

V. Počítačová grafika a analýza obrazu

Update: 12. května 2018

Obsah

1	Osvětlovací modely a systémy barev v počítačové grafice.	2
2	Afinní a projektivní prostor. Afinní a projektivní transformace a jejich matematický zápis. Aplikace v počítačové grafice. Modelovací a zobrazovací transformace.	7
3	Křivky a plochy: teoretické základy (definice, rovnice, tečný a normálový vektor, křivosti, C _n a G _n spojitost), použití (Bézier, Coons, NURBS).	16
4	Geometrické a objemové modelování. Hraniční metoda, metoda CSG, výčet prostoru, oktantové stromy.	25
5	Standardní zobrazovací řetězec a realizace jeho jednotlivých kroků. Gouraudovo a Phongovo stínování. Řešení viditelnosti. Grafický standard OpenGL: stručná charakteristika.	32
6	Metody získávání fotorealistických obrázků (rekurzivní sledování paprsku, vyzařovací metoda, renderovací rovnice).	41
7	Komprese obrazu a videa; principy úprav obrazu v prostorové a frekvenční doméně.	48
8	Základní metody úpravy a segmentace obrazu (filtrace, prahování, hrany).	59
9	Základní metody rozpoznávání objektů (příznakové rozpoznávání).	65

1 Osvětlovací modely a systémy barev v počítačové grafice.

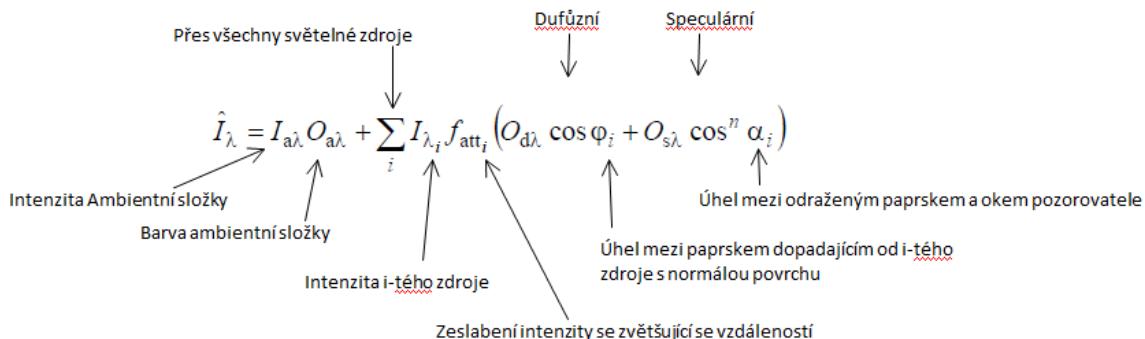
1.1 Osvětlovací modely

Výpočet osvětlení, záleží na druhu osvětlení a optických vlastnostech těles. Existují 3 základní druhy osvětlení:

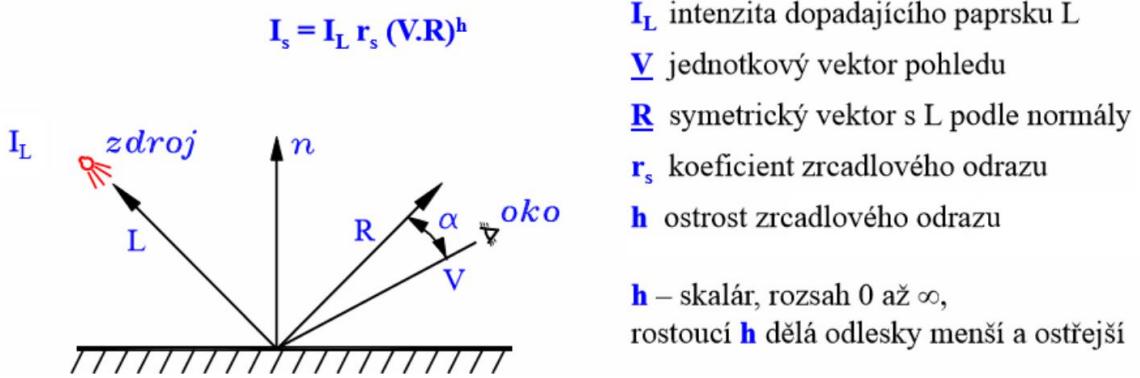
- **Ambientní** - fiktivní, všudypřítomné, rozptýlené (dopadá na těleso rovnoměrně ze všech směrů).
- **Difúzní** – matný odraz (intenzita, která se odráží od matného materiálu do všech směrů).
- **Spekulární** - odlesk (odrazivá složka – podle zákonů lomu).

1.1.1 Phongův osvětlovací model

- $I = I_a + I_d + I_s$
- Phongův osvětlovací model je **empirický** (na fyzice nezaložený) osvětlovací model pro výpočet osvětlení povrchu nějakého objektu.
- Ačkoliv tento model není fyzikálně zcela přesný, výsledky, které podává, jsou natolik zdařilé, že se Phongův model v počítačové grafice široce využívá již několik desítek let, zejména pak pro svou rychlosť našel uplatnění v real-time grafice.
- Jeho **nevýhodou** je, že nedokáže simulovat některé optické jevy globálního osvětlení, jako např. **Kaustiky** (Kaustika je obálka světelných paprsků odražených nebo zlomených nějakou zakřivenou plochou nebo předmětem, paprsky na dně bazénu).



- **Zeslabení intenzity i-tého světelného zdroje** – kde c_0, c_1, c_2 jsou konstanty popisující úbytek intenzity světla a d_i je vzdálenost $f_{att_i} = \min\left\{\frac{1}{c_0+c_1d_i+c_2d_i^2}, 1\right\}$.
- **Úhel mezi paprskem a normálou** – kde n je vektor normály plochy, l_i je vektor směřující od místa, kde osvětlení počítám k i-tému světelnému zdroji $\cos \varphi_i = n \cdot l_i$ (skalární součin).



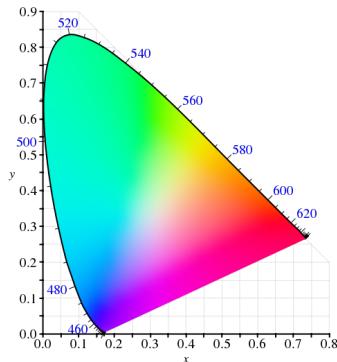
1.1.2 Blinn-Phong osvětlovací model

- Výchozí osvětlovací model pro standardní zobrazovací řetězec v OpenGL/DirectX.
Zavádí tzv. **half vector**: $H = \frac{L+V}{|L+V|}$
- Místo $R \cdot V$ pak použijeme při výpočtu $N \cdot H$.
- Blinn-Phong je **pomalejší** než původní Phong, protože při výpočtu half vectoru je nutné provést odmocninu (při normalizaci).
- Jelikož je odmocnina **hardwareově implementovaná**, rozdíl je zanedbatelný.
- Je však **rychlejší**, když se pozorovatel a světlo předpokládá v nekonečnu – směrová světla.
- V tom případě je half vector **nezávislý na poloze a zakřivení povrchu**, stačí jej vypočítat pouze jednou pro každé světlo.

1.2 Systémy barev v PG

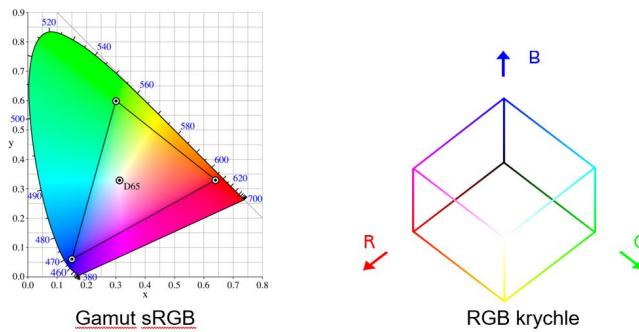
- Základem barevného prostoru je **barevný model**, který nám dává abstraktní matematický popis, jak lze barvy vyjádřit pomocí n-tic čísel, nejčastěji trojic.
- Mezi nejznámější barevné modely v dnešní době patří **RGB model**.
- Model RGB pracuje se třemi základními barvami: **červenou**, **zelenou** a **modrou**, z nichž se odvíjí i jeho název.
- Tyto barvy byly zvoleny na základně toho, jak **čípky v lidském oku** vnímají jednotlivé záření.
- Zároveň je RGB **aditivní barevný model**, což znamená, že se jednotlivé barevné složky **míchají** (nové barvy získáváme přidáváním větší intenzity jednotlivých složek) a výsledkem jsou další barevné odstíny, případně vyšší intenzita barvy.
- Když k tomuto modelu definujeme, jak mají být tyto n-tice interpretovány, dostáváme **barevný prostor** – je předem definovaná množina barev, kterou je schopno určité zařízení snímat, zobrazit nebo reprodukovat.
- Barevný prostor je tedy **definován rozsahem barev**, které dokáže zobrazit.

