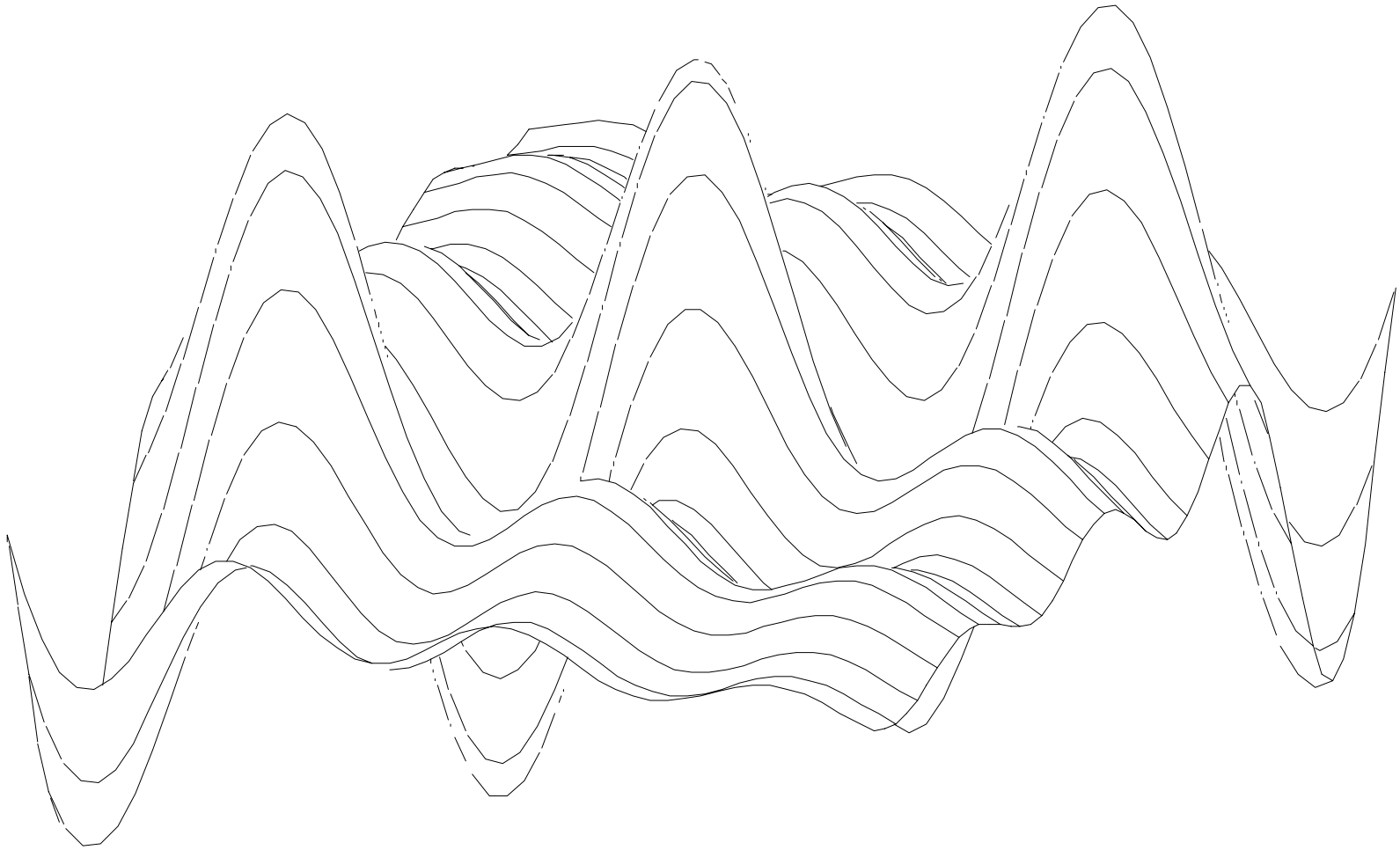
e-mail: Josef.Pelikan@mff.cuni.cz

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

Algoritmus plovoucího horizontu

- pouze pro **graf funkce dvou proměnných**
  - plocha **z = f(x,y)**
- pro některé parametry **středové projekce** nedává správné výsledky
- generuje **čárovou kresbu**
  - umí vytvořit vektorový výstup na plotter
- rychlejší **rastrová varianta**
  - exituje i varianta vyplňující plochy

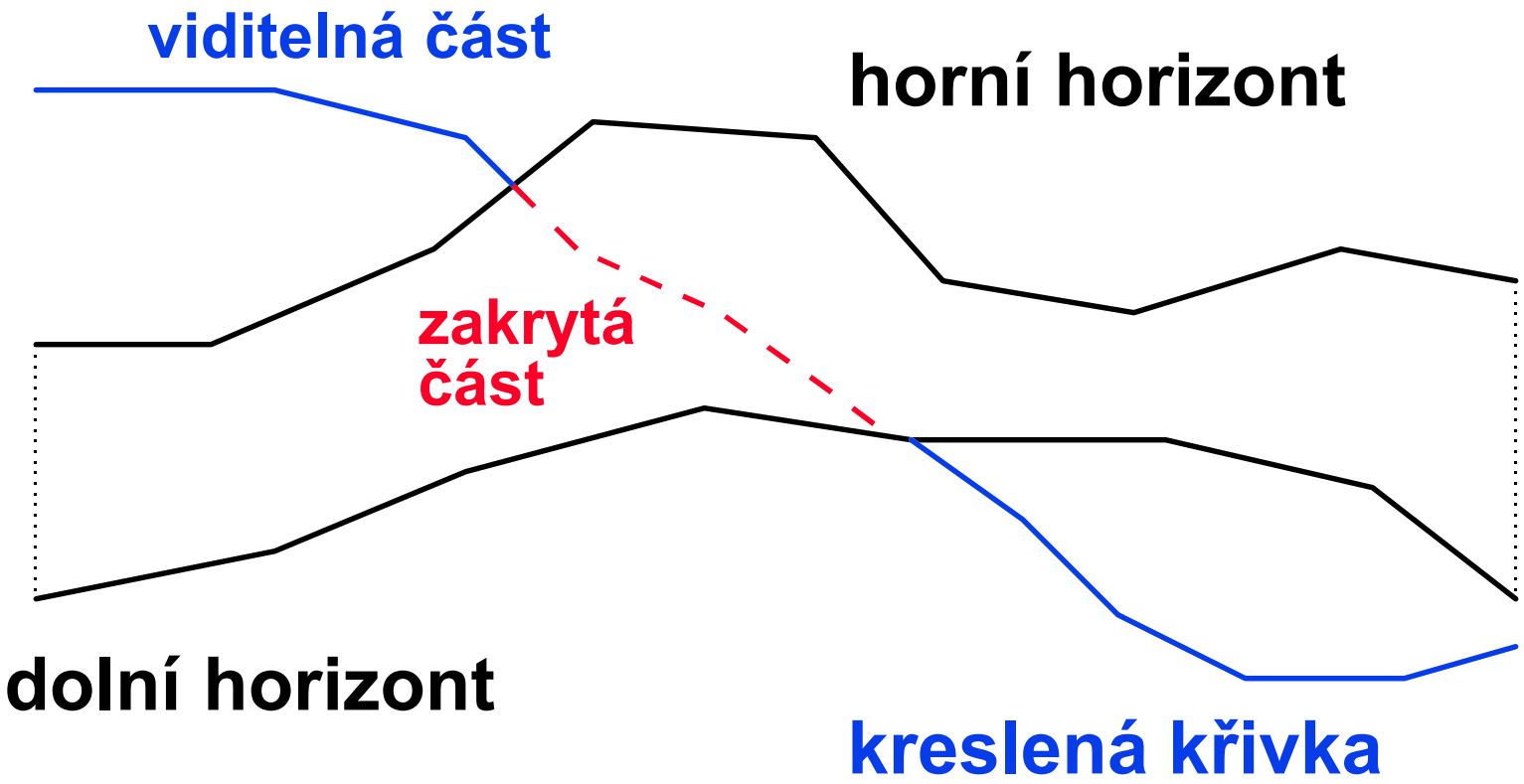
Odstranění neviditelných čar



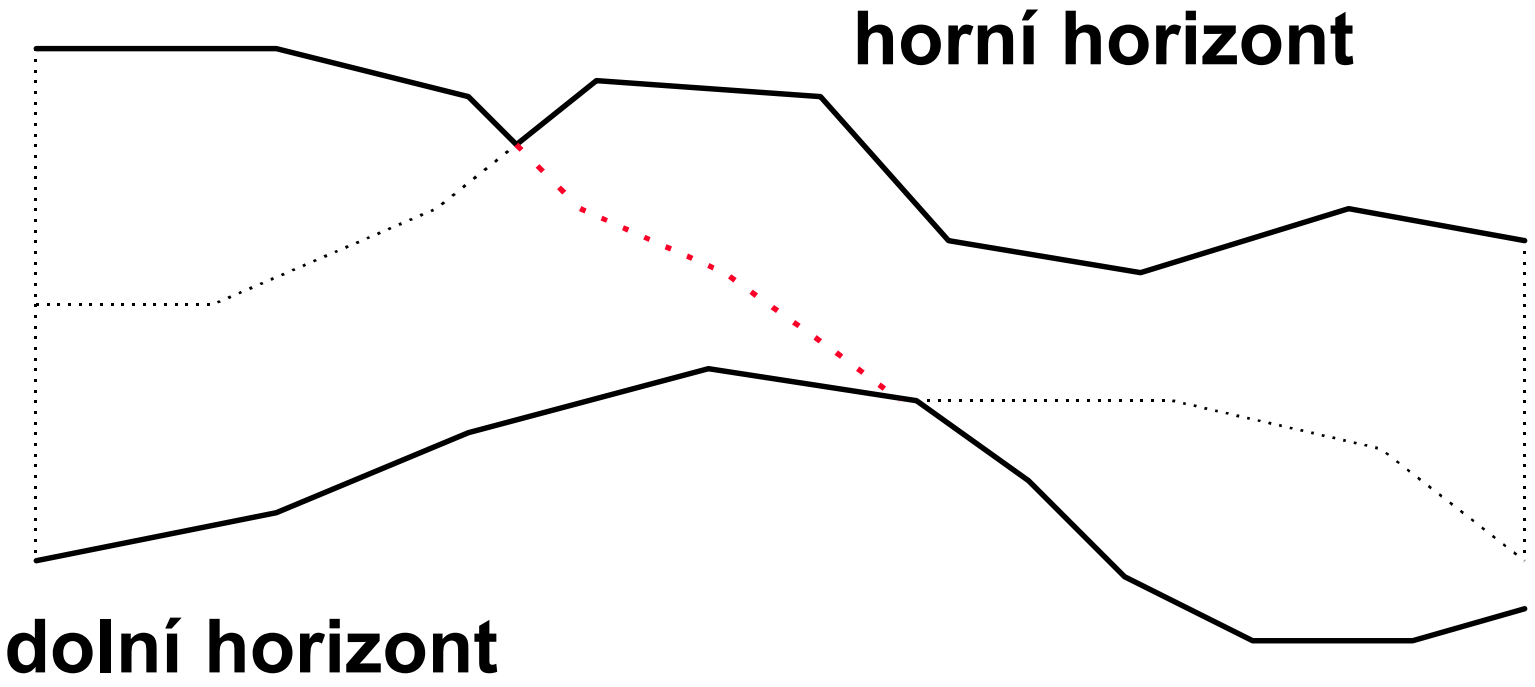
Princip práce algoritmu

- povrchové křivky dané plochy kreslím **odpředu dozadu**
  - objekt může být zakryt pouze objekty nakreslenými již dříve
- udržuji si **obrys** dosud nakreslené části roviny
  - graf funkce dvou proměnných: dvě křivky (lomené čáry) - **horní a dolní horizont**
- viditelné části křivek musí ležet **mimo aktuální obrys**

