# V. Počítačová grafika a analýza obrazu

Update: 9. května 2018

## Obsah

# 1 Osvětlovací modely a systémy barev v počítačové grafice.

# 2 Afinní a projektivní prostor. Afinní a projektivní transformace a jejich matematický zápis. Aplikace v počítačové grafice. Modelovací a zobrazovací transformace.

## 2.1 Afinní prostor - $A_n$

- je to prostor s body
- dále obsahuje **přidružený vektorový prostor** (souřadný systém) pomocí kterého je možné jednotlivé body prostoru zaměřit
- součásti afinního prostoru je také zobrazení, které přiřadí dvojici bodů vektor
- dimenze vektorového prostoru určuje dimenzi afinního prostoru
- ukázka afinného prostoru
  - v trojrozměrném afinném prostoru $A_3$ máme bod $X$ se souřadnicemi $X = (x_1, x_2, x_3)$
  - vektor **x** je prvek onoho přidruženého vektorového prostoru

## 2.2 Euklidovský prostor - $E_n$

- afinní prostor ve kterém je zaveden skalární součin(1) a norma(2) (velikost vektoru), to umožnuje měřit délku vektorů a úhly mezi nimi
- souřadný systím (kartézský, polární, válcový atd.)

$$Vektory a = (a_1, a_2, a_3), b = (b_1, b_2, b_3)$$
$$a \cdot b = |a| \cdot |b| \cos \alpha, \tag{1}$$
$$a \cdot b = a_1 b_1 + a_2 b_2 + a_3 b_3$$

$$|a| = \sqrt{a_1^2 + a_2^2 + a_3^2} \tag{2}$$

## 2.3 Kartézská souřadná soustava

- souřadné osy **vzájemně kolmé**
- protínají se v jednom bodě – **počátku soustavy souřadnic**
- jednotka se obvykle volí na všech osách stejně velká
- souřadnice polohy bodu je možno dostat jako kolmé průměty polohy bodu k jednotlivým osám

## 2.4 Afinní transformace

- Afinné transformace je zobrazení bodů jednoho afinního prostoru do jiného afinního prostoru (speciální případ: zobrazení do téhož afinního prostoru (bijekce); tomu se říká afinita).

- Afinní transformace souřadnic je geometrickou transformací bodu $P = [x, y]$, jehož obrazem je bod $Q = [x', y']$, které spočívá v **posunutí** (translation), **otáčení** (rotation), **změně měřítka** (scaling), **zkosení** (shearing) nebo operaci **vzniklé jejim skládáním**.
- Afinní – rovnoběžným přímkám odpovídají opět rovnoběžné přímky, které však nemusí být rovnoběžné s původními přímkami.
- Geometrické transformace jsou jedněmi z nejčastěji používaných operací v PG.

Když zavedeme následující vektory a matici:

$$y = (y_1, y_2, y_3), x = (x_1, x_2, x_3), t = (t_1, t_2, t_3),$$

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}.$$

Můžeme afinní transformaci zapsat jako

$$y = xA + t.$$

**y** je bod, do něho je transformován bod **x** pomocí matice **A** a vektoru **t**. Vektor **t** slouží k posunutí středu souřadné soustavy, matice **A** mění osy souřadné soustavy. Známe–li **A** a vektor **t**, můžeme transformaci jednoduše provést. Jindy se musí určit ze zadání.

## 2.5 Ortonormalita afinní transformace

- jsou takové transformace, které **nemění délky** ani **úhly**
- délky a úhly souvisejí se **skalárním součinem**, když se tento součin po transformaci **nezmění**, jsou zachovány délky i úhly
- vlastnosti ortonormální transformace:
  - afinní transformace bude zachovávat hodnotu prévě tehdy, když $AA^T = I$, kde $I$ je jednotková matice (**nutná a postačující podmínka ortonormality**)
  - také když uvážíme, že $A^{-1} = A^T$, pak platí $A^{-1}A^T = I$
  - determinant matice $detA$ musí být roven $\pm 1$, protože $detAA^T = detI$ a $detI = 1$

## 2.6 Afinní transformace (2D)

- **posunutí** (translation)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b_x \\ b_y \end{bmatrix}$$

- **otáčení** (rotation) –ve směru ručiček (naopak prohodíš sin)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

- **změna měřítka** (scaling)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

- **zkosení** (shearing)

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & sh_y \\ sh_x & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Skládání transformací(příklad):

Transformujte bod $A[3,4]$ posunem o vektor $(-5,1)$ a rotací o úhel 90° stupňů

$$posun: \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} -5 \\ 1 \end{bmatrix} \qquad rotace: \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} -5 \\ -2 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \left( \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 4 \end{bmatrix} + \begin{bmatrix} -5 \\ 1 \end{bmatrix} \right) + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

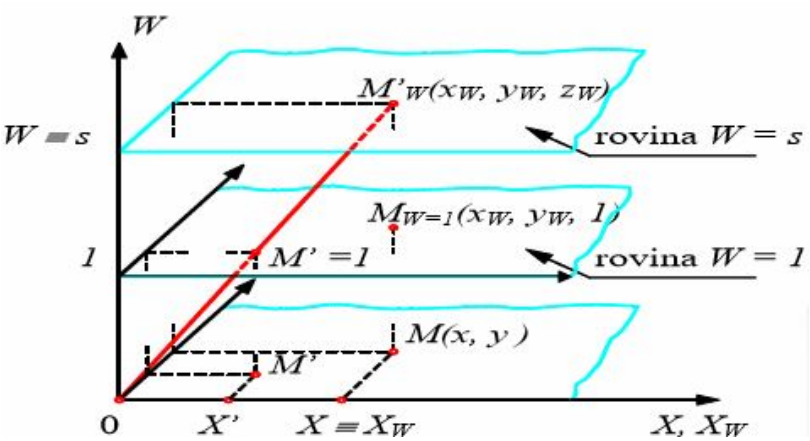
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a'_{11} & a'_{12} \\ a'_{21} & a'_{22} \end{bmatrix} \left( \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \right) + \begin{bmatrix} b'_1 \\ b'_2 \end{bmatrix}$$

Obecná skládání transformací v kartézské souřadné soustavě

$$[X'] = A_n ... (A_2(A_1 \cdot X + b_1) + b_2) ... + b_n$$

## 2.7 Homogenní souřadnice

- myšlenkou je reprezentace bodu ve vektorovém prostoru o jednu dimenzi větší
- rozšíření o jednu dimenzi (expanze z 2D do 3D, popř. z 3D do 4D)
- bod $(x, y)$ v homogenních souřadnicích $(wx, wy, w)$ kde $w \neq 0$
- nejčastěji volíme homogenní souřadnici $w = 1$
- bod se souřadnicemi $(X, Y, W)$ má kartézské souřadnice $x = X/W$ a $y = Y/W$




- transformace se při použití homogenních souřadnic omezí pouze na násobení matic (jednodušší) $Q = M \cdot P$

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

- umožnují použití projektivním transformací (perspektivní promítání)

- **posunutí** (translation)

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

- **otočení kolem osy x** (rotation)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **otočení kolem osy y** (rotation)

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **otočení kolem osy z** (rotation)

$$\begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **změna měřítka** (scaling)

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **zkosení** (shearing)

$$\begin{bmatrix} 1 & sh_y & 0 & 0 \\ sh_x & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

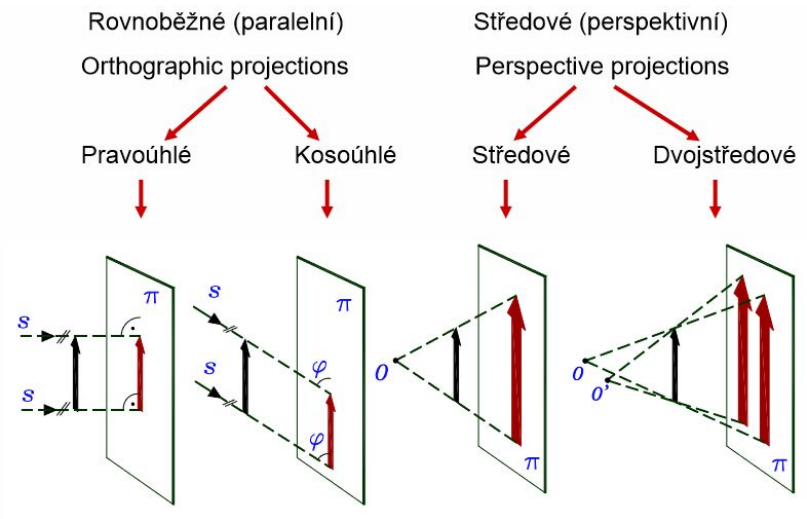
## 2.8 Projektivní transformace - kolineace

- kolineace je zobrazení bodů jednoho prostoru na body stejného nebo jiného prostoru
- matematický záps kolineace je $y = xT$
- sehrává zásadní úlohu v grafických systémech

## 2.9 Promítání

- v geometrii nejprve volíme promítací metodu a potom v této zobrazujeme objekty (v PG naopak)
- nejprve vytvoříme objekt a následně volíme zobrazovací metodu vhodnou pro požadovaný účel
- definice promítání:
  - **promítací paprsky** – polopřímka, vycházející z promítacího bodu, směr závisí na typu promítání
  - **průmětna** (viewing plane) – plocha v prostoru, na kterou dopadají promítací paprsky (paprsky vytvářející průmět)
  - průmětnou nemusí být pouze rovina (polokoule, NURBS plocha...)

## 2.10 Klasifikace promítacích metod



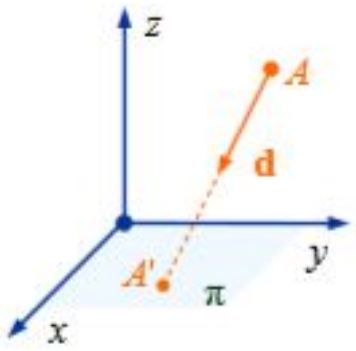

### 2.10.1 Rovnoběžné promítání

Orthographic nebo orthogonal projection

- Promítání je určeno průmětnou **a** a směrem **s**, který není rovnoběžný s průmětnou
- z řeckého „orthos“ rovný a „graphe“ kreslení.

$$\begin{bmatrix} 1 & 0 & -\frac{d_x}{d_z} & 0 \\ 0 & 1 & -\frac{d_y}{d_z} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

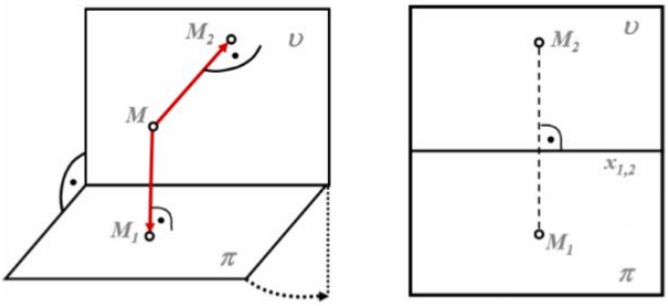
Matice popisuje rovnoběžné promítání na rovinu $xy$. Směr promítacího paprsku je $d = (d_x, d_y, d_z)$(viz obr).




Jednička na pozici $3,3$ v matici zajišťuje, že se transformací nezmění souřadnice z bodu. Toho opět využívá řešení viditelnosti.

## 2.11 Mongeova projekce

- nejprve promítáme kolmo na vodorovnou rovinu $\pi$ (půdorysnu) – promítací přímky jsou svislé, jde tedy o pohled shora (půdorys)
- poté promítáme kolmo na svislo rovinu $v$ (nárysnu) – promítací přímky jsou kolmé, jde tedy o pohled zpředu (nárys)
- pohledy kreslíme bez přihlížení k obsahu sklopené druhé průmětny, tudíž se obrazy v jednotlivých průmětnách prolínají a jejich polohu v souřadnicovém systému popisuje vzdálenost od základnice (osa $Y$) potažmo od nulového bodu




Souřadnice bodů: $M(x, y, z)$... 1. průmět $M1(x, y, 0)$ 2. průmět $M2(x, 0, z)$
V Mongeově projekci je těleso určeno svým nárysem a půdorysem

## 2.12 Kosoúhlé promítání

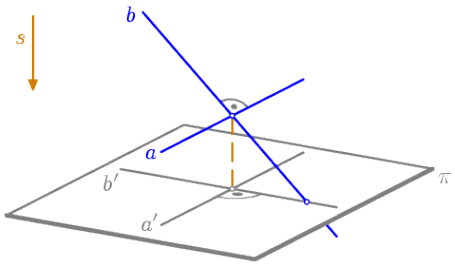
- je rovnoběžné promítání na jednu průmětnu směrem, který má odchulku $\varphi$ jinou než $90°$ od průmětny, promítací paprsky $S$ jsou tak rovnoběžné a ne kolmé k průmětně $\pi$. Průmětna $\pi$ je rovnoběžná s některou hlavních rovin
- výhodou tohoto způsobu je skutečnost, že předměty, které se nacházejí v nárysně jsou zobrazeny v reálné velikosti

## 2.13 Axonometrie - rovnoběžné, pravoúhlé promítání

- axonometrie nebo axonometrické projekce je jednoduchý způsob promítání prostorových těles a trojrozměrných struktur do roviny
- v rovině se nejprve zvolí tři osy $x, y, z$, jež spolu svírají stejné nebo nestejné úhly
- rozměry těles se pak nanášejí v určitém měřítku rovnoběžně s těmito osami
- hlavní výhoda axonometrie proti složitějším metodám promítáné je v tom, že průmět se snadno konstruuje, a že se z něho dají rozměry odečíst
- nevýhoda může být v tom, že v axonometrické projekci se rovnoběžky nesbíhají a tak je perspektivní dojem nedokonalý (může působit vizuální paradox)

## 2.14 Ortogonální promítání

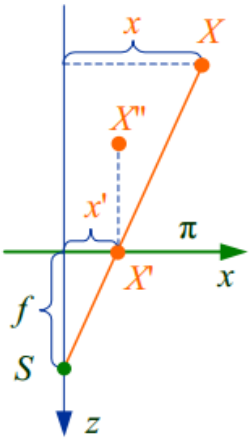
- směr promítání kolmý k průmětně (jedná se tedy o speciální případ rovnoběžného promítání)
- zachovávají se všechny vlastnosti rovnoběžného promítání




## 2.15 Perspektiva – středové promítání

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

Matice popisuje projekci ze středu o souřadnicích $(0, 0, f)$ na rovinu $z = 0$ (tedy na rovinu $xy$).