- Tomuto rozsahu se také říká **gamut**. Ten se zpravidla zobrazuje jako oblast v CIE 1931 chromatickém diagramu



1.2.1 RGB

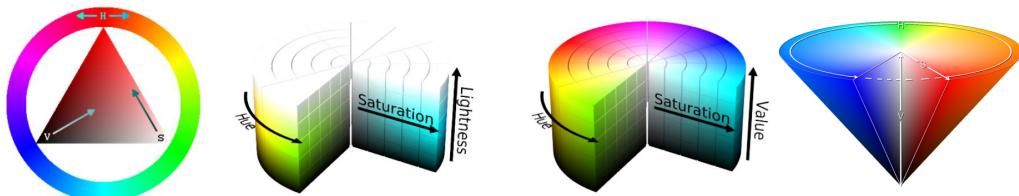
- Nejrozšířenější barevný prostor postavený na RGB barevném modelu je **sRGB** - standardní RGB.
- Jeho určení je pro zobrazování **na monitorech** nebo **kódování barev** na internetu.
- Pro všechny tři barevné složky má definované barvy v **chromatickém diagramu**, které vymezují jeho gamut.
- Každá barva, kterou tento prostor zobrazuje, je dána zastoupením jednotlivých barevných složek, buďto relativně (hodnoty jsou v rozmezí 0 - 1) nebo absolutně (konkrétní „bitové“ hodnoty, zpravidla 0 - 255, 24-bitů).
- RGB je možné zobrazit jako krychli.
- Často se přidává **Alpha kanál** pro průhlednost - **RGBA** (32-bitů).



1.2.2 HSV a HSL

- Hue, Saturation, Value/Lightness** - barevný model, který nejvíce odpovídá lidskému vnímání barev.
- Barvy popisuje pomocí 3 hodnot, které však samy barvy nereprezentují:

- **Hue** - barevný tón, převládající. Neboli **odstín** - barva **odražená** nebo **procházející** objektem. Měří se jako poloha na standardním barevném kole (0° až 360°). Obecně se odstín označuje názvem barvy. 0° - červená, 120° - zelená, 240° - modrá.
 - **Saturation** - sytost barvy, příměs jiné barvy. Někdy též chroma, síla nebo čistota barvy, představuje množství šedi v poměru k odstínu, měří se v procentech od 0% (šedá) do 100% (plně sytá barva). Na barevném kole vzrůstá sytost od středu k okrajům.
 - **Value** - hodnota jasu, množství bílého světla. Relativní světlost nebo tmavost barvy. Jas vyjadřuje **kolik světla barva odráží**, dalo by se také říct přidávání černé do základní barvy.
- Nejčastěji se tato reprezentace (popř. **HSL**) používají v grafických nástrojích jako komponenty pro výběr barvy, protože je mnohem intuitivnější než RGB.
 - Vyberu si odstín, jak má být sytý a jasný a hotovo. Není třeba řešit jak smíchat 3 barevné složky, abych dostal to co chci.
 - Dále se využívá často v případě detekce objektů, kdy hodnota HUE (odstín), je nezávislý na osvětlení scény. Problém však nastává u bílých a černých objektů (kdy HUE může být různé), ty lze na základě value a saturation mapovat do podobných barev (žlutá a černá).
 - Mimo níže uvedené zobrazení **válcem**, lze také zobrazit **kuželem** a



1.2.3 CMY a CMYK

- Substraktivní barevné systémy (barvy se „**odečítají**“ od bílé, přidáváním jednotlivých složek až po černou), **Cyan**, **Magenta**, **Yellow** a **Key (Black)**.
- Používá se **pro tisk**.
- Černá se přidala, protože smíchání CMY nedává plně černou barvu, navíc je černý inkoust levnější než barevný.
- Nevýhodou je, že **nedokáže správně zobrazit** sytě červenou, zelenou a modrou.
- Při tisku to však není poznat.
- Před tiskem se RGB obraz převádí do CMYK.
- To provádí buďto ovladač tiskárny nebo RIP (Raster Image Processor - u profi tiskáren).

- RGB se používá pro aktivní zdroje světla, CMYK jsou **pasivní** (světlo pouze **odrážejí**), proto nedokáží udělat tak jasné odstíny.



1.2.4 YCbCr

- Barva je reprezentována **jasovou složkou Y** a modrou a červenou **chrominanční komponentou**.
- Není to absolutní barevný model, jedná se o **způsob kódování RGB** informací.
- Využívá se nejčastěji u videa a barevných obrázků, kde je využito faktu, že **lidské oko nejvíce vnímá jas**, který je reprezentovaný složkou Y. Barvy už tak důležité nejsou a proto se můžou například více **komprimovat** bez výraznější ztráty kvality obrazu (JPEG).
- Jasová složka je kódována v intervalu $\langle 0, 1 \rangle$ a chrominanční složky v intervalu $\langle -0.5, 0.5 \rangle$

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 **obsahující body**. Navíc musí být současně dán **přidružený vektorový prostor** (souřadný systém) a nějaké **zobrazení**, které každé dvojici bodů přiřadí vektor (prvek z přidruženého vektorového prostoru).
- Cílem affinního prostoru je mít možnost **jednoznačně specifikovat body pomocí** jejich souřadnice a **manipulovat** s nimi s využitím prostředků lineární algebry.
- **Dimenze** vektorového prostoru určuje dimenzi affinního prostoru.
- Ukázka affinného prostoru:
 - v trojrozměrném affinném prostoru A_3 máme bod X se souřadnicemi $X = (x_1, x_2, x_3)$,
 - vektor \mathbf{x} je prvek onoho přidruženého vektorového prostoru.

2.1.1 Euklidovský prostor - E_n

- Afinní prostor ve kterém je zaveden **skalární součin**(1) a **norma**(2) (velikost vektoru), to umožňuje měřit délku vektorů a úhly mezi nimi.
- Souřadný systém (kartézský, polární, válcový atd.).

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

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

2.1.2 Kartézská souřadná soustava

- Souřadné osy jsou **vzájemně kolmé**.
- Protínají se v jednom bodě – **počátku souřadná soustava**.
- Jednotka se obvykle volí na všech osách stejně velká.
- Souřadnice polohy bodu je možno dostat jako kolmé průměty polohy bodu k jednotlivým osám.

2.2 Afinní transformace

- Afinní transformace je **zobrazení bodů jednoho affinního prostoru do jiného affinního prostoru** (speciální případ: zobrazení do téhož affinního 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 jejím 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:

$$y = xA + t,$$

kde \mathbf{y} je bod, do kterého je transformován bod \mathbf{x} pomocí matice \mathbf{A} a **translačního** vektoru \mathbf{t} . Vektor \mathbf{t} slouží k posunutí středu souřadné soustavy, matice \mathbf{A} mění osy souřadné soustavy. Známe-li \mathbf{A} a vektor \mathbf{t} , můžeme transformaci jednoduše provést. Jindy se musí určit ze zadání.

2.2.1 Ortonormalita affiní transformace

- Ortonormální 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:
 - affiní transformace bude zachovávat hodnotu právě 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 $\det(A)$ musí být roven ± 1 , protože $\det(AA^T) = \det(I)$ a $\det(I) = 1$.

2.2.2 Afinní transformace (2D)

- **posunutí** (translation) – posun ve směru x a y je dán hodnotou translačního vektoru,

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

- **otáčení** (rotation) – ve směru ručiček (**naopak** prohodíš znaménka u 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:

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

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

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

Obecné **skládání transformací** v kartézské souřadné soustavě:

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

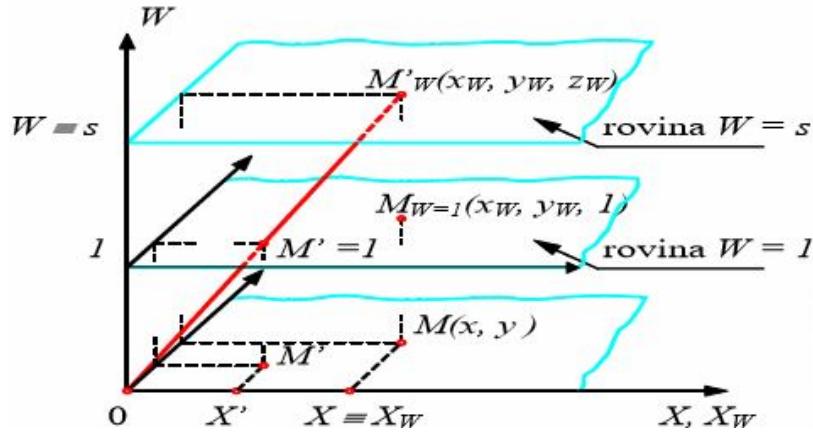
2.3 Projektivní prostor

Při použití afinních prostorů v PG lze v některých situacích narazit na jisté problémy, zejména to, že nelze zobrazovat body v nekonečnu (tzv. nevlastní body). I když jsou praktické scény konečné, mohou i zde vzniknout **nevlastní body** (středové promítání).