Výpočet viditelnosti



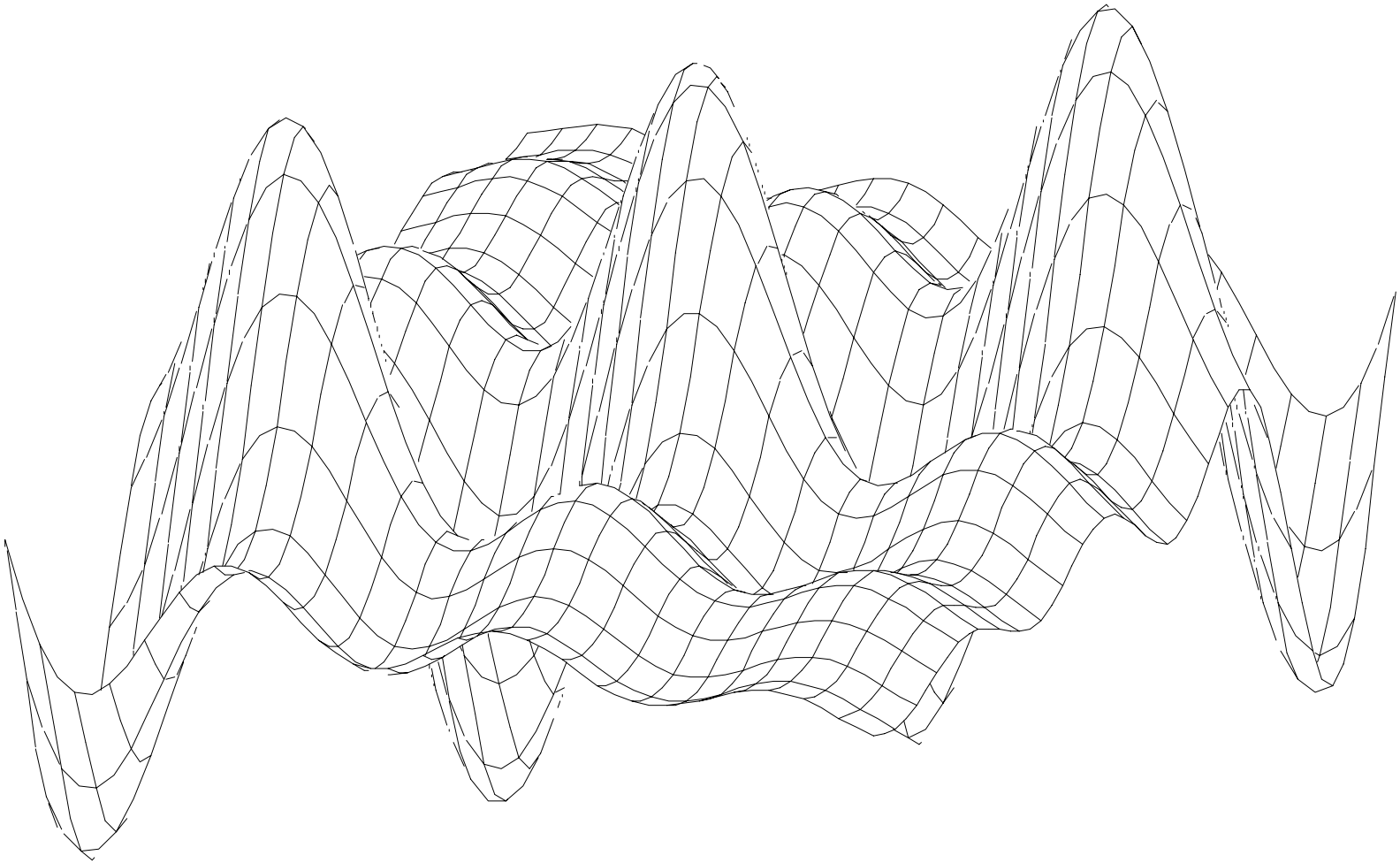
Oprava obrysových křivek



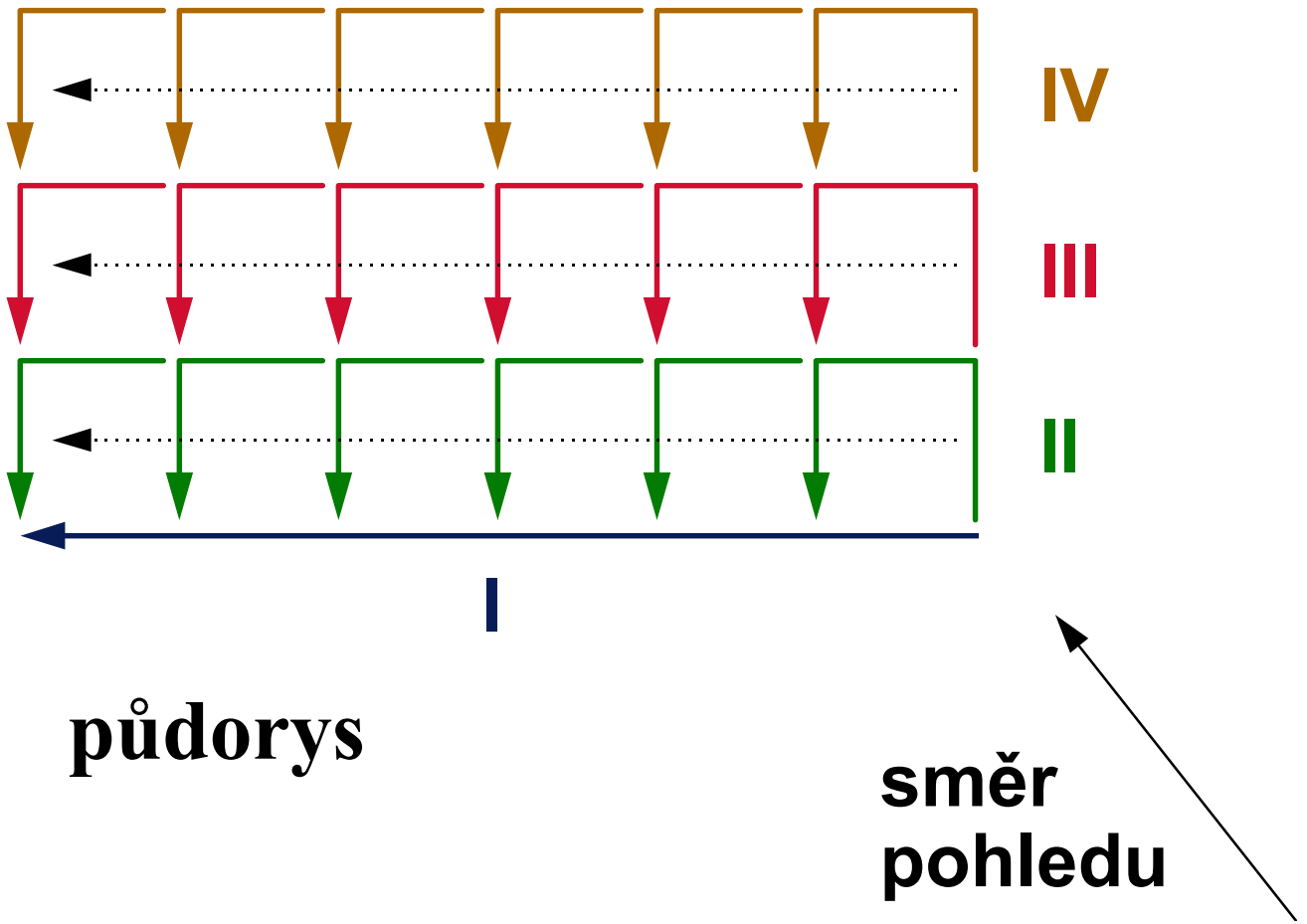
Implementace horizontu

1. **lomené čáry**: $[x_1,y_1]$, $[x_2,y_2]$ ... $[x_n,y_n]$, $x_i < x_{i+1}$
   - vektorový výstup (velká přesnost)
   - nutnost počítat průsečíky lomených čar
2. **pole mezních hodnot**: $y_{min}[i]$, $y_{max}[i]$, $0 \leq i < x_{res}$
   - rastrový výstup (omezená přesnost)
   - testování na úrovni pixelů (pro menší rozlišení je efektivnější)
   - snadnější implementace

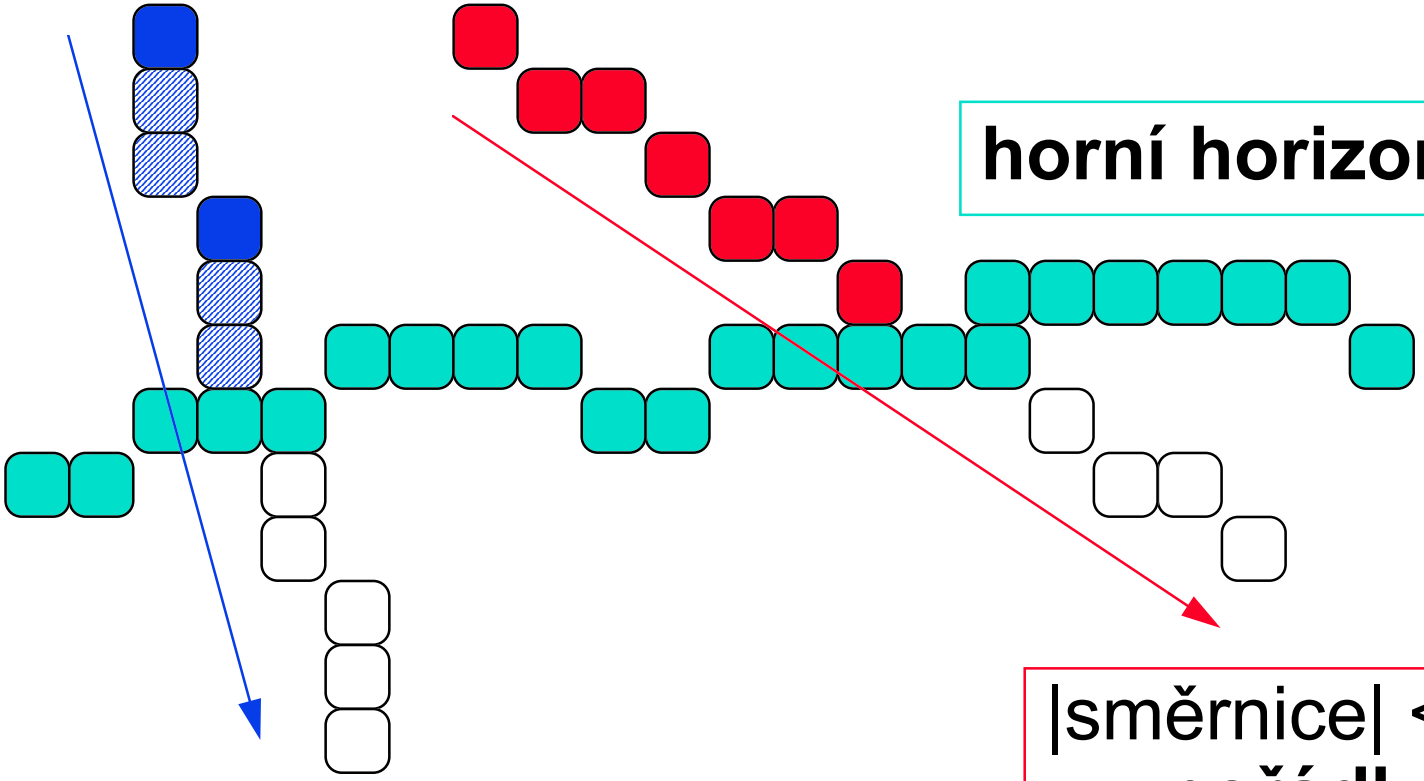
Dvě sítě povrchových křivek



Pořadí vykreslování



Úsečky s velkou směrnicí



horní horizont

|směrnice| < 1
v pořádku

|směrnice| > 1
vynechané pixely

Oprava rastrového algoritmu

- **odstředivé kreslení**
  - úsečku kreslím ve směru od horizontu
  - obtížné úpravy pro velmi dlouhé úsečky
- **dvě fáze zpracování** každé úsečky
  1. kreslím úsečku (testuji každý pixel)
  2. opravuji horizonty
- **dvě sady horizontů** (ztotožňované po každé úsečce)
  1. sadu používám k ořezávání kreslené úsečky
  2. sadu aktualizuji podle kreslené úsečky