Ačkoliv se očekává, že po transformaci bodu bude jeho z souřadnice rovna 0, není to pravda. Tuto nenulovou hodnotu však lze využít například při řešení viditelnosti. Souřadnice $x$ a $y$ slouží k vykreslení na obrazovku.

## 2.16 Modelovací transformace

Jsou všechny transformace, pomocí nichž se vytváří scéna: posun, zkosení, rotace, změna velikosti.

## 2.17 Zobrazovací transformace

Transformace používané k zobrazení scény: středové a rovnoběžné zobrazení. Jsou dělány tak, aby výsledky padly do jednotkového zobrazovacího objemu - souřadnice z intervalu$< -1, 1 >$.

# 3 Křivky a plochy: teoretické základy (definice, rovnice, tečný a normálový vektor, křivosti, Cn a Gn spojitost), použití (Bézier, Coons, NURBS).

## 3.1 Křivky

Křivky dělíme na: **rovinné**, **prostorové**, **interpolační**, **aproximační**.

V počítači jsou reprezentovány jako soustava parametrů nějaké rovnice, která je posléze generativně zobrazována. Toto vyjádření může být v podstatě trojího druhu:

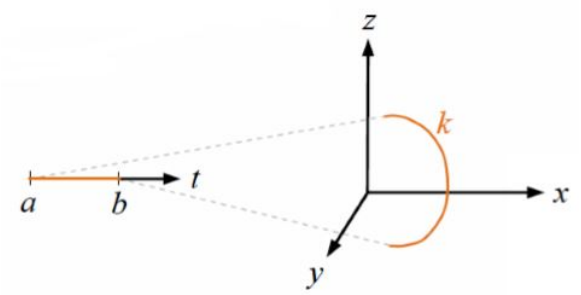
- **explicitní** – $y = f(x)$ kde $x \quad \varepsilon \quad \mathbb{I}.$,
  např. $y = x^2 + x + 1$, jedná se o parabolu.
- **implicitní** – $F(x, y) = 0$,
  $(x + 9)^2 + (y - 2)^2 - 4 = 0$, kružnice se středem $[-9, 2]$ a $r = 2$.
- **parametrické** –

$$x = f_x(t), \quad y = f_y(t), \quad z = f_z(t), parametr \quad t \quad \varepsilon < a, b > .$$

  $x = t, y = t^2, kde \quad t \quad \varepsilon \quad \mathbb{R}$, parabola s vrcholem v počátku.
- Jednoduchý příklad křivky je například kružnice nebo přímka.

### 3.1.1 Parametrické křivky

- Mějme interval $I =< a, b > \subseteq \mathbb{R}$.
- Parametricky vyjádřenou (parametrizovanou) křivku $k$ v $\mathbb{R}^n$ nazýváme diferencovatelné zobrazení $\varphi : I \to \mathbb{R}^n$




- V trojrozměrném euklidovském prostoru každému číslo $t$ odpovídá na křivce příslušný bod $P(t) = [x(t), y(t), z(t)]$.
- Polohový vektor $\mathbf{P}$ (vektor daný počátkem souřadné soustavy a souřadnicemi příslušného bodu $P$).

### 3.1.2 Interpolační křivky

- V PG nám pro definování (kreslení) křivky slouží její interpolace.
- Interpolační křivka k dané množině bodů je taková křivka, která jimi prochází.
- Obecně se dá křivka pro n bodů vyjádřit polynomem n1 řádu. To znamená, najít řešení soustavy pro každé:

$$x_0, x_1, ..., x_n, \qquad x_i \neq x_j \quad proi \neq j,$$

Hledáme interpolační polynom $P_n(x)$ stupně nejvýše $n$, který splňuje interpolační podmínky

$$P_n(x_i) = y_j \qquad i = 0, 1, ..., n.$$

### 3.1.3 Fergusonova křivka

- Křivka je pojmenovaná po James C. Ferguson z The Boeing Company.
- Je generování křivek **řízené dvěma body** a **vektory** v nich.
- **Hermitova interpolace** aplikovaná na složky vektorového vyjádření křivky pro jednotkovou změnu parametru na obloucích, získáme tzv. **Fergusonovy křivky**.

Ve vztahu obecně:

$$R(t) = F_0(t)G + F_1(t)H + F_2(t)g + F_3(t)h,$$

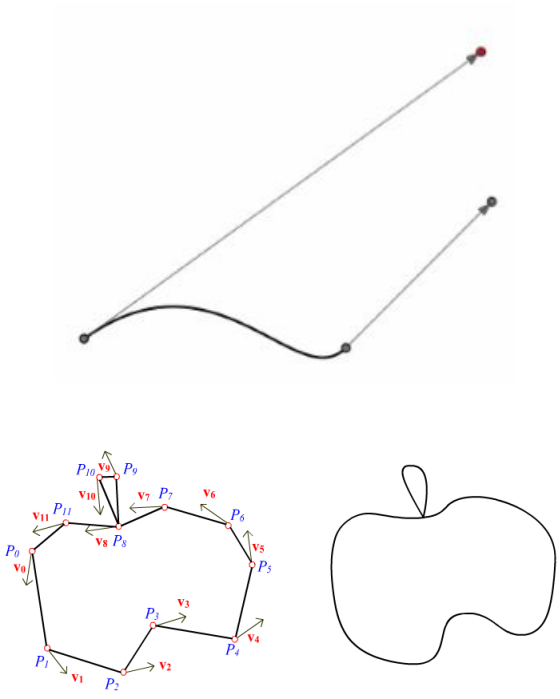
kde: F, G jsou polohové vektory bodů,
g,h jsou vektory tečen v bodech, ve kterých je Fergusonova křivka jednoznačně určena,
pro $F_i(t), i = 0, 1, 2, 3$, platí:

$$F_0(t) = 2t^3 - 3t^2 + 1,$$
$$F_1(t) = -2t^3 + 3t^2,$$
$$F_2(t) = t^3 - 2t^2 + t,$$
$$F_3(t) = t^3 - t^2, t\varepsilon < 0, 1 > .$$

Výsledný tvar Fergusonovy kubiky, lze ovlivnit třemi způsoby:

- polohou řídících bodů $V_0$ a $V_1$,
- směrem tečných vektorů $v_0$ a $v_1$,
- velikostí tečných vektorů $v_0$ a $v_1$.

Velikost vektorů $v_0$ a $v_1$ významně ovlivňuje výsledný tvar křivky. Čím délka tečných vektorů je větší, tím více se křivka přimyká k příslušnému tečnému vektoru.




Obrázek 1: Spline

### 3.1.4 Aproximační přímky

- Někdy není možné body proložit funkcí a proto se využívá aproximačních křivek, které procházejí v blízkosti bodů.
- Je dáno $n$ bodů. Úlohou je nalézt aproximační funkci, která nemusí procházet danými body, ale která co nejlépe vystihuje funkční závislost.
- Metoda nejmenších čtverců.

$$f : y = ax + b. \tag{3}$$

Cílem metody je dosáhnout co nejmenší počet kvadrátů euklidovského rozdělení mezi aproximovanou přímkou $f(y)$ a zadanými hodnoty $y_i$. Dále si nadefinujeme funci $c(a, b)$, která reprezentuje součet aproximované přímky a hodnotami $y_i$:

$$c(a, b) = \sum_{i=0} n(f(x) - y_i)^2, \tag{4}$$

kde $a, b$ jsou koeficienty aproximované přímky, $n$ je počet zadaných bodů.

Minimum této funkce $c(a, b)$ získáme parciální derivací jejími argumenty:

$$\frac{\partial c(a, b)}{\partial a} = 0, \qquad \frac{\partial c(a, b)}{\partial b} = 0 \tag{5}$$

Po vypočítání parciálních derivací získáme soustavu dvou rovnic o dvou neznýmých a dopočítáme a,b. Tyto koeficienty budou reprezentovat nalezenou aproximaci přímky, kde funkce $c(a, b)$ je nejmenší.

**Příklad**

Zadání :

$$A[1,1], \quad B[3,2], \quad C[2,3].$$

Nyní tyto body dosadíme do (3) a získáme vzdálenost mezi zadaným bodem a aproximovanou přímkou.

$$1 = a + b, 2 = 3a + b, 3 = 2a + b,$$

Nyní tuto soustavu rovnic můžeme dosadit do vzorce (4):

$$c(a,b) = (a + b - 1)^2 + (3a + b - 2)^2 + (2a + b - 3)^2.$$

Nyní funkci parciálně derivujeme dle [5]

$$\begin{aligned} \frac{\partial c(a,b)}{\partial a} &= 2(a + b - 1) + 2(3a + b - 2)3 + 2(2a + b - 3) &&= 28a + 12b - 26 \\ \frac{\partial c(a,b)}{\partial b} &= 2(a + b - 1) + 2(3a + b - 2) + 2(2a + b - 3) &&= 12a + 6b - 12 \end{aligned}$$

Nyní vyřešíme soustavu rovnic a získáme koeficienty aproximované přímky:

$$\begin{gathered} 28a + 12b - 26 = 0 \\ 12a + 6b - 12 = 02a + 1 = 0 \\ a = -\tfrac{1}{2} \qquad b = 3 \end{gathered}$$

Našli jsme aproximaci:

$$f : y = -\frac{1}{2}x + 3.$$

Pokud provedeme derivaci této přímky, dostaneme $-\frac{1}{2}$. Tato hodnota je taky nazývána jako směrnice přímky a je rovna tangentě mezi přímkou a mezi kladnou poloosou x neboli nám říká, jaký je úhel mezi přímkou a poloosou x. V příloze nalezneme obrázek k tomuto příkladu.

- Bezierova – pomocí algoritmu Casteljau (změna parametru $t$) , Kubický Coonsův B–spline, NURBS

### 3.1.5 Tečný a normálový vektor křivky

**Tečna a tečný vektor**

- Parametricky vyjádřená přímka $P(t) = (x(t), y(t), z(t))$
- **Tečný vektor** v bodě $t = t_0$ je dán jako $P'(t_0) = \frac{dP}{dt}(t_0) = (x'(t_0), y'(t_0), z'(t_0))$
- **Tečna** (přímka, která se v daném bodě křivky dotýká) je dána bodem dotyku a tečným vektorem. $Q(u) = P(t_0) + uP'(t_0)$, kde $u\varepsilon\mathbb{R}$
- Tečna je limitní polohou sečny, kdy oba průsečíky splynou v jeden.
- **Normálový vektor** je **kolmý** na tečný vektor. Skalární součin je tedy roven 0. Potřebujeme ho pro **obecnou rovnici** přímky/roviny. Získáme jej **prohozením souřadnic směrového vektoru** a u jedné souřadnice změníme znaménko.

## 3.2 Křivost křivky

Křivost křivky je jedna ze **základních vlastností**, které charakterizují křivky. Rozlišujeme dva typy křivostí:

- **první křivost (flexe)** – obvykle označována pojmem „křivost“ a udává velikost odchýlení od křivky $P(t)$ v daném bodě $P(t_0)$. V inflexních bodech je křivost nulová.

$$k_1(t) = \frac{|P'(t) \times P''(t)|}{|P'(t)|^3}$$

- **druhá křivost (torze)** -– je mírou odchýlení křivky $P(t)$ v daném bodě $P(t_0)$ z její oskulační roviny (viz. oskulační rovina) do prostoru.

$$k_2(t) = \frac{(P'(t) \times P''(t)) \cdot P'''(t)}{|P'(t) \times P''(t)|^2}$$

### 3.2.1 Oskulační rovina

Každá rovina procházející tečnou křivky v bodě $P(t0)$ se nazývá **tečná rovina**. Oskulační rovina je **limitní rovinou** těchto rovin. Pomocí bodu $P(t0)$ a dvou vektorů $P'(t0)$ a$P''(t0)$ zjistíme oskulační rovinu. Jsou–li tyto dva vektory lineárně nezávislé, existuje právě jedna oskulační rovina v daném bodě. V opačném případě je oskulační rovinou každá tečná rovina.

## 3.3 Plochy

- **Rozšířením křivek** se dostaneme k plochám, které mají však s křivkami hodně společného. Zejména některé vznikly rozšířením křivek (Bezier, NURBS).

- Z parametrického vyjádření je snadné získat jednotlivé body, z implicitního můžeme jednoduše testovat, zda bod patří do plochy nebo ne.
- **Nejvyužívanější** plochy jsou **parametrické**.
- Plochy mohou být zadány **analytickým předpisem**, **hraničními křivkami**, **sítí bodů** (NURBS, Bezier) nebo plochy vytvořené **kinematicky** (rotační plochy, plochy vzniklé skládáním pohybu).