Problém spočívá v tom, že vektor reprezentovaný nevlastními body má jednu nebo více složek rovnou $\pm\infty$, kterou nelze v počítači jednoduše reprezentovat. Projektivní prostor navíc umožňuje promítání **středové** i **rovnoběžné** oproti prostoru affinnímu. Projektivní prostor tedy zavádí **homogenní souřadnice**, které jednotlivé body reprezentují.

2.4 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) je v homogenních souřadnicích reprezentován jako (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 $y' = y/w$.



2.5 Projektivní transformace - kolineace

- Kolineace je **zobrazení** bodů jednoho prostoru na body stejného nebo jiného prostoru.
- Kolineaci (projektivní transformaci) lze matematicky **popsat vztahem** $y = xT$, kde vektory x a y reprezentují před a po transformaci a matici T charakterizuje **kolineaci** (transformační matice).
- Sehrává zásadní úlohu v grafických systémech, ty pracují nejčastěji s trojrozměrným projektivním prostorem, kde T je ve tvaru ve tvaru 4×4 a x, y jsou čtyřprvkové vektory homogenních souřadnic.

2.5.1 Základní transformace

U projektivních transformací se můžeme setkat s těmito základními transformacemi:

- **posunutí** (translation),

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

- otočení kolem osy x (rotation),

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

- otočení kolem osy y (rotation),

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

- otočení kolem osy z (rotation),

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

- změna měřítka (scaling),

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

- zkosení (shearing),

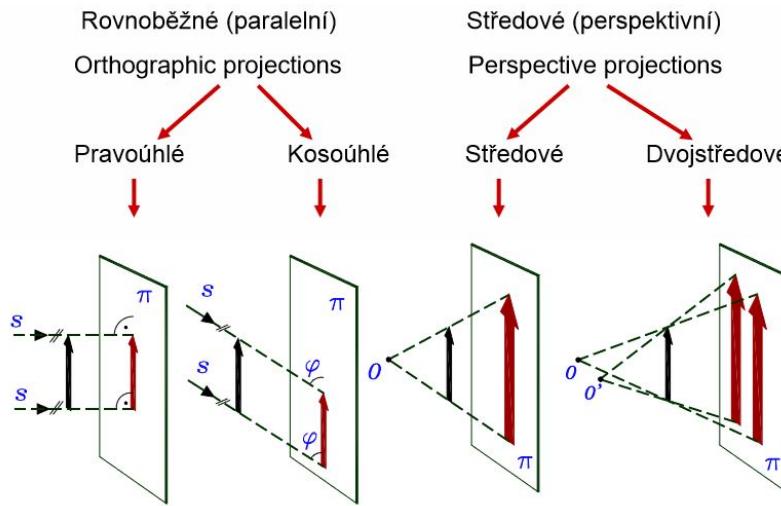
$$T = \begin{bmatrix} 1 & sh_y & 0 & 0 \\ sh_x & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.6 Promítání

- **Promítání** – zobrazení jednotlivých bodů na předem danou průmětnu.
- 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...).

2.6.1 Klasifikace promítacích metod

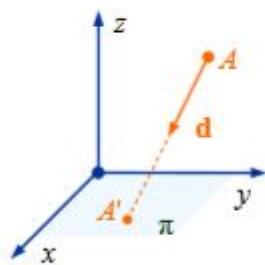


2.6.2 Rovnoběžné promítání

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

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

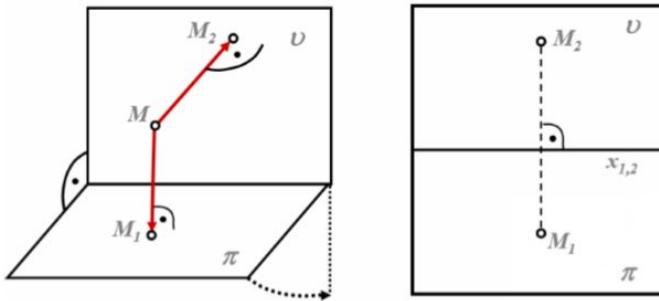
Matice popisuje rovnoběžné promítání na rovinu xy . Směr promítacího paprsku je $d = (d_x, d_y, d_z)$ (viz obr.).



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

2.7 Mongeova projekce

- Nejprve promítáme kolmo na vodorovnou rovinu π (půdorysu) – 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**.

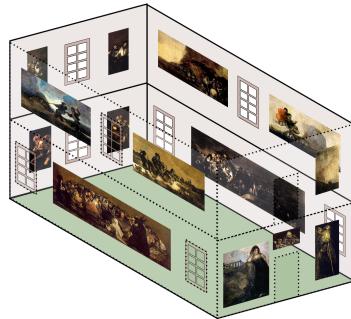
2.8 Kosoúhlé promítání

- Je rovnoběžné promítání na jednu průmětnu směrem, který má odchulku φ jinou než 90° od průmětny, promítací paprsky S jsou tak rovnoběžné a ne kolmé k průmětně π . Průmětna π je rovnoběžná s některou hlavními rovinami.
- **Výhodou** tohoto způsobu je skutečnost, že **předměty**, které se nacházejí v nárysně **jsou zobrazeny v reálné velikosti**.

2.9 Axonometrie - rovnoběžné, pravoúhlé promítání

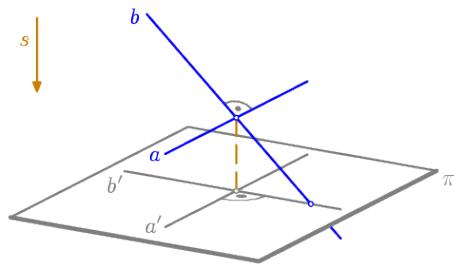
- Axonometrie nebo axonometrické projekce je jednoduchý způsob promítání prostorových těles a trojrozměrných struktur do roviny.
- V rovině se nejprve zvolí tři osy x, y, z , jež spolu svírají stejné nebo nestejně úhly.
- Rozměry těles se pak nanášejí v určitém měřítku rovnoběžně s těmito osami.
- Hlavní výhoda axonometrie proti složitějším metodám promítání 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).



2.10 Ortogonální promítání

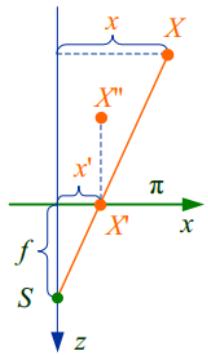
- Směr promítání kolmý k průmětně (jedná se tedy o speciální případ rovnoběžného promítání).
- Zachovávají se všechny vlastnosti rovnoběžného promítání.



2.11 Perspektiva – středové promítání

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

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.

2.12 Modelovací transformace

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

2.13 Zobrazovací transformace

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

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

3.1 Křivky

Křivky dělíme na: **rovinné, prostorové, interpolační, approximační**. Jednoduchý příklad křivky je například **kružnice** nebo **přímka**. Vyjádření může křivek být v podstatě trojho druhu:

- **Explicitní** – $y = f(x)$ kde $x \in \mathbb{I}$, 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** – souřadnice bodů parametricky jsou vyjádřeny jako funkce proměnného parametru (například t) definovaného na určitém intervalu.

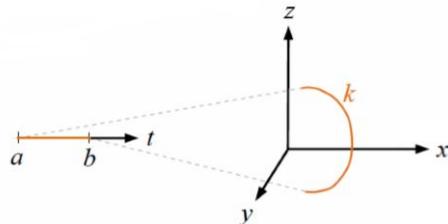
$$P(t) = A + \vec{u}t, \quad \text{parametr } t \in \langle a, b \rangle.$$

Obecně jsou křivky reprezentovány **polynomy stupně $n-1$** : $y = a_1x^{n-1} + a_2x^{n-2} + \dots + a_n$,

- čím vyšší stupeň, tím je křivka více náhylná na „rozkmitání“, snažím se proto dávat stupeň co nejmenší,
- **derivace** v bodě nám dá **tečnu** v bodě (lze získat derivací jednotlivých složek přímky), **první derivace** (směr), **druhá derivace** (zrychlení směru).