Konec

Další informace:

- **J. Foley, A. van Dam, S. Feiner, J. Hughes**: ***Computer Graphics, Principles and Practice***, 651-656
- **Jiří Žára a kol.**: ***Počítačová grafika***, principy a algoritmy, 307-311
- **LAN** na Malé Straně:
  - **barbora\usr:\vyuka\pelikan\6**

Appelův algoritmus

KSVI MFF UK Praha

e-mail: Josef.Pelikan@mff.cuni.cz

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

Appelův algoritmus

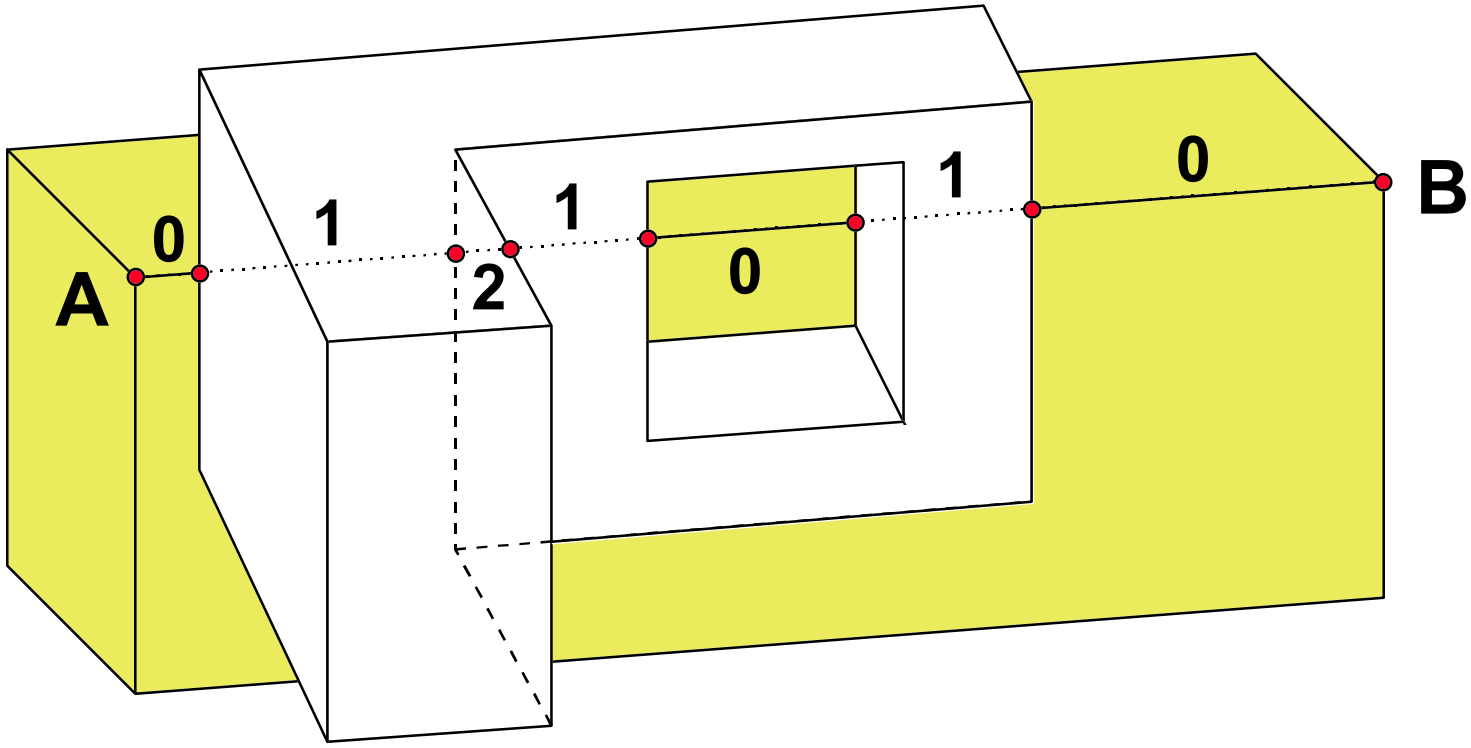
- **generuje vektorovou čárovou kresbu**
  - vhodný např. pro výstup na plotter
  - velká přesnost výstupu (možnost škálování)
  - nelze vyplňovat
- **scéna složená z** (uzavřených) **mnohostěnů**

➡ viditelnost mohou změnit pouze **obrysové hrany**

➡ **koeficient zakrytí** bodů na úsečce

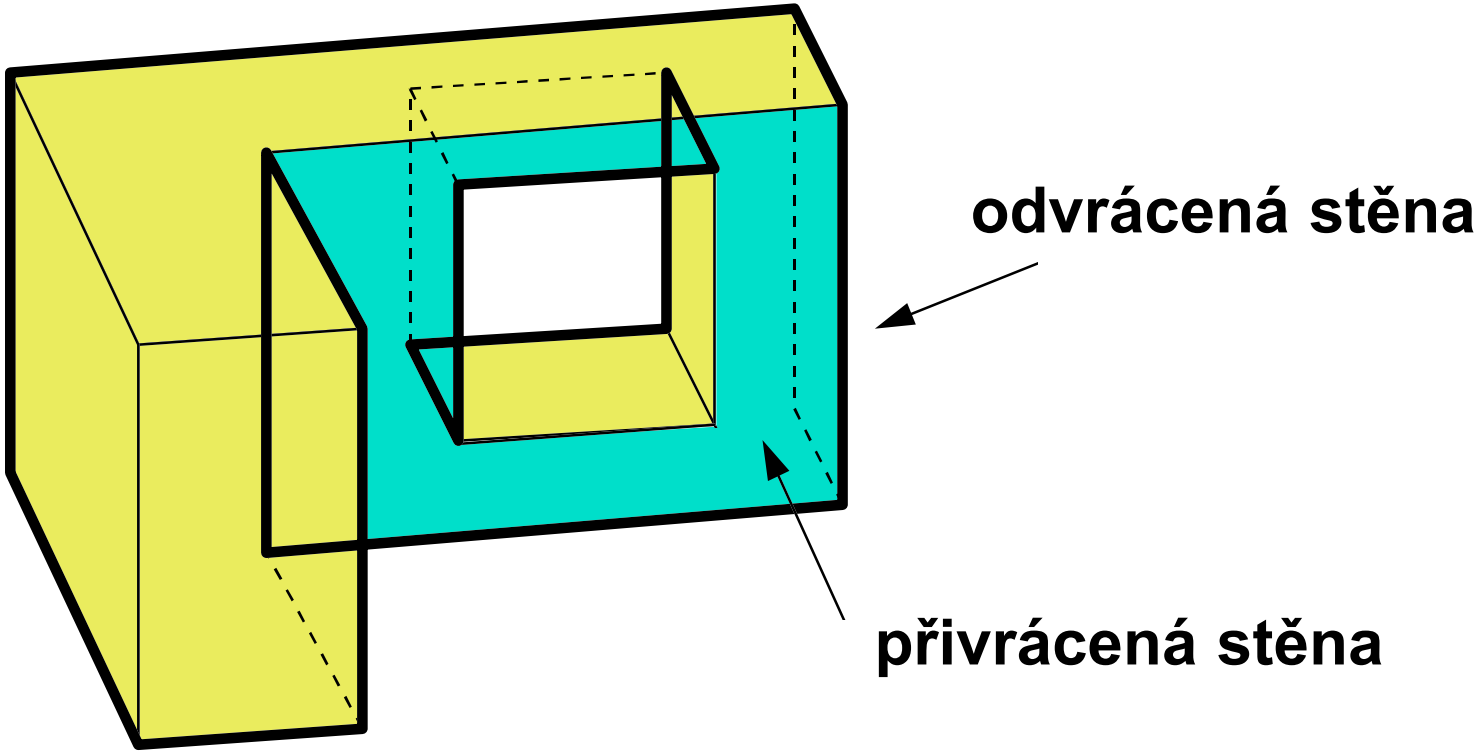
Koeficient zakrytí

Počet přivrácených stěn, které daný bod zakrývají



Obrysová hrana

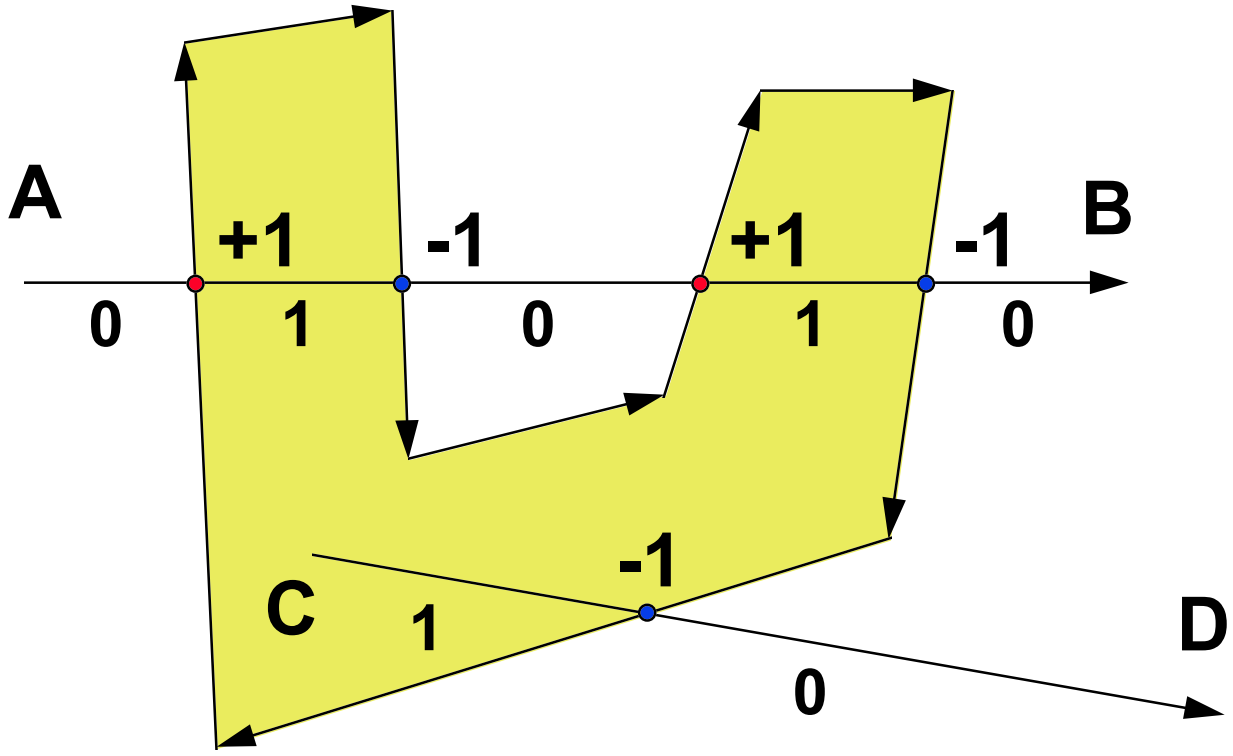
Hrana ležící mezi <u>přivrácenou</u> a <u>odvrácenou</u> stěnou nebo hrana, která nesousedí se dvěma stěnami



Postup výpočtu

1. výpočet **koeficientu zakrytí** dosud nezpracovaného vrcholu **A**
   - hrubou silou (podle definice)
2. **výpočet viditelnosti úsečky AB**
   - v průmětu: nalezení průsečíků s obrysovými hranami ležícími před úsečkou **AB**
   - uspořádání průsečíků a určení jejich příspěvků +1/−1 (orientované smyčky obrysových hran)
   - výpočet koeficientů zakrytí (0: úsek hrany je vidět)