Vyjádření ploch analytickým předpisem:

- **explicitní** – $z = f(x, y)$,
- **implicitní** – $F(x, y, z) = 0$,
- **parametrické** –

$$x = f_x(u, v), y = f_y(u, v), z = f_z(u, v)..$$

Tečné vektory plochy:

$$\mathbf{t}_u(u,v) = \frac{\partial \mathbf{Q}(u,v)}{\partial u} = \left( \frac{\partial x(u,v)}{\partial u}, \frac{\partial y(u,v)}{\partial u}, \frac{\partial z(u,v)}{\partial u} \right),$$

$$\mathbf{t}_v(u,v) = \frac{\partial \mathbf{Q}(u,v)}{\partial v} = \left( \frac{\partial x(u,v)}{\partial v}, \frac{\partial y(u,v)}{\partial v}, \frac{\partial z(u,v)}{\partial v} \right).$$

Tečná rovina:

$$\nu(r,s) = \mathbf{Q}(u,v) + r\mathbf{t}_u(u,v) + s\mathbf{t}_v(u,v), \text{ kde } r,s \in \mathbb{R}.$$

Normála: Určíme jako vektorový součin tečných vektorů. Jednotkový normálový vektor $= n/|n|$

### 3.3.1 Křivost plochy

- **Normálová křivost** křivky – určuje se v regulárním bodě plochy pro konkrétní tečnu a křivku procházející tímto bodem $k_n = k_1(n_k \cdot n) = k_1 \cos\gamma$
  $\mathbf{k_1}$ – první křivost křivky, $\mathbf{n_k}$ – hlavní normála křivky, $\mathbf{n}$ – normála plochy, $\gamma$ – úhel mezi $\mathbf{n}$ a $\mathbf{n_k}$.
- **Gaussova křivost plochy** – $k_G = k_{n,min} \cdot k_{n,max}$, min. normálová křivost($\mathbf{k_{n,min}}$), a max. normálová křivost($\mathbf{k_{n,max}}$).
- **Střední křivost plochy** – $k_H = \frac{k_{n,min} \cdot k_{n,max}}{2}$.
- **Absolutní křivost plochy** – $k_abs = |k_{n,min}| + |k_{n,max}|$.

## 3.4 Cn a Gn Spojitost

- Při navazování oblouků je významným faktorem spojitost křivek.
- Výsledná křivka je **spojitá**, pokud je spojitá **ve všech svých bodech**, a tedy zejména v navazovacích bodech.
- Křivka je **hladká**, pokud jsou ve všech jejích bodech **spojité i její první derivace**. Pro vyšší derivace říkáme, že křivka má spojitost druhého, třetího a obecně $n$–tého řádu.
- Význam spojitosti křivek:
  - vizuální stránka napojení dvou křivek,
  - animace křivky.

$\mathbf{C_n}$ – parametrická spojitost:

- $\mathbf{c_0}$ – koncový bod prvního segmentu je počátečním bodem segmentu druhého,
- $\mathbf{c_1}$ – rovnost tečných vektorů v daném uzlu,

- $\mathbf{c_2}$ – rovnost prvních derivací tečných vektorů v daném uzlu.

Čím vyšší spojitost je požadována, tím delší "dobu"(ve smyslu parametru $\mathbf{t}$) se oba segmenty k sobě přimykají. Ze spojitosti $\mathbf{c_0}$ plyne, že bod se pohybuje po spojité dráze, ale v uzlu může měnit skokem směr pohybu, rychlost i zrychlení. Směr pohybu a velikost rychlosti se nemůže měnit skokem při spojitosti $\mathbf{c_1}$ a zrychlení zůstává nezměněné při spojitosti $\mathbf{c_2}$.

$\mathbf{G_n}$ – geometrická spojitost:

- $\mathbf{g_0}$ – koncový bod prvního segmentu je počátečním bodem segmentu druhého
- $\mathbf{g_1}$ – tečné vektory jsou lineárně závislé

Opticky zaručuje $\mathbf{g_1}$ spojitost "skoro stejnou"hladkost jako $\mathbf{c_1}$. Z hlediska použití bývá jednodušší zaručit spojitost $\mathbf{g_1}$ nežli $\mathbf{c_1}$.

### 3.5 Bézier

- Pierre Étienne Bézier (1910 - 1999) Renault
- mějme zadání $n+1$ řídících bodů $P_0, P_1, ..., P_n$, kde $n \geq 1$.
- Bézierova křivka je zadána jako

$$P(t) = \sum_{k=0}^{n} P_i B_i^n(t),$$

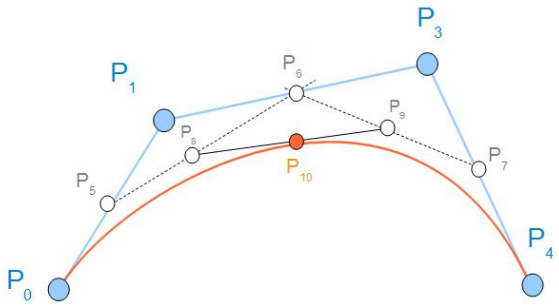
  kde: $t\varepsilon < 0, 1 >$, $P_i$ je počet bodů a $B_i^n(t)$ reprezentuje Bersteinovy polynomy.
- Pro výpočet bázových funkcí se využívá Bersteinových polynomů:

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

  kde: $t\varepsilon < 0, 1 >$, $n$ je polynomiální stupeň (počet bodů), $i$ je index a $t$ parametr.
- Jednoduchá Bézierová křivka je přímka z bodu $P_0$ do bodu $P_1$.
- Kvadratická Bézierová křivka je definována třemi kontrolními body.
- Kubická Bézierová křivka je definována čtyřmi kontrolními body $P_0$, $P_1$, $P_2$ a $P_3$. Body $P_0$, $P_2$ a tyto tečné vektory dosadíme do vzorce:

$$P(t) = P_0 \binom{3}{0} t^0 (t-1)^3 + P_1 \binom{3}{1} t^1 (t-1)^2 + P_2 \binom{3}{2} t^2 (t-1)^1 + P_3 \binom{3}{3} t^3 (t-1)^0$$

### Konstrukce Bézierovy křivky




Pro geometrickou konstrukci Beziérovy křivky zvolíme poměr t v kterém dělíme lomenou řídící čáru, jak je vidět na obrázku nahoře, kde je $t = 0,5$.

Takto jsme vykreslili první bod křivky $\mathbf{P_10}$. Konstrukcí bodu $\mathbf{P_10}$ jsme získali nové řídící body, které použijeme pro získání dalších bodů křivky. Další dva body křivky tedy získáme stejným způsobem za použití řídících bodů $\mathbf{P_0}$, $\mathbf{P_5}$, $\mathbf{P_8}$, $\mathbf{P_10}$ a $\mathbf{P_10}$, $\mathbf{P_9}$, $\mathbf{P_7}$, $\mathbf{P_4}$. Tímto rekurzivním způsobem postupně vykreslíme celou křivku.

Výhoda této konstrukce je, že můžeme ovlivnit hustotu vykreslování dle potřeby. Například v oblasti velkého zakřivení.

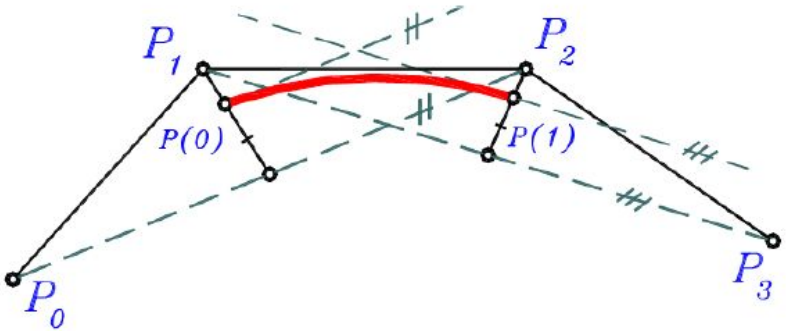
### Převod Fergusonovy kubiky na Beziérovu kubika

Fergusonovu křivku lze převést na Beziérovu křivku, pokud se budeme při výpočtu držet následujícího pravidla:

$$\begin{aligned} P_0 &= V_0, \\ P_1 &= V_0 + \tfrac{1}{3}\vec{u}, \\ P_2 &= V_1 - \tfrac{1}{3}\vec{v}, \\ P_3 &= V_1. \end{aligned}$$

## 3.6 Coons

- Coonsnová kubická B-spline křivka vznikne pospojováním Connsnových kubik, tak aby byla zajištěna spojitost druhého řádu.
- Coonsnová kubika je parametrická křivka dána čtyřmi body $P_0, P_1, P_2, P_3$ a tímto vztahem:
$$P(t) = \frac{1}{6}(P_0C_0(t) + P_1C_1(t) + P_2C_2(t) + P_3C_3(t))$$
kde bázové funkce jsou:
$$\begin{aligned} C_0(t) &= -t^3 + 3t^2 - 3t + 1, \\ C_1(t) &= 3t^3 + 6t^2 + 4, \\ C_2(t) &= -3t^3 + 3t^2 + 3t + 1, \\ C_3(t) &= t^3 \end{aligned}$$




### 3.7 NURBS

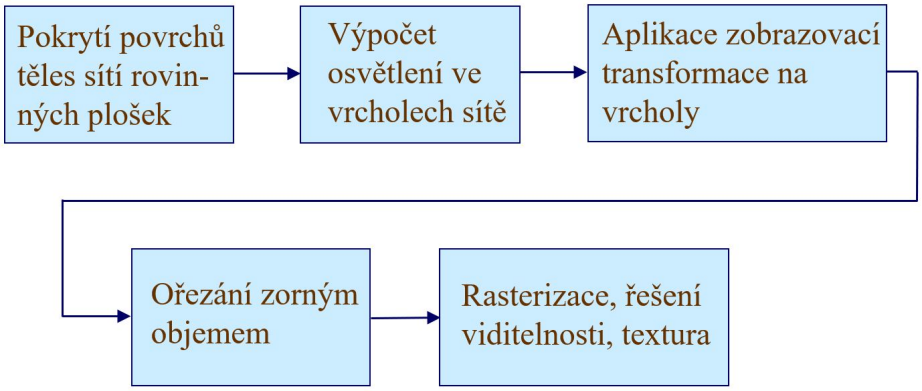
- Non-uniform rational basis spline (NURBS)

# 4 Geometrické a objemové modelování. Hraniční metoda, metoda CSG, výčet prostoru, oktantové stromy.

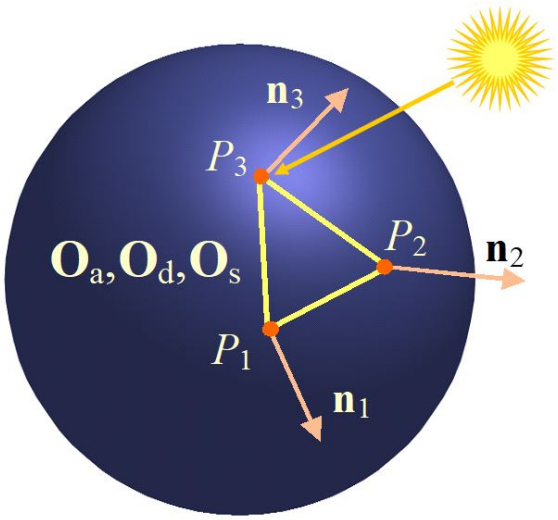
# 5 Standardní zobrazovací řetězec a realizace jeho jednotlivých kroků. Gouraudovo a Phongovo stínování. Řešení viditelnosti. Grafický standard OpenGL: stručná charakteristika.

## 5.1 Standardní zobrazovací řetěz

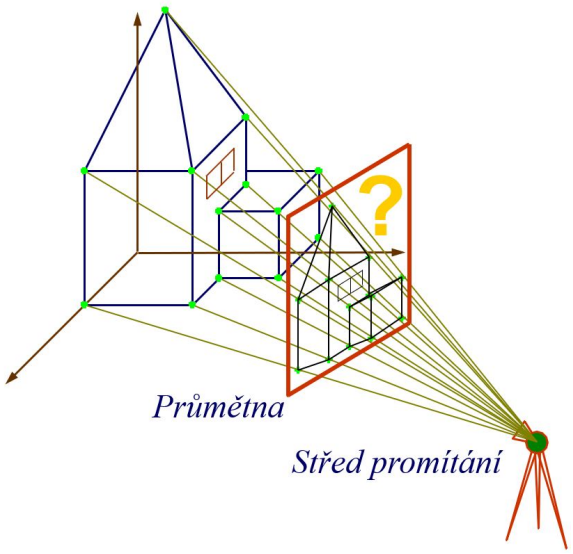
- Klade důraz na rychlost nikoli na kvalitu.
- Realizuje ho OpenGL.