3.1.1 Parametrické křivky

- Mějme interval $I = \langle a, b \rangle \subseteq \mathbb{R}$, pak parametricky vyjádřenou (parametrizovanou) křivku k v \mathbb{R}^n nazýváme **diferencovatelné zobrazení** $\varphi : I \rightarrow \mathbb{R}^n$.



- V trojrozměrném euklidovském prostoru každému číslo t odpovídá na křivce příslušný bod $P(t) = [x(t), y(t), z(t)]$, t je obecně definováno v intervalu $\langle 0, 1 \rangle$. Kde 0 značí **počátek** a 1 **konec** křivky.
- **P** je polohový vektor (vektor daný počátkem souřadné soustavy a souřadnicemi příslušného bodu P).

3.1.2 Interpolaci křivky

- V PG nám pro definování (kreslení) křivky slouží její **interpolace** (kreslíme jen vybrané body a mezi nimi interpolujeme).
- Interpolaci křivka k dané množině bodů je taková křivka, která jimi prochází.
- Obecně se dá křivka pro n bodů **vyjádřit polynomem stupně n** . To znamená, najít řešení soustavy $y = a_1x^{n-1} + a_2x^{n-2} + \dots + a_n$ pro každý bod:

$$x_0, x_1, \dots, x_n, \quad x_i \neq x_j \quad \text{pro } i \neq j,$$

- Hledáme interpolaci polynom $P_n(x)$ stupně nejvýše n , který splňuje **interpolaci podmínky**:

$$P_n(x_i) = y_i \quad i = 0, 1, \dots, n.$$

3.1.3 Fergusonova křivka

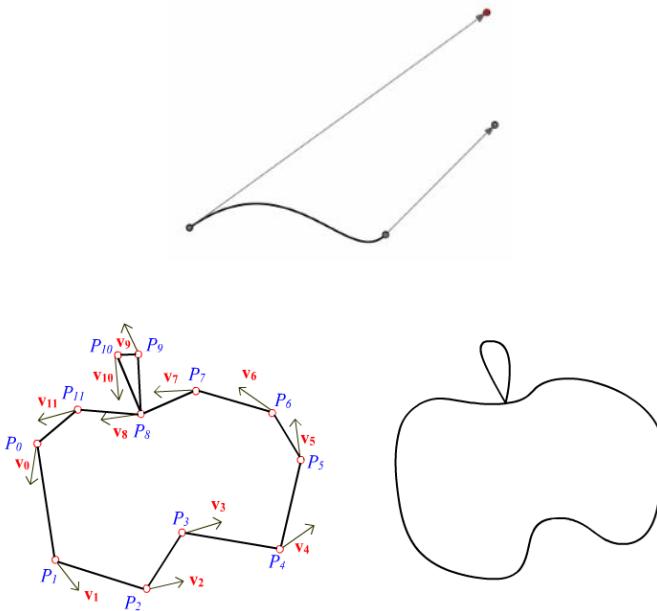
- Jedná se o jednu z nejpoužívanějších druhů interpolaci křivek, která je **třetího stupně**.
- Generování křivek je **řízené dvěma krajními body a dvěma vektory** v těchto bodech, jejíž parametrické vyjádření vypadá následovně:

$$P(t) = P_0F_0(t) + P_1F_1(t) + \vec{u}F_2(t) + \vec{v}F_3(t),$$

kde P_0, P_1 jsou **počáteční body**, \vec{u}, \vec{v} **tečné vektory** a F_0, F_1, F_2, F_3 nazýváme takzvanými hermitovské polynomy, které jsou dány následujícím předpisem:

$$\begin{aligned} F_0 &= 2t^3 - 3t^2 + 1, \\ F_1 &= -2t^3 + 3t^2, \\ F_2 &= t^3 - 2t^2 + t, \\ F_3 &= t^3 - t^2. \end{aligned}$$

- 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.

3.1.5 Metoda nejmenších čtverců

Před samotným popisem této metody je nutné si připomenout **explicitně zadanou rovnici přímky**:

$$f(y) = ax + b. \quad (3)$$

Cílem metody nejmenších čtverců je najít takovou rovnici přímky, pro kterou **součet čtverců odchylek zadané souřadnice y a souřadnice $ax + b$ je co nejmenší**. Tento součet vyjadřuje funkce $f(a, b)$, kde a, b jsou koeficienty dané přímky a n je počet zadaných bodů:

$$f(a, b) = \sum_{i=0}^n (y_i - (a + bx))^2. \quad (4)$$

Minimum funkce $f(a, b)$ získáme **parciální derivací** jejich koeficientů:

$$\frac{\partial f(a, b)}{\partial a}, \quad \frac{\partial f(a, b)}{\partial b}. \quad (5)$$

Parciální derivací (5) získáme soustavu dvou rovnic, o dvou neznámých. Výsledné koeficienty této soustavy rovnic dosadíme do explicitní rovnice přímky (3) a získáme hledanou rovnici approximované přímky.

Příklad

Zadání: $A[1, 1]$, $B[3, 2]$, $C[2, 3]$. Nyní tyto body dosadíme do (3) a získáme vzdálenost mezi zadaným bodem a approximovanou přímkou.

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

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

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

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 approximované přímky:

$$\begin{aligned} 28a + 12b - 26 &= 0 \\ 12a + 6b - 12 &= 0 \\ 2a + 1 &= 0 \\ a &= -\frac{1}{2} \quad b = 3 \end{aligned}$$

Našli jsme **approximaci**:

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

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**.

3.2 Tečný a normálový vektor křivky

- Parametrické vyjádření přímky je dáno počátkem v bodě A , vektorem \vec{u} , parametrem t a předpisem: $P(t) = A + \vec{u}t$.
- Tečný vektor** v bodě $t = t_0$ je dán jako **parciální derivace** $P'(t_0) = \frac{dP}{dt}(t_0) = (x'(t_0), y'(t_0), z'(t_0))$
- 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 $u \in \mathbb{R}$
- Tečna je limitní polohou sečny (protíná dva body), 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.

3.3 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** (kde se funkce mění z konvexní na konkávní nebo naopak) 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.

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

3.3.1 Oskulační rovina

Každá rovina procházející tečnou křivky v bodě $P(t_0)$ se nazývá **tečná rovina**. Oskulační rovina je **limitní rovinou** těchto rovin. Pomocí bodu $P(t_0)$ a dvou vektorů $P'(t_0)$ a $P''(t_0)$ 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.

3.4 Plochy

- **Rozšířením křivek** se dostaneme k plochám, které mají však s křivkami hodně spořeňho. 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** – $P(t) = A + \vec{u}t + \vec{v}t$.

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 $= \frac{n}{|n|}$.

3.4.1 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,
 - γ – 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** – $k_{abs} = |k_{n,min}| + |k_{n,max}|$.

3.5 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 **spojeté i její první derivace**. Pro vyšší derivace říkáme, že křivka má spojitost druhého, třetího a obecně n -tého rádu.
- Význam spojitosti křivek:
 - vizuální stránka napojení dvou křivek,
 - animace křivky.

3.5.1 Parametrická spojitost C_n

- \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 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, rychlosť i zrychlení. Směr pohybu a velikosť rychlosťi se nemůže měnit skokem při spojitosti \mathbf{c}_1 a zrychlení zůstává nezměněné při spojitosti \mathbf{c}_2 .

3.5.2 Geometrická spojitost G_n

- \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 stejno“ hladkost jako \mathbf{c}_1 . Z hlediska použití bývá jednodušší zaručit spojitost \mathbf{g}_1 nežli \mathbf{c}_1 .

3.6 Bézier

Jedná se o nejčastěji používaný typ interpolační křivky zejména v 2D i 3D počítačové grafice. Oproti Fergusonově křivce jsou Beziérové **křivky zadány pouze řídícími body**. Tyto křivky vždy prochází počátečním a koncovým bodem a zároveň nikdy neopustí svůj **konvexní obal**, který je dán řídícími body. Beziérova křivka je dána předpisem:

$$X(t) = \sum_{i=0}^n P_i B_i^n(t),$$

kde B_i^n jsou Bernsteinovy polynomy, které jsou definovány:

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

- **Lineární** Bézierová křivka je přímka z bodu P_0 do bodu P_1 .
- **Kvadratická** Bézierová křivka je definována **třemi** řídícími body.
- **Kubická** Bézierová křivka je definována **čtyřmi** řídícími body.