Koeficient zakrytí na úsečce



Postup výpočtu

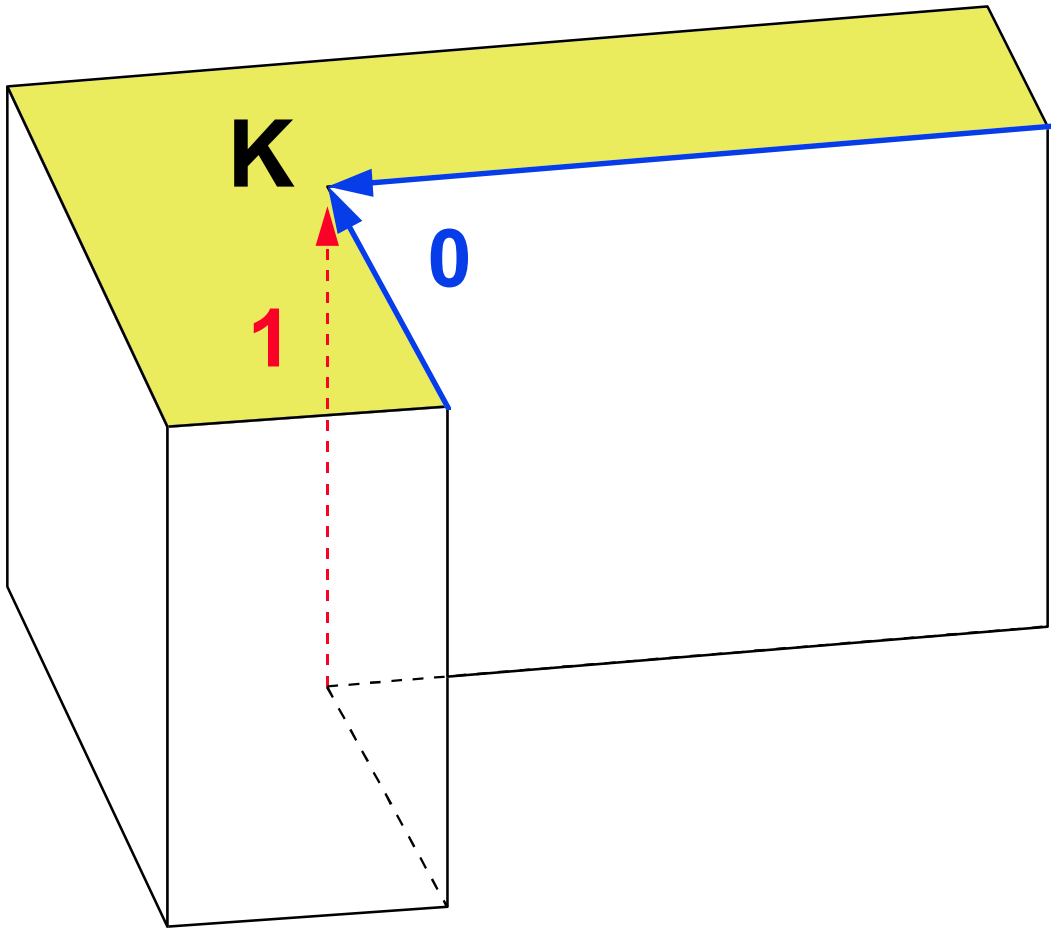
❸ krok ❷ se opakuje pro všechny **potenciálně viditelné hrany**

- hrany sousedící alespoň s jednou přivrácenou stěnou
- v případě potřeby se provádí krok ❶

➡ propagace koeficientu zakrytí z **obrysových vrcholů**

- některé úsečky mohou být zakryté přivrácenou stěnou obsahující daný vrchol
- speciální test v obrysových vrcholech

Konflikt v obrysovém vrcholu



Konec

Další informace:

- **J. Foley, A. van Dam, S. Feiner, J. Hughes**: ***Computer Graphics, Principles and Practice***, 666-667

Malířův algoritmus

KSVI MFF UK Praha

e-mail: Josef.Pelikan@mff.cuni.cz

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

Malířův algoritmus

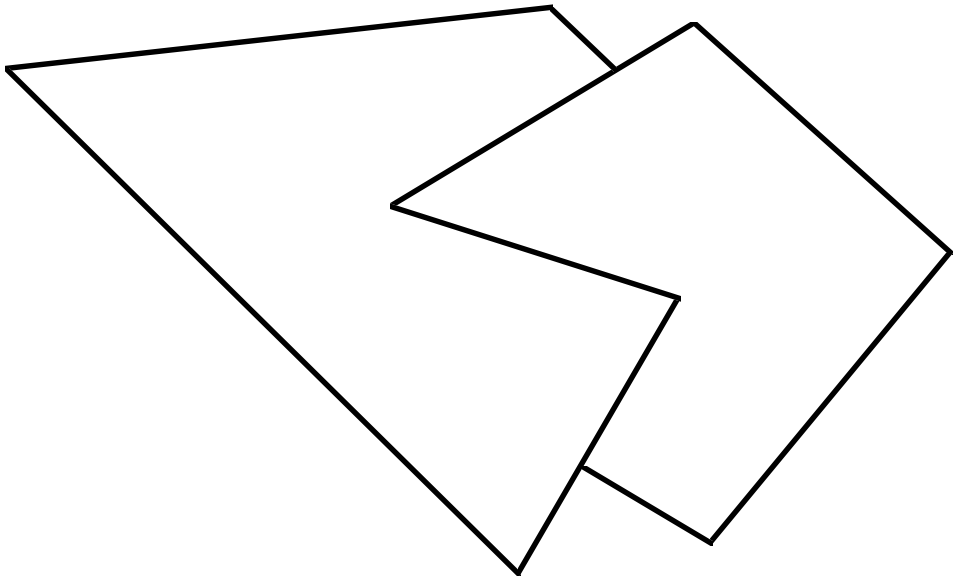
- **kreslení do bufferu**
  - video-RAM, rastrová tiskárna s bufferem
- **vyplňování ploch**
  - lze i stínovat
- **kreslení odzadu dopředu**
  - **překreslování** dříve nakreslených objektů
- **určení správného pořadí ploch**

Zjednodušené varianty

- **explicitní pořadí kreslení**
  - např. funkce dvou proměnných: $z = f(x,y)$
- **hloubkové třídění** (“depth-sort”)
  - **setřídění objektů** (plošek) **podle souřadnice *z***
    (střed, těžiště)
  - dobře funguje při velkém množství malých objektů
  - nesprávná kresba velkých ploch (velká stolní deska s malými předměty)

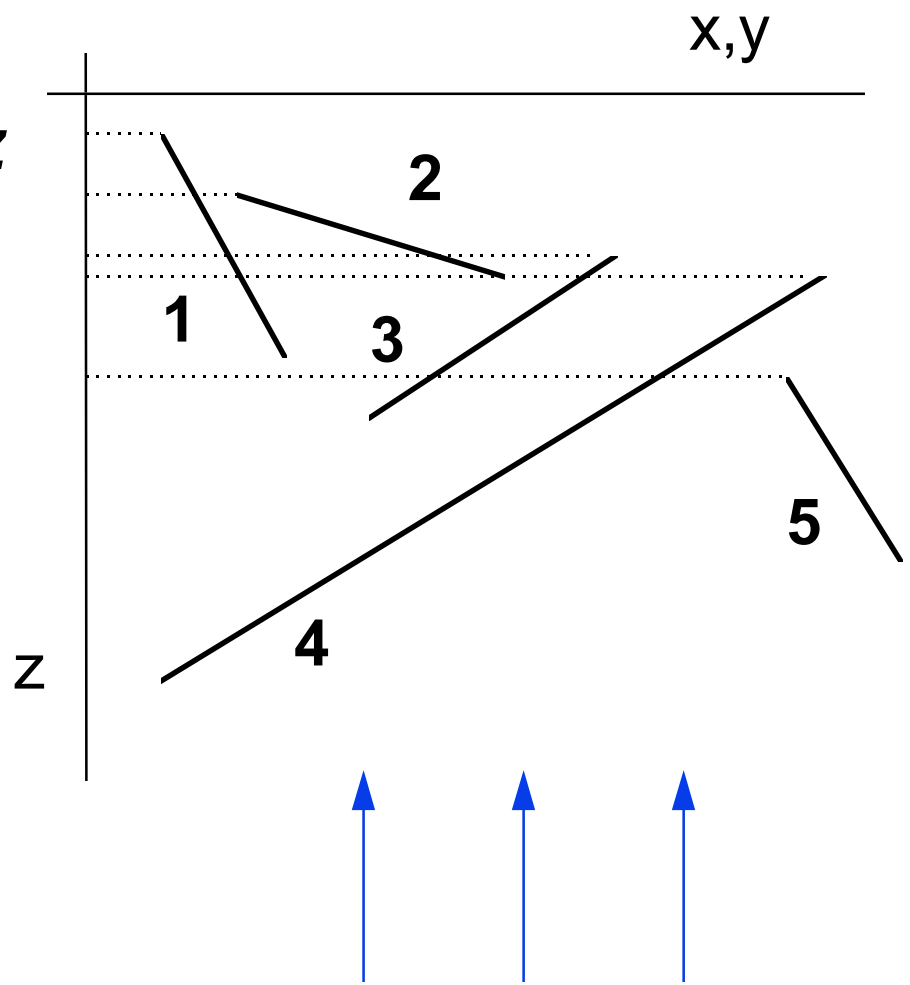
Korektní algoritmus:

- scéna je složena z **rovinných plošek**
- plošky mohou mít společné body **pouze na obvodu** (nesmějí se prosekávat)



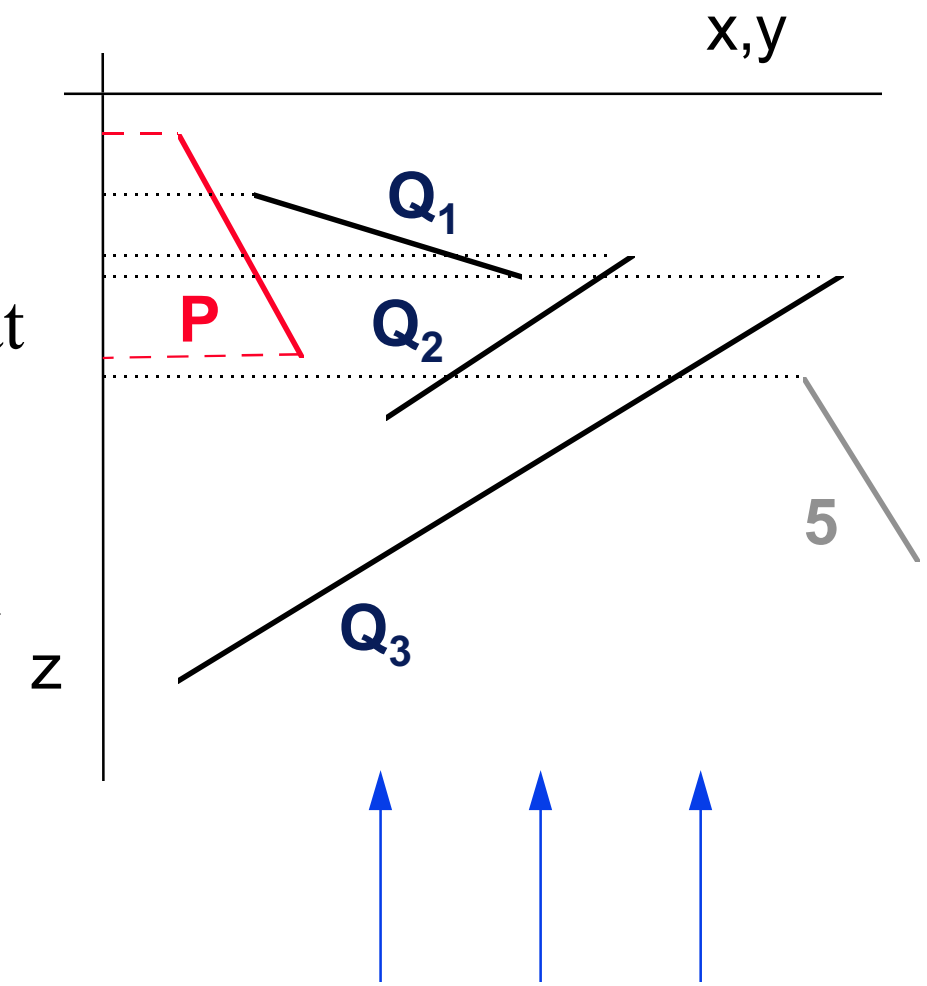
1. fáze: třídění

① plošky setřídíme podle **minimální souřadnice *z* vzestupně** - tj. odzadu dopředu - vytvoříme tak **vstupní seznam *S***