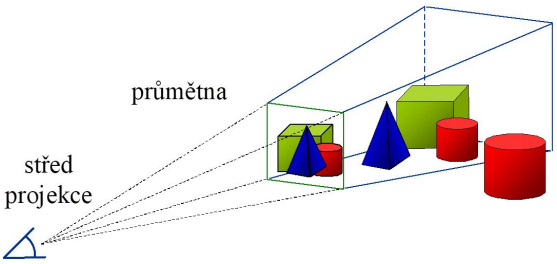

- **Pokrytí povrchu objektů sítí rovinných plošek:**
  - Ploškami bývají nejčastěji trojúhelníky nebo čtyřúhelníky.
  - Pro objekty ve tvaru mnohostěnu je takové dělní vcelku samozřejmé.
  - K přesnějšímu výpočtu barev bývá, ale někdy dělení na plošky jemnější.
  - Někdy síť rovinných plošek žádaný povrch pouze aproximuje.
- **Výpočet osvětlení ve vrcholech sítě**
- k tomu známe:
  - Polohu, intenzitu a barvu světelných zdrojů.
  - Souřadnice vrcholů ($P$), normál ($n$) a konstanty popisující optické vlastnosti materiálu ($O_a, O_d, O_s$)



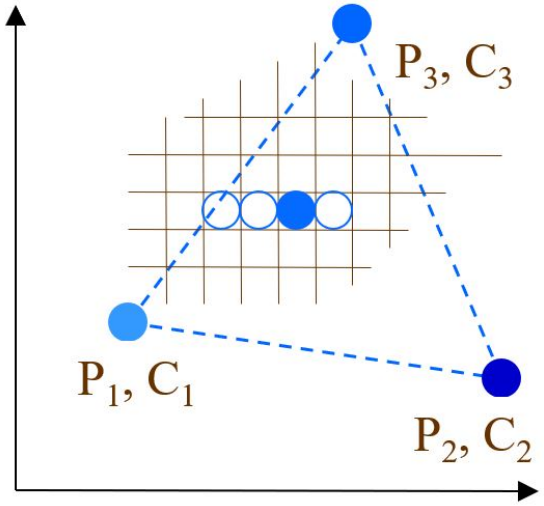

- **Aplikace zobrazovací transformace na vrcholy**
  - Oblíbenou technikou je středové promítání. To je zadáno:
    - Polohou průmětny.
    - Polohou středu promítání.



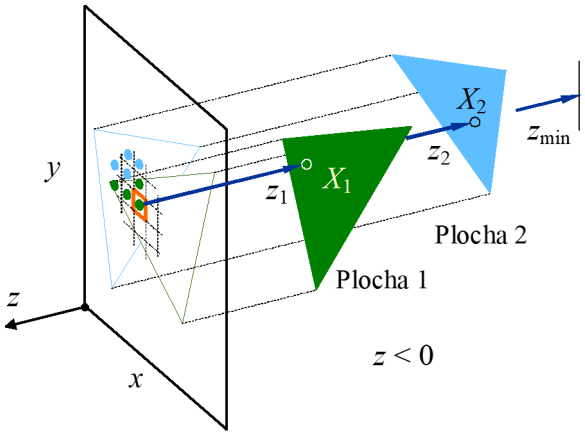

- **Ořezání zorným objemem**
  - Objekty nebo jejich části, nacházející se mimo zorný objem (obvykle jehlan) jsou odstraněny.




- **Rasterizace plošek**
  - Postupně zpracovávány všechny plošky.
  - Pro každou plošku rozsvěceny všechny její pixely.
  - Barva každého pixelu se stanoví interpolací mezi hodnotami ve vrcholech.




- **Řešení viditelnosti (z–buffer)**
  - Pro rozhodnutí viditelnosti se použijí hodnoty souřadnice $z$ (zde je $z_1 > z_2$).
  - Před řešením viditelnosti bývá centrálním promítání převedeno na rovnoběžné.



- **Nanášení textury**
  - Vzhled obrázků lze vylepšit nánášením textury.

## 5.2 Stínování (shading)

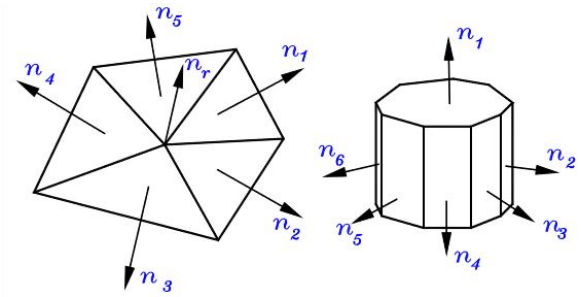
- Vykreslování barevných objektů různými odstíny barev.
- Lze odlišit křivosti ploch a tím docílit lepšího prostorového vjemu.
- Neplést s výpočtem vrženého stínu.
- Základní typy: Konstantní stínování, Gouraudovo stínování (Interpolace barvou), Phongovo stínování (Interpolace normálových vektorů)

## 5.3 Gouraudovo stínování (Interpolace barvou)

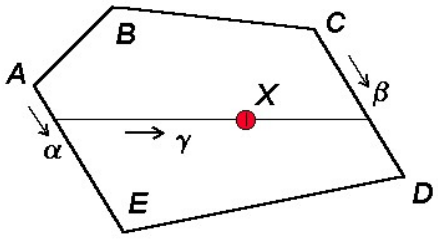
Princip metody spočívá v tom, že pokud budeme znát normálu v každém vrcholu každé plochy objektu, pak lze vypočítat barvu v tomto vrcholu a interpolací vypočítat barvu pixelu uvnitř plošky (bilineární interpolace).

Přesto ani tento způsob stínování neposkytuje zcela věrný obraz reálných objektů - interpolace samotného odstínu barvy totiž nemůže způsobit místní zvýšení jasu na plošce, stejně jako nemůže kvalitně vytvořit odlesky způsobené odraženým světlem. Dá se říci, že tato metoda zahlazuje barevné rozdíly u místních nerovností povrchu.

Normálový vektor $n_r$ vypočteme jako aritmetický průměr vektorů okolních plošek.




- Vypočteme normálové vektory pro všechny plošky ze kterých je objekt složený.
- Pro každý vrchol spočítáme normálový vektor v tomto vrcholu jako průměr normálových vektorů plošek, které se v tomto vrcholu stýkají.
- Z normálových vektorů ve vrcholech a pozice světelného zdroje vypočteme barvy ve vrcholech plošek.
- Provedeme interpolaci barvy pro body jednotlivých plošek.




$$\mathbf{f_X = (1\text{-}\gamma) \cdot [(1\text{-}\alpha) \cdot f_A + \alpha \cdot f_E] +}$$
$$\mathbf{+ \gamma \cdot [(1\text{-}\beta) \cdot f_C + \beta \cdot f_D]}$$

**Výhody**

+ umožnuje dobře zobrazit i hladké objekty,
+ používá se jako nejčastější metoda stínování.

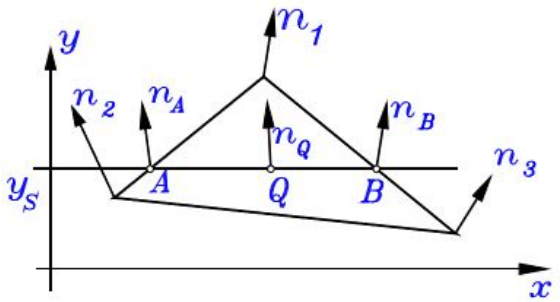
**Nevýhody**

– nevnikají ostré odlesky uprostřed polygonů.

## 5.4 Phongovo stínování (Interpolace normálových vektorů)

- Interpolaci provádíme po řádcích.
- Touto metodou se odstraní problém neostrých odlesků.
- Je ale bohužel náročná na výpočet.
- Pro normálové vektory lze psát:

$$n_A = n_1 + (n_2 - n_1) \cdot u; u < 0, 1 >,$$
$$n_B = n_1 + (n_3 - n_1) \cdot w; w < 0, 1 >,$$
$$n_Q = n_A + (n_B - n_A) \cdot t; t < 0, 1 >,$$

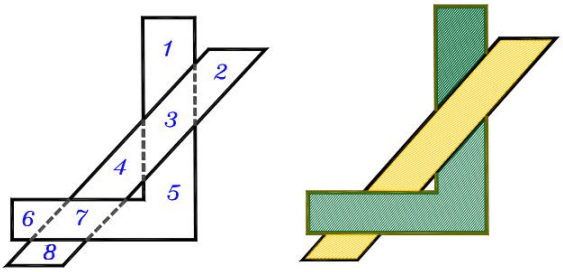


## 5.5 Řešení viditelnosti

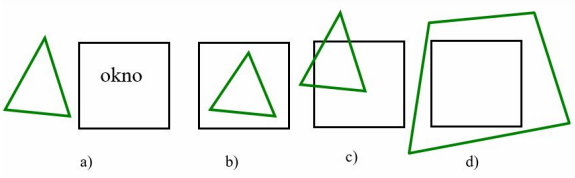
- Podle výsledných dat
  - **Vektorové algoritmy** – geometrické prvky vrcholy, hrany a stěny. Výstupem je vektorové řešení.
  - **Rastrové algoritmy** – výsledkem je rastrový obraz (jednotlivé pixely obsahují barvu), většina současných metod.
- Podle místa řešení
  - **Řešení v prostoru objektů** – proovnávání vzájemné polohy těles $0(n^2)$
  - **Řešení v prostoru obrazu** – pracujeme s promítnutými a rasterizovanými objekty. Pro pixely hledáme nejbližší objekty.

Rastrové algoritmy:

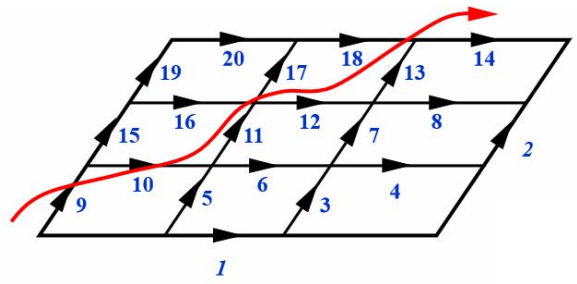
- **Malířův algoritmus** (Painter's algorithm) – porovnává plochy z hlediska jejich $z$–tových souřadnic (plocha s menší $z$–tovou souřadnící bude kreslena první), jestliže se plochy **nepřekrývají**, potom na pořadí kresby nezáleží, pokud se protínají – rozdělit na nepřekrývající se plochy.




- **Dělení obrazovky** (Warnock subdivision)
  1. Všechny plošky leží mimo zónu - zůstane barva pozadí. (a)
  2. Oblast obsahuje právě jeden celý n-úhelník. Daná oblast se vyplní barvou a zbytek -pozadím. (b)
  3. Oblast protíná právě jeden n-úhelník. Daná část se vyplní barvou, zbytek pozadí. (c)
  4. Pokud zobrazovaná část je celá uvnitř jednoho n-úhelníka, potom se celá oblast zobrazí barvou nejbližšího n-úhelníka, který oblast obklopuje. (d)
  5. Pokud nenastane jeden z vyjmenovaných případů - oblast se rozdělí.

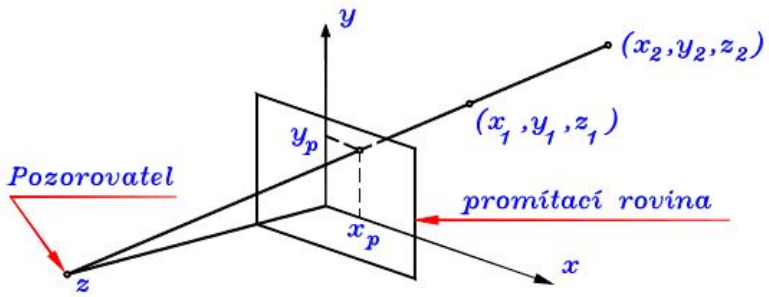


- **Plovoucí horizont** (Floating Horizon Algorithm) – metoda „zig–zag“, počítáme od „nejbližšího“ rohu plochy k „oku“ pozorovatele.




- **Paměť hloubky** (**Z–buffer**, depth–buffer)

  1. Vyplň obrazovou paměť barvou pozadí.
  2. Vyplň paměť hloubky – nekonečnem
  3. Pro každou plochu najdi její průměť (rasterizaci) nalezenému pixelu $[x_i, y_i]$ přiřaď hloubku $z_i$
  4. Porovnej hloubku a zapiš do paměti




  – Nejznámější a nejefektivnější metoda.
  – Každá plocha se zpracovává pouze jednou.
  – Doba zpracování roste s počtem ploch lineárně (záleží i na velikosti ploch).
  – Není potřeba žádné třídění nebo pomocné datové struktury.
  – Možnost paralelních procesů.

- Z–buffer – paměť hlouby – průhlednost - princip

  1. Inicializuj color buffer a depth buffer.
  2. Postupně načti všechny plochy, neprůhledné zpracuj, průhledné si zapamatuj a odlož pro následné zpracování.
  3. Po zpracování neprůhledných ploch setřiď průhledné plochy podle vzdálenosti.
  4. Zpracuj průhledné plochy s použitím alfa míchání.

## 5.6 Grafický standard OpenGL: stručná charakteristika

OpenGL (Open Graphics Library) je grafická multiplatformní knihovna pro **tvorbu** a **zobrazování 2D a 3D objektů** vyvinutá firmou SGI (Silicon Graphics Inc.) v 90.letech. Dnes jde o všeobecně uznávaný **standard** podporován výrobci grafických karet. Standard OpenGL definuje množinu funkcí, které se volají z programu. Pokud nejsou některé z těchto funkcí podporovány na technické úrovni, je podpora realizována programově, což zajišťuje široké využití i při zachování techniceké nezávislosti programu.

Používá se pro tvorbu **PC her**, **CAD programů**, aplikací **virtuální reality** či **vědeckotechnické vizualizace** apod.