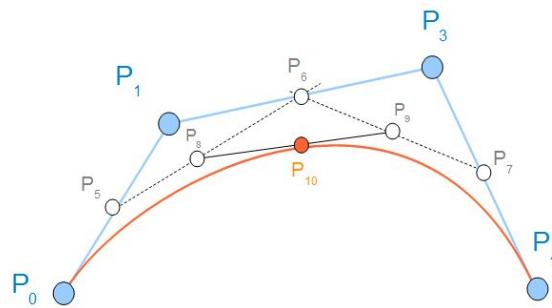
3.6.1 Beziérova kubika

Kubické Bézierové křivky mají velký význam pro praxi, protože jsou skládány z menších dílů, kde můžeme jednoduše definovat spojitost v řídících (koncových) bodech jednotlivých

segmentů. Další výhodou je omezený vliv změny polohy jednoho bodu na tvar celé křivky. Bernsteinovy polynomy pro Beziérovu kubiku vypadají následovně:

$$\begin{aligned}B_0^3(t) &= -t^3 + 3t^2 - 3t + 1, \\B_1^3(t) &= 3t^3 - 6t^2 + 3t, \\B_2^3(t) &= -3t^3 + 3t^2, \\B_3^3(t) &= t^3.\end{aligned}$$

3.6.2 Konstrukce Bézierovy křivky



Pro geometrickou konstrukci Beziérové 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 $\mathbf{P}_1, \mathbf{P}_6, \mathbf{P}_9, \mathbf{P}_7, \mathbf{P}_4$. Tímto rekurzivním způsobem postupně vykreslíme celou křivku.

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

3.6.3 Převod Fergusonovy kubiky na Beziérovu kubiku

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

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

3.7 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:

$$P(t) = \frac{1}{6}(P_0C_0(t) + P_1C_1(t) + P_2C_2(t) + P_3C_3(t))$$

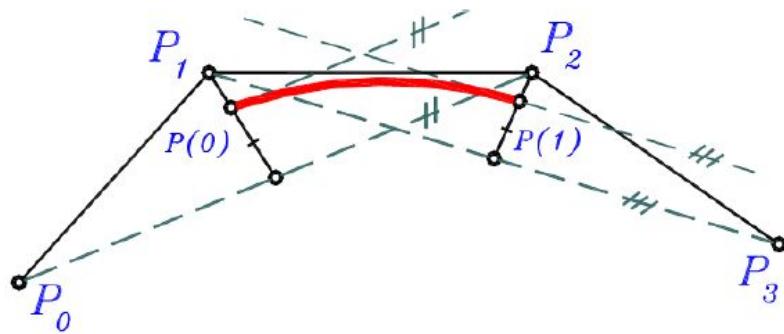
kde bázové funkce jsou:

$$C_0(t) = -t^3 + 3t^2 - 3t + 1,$$

$$C_1(t) = 3t^3 + 6t^2 + 4,$$

$$C_2(t) = -3t^3 + 3t^2 + 3t + 1,$$

$$C_3(t) = t^3$$



3.8 NURBS (Non-uniform rational b-spline)

- Obyčejným beziérem nejde udělat dokonalý kruh, proto se přidávají **váhy** w , kterými se přenásobují jednotlivé Bernsteinovy polynomy. Tento způsob reprezentuje vlastnost **racionality**.
- **Neuniformnost** znamená, že parametry t můžou nabývat různých hodnot i mimo interval $\langle 0, 1 \rangle$ a pro každou část můžou být jinak „rychlé“.

$$C(t) = \frac{\sum_{i=0}^m w_i P_n N_i^n(t)}{\sum_{i=0}^m w_i N_i^n(t)}, \quad \text{kde: } t \in \langle 0t_n, t_{m+1} \rangle$$

- Bázová funkce N_i^n je definována rekurentně: nechť $t = (t_0, t_1 \dots t_i)$ je **uzlový vektor**, b-spline funkce stupně n je definována jako:

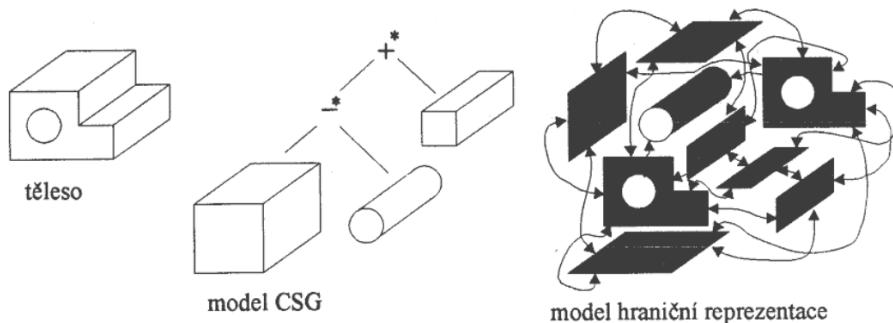
$$N_i^0(t) = \begin{cases} 1 & \text{pro } t \in \langle t_i, t_i + 1 \rangle \\ 0 & \text{jinak} \end{cases}$$

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

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

4.1 Geometrické a objemové modelování

- Geometrické modelování **zkoumá reálné objekty z hlediska geometrických vlastností**.
- Je to **soubor metod k popisu těles**. Rozlišujeme **3 základní typy modelů** (příklady jsou uvedeny pro *trojboký hranol*):
 - Drátěný model** – je určen seznamem devíti **hran**.
 - Hraniční model** – je určen seznamem pěti **stěn**.
 - Objemový model** – je určen **velikostí stran podstavy a výškou** (CSG).
- Objemové reprezentace** těles nesou přímé informace o vnitřních objemech těles a používají se i jako pomocné datové struktury vzhledem k rychlému zpracování. Oproti tomu povrchové reprezentace těles nesou přímé informace o povrchu tělesa (např. hrany, stěny) a poměrně snadno se zobrazují.



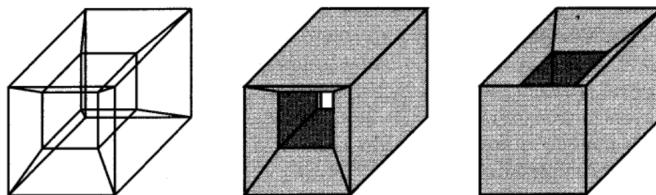
4.2 N-manifold v E^M

N-manifold v topologii představuje topologický prostor, který lokálně připomíná reálný n-dimenzionální prostor. N-manifold je takový topologický manifold, jehož **každý bod** má okolí homeomorfní s \mathbb{R}^n . V podstatě definuje hranice (**povrch**) tělesa. **Je taková podmnožina E^m , kde $m \geq n$, která je homeomorfní s E^n .**

- 1-manifold** je například kruh (nemá žádný povrch).
- 2-manifold** (v 3D prostoru) je **povrch tělesa**, s kterým lze jednoduše manipulovat (rozdělit rovinou, naříznout (nová hrana)).
- 3-manifold** (v 3D prostoru) je **povrch s dírami**, s tímto tělesem už se tak jednoduše nedá manipulovat.

4.3 Drátěné (hranové) modely (wireframe model)

- Drátěné modely jsou reprezentovány **hranami objektu** a skládají se z **bodů, přímek a křivek**.
- První wireframe systémy byly pouze dvou-dimenzionální, používaly se především na návrhy tištěných spojů. Základními elementy těchto systémů byly body, přímky a oblouky některých kuželoseček. Vnitřní reprezentaci tvořil obvykle seznam úseček a oblouků.
- V 70. letech byla wireframe reprezentace použita i pro **modelování v prostoru**. Zde však vznikaly dost závažné **problémy**:
 - možnost vytvoření nejednoznačných modelů,
 - možnost vytvoření nesmyslných modelů,
 - problém velkého počtu dat pro „drátěný“ popis objektu.
- Průmět drátěného modelu může mít více významů.
- Pokud u modelu **vynecháme** některé **hrany**, dostaneme **nesmyslný** (nonsense) model.
- U wireframe modelů je velice obtížné kontrolovat jejich **logickou správnost**.
- Poslední nevýhodou je **značné množství dat**, které jsou nutné pro úplný popis objektu.
- **Například:** obecně pro jednoznačné určení kvádru v prostoru stačí výška, šířka, hloubka a poloha. Wireframe model kvádru musí obsahovat souřadnice osmi vrcholů, musí být určeno dvanáct hran a ještě není model jednoznačný.