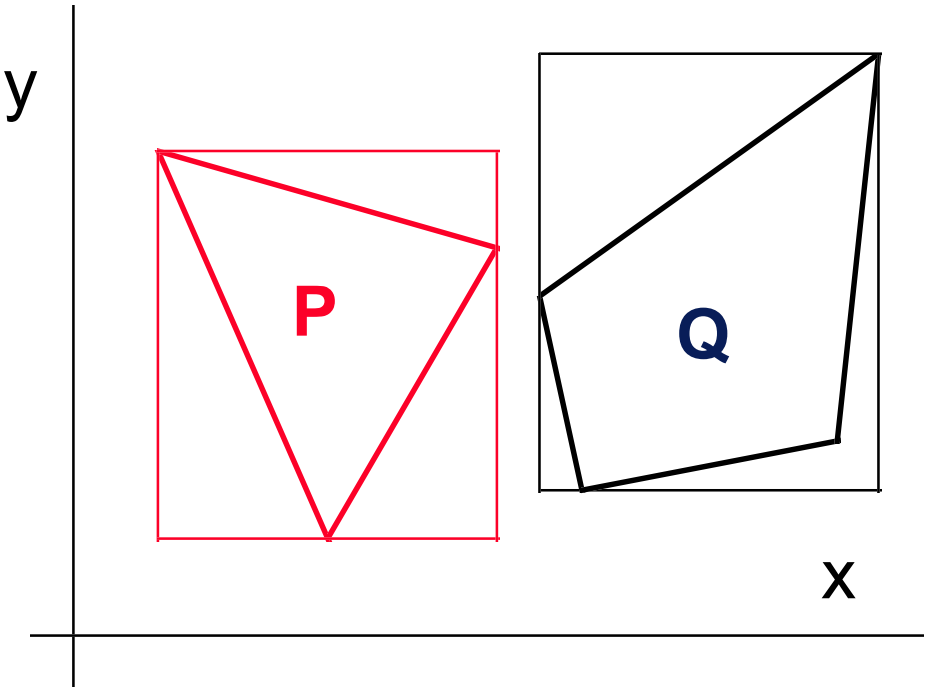
2. fáze: kontrola pořadí

② ze začátku seznamu ***S*** vezmeme plošku ***P*** - **kandidáta** pro kresbu. Proti ***P*** musíme otestovat ostatní plošky, které s ní mohou kolidovat. Právě testovanou plošku označíme ***Q***



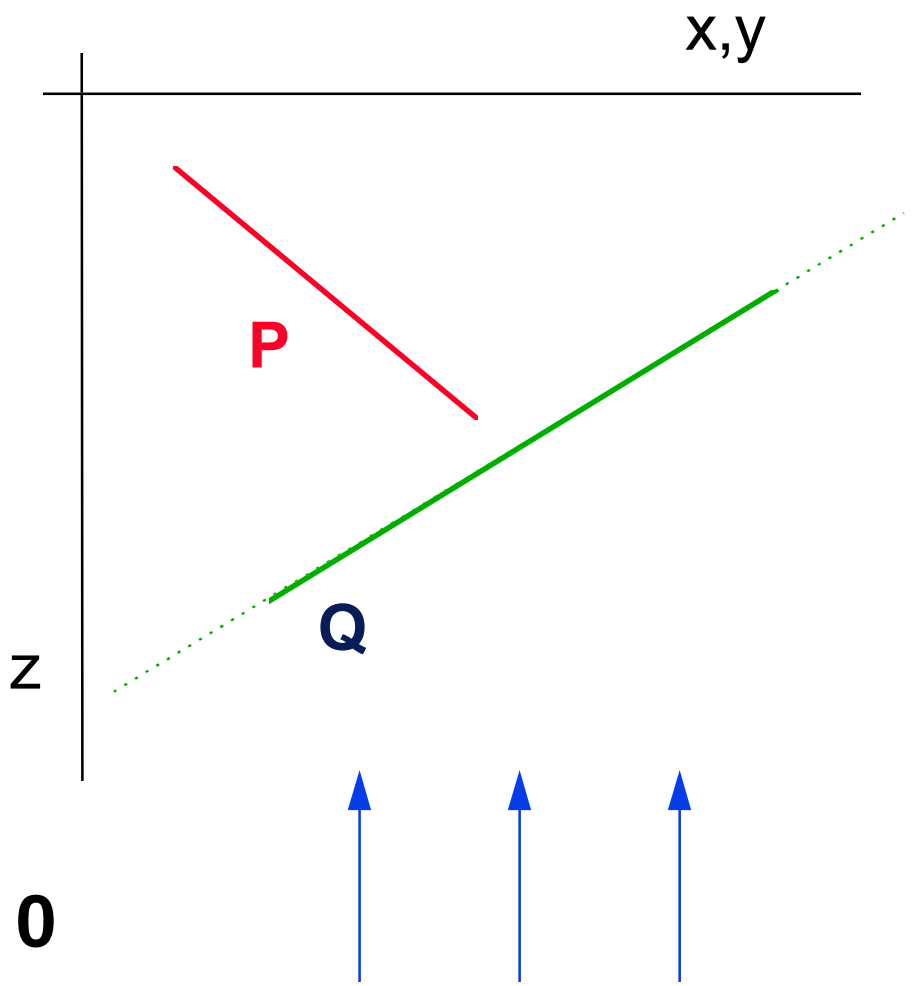
2.A fáze: “minimax test”

1. nejprve provedeme **nejjednodušší test - v průmětu** porovnáme obdélníky opsané oběma ploškám. Jestliže nemají společný bod, testování ***Q*** končí.
   Jinak pokračujeme dalším testem ***P*** a ***Q***



2.B fáze: *P* versus rovina *Q*

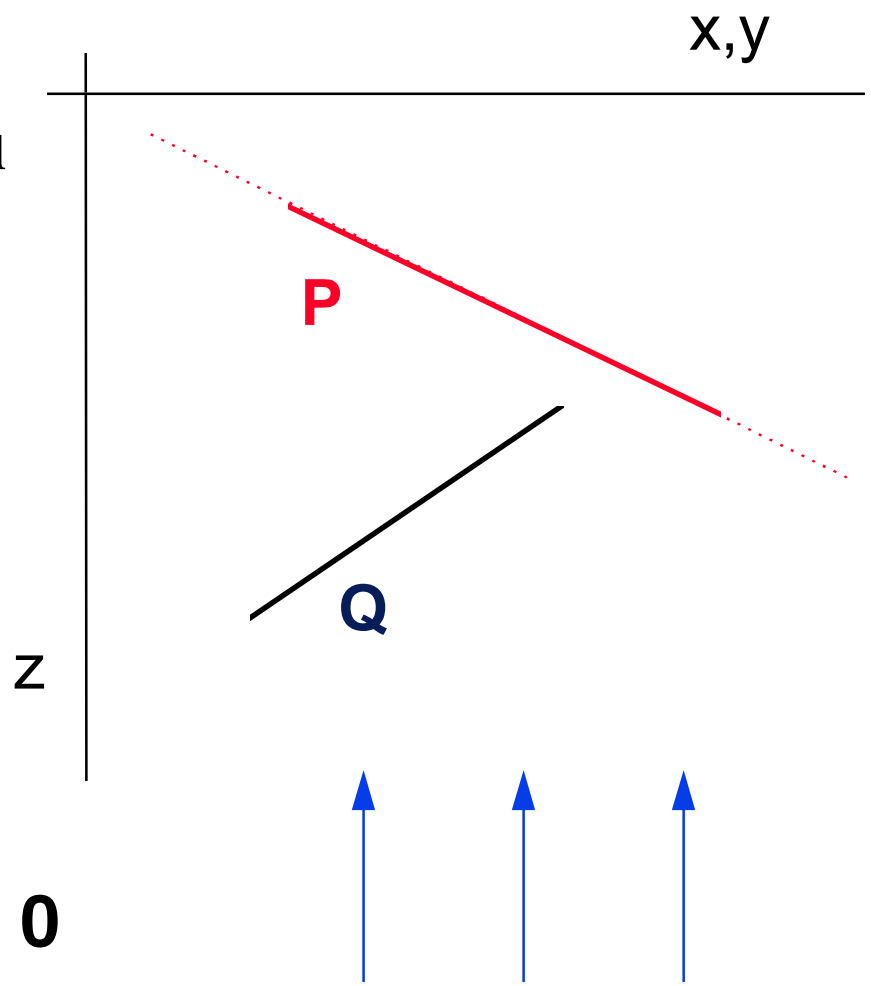
❷ dále testujeme, zda ploška ***P*** neleží celá **za rovinou** danou ploškou ***Q***. V kladném případě testování ***Q*** končí. Jinak pokračujeme dalším testem ***P*** a ***Q***



a· *x* + b· *y* + c· *z* + d < 0

2.C fáze: ***Q*** versus rovina ***P***

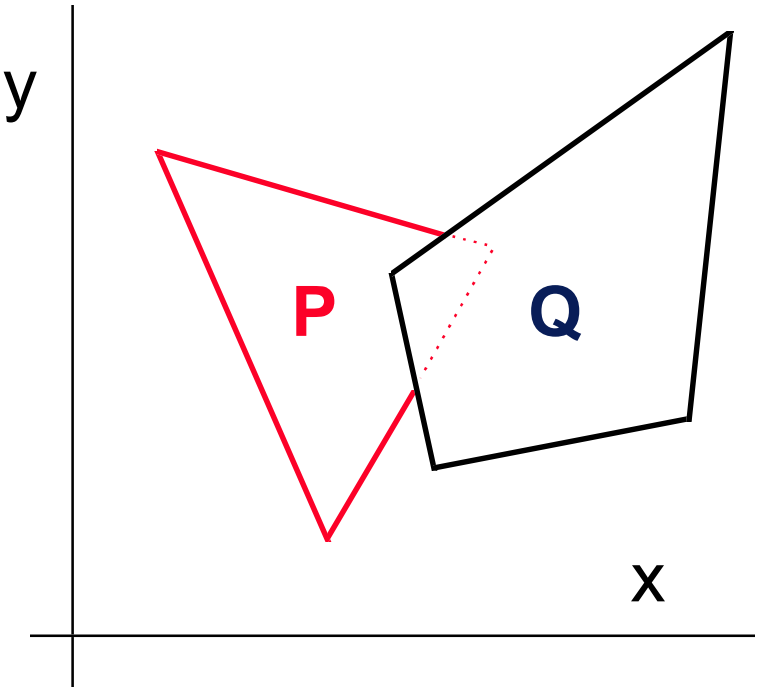
❸ testujeme, zda ploška ***Q*** neleží celá **před rovinou** danou ploškou ***P***.
V kladném případě testování ***Q*** končí.
Jinak pokračujeme dalším testem ***P*** a ***Q***



a· *x* + b· *y* + c· *z* + d > 0

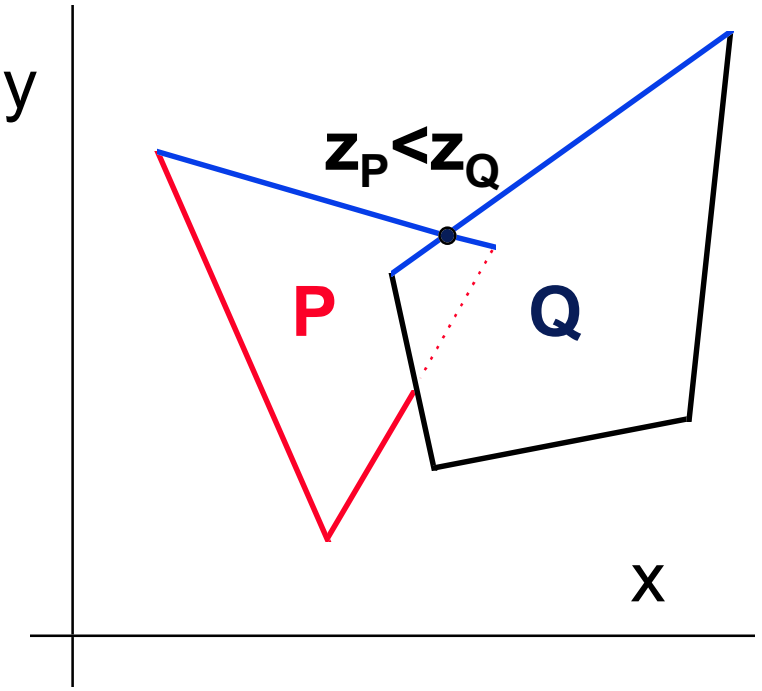
2.D fáze: úplný test v průmětu

❹ pokud předchozí testy neuspěly, musíme provést **úplný test** plošek ***P*** a ***Q*** **v průmětu**.
Je potřeba zjistit, zda není některá část ***Q*** překrytá ploškou ***P***.
V takovém případě by nešlo nakreslit ***P*** před ***Q***!