OpenGL je jednoduchý, **nepodporuje objektově orientované programování**. Přesto nabízí široké možnosti a urychluje práci s grafikou. Kromě vykreslování základních typů objektů, umožňuje OpenGL transformace. A to **transformace zobrazovací** a **transformace modelovací**. Při těchto transformací se pracuje s transformačními maticemi.

Na některých platformách je možné rozdělení aplikace na dvě relativně samostatné části – **serverovou** a **klientskou**. Při vykreslování se potom jednotlivé příkazy (což jsou většinou parametry funkcí OpenGL) přenášejí přes síťové rozhraní. Knihovna OpenGL (narozdíl od IRIS GL nebo Direct 3D) byla vytvořena tak, aby byla **nezávislá** na použitém operačním systému, grafických ovladačích a správcích oken (Window Managers). Proto také neobsahuje žádné funkce pro práci s okny (otevírání, zrušení, změnu velikosti), pro vytváření grafického uživatelského rozhraní (Graphical User Interface – GUI) ani pro zpracování událostí

Pro dosažení co největší nezávislosti na použité platformě zavádí knihovna OpenGL vlastní primitivní datové typy, například **GLbyte**, **GLint** nebo **GLdouble**.

Programátorské rozhraní knihovny OpenGL je vytvořeno tak, aby knihovna byla použitelná v téměř libovolném programovacím jazyce. Primárně je k dispozici hlavičkový soubor pro jazyky C a C++. Existují však i podobné soubory s deklaracemi pro další programovací jazyky, například Fortran, Object Pascal či Javu; tyto soubory jsou většinou automaticky vytvářeny z Cčkovských hlavičkových souborů.

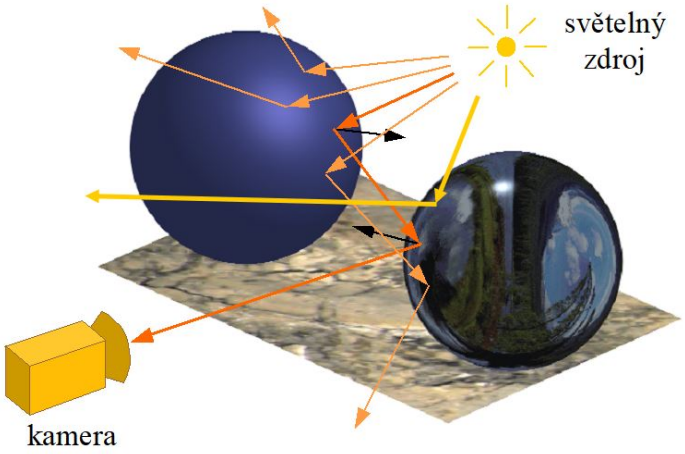
Z programátorského hlediska se OpenGL chová jako **stavový automat**. To znamená, že během zadávání příkazů pro vykreslování **lze průběžně měnit vlastnosti vykreslovaných primitiv** (barva, průhlednost) nebo celé scény (volba způsobu vykreslování, transformace) a toto nastavení zůstane zachováno do té doby, než ho explicitně změníme. Výhoda tohoto přístupu spočívá především v tom, že funkce pro vykreslování mají menší počet parametrů a že jedním příkazem lze globálně změnit způsob vykreslení celé scény, například volbu drátového zobrazení modelu (wireframe model) nebo zobrazení pomocí vyplněných polygonů (filled model). Vykreslování scény se provádí **procedurálně** – voláním funkcí OpenGL se vykreslí výsledný rastrový obrázek. Výsledkem volání těchto funkcí je rastrový obrázek uložený v tzv. framebufferu, kde je každému pixelu přiřazena barva, hloubka, alfa složka popř. i další atributy.

# 6 Metody získávání fotorealistických obrázků (rekurzivní sledování paprsku, vyzařovací metoda, renderovací rovnice).

- Syntetizace fotorealistických obrazů je oblastí PG, která dovoluje vykreslit jakoukoliv uměle vytvořenou scénu tak, jak by vypadala v reálném světě.
- Toho dosahuje díky implementaci optických zákonů, které lze běžně pozorovat.

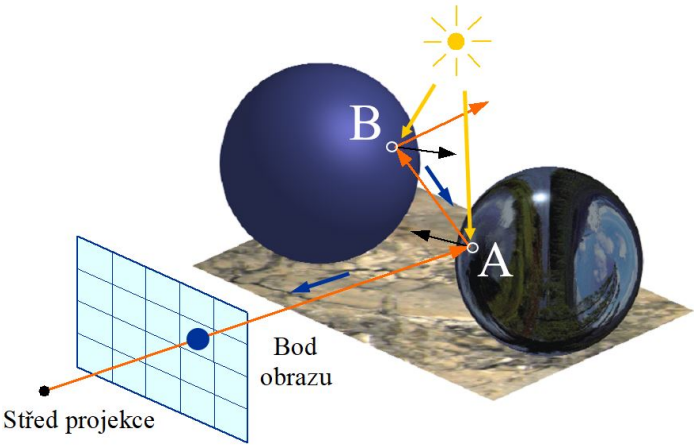
## 6.1 Sledování paprsku - ray tracing

- Metoda sleduje šíření paprsků ve scéně.
- Tyto paprsky začínají ve světelném zdroji, odráží se o tělesa v prostoru a některé z nich nakonec dopadnou do průmětny.
- Paprsky, které takto prochází scénu, lze znázornit jako strom.
- Tento přístup je však neefektivní, protože velká část paprsků do průmětny nikdy nedopadne, takže nemají přínos pro výsledný obraz a zbytečně zvyšují výpočetní čas.



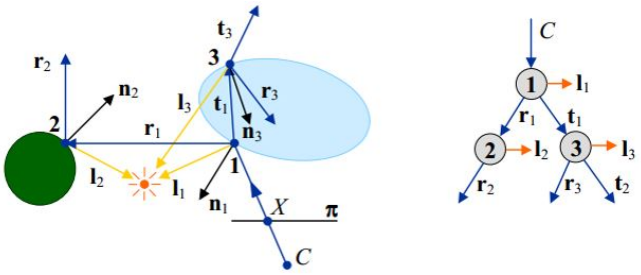

## 6.2 Zpětné sledování paprsku

- Funguje stejně jako běžné sledování paprsku, ovšem paprsky jsou vysílány z kamery do scény.
- Tím se eliminuje možnost, že by paprsek nepřinesl žádný prospěch výslednému obrazu.




## 6.3 Rekurzivní sledování paprsku

- Metoda vyšetřuje „běh“ světelných paprsků ve scéně.
- Světlo je reprezentovanáno paprsky, které jsou do scény vyzařovány světelnými zdroji a putují prostorem scény, některé dopadnou na povrchy těles, jiné odletí ze scény.
- Paprsek, který dopadne na povrch tělesa se může odrazit (zákon odrazu) nebo pokud je těleso průhledné, může se paprsek zlomit (zákon lomu) – oba druhy paprsků mohou opět dopadnout na povrch těles, kde se celý proces znovu opakuje.
- Do scény se vyšle velké množství paprsků, ale podstatné jsou ty, které projdou objektivem myšlené kamery, pokud na průmětnu dopadne dostatečný počet paprsků, vykreslí se obrázek.
- 
- Metoda je sice jasná a fyzikálně podložená, ale nepoužívá se, protože se obtížně realizuje. (Je potřeba vyslat velké množství paprsků, ale k objektivu kamery by jich dorazilo jen malé množství a ostatní by se sledovaly zbytečně).
- Řešením je otočit paprsky a vyslat je od kamery ke světelnému zdroji. Principiálně to pak funguje stejně.




- Rovnice výpočtu lokálního osvětlení: $I = I_l + k_r I_r + k_t I_t$
  $I_l = I_a O_a + \sum_i S_i f_{att_i} I_i (O_d \cos\varphi_i + O_s \cos^n \alpha_i)$. Hodnota Si představuje viditelnost i-tého zdroje světla v daném bodě.

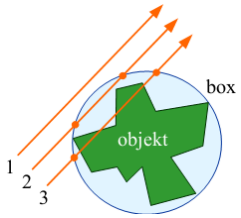
## 6.4 Urychlování trasování

Největším problémem je hledání průsečíků paprsků s objekty scény.

1. Nejjednodušším řešením je využít **ohraničujících ploch** („bounding boxů“). Ohraničující plocha se vytvoří kolem každého objektu ve scéně. Nejlépe když má plocha následující vlastnosti

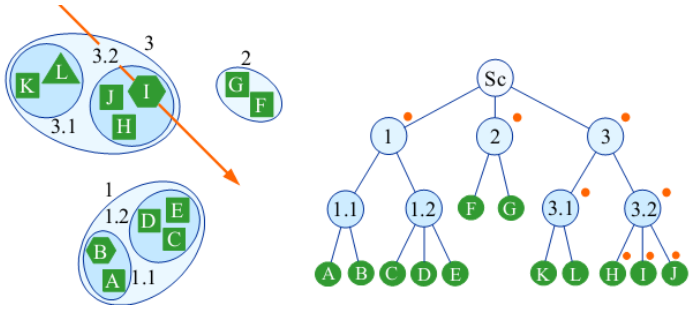
   - objekt leží **celý uvnitř** ohraničující plochy, ale plocha jej obepíná co **nejtěsněji**,
   - průsečíky paprsků s plochou musí jít spočítat, co nejjednodušším výpočtem,
   - plochu musí být možné pro jednotlivé objekty dostatečně jednoduše nalézt.

   Je možné použít **kulovou plochu** (ne moc vhodné, objekty můžou být protáhlé a tato plocha by je neobepínala dostatešně těsně). Další variantou plochy je **kvádr** (taky sice není moc vhodný, protože těleso může být našikmo a taky by jej neobepínal moc natěsno, nicméně nalezení ohraničující plochy je snadné – minimální a maximální hodnoty obepínaného tělesa).

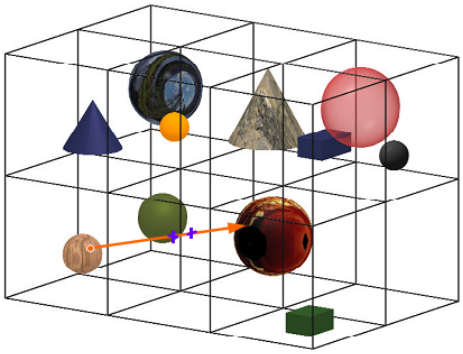
   Princip ohranučujících ploch spočívá v tom, že pokud paprsek neprotne ohraničující plochu, pak neprotne ani těleso uvnitř (velmi časté). Odhalením této situace dojde ke značnému zrychlení. Pokud je to naopak, hledají se průsečíky s ohraničeným tělesem.

2. Rozšířením předchozího je **organizování ohraničujících ploch do hierarchických struktur**. Princip je stejný, pokud se neprotne rodičovská plocha, nehledají se dále ani průsečíky s potomky. Nevýhodou je, že nelze jednoduše takovouto strukturu automatizovaně nalézt.

3. Další metodou je **Dělení prostoru scény na podprostory**. Obykle se dělí rovinami souřadné soustavy $xy, xz, yz \rightarrow$ vznikají tak velké kvádry (stejně velké / různě velké). Princip metody:

- u neurychlené metody byly všechny objekty organizovány v 1 velkém seznamu,
- nyní je zřízeno tolik seznamů, kolik je objemových elementů vzniklých dělením prostoru, každý element bude mít svůj seznam objektů, které do něj aspoň z části zasahují (pokud objekt zasehuje do více elemntů, bude v seznamu každého z nich) – hledání průsečíků začíná v tom elementu, kde je počátek paprsku, při opouštění elementu lze zjistit, do kterého elementu vstupuje, paprsek kontroluje pouze průsečíky s objekty, které jsou v seznamu daného elementu.



4. Dalším etodou je **Adaptivní hloubka rekurze**. Odhaduje se, zda je paprsek pro stanovení intenzity ve zkoumaném obrazovém bodě dostatečně užitečný. Pokud ne, tak se nevyšle. (např. u odrazů či průchodů tělesy).