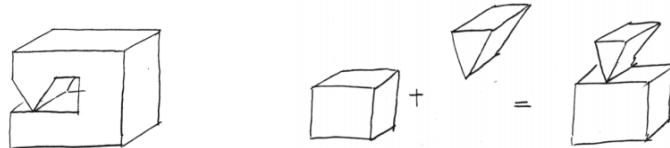


Obrázek: Nejednoznačnost drátového modelu

4.4 Hraniční metoda (hraniční model)

- Tato reprezentace patří k **nejpoužívanějším** typům při geometrickém popisu objektů.
- Historicky vyžadovala hranici **2-manifold v E3**, což znemožňovalo použití např. těles, která se sama sebe dotýkají (obrázek dole vlevo). Na druhou stranu byly pro uvedenou třídu těles velice dobře prozkoumány topologické vlastnosti. Tento požadavek byl však později **revidován** (boleaovské operace nemohou zaručit výsledek v 2-manifold E³, obrázek dole vpravo).

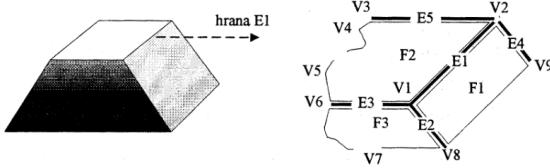
- Metoda reprezentace tělesa pomocí hranice, spočívá v tom, že **těleso je dáno svou hranicí** (povrchem), která je **orientována** a dělí tedy prostor na dvě části - **vnitřek** tělesa a **vnějšek** tělesa.
- Popis tělesa je rozdělen na dvě části:
 - **topologická** - popisuje vzájemné **vztahy** mezi entitami tělesa (vztahy mezi stěnami, hranami a vrcholy),
 - **geometrická** - popisuje **umístění** entit tělesa v **prostoru** (souřadnice prostorů, rovnice ploch, ...).
- Těleso je definováno těmito informacemi:
 - **seznam povrchů (shell)**, které těleso omezují,
 - **seznam stěn (face)** definující povrch tělesa,
 - **seznam smyček (loop)**, které definují stěny. Jde o **orientované hrany**. Obsahuje-li stěna jednu smyčku, jde o stěnu bez děr. Je li dána více smyčkami jsou ve stěně tunely. **Směr rotace** smyčky udává **normálu** stěny a určuje tedy na které straně stěny je **vnitřek** tělesa a která strana je **vnější** (smyčka proti směru hodinových ručiček – stěna z vnějšku). Je-li uprostřed stěny, jde o díru (tunel).
 - **seznam hran (edge)** definující smyčky.
- Ně všechny uvedené informace (o dírách, stěnách atd) musí být explicitně zaznamenány v datové struktuře, je však žádoucí aby je bylo možné **rychle dopočítat**.
- Jednou z hraniční reprezentací těles je datová struktura zvaná **okřídlená hrana** daná hranou a dvěma plochami k ní přiléhajícími.



4.4.1 Okřídlená hrana

Okřídlená hrana je **datová struktura** pro reprezentaci polygonových modelů taková, že jasně definuje jejich geometrii a topologii stěn, hran a vrcholů.

- Umožňuje jednoduchý průchod po jednotlivě spojených hranách, stěnách a bodech díky vzájemně propojeným strukturám.
- Okřídlené hraně přiléhají 2 stěny **nalevo** a **napravo** od ní. Zároveň si uchovává informaci o **následujících** a **předcházejících** hranách podél každé ze stěn, které jsou uspořádány po směru hodinových ručiček.



4.4.2 Eulerovy operátory hraniční reprezentace

Slouží k obsluze datových struktur hraniční reprezentace. Modifikují datovou strukturu tak, aby vždy popisovala nějaké těleso (jinými slovy, lze pomocí nich vytvářet nová tělesa). Pro jednoduché mnohostěny (uzavřené bez otvorů) platí následující **Eulerova formule**:

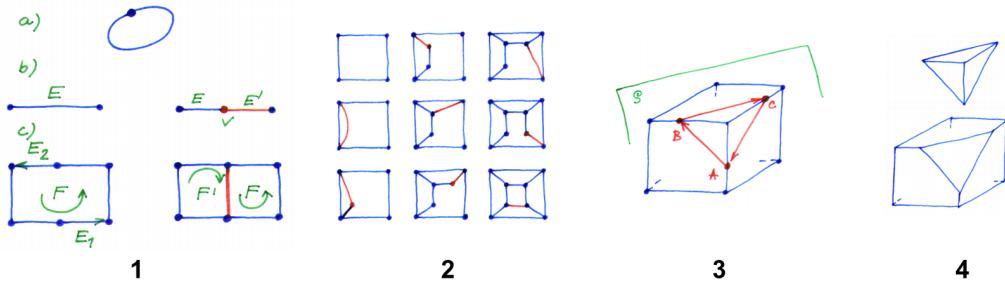
$$V + F - E = 2,$$

kde V je počet vrcholů, E je počet hran a F je počet stěn. Pro mnohostěny s otvory platí **zobecněná Eulerova formule**:

$$V + F - E = H + 2(B - P),$$

kde B (Body) je počet disjunktních částí tělesa, H (Hole) je počet otvorů ve stěnách a P (Passage) je počet děr v tělese.

- **Základní operace** – `createSolid` (vytvoří minimální neprázdné těleso), `splitEdge` (rozdělí zadanou hranu vložením nového vrcholu), `splitFace` (rozdělí zadanou stěnu vložením nové hrany).
- **Další operace** – `splitSolid` (provede řez nějakého tělesa ořezovou rovinou).

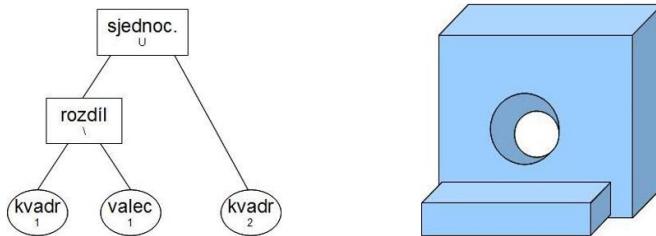


Příklad: Na obrázku výše (číslo 2) je postup vytvoření kvádru. Nejprve se zavolá `createSolid` a pak $3 \times$ `splitEdge` → tím výchozí výsledek čtverec (vlevo nahoře). Dále se pak střídá použití operátorů `splitFace` a `splitEdge` dokud není vytvořena taková struktura hran a vrcholů jako obrázek vpravo dole.

4.5 Metoda CSG (objemový model)

- **Constructive Solid Geometry** je metoda reprezentace těles, kdy jsou tělesa tvořena za pomocí standardních **primitivních těles** (kvádr, hranol, válec, kužel, koule, toroid, ...), **regularizovaných booleovských operací** a **geometrických transformací**.

- Uživatel si naskládá primitivní (či dříve vytvořené) tělesa do prostoru a aplikuje na ně boolovské operace jako sjednocení, průnik, rozdíl, čímž vznikne nový útvar.
- Z hlediska **datové struktury** jsou tělesa popsány stromem. **Listy** stromu odpovídají primitivním tělesům a transformacím, vyčteme z nich tedy rozmístění a velikosti těles. Zbylé uzly tvoří **boolovské operace** které jsou nad tělesy vykonány.
- CSG strom je **binární**, pokud chápeme boolovské operace jako binární, jinak tomu být nemusí.

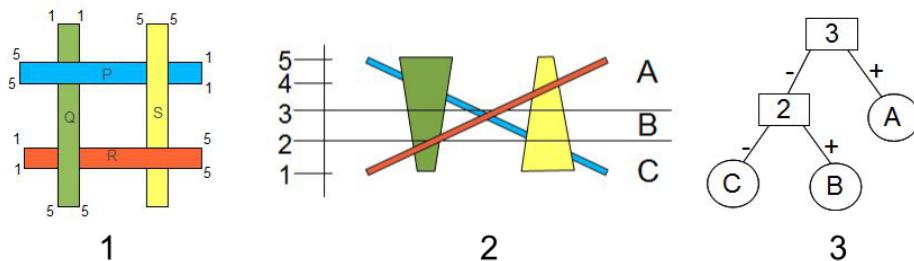


- Výhoda stromové struktury je že uchovává **historii** modelace tělesa, kterou lze využít při další úpravě.
- Další výhodou CSG tohoto přístupu je **jednoduchost boolovských operací**.
- Na rozdíl od hraniční metody, není třeba náročného výpočtu hranic nového tělesa.