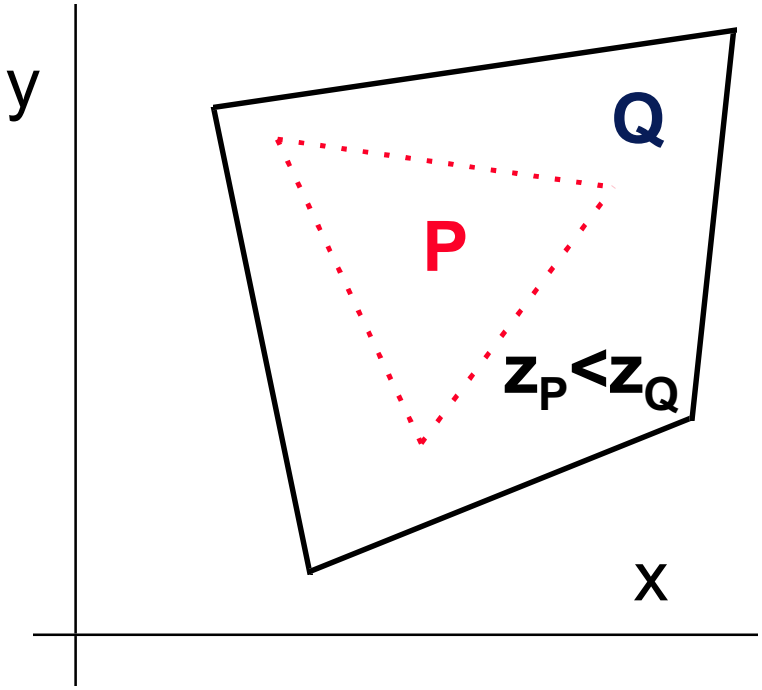
2.D fáze: úplný test v průmětu

- 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. Jinak nelze ***P*** nakreslit před ***Q***!



2.D fáze: úplný test v průmětu

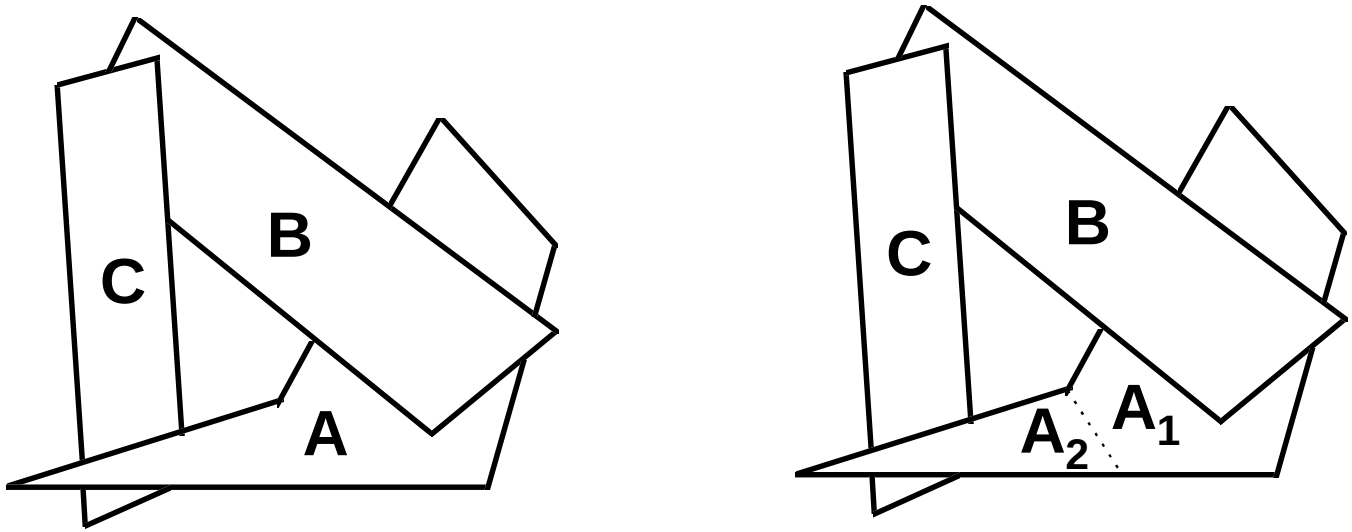
- jestliže neexistuje průsečík hran ***P*** a ***Q***, je třeba ještě zkontrolovat, zda neleží ploška ***P*** **celá uvnitř** ***Q*** nebo naopak. V takovém případě opět porovnáme souřadnice ***z***



2. fáze: změna pořadí

- jestliže nelze z nějakého důvodu nakreslit ***P*** před ***Q***, zkusíme přesunout plošku ***Q*** **na začátek seznamu *S*** (ještě před P)
  - pro ***Q*** budeme opět provádět všechny testy 2. fáze (jak jsme je popsali s ploškou ***P***)
  - testy nového kandidáta ***Q*** proti ***P*** už byly z velké části provedeny, stačí pouze doplnit obrácené testy **B** a **C**
- kvůli **zacyklení** se musí každý kandidát označit zvláštním příznakem

2. fáze: zacyklení



- jestliže je testován některý kandidát podruhé, došlo k **zacyklení**
- cyklus lze odstranit **rozštěpením** některé plošky (správné pořadí je pak **A_1**, **B**, **C**, **A_2**)

Konec

Další informace:

- **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
- **LAN** na Malé Straně:
  - **barbora\usr:\vyuka\pelikan\6**

Z-buffer (paměť hloubky)

KSVI MFF UK Praha

e-mail: Josef.Pelikan@mff.cuni.cz

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

Z-buffer (paměť hloubky)

- **kreslení do bufferu**
  – video-RAM, rastrová tiskárna s bufferem
- **vyplňování ploch**
  – lze i stínovat

➡ **není třeba třídit**

➡ **správné vykreslení nestandardních situací**
  – prosekávání ploch, cyklické zákryty, ...

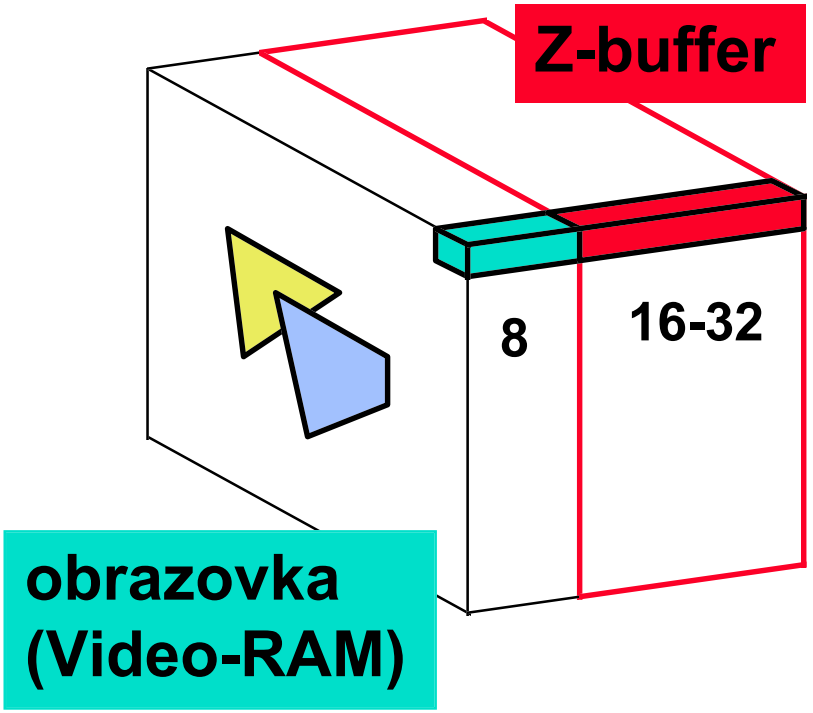
Paměť hloubky

Pro každý pixel ukládám:

- **barvu** (Video-RAM)
- **hloubku** = vzdálenost od pozorovatele (**Z-buffer**)

real: jednodušší

integer: rychlejší, úspora paměti (16-32 bitů)



Algoritmus:

1. **inicializace:**
   - Video-RAM := *barva pozadí*
   - Z-buffer := "*nekonečno*"
2. **zápis všech objektů do Z-bufferu:**
   - rozložení na jednotlivé pixely (vyplňování)
   - test hloubky

```
WritePixel ( x, y, z, barva : integer );
  if z < Zbuf[x,y] then begin
   Zbuf[x,y] := z;
   PutPixel(x,y,barva);
  end;
```

Výhody Z-bufferu

- **jednoduchost výpočtů**
  - celočíselná aritmetika
  - HW implementace 500k až 20M plošek/s
- **není nutné třídění**
- správné vykreslení **nestandardních situací**
- **nemusím** kreslit pouze **rovinné plošky**
  - rutina pro rozklad objektu na pixely (s výpočtem hloubky ***z***)

Nevýhody Z-bufferu

- **velká spotřeba paměti** (kdysi)
  - 800 x 600 x 16 bitů = 937 KB
  - 1024 x 768 x 24 bitů = 2.25 MB
- některé pixely ve Video-RAM se **několikanásobně překreslují**

Úspora paměti hloubky

- **kreslení po pásech**
  - Z-buffer pro **vodorovný pás** obrazu
  - více průchodů scénou (pro každý pás jeden)
  - ořezávání
- “**řádkový Z-buffer**”
  - každá řádka se kreslí zvlášť
  - větší efektivita: seznam aktuálních stěn (hran, objektů)

Konec

Další informace:

- **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
- **LAN** na Malé Straně:
  - **barbora\usr:\vyuka\pelikan\6**

Watkinsův algoritmus řádkového rozkladu

KSVI MFF UK Praha

e-mail: Josef.Pelikan@mff.cuni.cz

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

Watkinsův algoritmus

- **nepotřebuje výstupní buffer**
  - rastrový výstup generuje po jednotlivých řádkách
- **vyplňuje plochy**
  - lze i stínovat
- **žádné pixely se nekreslí zbytečně**
  - nepřekresluje se
  - výhodné zejména pro stínování
- vychází z 2D algoritmu - **vyplňování n-úhelníka v rovině**