## 6.5 Vyzařovací metoda - radiozita

Na rozdíl od předchozí metody rekurzivního sledování paprsku (dobře zobrazuje lesklé, dobře osvícený předměty) je tato metoda spíše protikladná. Zaměřuje se na difúzní odrazy světla – vhodná pro matné povrchy a rozptýlené světlo (např. interiéry). Princip:

- Vypočítá se, jak jsou osvětlena jednotlivá místa scény.
- Podle toho se povrchy těles pokryjí sítí (v místech kde je komplikovaný průběh osvětlení je síť hustá – zlom světla a stínu).
- Pro každou plošku jsou spočítany hodnoty RGB – vyzařování je konstantní na celém povrchu plošky.
  **PROBLÉM:** kdyby se takto plošky zobrazovaly, mohly by se sousedící plošky výrazně lišit intenzitou a nebylo by to pěkné.
  **ŘEŠENÍ:** po výpočtu intenzit plošek se intenzity přenesou do jednotlivých uzlů sítě (zprůměrování intezit okolních plošek, které obklopují uzel) a následně se intenzity interpolují. (proto i to hustší dělení, kde je přechod světlo–stín ...).
- Nejjednodušším, ale ne zrovna nejsprávnějším zobrazením scény a interpolací je pomocí Gouradova stínování.
  **PROBLÉM:** osvětlení bylo spočítáno v prostoru scény a tam by se měla provádět i interpolace, ale Gouraudovo stínování interpoluje v prostoru obrazu. Problém je, že

při středové projekci se nezachovává dělící poměr a proto budou výsledky v prostoru obrazu rozdílné od výsledků z prostoru scény. Nicméně Gouraudovo stínování se používá, protože je rychlé.
Vytváření sítě probíhá v několika iteracích: nejdřív se hustota odhadne, pak se spočítá osvětlení a dle výsledků se síť dohustí tam, kde je třeba.
Základní myšlenou je, že na všech ploškách ustanov energetická rovnováha :
**výkon vyzařovaný + výkon absorvovaný = výkon na plošku dopadající od jiných ploch + výkon, který ploška sama vyzařuje**

$$B_i = E_i + p_i \sum_{i=i}^{n} B_j F_{j \to i} \frac{A_j}{A_i}.$$

kde:

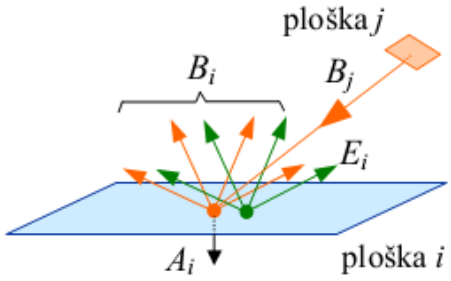
$B_j$ výkon vyzářený ploškou $j$,

$p_i$ míra odrazu (optické vlastnosti materiálu),

$E_i$ hodnota výkonu vlastního vyzařování plošky,

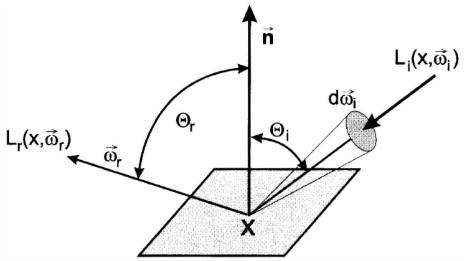
$A_i$,$A_j$ velikost plošek (plošný obsah),

$F_{j \to i}$ konfigurační koeficient říká, jaká část výkonu vyzařeného ploškou $j$ dopadne na plošku $i$ (jedná se o $int < 0, 1 >$, záleží na pořadí indexů)




## 6.6 BRDF (Bidirectional Reflectance Distribution Function)

- Charakterizuje **odrazové schopnosti povrchu materiálu** v určitém bodě **x**.
- Jedná se o **poměr odraženého zářezí** ke vstupnímu diferenciálnímu zařízení, promítnutému na kolmou plochu.
- BRDF v daném bodě zůstává stejná i když změníme směr paprsku.
- **Pozitivita BRDF**: funkce není nikdy záporná.
- **Zákon zachování energie**: plocha nemůže odrazit víc než je celková přijatá energie
- **Odrazivost** $p(x) = \frac{d\Phi_r(x)}{d\Phi_i(x)}; d\Phi_r(x)$ je odražený světelný tok, $d\Phi_i(x)$ je dopadající světelný tok.
- Obor hodnot odrazivosti je na intervalu $< 0, 1 >, 1 =$ plný odraz

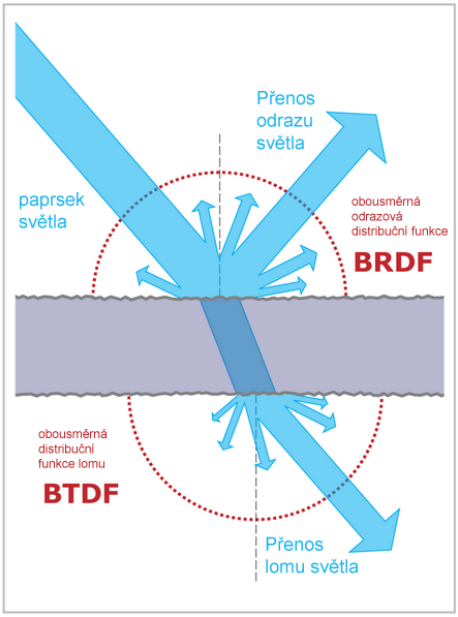



## 6.7 BTDF (Bidirectional Transmittance Distribution Function)

Dvousměrná distribuční fuknce lomu. Popisuje průchod světla povrchem.

## 6.8 BSDF (Bidirectional Scattering Distribution Function)

- Obousměrná distribuční funkce **rozptylu**.
- Je to souhrn dvou distribučních fuknců, a to funkce odrazu (BRDF) a lomu (BTDF).
- **BSDF + BTDF + BRDF**




## 6.9 Renderovací rovnice

Rekurzivní diferenciální rovnice

$$L(x,\omega_0) = L_e(x,\omega_0) + \int_\Omega L(r(x,\omega_i) - \omega_i) \cdot BRDF(\omega_i, x, \omega_0) \cos\theta_i d\omega_i$$

Zjednodušeně:
**osvětlení povrchu = samovolně vyzařované světlo + součet příchozího osvěrlení ze všech směrů krát BRDF**

# 7 Komprese obrazu a videa; principy úprav obrazu v prostorové a frekvenční doméně.

# 8 Základní metody úpravy a segmentace obrazu (filtrace, prahování, hrany).

# 9 Základní metody rozpoznávání objektů (příznakové rozpoznávání).

## 9.1 Histogram orientovaných gradientů HOG

Základní myšlenkou je, že objekt v obraze může být pomocí vzhledu a tvaru charakterizován pomocí intenzity gradientů, i přestože neznáme jejich přesnou polohu v obraze. Autoři jsou N. Dalal a B. Triggs (2005).

1. před započetím výpočtů je třeba normalizovat, například normalizaci barev a gamy, v případě černobílých obrázků k normalizaci kontrastu (tento krok může být přeskočen,dle Dalal a Triggs $\rightarrow$ předzpracování má malý vliv na výkon)
2. obraz se **rozdělí** na malé prostorové oblasti (buňky, například $8 \times 8$ pixelů)
3. pro každou buňku se vypočítá 1-D **histogram**, který je vypočítán ze všech pixelů z buňky (hodnoty buněk jsou rovnoměrně rozloženy do histogramu o 9 kanálech (binech) po 20°; rozsah 0°–180°) $\rightarrow$ výjde nám vektor o velikosti 9
4. buňky spojíme do větších **propojených bloků** $16 \times 16$ z důvodu normalizace osvětlení a kontrastu. Pro chodce se používá L2–norm normalizace, dle vztahu (6) $\rightarrow$ vznikne vektor velikosti $9 \times 4 = 36$ (čtyři $8 \times 8$ bloky)
5. vektor normalizovaných histogramů pro jeden blok nazýváme **deskriptor**.
6. spojíme tyto normalizované vektory do jednoho a získáme „trénovací“ vektor příznaků (features)

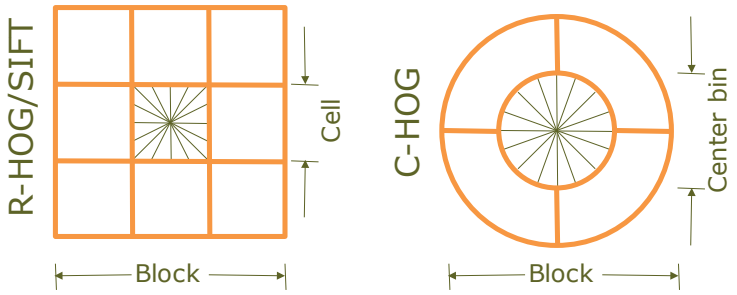
Existují dvě varianty spojení bloků, tzv. obdélníkové bloky (R–HOG) a kruhové bloky (C–HOG). Rozdělení do rozsahu 0–180° proto, že se jedná o bezznaménkové gradienty (unsigned) a bylo dokázáno, že fungují lépe než znaménkové (signed) 0–360°. Některé implementace HOG umožní určit, zda chceme používat signed gradienty.

$$\begin{aligned} L2-norm: \qquad f &= \frac{v}{\sqrt{\|v\|_2^2+e^2}} \\ L1-sqrt: \qquad f &= \sqrt{\frac{v}{\|v\|_1+e}} \end{aligned} \tag{6}$$

Nechť $v$ je nenormalizovaný vektor obsahující všechny histogramy v daném bloku, $\|v\|_k$ je jeho k–norm pro $k = 1, 2$, a $e$ je malá konstanta.

**C–HOG** (Kruhové HOG bloky) - lze nalézt ve dvou variantách: *s jedinou, centrální buňkou* a *úhlově rozdělenou centrální buňkou*. Dají se popsat čtyřmi parametry: počtem úhlů a radiálních kanálů (binů), poloměrem centrálního binu a faktorem roztažení pro poloměr dalších radiálních binů.

**R–HOG** (Obdélníkové HOG bloky) - tyto bloky jsou v praxi nejčastěji používané a reprezentují se třemi parametry: *počet buněk na blok*, *počet pixelů na buňku* a *počet binů (kanálů) na jeden histogram*. R–HOG bloky se také používají pro kódování informací.




Obrázek 2: Varianty geometrie spojení bloků

## 9.2 SIFT/SURF - detektory a popisovače klíčových bodů

Využívá se stejný princip tvoření histogramu jako u metody HOG

- klíčové body jsou nezávislé na osvětlení, velikosti, orientaci, pozici
- Octave - úroveň skálování
  - 4 oktávy a 5 rozmazání "ideál" dle prezentace, každá oktáva se 5x rozmaže
  - vypočítá se rozdíl mezi rozmazanými obrázky
  - klíčový bod najdeme jako minimum/maximum mezi ruznými urovněmi rozmazání
- poté musíme provést eliminaci slabých bodů
  - odebereme ty s malou intenzitou
  - odstraníme klíčové body, které leží na hraně - využit princip Harrisova detektoru hran - Hessian matice (matice druhých derivací)
- orientace bodu se vypočítá pomocí histogramu směrů gradientů v okolních bodech, zajišťuje nám to invarianci vůči rotaci (36 košů)
- Descriptor
  - 16x16 matice okolo klíčových bodů
  - rozdělit na 4x4 subbloky (16 bloků v 16x16 okolí)
    - v nich vypočítat histogram orientací gradientu
    - poté tento histogram převést do vektoru (viz HOG)
    - spojit pro všechny bloky a máme výsledný vektor
- SURF má stejné kroky jako SIFT, jen má jiné „implementace" kroků

## 9.3 Haarovy příznaky

- Na tomto přístupu je založen objektový detektor **Viola–Jones** (*Viola–Jones object detector framework*).
- Poskytuje v reálném čase **spolehlivou** a **konkurenceschopnou** detekci objektů.

- Může být vytrénován pro detekci různých objektových tříd (primárně určen pro **detekci obličejů**).
- Detektor pracuje s obrazy ve stupních šedi a skládá se ze tří částí. (Integrální obraz, Haar příznaků a AdaBoost algoritmus)

Integrální obraz je takový obraz (obrázek 5), kde každý bod $x$ představuje součet hodnot předchozích pixelů doleva a nahoru. Spodní pravý bod obsahuje součet všech pixelů v obraze. Zápis integrálního obrazu je:

$$I(x,y) = \sum_{\substack{x' \leq x \\ y' \leq y}} i(x',y'),$$

kde $i(x',y')$ je hodnota pixelu na pozici $(x,y)$.

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

Obrázek 3: Vstupní obraz

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 2 | 4 | 6 |
| 3 | 6 | 9 |

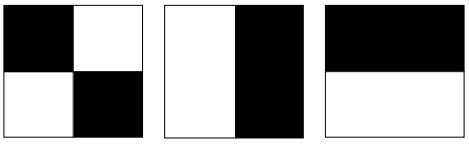
Obrázek 4: Integrální obraz

Obrázek 5: Převod obrazu na integrální obraz

Princip využití Haarových příznaků v obrazech je založen na pozorování, že lidská těla a obličeje mají některé podobné rysy. Právě tyto rysy mohou být porovnány pomocí Haarových příznaků. Jedná se například o tyto rysy:

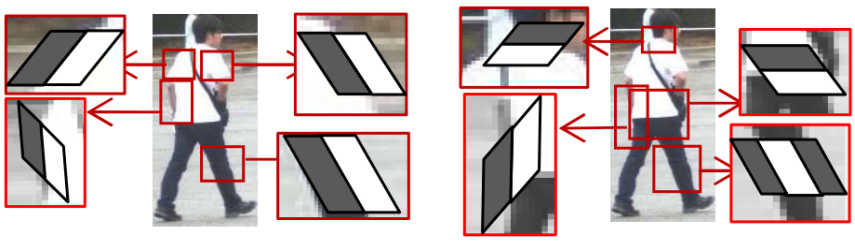
- Oční oblast je tmavší než oblast nosního mostu,
- hlava člověka je tmavší než její okolí,
- oblast mezi dolními končetinami je světlejší než samotné nohy.

Sada Haarových vlnek je na obrázku 6, jedná se pouze o základní sadu příznaků.



Obrázek 6: Základní sada Haarových příznaků

Pro identifikaci lidských postav se používá rozšířená sada vlnek, tzv. Haar–like příznaky. Klasifikační systém založený na těchto Haar–like příznacích dosahuje nižší falešně pozitivní detekce než původní Haar příznaky. Na obrázku 7 je příklad detekce pomocí Haar–like vlnek. Hodnota příznaku je rozdíl mezi sumou hodnot pixelů v bílé a černé oblasti Haarových vlnek.



Obrázek 7: Použití Haar–like příznaků na chodcích

## 9.4 LBP Lokální binární vzor

Hlavní myšlenkou LBP je, že struktury obrazu mohou být efektivně zakódovány porovnáním hodnot jednotlivých pixelů a jejich okolí. Tato metoda je odolná vůči jasovým změnám obrazu.