4.6 Výčet prostoru - BSP stromy

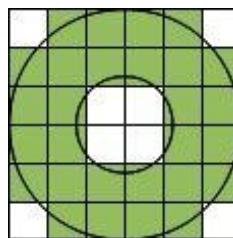
Při vykreslování objektů v prostoru narazíme na problém co vykreslit dřív a co později, aby to co je blíže pozorovateli překrývalo to co je od něj dál. Nejjednodušším algoritmem je **malířův algoritmus**, ten postupně vykresluje objekty seřazené od nejvzdálenějšího k nejbližšímu. Na obrázku (1) níže je však ukázaný problém, s kterým si neporadí. Toto řeší BSP stromy.

BSP strom je binární strom nesoucí informaci o rozdelení prostorů. Představuje **hierarchické rekurzivní** dělení N-dimenzionální prostoru na konvexní podprostory. BSP stromy uchovávají toto rozdelení tak že vlevo je **horní poloprostor** a vpravo **dolní poloprostor** (obr dole č. 3). Každý poloprostor poté obsahuje množinu objektů, které se v něm nachází.



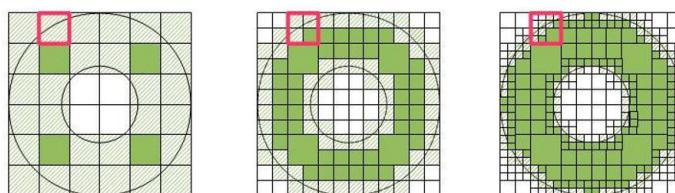
4.7 Oktantové stromy

- Jedna z metod objemového modelování.
- Těleso je definováno **objemovými elementy (voxel)**, což je v podstatě 3D pixel. Jde o **malé krychličky**, na které je rozdělen prostor a které bud' jsou v tělese nebo nejsou.
- Na obrázku níže je zobrazen tento způsob pro 2D. Zápis takového 2D tělesa by mohl vypadat takto: [0, 0, false], [0, 1, true], [0, 2, true], ...
- Čím **jemnější detaily** potřebujeme modelovat, tím jemněji potřebujeme plochu rozdělit.
- Jenže tím se nám taky **zvedají nároky na paměť**. Proto zavádíme **oktanové stromy**, které tento problém eliminují.

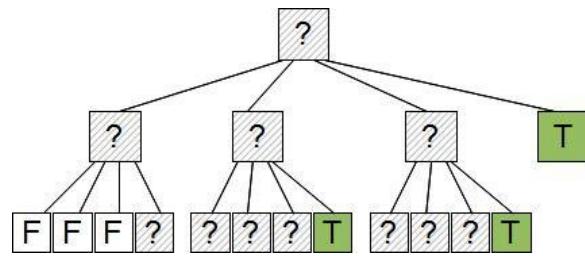


4.7.1 Struktura oktantových stromů

- Oktantové stromy **zmenšují paměťovou náročnost** detailních modelů, tak že zjemní pouze elementy, kterých se detail týká.
- Začíná se tedy na velkých voxelech, ale ty nyní mohou nabývat třech stavů:
 - voxel je obsažen v tělese
 - voxel není obsažen v tělese
 - voxel je částečně obsažen v tělese.
- V dalším kroku se řeší ty voxely, které jsou nerozhodné, tedy částečně obsaženy v tělese. Takový voxel ležící na hranici tělesa **rozdělíme na osm menších voxelů**. (Na obrázku níže je tento postup znázorněn ve 2D prostoru, kde rozdělíme na 4 pixely).



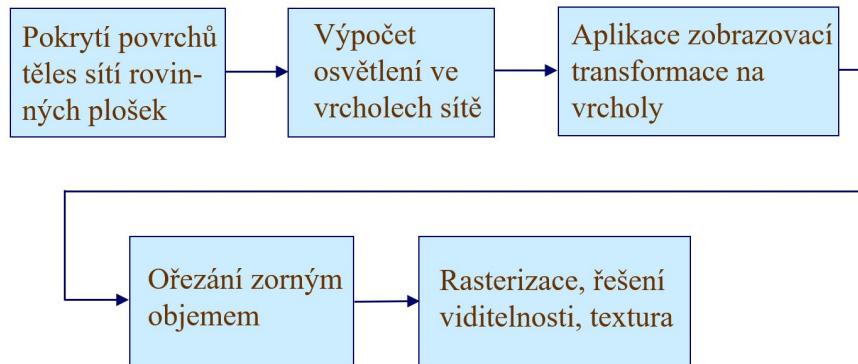
- Výsledná struktura je pak zapsána do **oktanových stromů** (stromy s max. osmi větvemi). Dole máme oktanový (respektive kvartálový, protože jsme jen v 2D prostoru) strom popisující oblast, která je vyznačená červeně na obrázku výše.



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 rychlosť** nikoli na kvalitu, realizuje ho **OpenGL**.

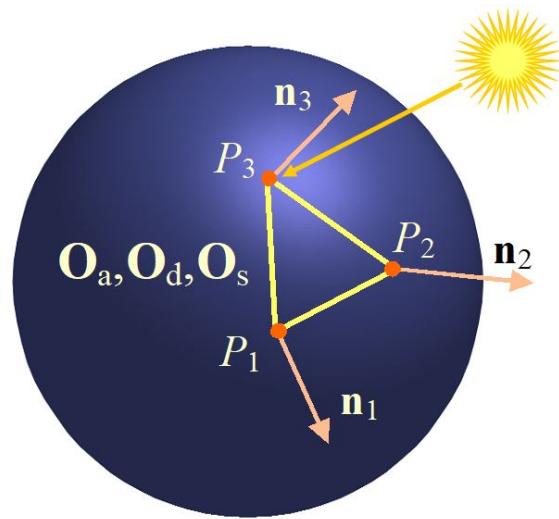


1. Pokrytí povrchu objektů sítí rovinných plošek:

- Ploškami bývají nejčastěji **trojúhelníky** nebo čtyřúhelníky.
- Pro objekty ve tvaru mnohostěnu je takové dělení vcelku samozřejmé.
- K **přesnějšímu výpočtu** barev bývá, ale někdy dělení na plošky **jemnější**.
- Někdy síť rovinných plošek žádaný povrch pouze approximuje.

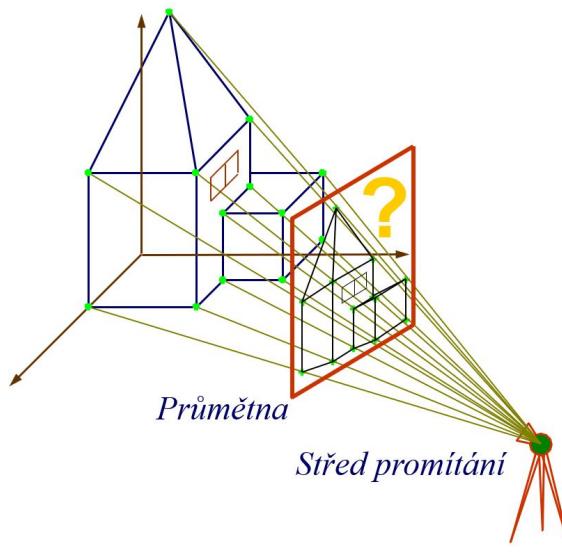
2. Výpočet osvětlení ve vrcholech sítě – k tomu známe:

- Polohu, intenzitu a barvu světelných zdrojů.
- Souřadnice vrcholů (P), normál (n) a konstanty popisující optické vlastnosti materiálu (O_a, O_d, O_s).
- V tomto kroku je počítán barevný vjem každého vrcholu.



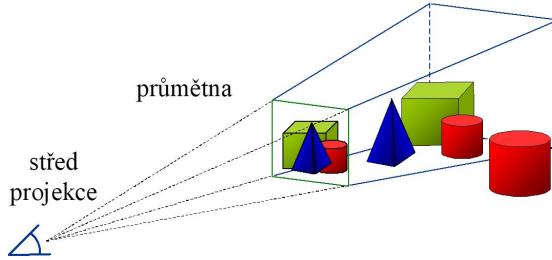
3. Aplikace zobrazovací transformace na vrcholy

- Oblíbenou technikou je středové promítání, to je zadáno:
 - Polohou průmětny.
 - Polohou středu promítání.