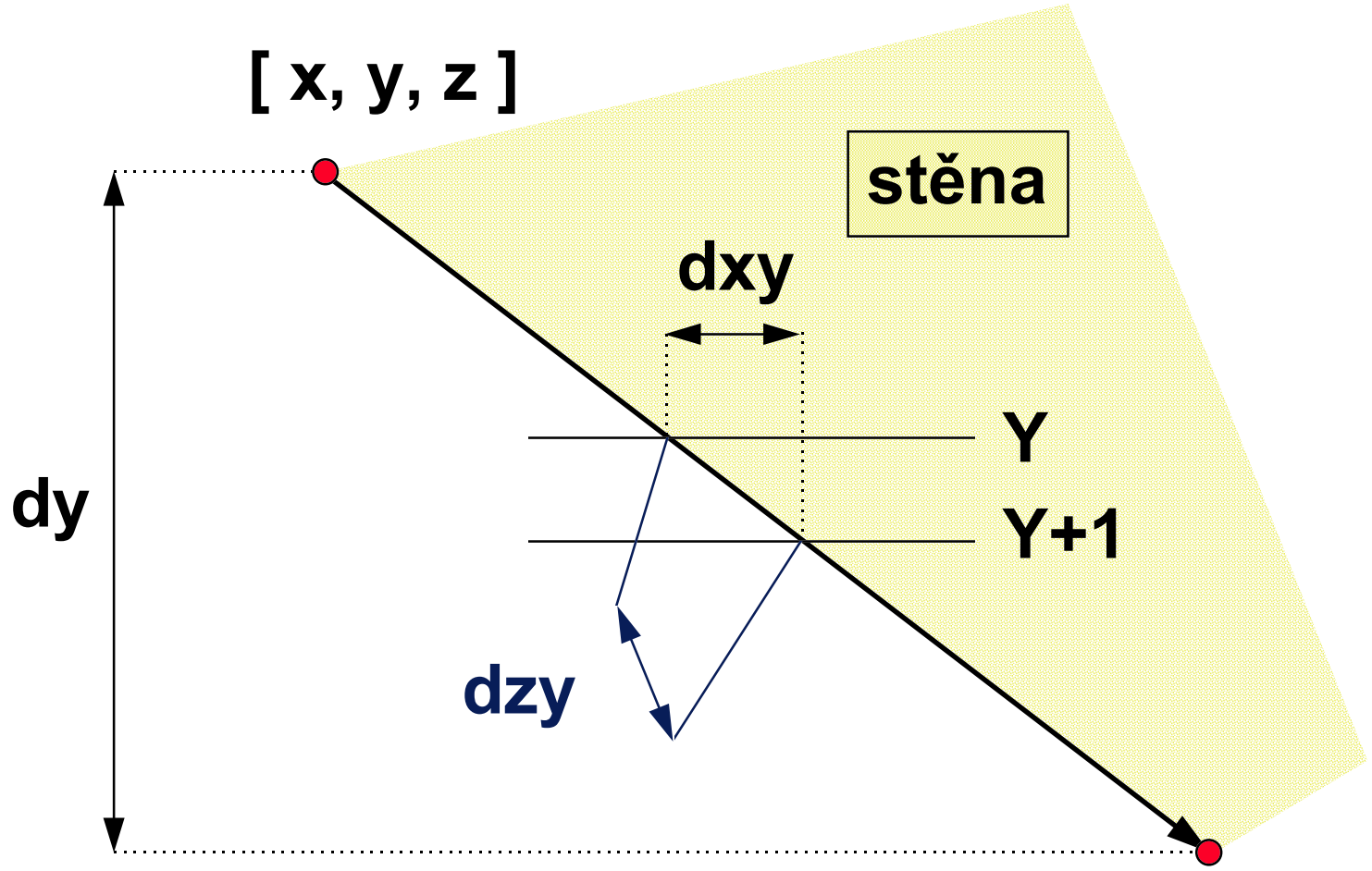
Předpoklady:

- scéna je složena z **rovinných plošek**
- každá ploška je zadána posloupností svých vrcholů (3D souřadnice, ***z*** je hloubka)
- ✓ **zjednodušení**: plošky smějí mít společné body pouze na obvodu (nesmějí se prosekávat)

1. předzpracování

- ze scény **odstraníme odvrácené plošky**
  – podle normálového vektoru
- přivrácené plošky rozložíme na **jednotlivé hrany**
- odstraníme **vodorovné hrany**
- pro ostatní hrany vytvoříme tzv. **pracovní záznamy**

pracovní záznam pro hranu:



Pracovní záznam pro hranu:

```
x : real;
```
{ ***x*** horního koncového bodu, později souřadnice průsečíku s aktuální řádkou }

```
y : integer;
```
{ ***y*** horního koncového bodu }

```
z : real;
```
{ ***z*** horního koncového bodu }

```
dy : integer;
```
{ výška hrany v pixelech: ***|y2-y|*** }

```
dxy : real;
```
{ změna ***x*** při posunutí na následující řádku (směrnice pro ***x***): ***(x2-x)/dy*** }

```
dzy : real;
```
{ změna ***z*** při posunutí na následující řádku (směrnice pro ***z***): ***(z2-z)/dy*** }

```
st : ^stena;
```
{ odkaz na stěnu, do které patří }

Záznam pro stěnu:

```
dzx : real;
```
{ změna ***z*** při posunutí na následující pixel (směrnice pro ***z***): ***dz/dx*** }

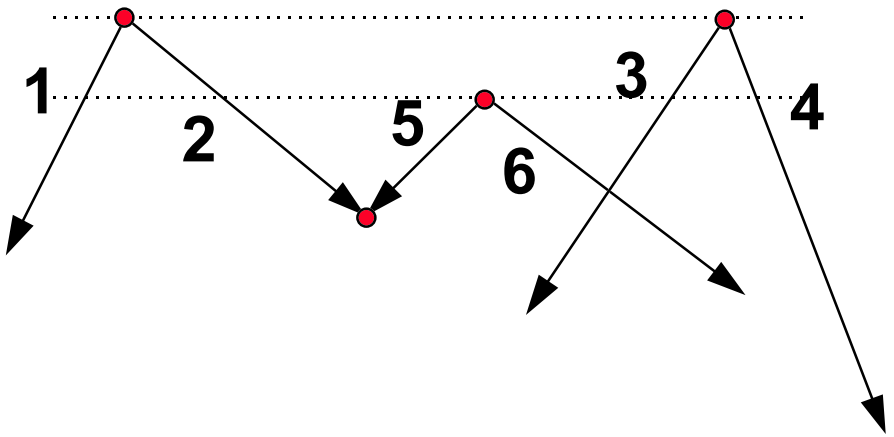
```
barva : barvy;
```
{ barva stěny }

```
. . .;
```
{ další pomocné údaje pro výpočet viditelnosti }

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***

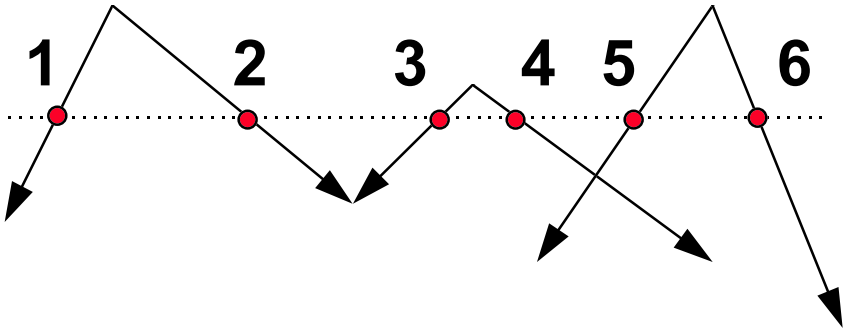


3. inicializace seznamu ***A***

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

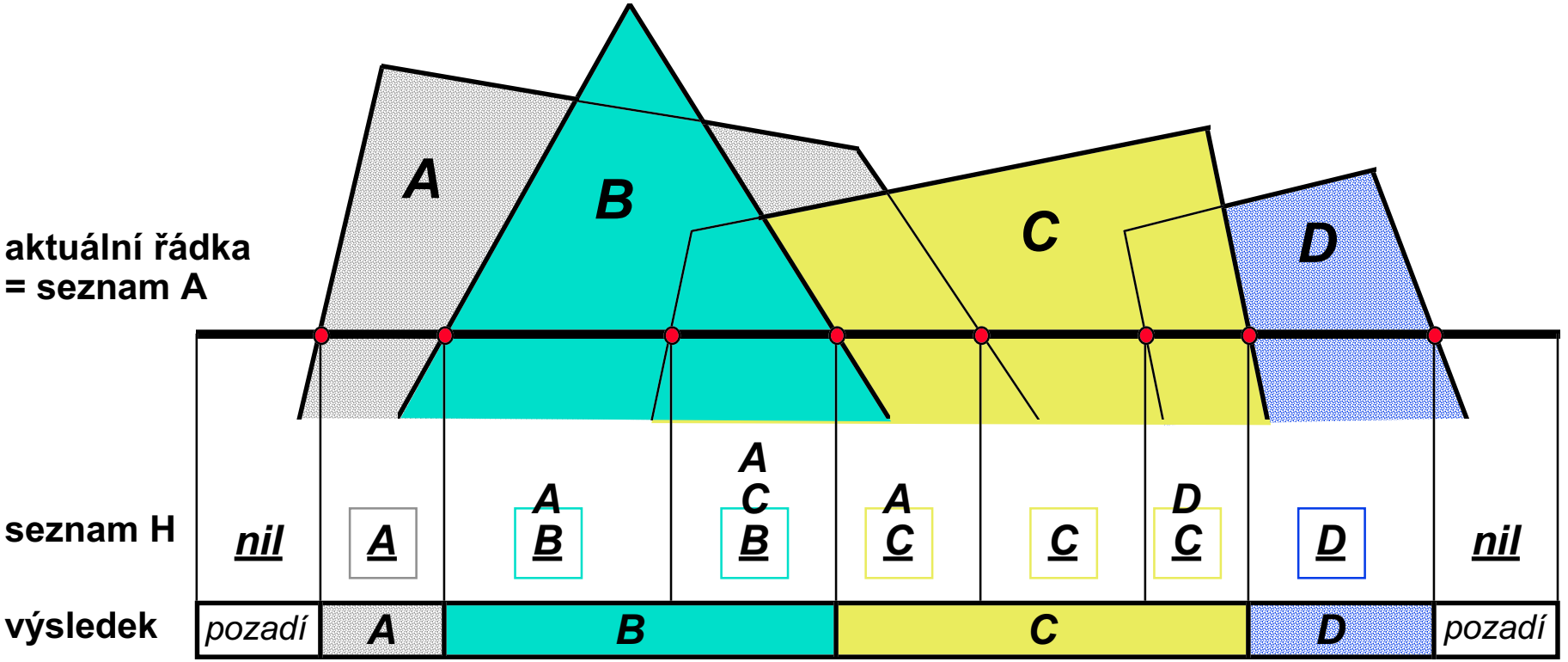
❷ vzestupně podle ***x***

❸ vzestupně podle ***dxy***



Na začátku zařadíme do ***A*** **počáteční úsek seznamu** ***S*** - hrany se shodným (minimálním) ***y***

4. výpočet viditelnosti na řádce



4. výpočet viditelnosti na řádce

Je třeba projít **aktuální seznam** ***A*** a určit viditelnost. K tomu se dá použít **pomocný seznam stěn** ***H*** seřazených podle hloubky:

① procházíme ***A*** a na stěnu každé hrany se podíváme do ***H***

② je-li stěna v seznamu ***H***, odstraníme ji a naopak

Při zatřiďování stěn do ***H*** používáme ***z***, ***dzy***

➡ **první prvek** ***H*** určuje viditelnou stěnu, pokud je ***H*** prázdný, kreslíme **barvu pozadí**

5. přechod na další řádku

Aktualizace seznamu *A*:

```
dy := dy - 1;
 if dy=0 then  "odstraň hranu ze seznamu A"
x := x + dxy;
z := z + dzy;
```

- kontrola setřídění *A*
- zatřídění nových hran z ***S*** do ***A*** (počáteční úsek ***S***)

6. podmínka ukončení cyklu

- jestliže je alespoň jeden ze seznamů ***A***, ***S*** **neprázdný** a ještě jsme nedokreslili celou obrazovku, výpočet pokračuje krokem **4**

- jinak algoritmus **končí**
  - případný nedokreslený zbytek obrazovky vybarvíme barvou pozadí

Průnik dvou ploch

- v seznamu ***H*** si vymění dvě plochy své pořadí během procházení aktuální řádky

- do ***A*** přidáme umělou **pomocnou hranu**

- v určování viditelnosti se vrátíme na jejího **předchůdce**

Konec

Další informace:

- **J. Foley, A. van Dam, S. Feiner, J. Hughes**: ***Computer Graphics, Principles and Practice***, 680-686

- **LAN** na Malé Straně:
  - **barbora\usr:\vyuka\pelikan\6**

Warnockův algoritmus (dělení obrazovky)