1. převod obrazu do stupňů šedi a jeho rozdělení do buněk
2. okolní hodnoty pixelů jsou porovnávány se středovým pixelem, pokud je jejich hodnota rovna nebo větší zapíše se na tuto pozici jednička v opačném případě nula
3. tyto hodnoty seřadíme dle hodinových ručiček nebo naopak a získáme osmimístné binární číslo a převedeme do dekadické soustavy
4. z čísel, které jsme získali kombinací pixelů v buňkách, vypočítáme histogram
5. zřetězíme všechny histogramy buněk a získáme vektor příznaků pro celý obraz (jedná se o 256–dimenzionální vektor příznaků)

Matematicky lze LPB vyjádřit jako:

$$LBP_{P,R} = \sum_{p=0} P - 1 s(g_p - g_c) 2^P, s(x) = \begin{cases} 1 & \text{pro } x \geq 0, \\ 0 & \text{pro } x < 0, \end{cases}$$

kde: $P$ je počet bodů v okolí, $R$ vyjadřuje vzdálenost bodů od středového pixelu, $g_c$ je středový pixel, $g_p$ je aktuální pixel.

Následující příklad se vztahuje k obrázku 11. Po porovnání pixelů se středovým pixelem jsme získali vzor 11110001. Tento vzor převedeme do dekadické soustavy a sečteme, $1 + 16 + 32 + 64 + 128 = 241$. Získali jsme hodnotu této buňky do vektoru příznaků.

| | | |
|---|---|---|
| 6 | 2 | 2 |
| 7 | 6 | 1 |
| 9 | 8 | 7 |

Obrázek 8: *
Vstupní buňka

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 1 | | 0 |
| 1 | 1 | 1 |

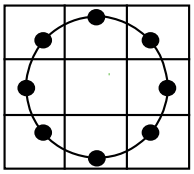
Obrázek 9: *
Prahové hodnoty

| | | |
|---|---|---|
| 1 | 2 | 4 |
| 128 | | 8 |
| 64 | 32 | 16 |

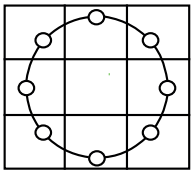
Obrázek 10: *
Pixely ohodnoceny váhou

Obrázek 11: Výpočet příznaku

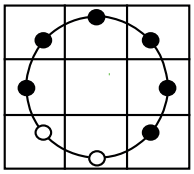
Výhoda této metody je její rychlý a snadný výpočet a odolnost vůči různým osvětlením. Na druhou stranu je těžší na trénování, protože výsledné dekadické číslo může mít obrovské množství možností (podle parametru $P$). K omezení lze využít uniformní vzory (obrázek 17). Pro parametr $P = 8$, získáme 59 vzorů.



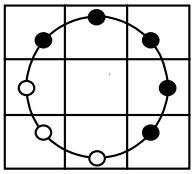
Obrázek 12: *
Bod



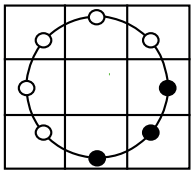
Obrázek 13: *
Bod/Plocha



Obrázek 14: *
Křivka



Obrázek 15: *
Roh



Obrázek 16: *
Hrana

Obrázek 17: Lokální okolí LBP metody

## 9.5 Klasifikátory

Klasifikace je obecný proces kategorizující objekty do určitých tříd. Termín klasifikátor někdy odkazuje také na matematickou funkci, implementovanou klasifikačním algoritmem.

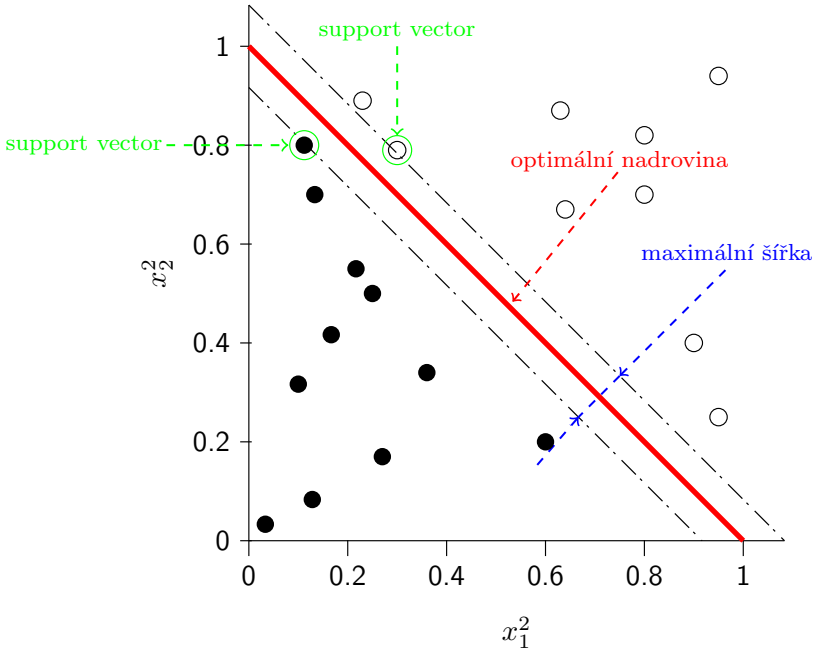
### SVM Support vector machines

- První algoritmus přisuzován Vladimíru Vapnikovi (1963)
- Učební modely, které jsou velmi populární v oblasti strojového učení.
- Založena na tzv. **jádrových algoritmech** (kernel machines) s využitím **podpůrných vektorů** (support vectors).
- Původně tato technika sloužila k vytvoření optimálního binárního klasifikátoru, později byla rozšířena na řešení **problému regrese** a **shlukování**.
- Byly úspěšně použity ve třech hlavních oblastech: kategorizace textu, rozpoznání obrazu a bioinformatika (např. třídění novinových zpráv, rozpoznávání ručně psaných čísel nebo například vzorky rakovinových tkání).

Primárním cílem SVM je nalézt **nadrovinu**, která **optimálně rozděluje prostor** příznaků tak, aby trénovací data náležela do konkrétních tříd. Tuto nadrovinu ilustruje obrázek

18. Pokud mezera mezi oddělující nadrovinou a nejbližšími vektory příznaků z obou kategorií (v případě binárního klasifikátoru) je maximální, jedná se o optimální řešení. Vektory příznaků v blízkosti této nadroviny se nazývají podpůrné vektory, což znamená, že pozice ostatních vektorů nemá vliv na nadrovinu (rozhodovací funkce).

Jinými slovy, se jedná o diskriminační klasifikátor formálně definovaný rozdělovací nadrovinou, která kategorizuje nové příklady.




Obrázek 18: Optimální oddělovací hranice

Implementaci SVM lze nalézt v již existujících knihovnách, jako jsou například LIBSVM, kernlab, scikit-learn, SVMLight..

- $C$–**Support vektorová klasifikace ($C$–SVC)** – Umožnuje nedokonalé oddělení tříd pro $n$–tříd ($n > 2$) s postihovým multiplikátorem $C$, pro odlehlé hodnoty ($C > 0$).
- $\nu$–**Support vektorová klasifikace ($\nu$–SVC)** – $n$–třídní klasifikace s možností nedokonalé separace. Tato klasifikace přidává nový parametr $\nu \in (0; 1\rangle$, čím větší je jeho hodnota, tím hladší je rozhodovací funkce.
- **Distribuční odhad (Jednotřídní SVM)** – Distribution Estimation (One-class SVM), jak již název sám o sobě napovídá všechny trénovací data pocházejí z jedné třídy, SVM vytvoří hranici, která odděluje třídu od zbývající části.
- $\varepsilon$–**Support vektorová regrese ($\varepsilon$–SVR)** – Vzdálenost mezi vektory příznaků a rozdělovací nadrovinou musí být menší než mez tolerance $\varepsilon$. Pro odlehlé hodnoty opět použijeme multiplikátor $C$. Musí tedy platit: $C > 0$ a $\varepsilon > 0$.
- $\nu$–**Support vektorová regrese ($\nu$–SVR)** – Tato klasifikace je podobná jako $\varepsilon$–SVR. Na místo $\varepsilon$ se použije parametr $\nu \in (0; 1\rangle$.

Účinnost SVM závisí na výběru správného jádra a jeho parametrů. Často se používá Gaussovo jádro s jedním parametrem $\gamma$. Díky jeho přesnosti, ale je časově náročné. V této knihovně se můžeme setkat s následujícími jádry.

- **Lineární jádro** – Použití tohoto jádra je velmi rychlé (bez jakékoliv transformace), jedná se o lineární diskriminaci a rozdělovací nadrovina bude vždy přímka. Pro toto jádro platí
$$K(x_i, x_j) = x_i^T x_j,$$
kde $x_i$ a $x_j$ jsou vektory vstupního prostoru.
- **Polynomické jádro** – Polynomické jádro umožňuje učení nelineárních modelů
$$K(x_i, x_j) = (\gamma x_i^T x_j + c)^d, \gamma > 0,$$
kde: $c \geq 0$, volný parametr, který vylučuje vliv vyššího řádu oproti polynomu nižšího řádu (pokud $c = 0$, jádro je homogenní), řád polynomu určuje parametr $d$.
- **Gaussovo jádro** – Gaussovo neboli RBF (Radial Basis Function) jádro se řadí mezi nejpoužívanější a je definované jako
$$K(x_i, x_j) = e^{-\gamma||x_i - x_j||^2}, \gamma > 0,$$
kde: $||x_i - x_j||^2$ značí kvadratickou euklidovskou vzdálenost mezi dvěma vektory příznaků.
- **Sigmoidní jádro** – toto jádro je podobné sigmoidní funkci v logistické regresi
$$K(x_i, x_j) = \tanh(\gamma x_i^T x_j + r),$$
kde $r$ je volitelný parametr.
- **Exponenciální jádro** – Exponenciální jádro $\chi 2$ je podobné RBF jádru a využívá se převážně na histogramy
$$K(x_i, x_j) = e^{-\gamma \chi^2(x_i, x_j)}, \chi^2(x_i, x_j) = \frac{(x_i - x_j)^2}{(x_i + x_j)}, \gamma > 0,$$
- **Jádro histogramu průsečíků** – Toto jádro je také známé jako *Min Kernel*, jedná se o nejnovější jádro v této knihovně a je velmi rychlé a užitečné při klasifikaci
$$K(x_i, x_j) = min(x_i, x_j).$$

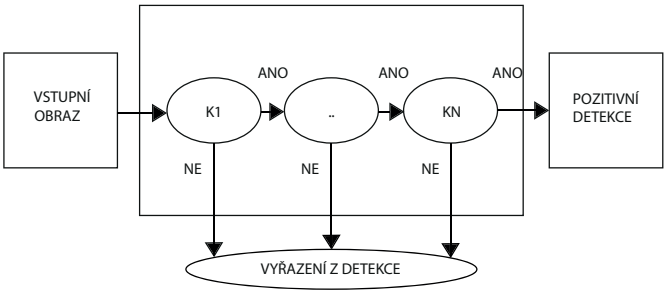
### 9.5.1 KNN - K-nearest neighbours

- velmi rychlý, jednoduchý
- pomocí k sousedů určíme label dotazovaného
- pokud bereme v potaz jejich vzdálenosti jedná se o modifikované KNN
- učení je velmi rychlé (jen se uloží data do struktur)
- určení se zpomaluje se zvyšujícím se k

### 9.5.2 Kaskádové klasifikátory

- Skládá **z více slabších** klasifikátorů umístěných v **kaskádách** za sebou.
- Požadavky na tento druh klasifikátoru byly **rychlost** detekce, aby mohl být implementován na procesorech s nižším výkonem. (v kamerách, v telefonech..)
- Klasifikátory si mezi sebou **předávají všechny** informace o vstupním obraze (může se redukovat čas, nutný pro detekci v daném obraze).
- Prvním takovým klasifikátorem byl detektor obličeje **Viola–Jones**

Klasifikátor na první vrstvě může vyfiltrovat většinu negativních oken. Na druhé vrstvě se mohou odfiltrovat „těžší“ negativní okna, která přežila z první vrstvy a tak dále. Subokno, které přežije všechny vrstvy, bude označeno jako pozitivní detekce. Příklad řetězce kaskádového klasifikátoru je ilustrován na obrázku 19, kde $K1$–$KN$ je klasifikátor první až $n$–té vrstvy.




Obrázek 19: Ukázka pipeline kaskádového klasifikátoru

### 9.5.3 AdaBoost

- **AdaBoost**, neboli Adaptive Boosting
- klasifikátor **kombinuje** slabé klasifikátory k vytvoření jednoho silného klasifikátoru

V kombinaci více klasifikátorů s výběrem trénovací sady v každé iteraci algoritmu a přidělení správné váhy na konci trénování, docílíme klasifikátoru s dobrou přesností. Klasifikátory v tomto řetězci, které mají klasifikační přesnost menší než 50%, jsou ohodnoceny zápornou vahou. Váhou nula jsou ohodnoceny klasifikátory, které mají přesnost 50%. Pouze ty, keré mají přesnost vyšší než 50%, jsou přínosné do této kombinace a můžeme hovořit o zesílení (boosting) klasifikace.

## 9.6 Template Matching

- vytvoříme si model objektu obsahující tvar, barvu a texturu
- následně se hledají v obraze jednotlivé prvky objektu samostatně a zjišťuje se míra podobnosti s vytvořeným modelem a tu pak v obrázku hledáme (pixel po pixelu)

Jednoznačnou výhodou tohoto přístupu je snadná implementace, ukázalo se však, že pro detekci obličejů není vhodná zdůvodů nízké odolnosti proti variabilitě Změna velikosti, pozice nebo tvaru objektu významně ovlivňuje výsledky metody.

Druhy metod:

- **SAD** suma absolutních rozdílů (Sum of Absolute Difference) – lze vypočítat získáním absolutních rozdílů napříč všemi pixely mezi vstupním obrazem S a odpovídající pozicí pixelu v templatu T. Sumu těchto hodnot lze následně použít jako koeficient míry podobnosti mezi obrázkem S a templatem T, kdy čím menší tato hodnota je, tím více se jednotlivé pixely na daných pozicích shodují. Tudíž lze předpokládat, že se zde nachází hledaný objekt.

$$\mathrm{SAD}(u,v) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} |T(m,n) - S(u+m, v\ +n)|$$

  Ve srovnání s ostatními metodami (SSD, NCC) je SAD přímočará, jednoduchá a výpočetně nenáročná. Může být však nespolehlivá a produkovat chybné výsledky v případě změn ve světelných podmínkách, barvě, velikosti či tvaru. Díky své rychlosti je však možné ji použít spolu s jinými metodami, jako je detekce hran, pro zlepšení spolehlivosti.

- **SSD** suma čtvercových rozdílů (Sum of Squared Difference) – představuje jednu z více používaných metod pro výpočet koeficientu míry podobnosti. Nejmenší hodnota pixelu opět představuje nejlepší shodu, jako tomu bylo v případě SAD.

$$\mathrm{SSD}(u,v) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} (T(m,n) - S(u+m, v\ +n))^2$$

  Ve srovnání s SAD se jedná o výpočetně náročnější, z důvodů nutnosti násobení, ale stále velice používanou metodu. Převážně vzhledem ke své jednoduchosti a stále relativně malé výpočetní náročnosti. NCC však ve většině případů produkuje přesnější a spolehlivější výsledky.

- **CC** vzájemná korelace (Cross-Correlation) – představuje sumu párových násobků hodnot jednotlivých pixelů.

$$\mathrm{CC}(u,v) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} (T(m,n)S(u+m, v\ +n))$$

  V reálných aplikacích se však tato metoda většinou nepoužívá. Přestože je relativně výpočetně nenáročná, tak vzhledem k její nespolehlivosti v případech, kdy se v obrázku nachází větší změny v měřítku nebo rotaci mezi hledaným objektem a vstupním

obrazem, se využívá spíše její normalizovaná varianta, a to i přes daleko větší výpočetní náročnost.

- **NCC** normalizovaná vzájemná korelace (Normalized Cross-Correlation) – představuje jednu z nejpoužívanějších metod pro výpočet míry podobnosti mezi dvěma obrazy. Její největší výhodou oproti typické CC je větší odolnost vůči změnám v osvětlení scény.

$$\mathrm{NCC}(u,v) = \frac{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}(T(m,n)S(u+m,v\ +n))}{\sqrt{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}T(m,n)^2\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}S(u+m,v\ +n)^2}}$$

  V porovnání s metodami SAD, SSD a CC je NCC ve většině situací nejpřesnější, avšak výpočetně mnohem náročnější. Při jejím použití je, pro urychlení výpočtu koeficientů, doporučeno provést jednoduché předfiltrování zpracovávaných oken např. aplikací kontroly salience.

Všechny výše zmíněné metody bohužel trpí stejnými nedostatky. Při vyhledávání jsou výskyty vzorků ve vstupním obraze nuceny zachovat orientaci referenčního obrázku. Zároveň je velice neefektivní a časově náročné počítat korelaci mezi templatem a vstupním obrazem pro obrazy středních avyšších rozlišení.

## 9.7 Deep learning

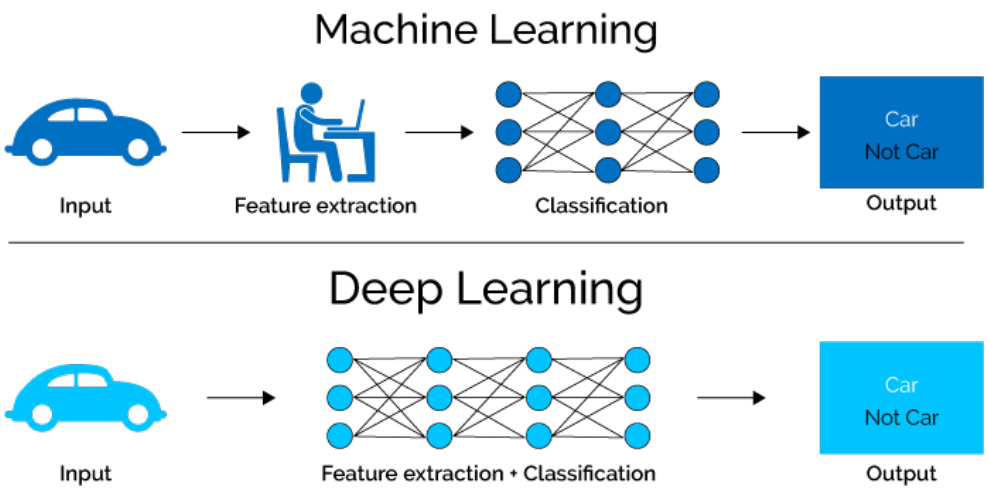
- Deep learning neboli **hluboké učení**, známé také jako hierarchické učení, je **sbírka algoritmů** používaných ve strojovém učení.
- Používají se k modelování abstrakcí na vysoké úrovni v datech za pomocí modelových architektur, které se skládají z několika nelineárních transformací.
- Hluboké učení je součástí široké skupiny metod používané pro strojové učení, které jsou založeny na učení reprezentace dat.

Hluboké strukturované učení může být:

- **Kontrolované (s učitelem)** - všechna data jsou kategorizovaná do tříd, algoritmy se učí předpovídat výstup ze vstupních dat.
- **Částečně kontrolované** - data jsou částečně kategorizovaná do tříd. Pří tomto přístupu učení lze využít kombinaci kontrolovaného a nekontrolovaného přístupu učení.
- **Nekontrolované (bez učitele)** - data nejsou kategorizovaná do tříd, algoritmy se učí ze struktury vstupních dat.

Hluboké učení je specifický přístup, použitý k budování a učení neuronových sítí, které jsou považovány za velmi spolehlivé rozhodovací uzly. Jestliže vstupní data algoritmu procházejí řadou nelinearit a nelineárních transformací, tak tento algoritmus je považován za „deep“ algoritmus.

Odstraňuje také ruční identifikaci příznaků (obrázek 20) z dat a místo toho se spoléhá na jakýkoliv trénovací proces, které má za úkol zjistit užitečné vzory ve vstupních příkladech. To dělá neuronovou síť jednodušší a rychlejší, a může přinést lepší výsledky než z oblasti umělé inteligence.

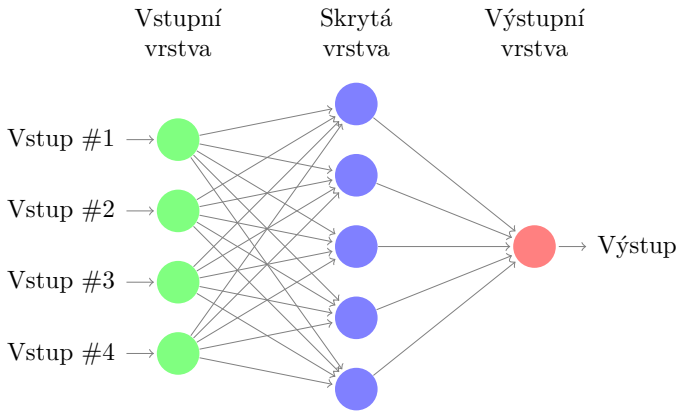



Obrázek 20: Hlavním rozdílem mezi strojovým a hlubokým učením je ten, že u strojového se příznaky musí extrahovat manuálně. [?]

### 9.7.1 Neuronové sítě *ANN* - Artificial Neural Network

- Inspirované **lidským mozkem**, který je složený z různých vzájemně propojených vrstev neuronů, kde každý z nich přijímá informaci z předchozího, zpracovává tuto informaci a odesílá jí do dalšího neuronu, dokud není přijat konečný výstup.
- Může se jednat o výstup **s danou kategorií**, jestliže se jedná o kontrolované učení nebo **o určitá kritéria** v případě nekontrolovaného učení.
- Umožňuje klasifikovat více tříd.

Příklad topologie neuronové sítě je na obrázku 21.




Obrázek 21: Neuronová síť je propojená skupinou uzlů, podobná síti neuronů v mozku.

Typickým příkladem neuronové sítě je **vícevrstvý perceptron** (ANN–MLP). Tato neuronová síť se skládá minimálně ze tří vrstev uzlů (vstupní, výstupní a skrytou). Každý z uzlů je **neuron**, který využívá nelineární aktivační funkci s výjimkou vstupních uzlů:

- **Vstupní vrstva** - jedná se o pasivní vrstvu, která nemodifikuje data, pouze je získává z okolního světa a pošle je dál do sítě. Počet uzlů v této vrstvě závisí na množství příznaků nebo deskriptivních informací, které chceme extrahovat z obrázku.
- **Skrytá vrstva** - v této vrstvě probíhá transformace vstupů do něčeho, co může výstupní nebo jiná skrytá vrstva využít (za předpokladu, že existuje více skrytých vrstev). Počet uzlů je určen složitostí problému a přesnosti, které chceme přidat do sítě.
- **Výstupní vrstva** - tato vrstva musí také vždy existovat v topologii sítě, ovšem počet uzlů v tomto případě bude definován vybranou neuronovou sítí. Pokud detekujeme na obrázku pouze jeden objekt, bude mít vrstva jen jeden uzel (lineární regrese) a bude vracet hodnotu definující pravděpodobnost konkrétního objektu v rozmezí $[-1, 1]$.
- Vysoká dimenze vstupního vektoru zvyšuje přesnost výsledků (ovšem zvyšuje výpočetní náklady)
- Aktivační funkce pro skrytou vrstvu, která umožnuje přizpůsobit nelineární hypotézy a získat lepší detekci vzoru v závislosti na poskytnutých datech (Sigmoid, tanh, ReLU).
- Hodnoty jsou získávány z předchozí vrstvy, sečteny s určitými váhami a hodnotou zkreslení. (Suma těchto hodnot je transformována pomocí aktivační funkce, může se lišit pro různé neurony)

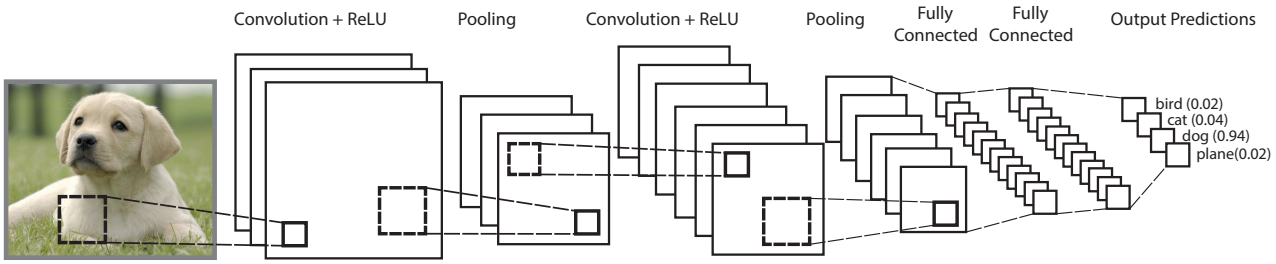
### 9.7.2 Konvoluční neuronové sítě *CNN* - Convolution neural network

- Speciálním druhem vícevrstvých neuronových sítí a jsou navrženy tak, aby rozpoznaly vizuální vzory přímo z pixelu obrazu s minimálním předzpracováním.
- Mohou rozpoznat vzory s extrémní variabilitou (například ručně psané znaky) a odolnost vůči deformacím a jednoduchým geometrickým transformacím.
- Síť využívá matematickou operaci zvanou konvoluce alespoň v jedné jejich vrstvě.

Nejznámější a nejvíce používanou konvoluční neuronovou sítí jsou modely LeNet. Hlavní kroky LeNet sítě jsou:

- **Konvoluce** - tyto vrstvy provádějí konvoluci nad vstupy do neuronové sítě.
- **Nelinearita (ReLU)** - tato vrstva je použita po každé konvoluční vrstvě a jejím cílem je nahrazení všech negativních pixelů nulou ve výstupu této vrstvy (příznaková mapa).
- **Pooling/sub sampling** - ze vstupního obrazu vyextrahuje pouze zajímavé části pomocí některých matematických operací (max, avg, sum), a tím se redukuje jeho dimenzionalita.

- **Fully connected layer/klasifikace** - tato vrstva vychází z původních umělých neuronových sítí, konkrétně z vícevrstvého perceptronu. Tato vrstva je typicky umístěna na konci sítě a je propojena s klasifikační vrstvou pro predikci.




Obrázek 22: Řetězec LeNet konvoluční neuronové sítě

## 9.8 Bag of words BoW

BoW model může být aplikován pro klasifikaci obrázků, zachází s příznaky obrázků jako se slovy

1. Vstupem je vektor příznaků
2. centroidy z výstup k-means se stanou „slovníkem“
3. poté když máme obraz, můžeme příznaku (slovu) přiřadit třídu ze slovníku
4. vytvoříme histogram počtu výskutů slov ze slovníku
5. tento histogramem dáme klasifikátoru