KSVI MFF UK Praha

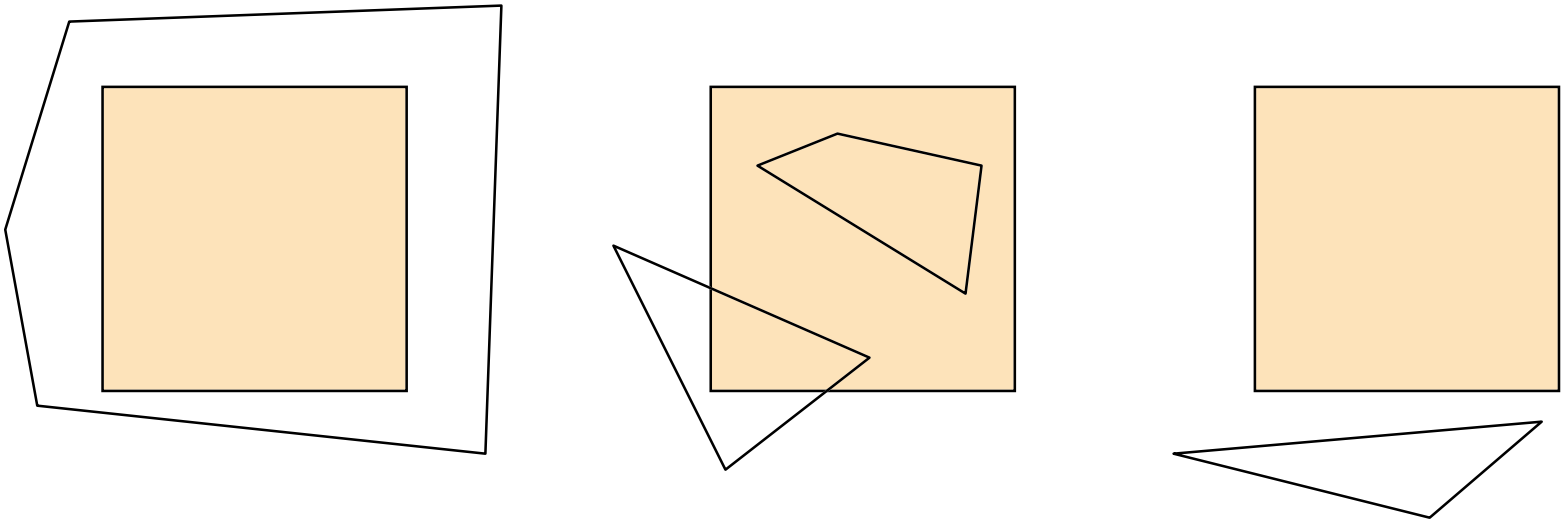
e-mail: Josef.Pelikan@mff.cuni.cz

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

Warnockův algoritmus

- **vyplňování ploch v rastrovém prostředí**
  - – každý pixel se kreslí pouze jednou
  - – lze stínovat
- **scéna složená z rovinných stěn**
  - – plošky se mohou prosekávat
- **metoda “rozděl a panuj”**
- **jednoduché případy** se kreslí přímo
  - – složitější situace se řeší rekurzivním dělením

Poloha stěny vzhledem k oknu



1. stěna pokrývá okno

2. stěna zasahuje do okna

3. stěna nezasahuje do okna

Výpočet viditelnosti v okně

1. žádná stěna **nezasahuje** ani **nepokrývá** okno
   – okno vyplníme barvou pozadí
2. jediná stěna **pokrývá** okno, ostatní do něj nezasahují
   – okno vyplníme barvou stěny
3. pouze jedna stěna **zasahuje** do okna
   – okno vyplníme barvou pozadí a pak nakreslíme stěnu (ořezanou vzhledem k oknu)

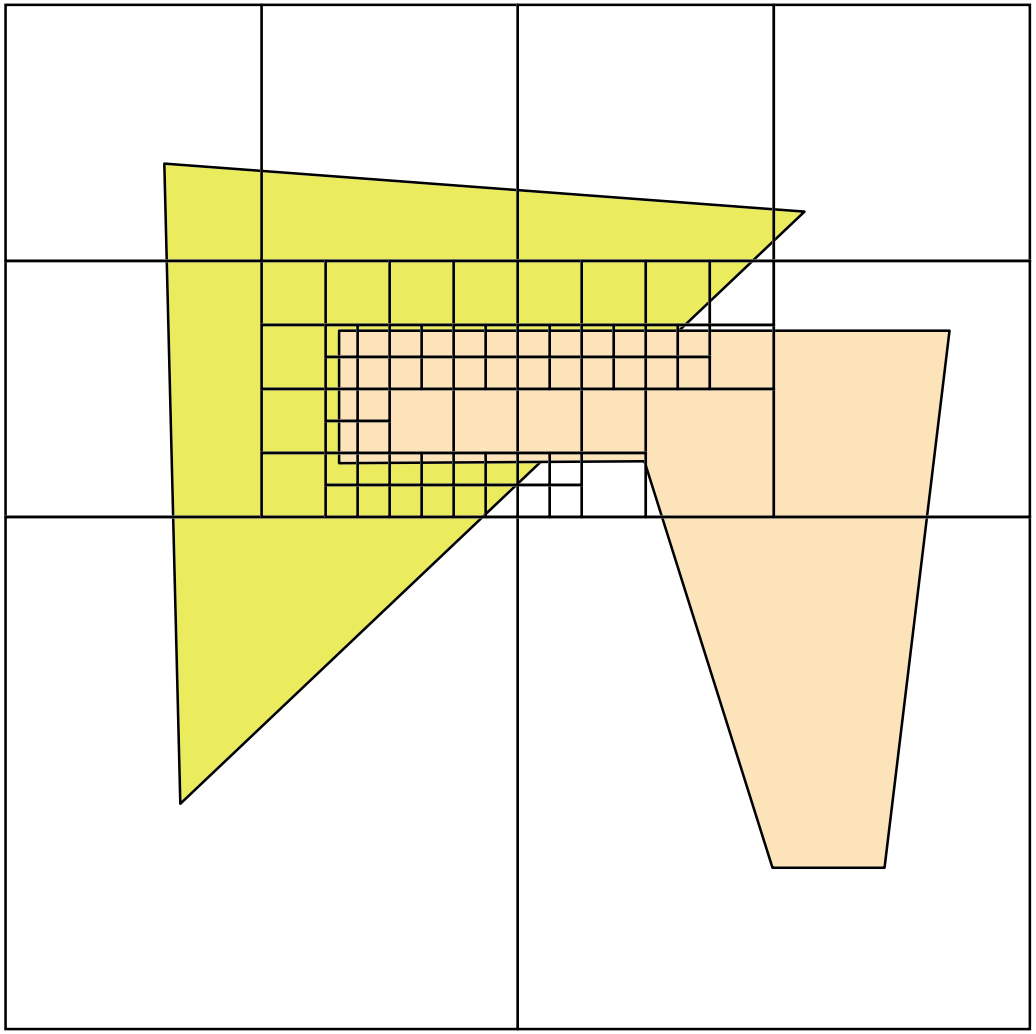
Výpočet viditelnosti v okně

❹ několik stěn **zasahuje** nebo **pokrývá** okno, jedna z pokrývajících leží před všemi ostatními
- testy se provádějí v rozích okna
- okno vyplníme barvou přední stěny

❺ nenastává žádný z předchozích případů
- okno **rozdělíme na čtyři** (shodné) **části** a v každé z nich aplikujeme stejný algoritmus rekurzivně

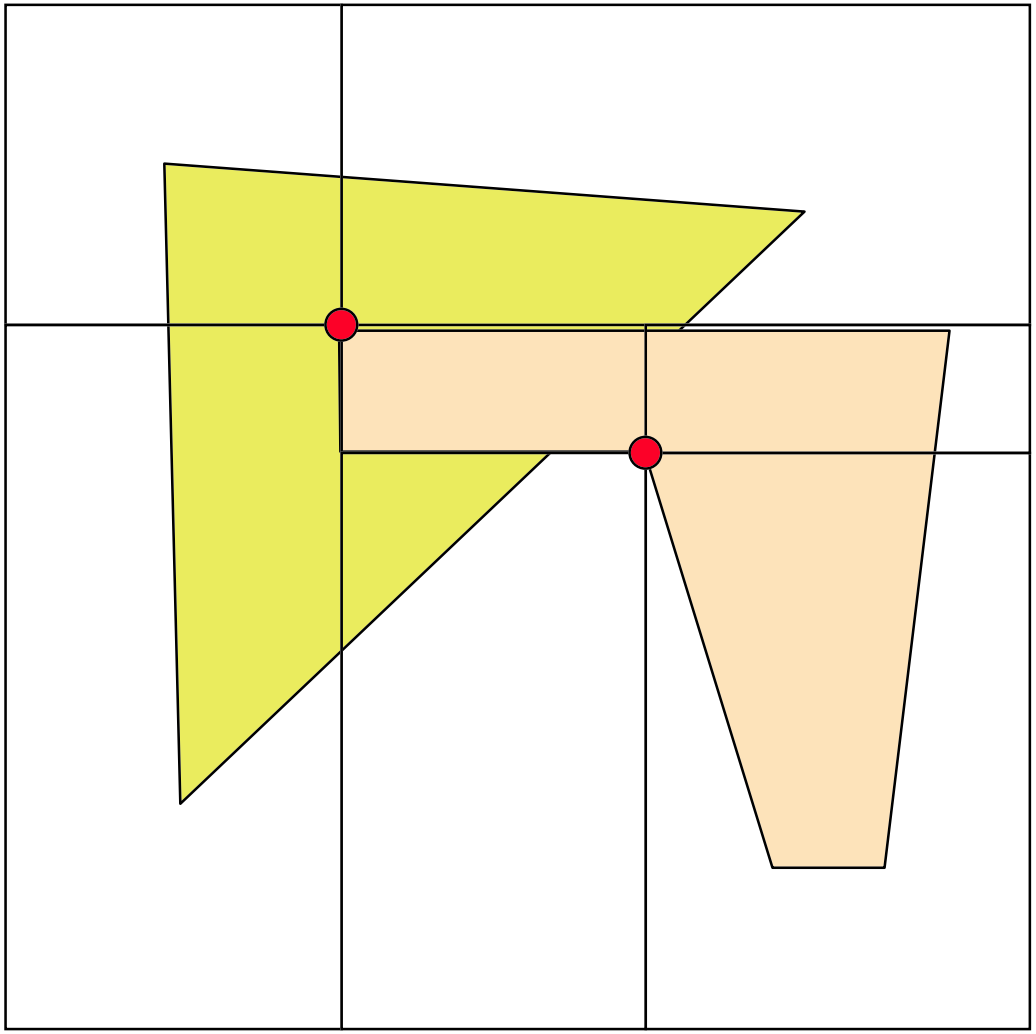
➡ v případě potřeby dělíme až na úroveň pixelů
- pixel nakreslíme barvou nejbližší stěny

Rekurzivní dělení okna



pravidelné dělení

Rekurzivní dělení okna



dělení podle vrcholů

Implementace

- testování **zákrytu stěn** v okně:
  - každou stěnu proložíme rovinou
  - porovnáváme hloubky (souřadnice **z**) v rozích okna
  - pokrývající stěna leží přede všemi ostatními, když její rovina je nejblíž ve **všech čtyřech rozích okna**

- při dělení okna počítáme **hloubky v nových vrcholech** pouze pomocí starých hodnot
  - dělení na shodné části: aritmetický průměr
  - obecné dělení: trojčlenka

Implementace

- **seznamy incidentních stěn** pro každé okno:
  - stěny pokrývající okno
  - stěny zasahující do okna
- **aktualizace seznamů** při dělení okna:
  - pokrývající stěny se dědí beze změny
  - zasahující stěny se testují proti novému oknu (mohou vypadnout ze hry, zůstat zasahujícími nebo se stát pokrývajícími)

Konec

Další informace:

- **J. Foley, A. van Dam, S. Feiner, J. Hughes**: ***Computer Graphics, Principles and Practice***, 686-689
- **Jiří Žára a kol.**: ***Počítačová grafika***, principy a algoritmy, 297-298