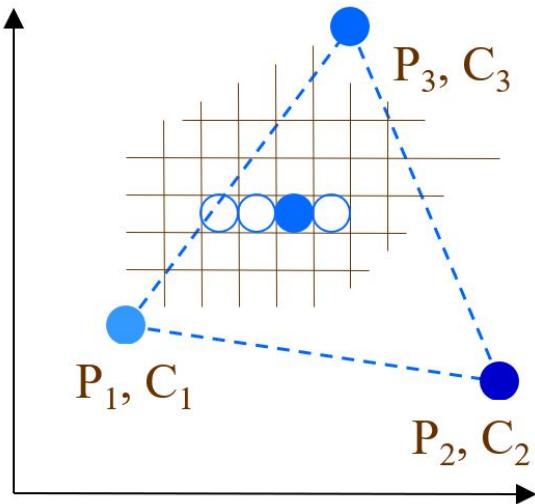
4. Ořezání zorným objemem

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



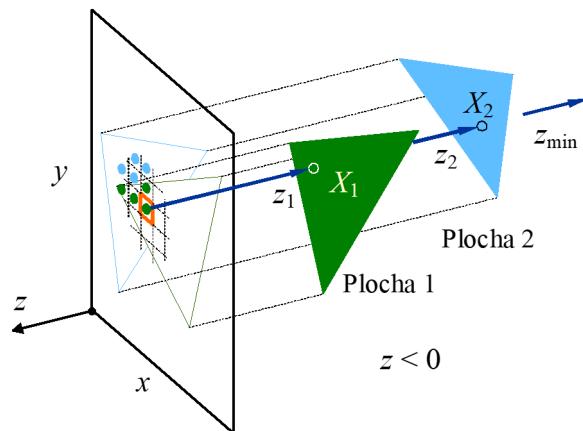
5. Rasterizace plošek

- Postupně se zpracovávají všechny plošky.
- Pro každou plošku jsou „rozsvěceny“ všechny její pixely.
- Barva každého pixelu se stanoví **interpolací mezi hodnotami ve vrcholech**.



5.1 Řešení viditelnosti (z-buffer)

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



5.2 Nanášení textury

- Vzhled obrázků lze vylepšit nánášením textury.

5.2 Stínování (shading)

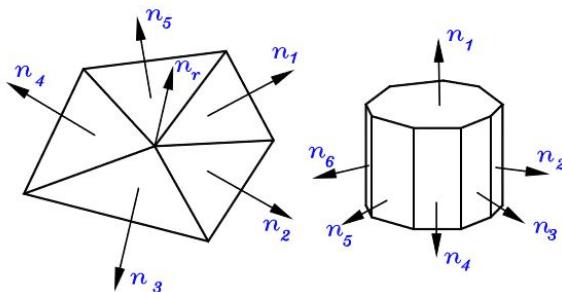
- Vykreslování barevných objektů **různými odstíny barev**.
- Lze odlišit křivosti ploch a tím docílit lepšího prostorového vjemu.
- Neplést s výpočtem vrženého stínu.
- Základní typy: **Konstantní** stínování, **Gouraudovo** stínování (Interpolace barvou), **Phongovo** stínování (Interpolace normálových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

5.3 Gouraudovo stínování (Interpolace barvou)

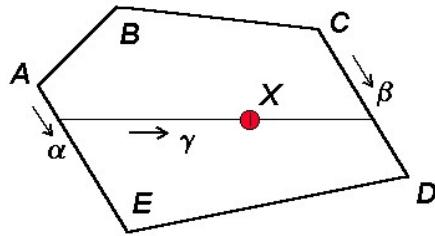
Princip metody spočívá v tom, že pokud budeme znát **normálu** v každém **vrcholu** každé plochy objektu, pak lze vypočítat barvu v tomto vrcholu a **interpolací** vypočítat **barvu pixelu uvnitř plošky** (bilineární interpolace).

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

Normálový vektor pro každý vrchol vypočteme jako aritmetický průměr normálových vektorů plošek, které se v tomto vrcholu stýkají.



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



$$\mathbf{f}_X = (1-\gamma) \cdot [(1-\alpha) \cdot \mathbf{f}_A + \alpha \cdot \mathbf{f}_E] + \\ + \gamma \cdot [(1-\beta) \cdot \mathbf{f}_C + \beta \cdot \mathbf{f}_D]$$

Výhody

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

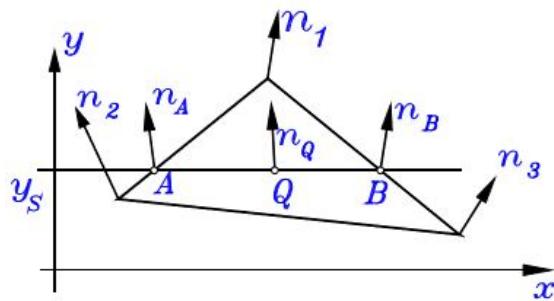
Nevýhody

- nevznikají ostré odlesky uprostřed polygonů.

5.4 Phongovo stínování (Interpolace normálových vektorů)

- Provádí se **interpolace normálových vektorů** jednotlivých vrcholů, Interpolaci provádíme po řádcích.
- Touto metodou se **odstraní problém neostrých odlesků**.
- Je **náročnější** na výpočet než goraudovo stínování, jelikož se počítá výsledná barva v každém bodě plošky (ne pouze ve vrcholech).
- Pro normálové vektory lze psát:

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

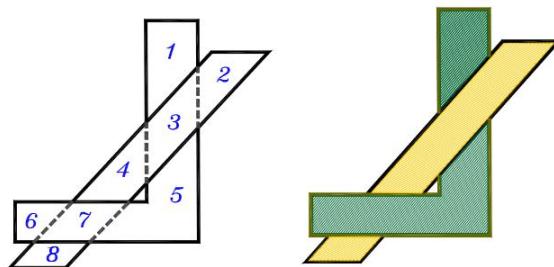


5.5 Řešení viditelnosti

- 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 $O(n^2)$.
 - **Řešení v prostoru obrazu** – pracujeme s promítnutými a rasterizovanými objekty. Pro pixely hledáme nejbližší objekty.

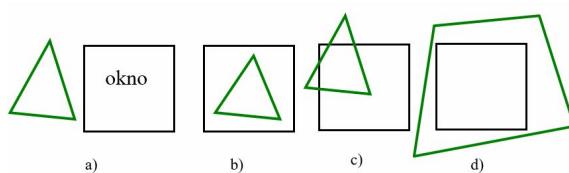
5.5.1 Rastrové algoritmy

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

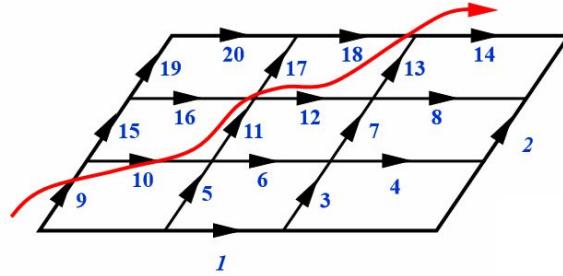


2. **Dělení obrazovky** (Warnock subdivision)

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

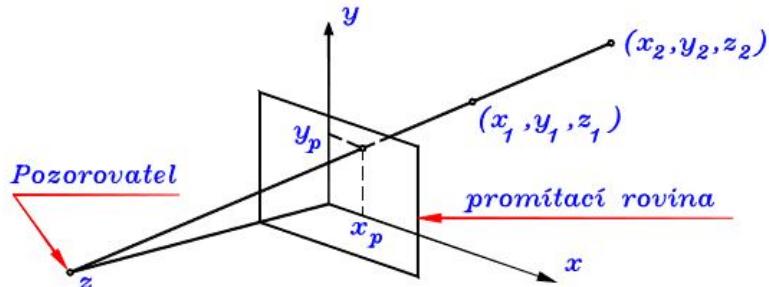


3. **Plovoucí horizont** (Floating Horizon Algorithm) – metoda „zig-zag“, počítáme od „nejbližšího“ rohu plochy k „oku“ pozorovatele.



4. **Paměť hloubky** (Z-buffer, depth-buffer)

- Vyplň obrazovou paměť barvou pozadí.
- Vyplň paměť hloubky – nekonečnem.
- Pro každou plochu najdi její průměr (rasterizaci) nalezenému pixelu $[x_i, y_i]$ přiřaď hloubku z_i .
- Porovnej hloubku a zapiš do paměti, pokud je hloubka menší překreslí se nový pixel daným objektem.



- Nejznámější a nejfektivnější 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 je často realizován jako 2D pole, kdy se ukládá pouze aktuální hodnota z (nejmenší).

5. **Z-buffer – paměť hloubky – průhlednost - princip**

- Inicializuj **color buffer** a **depth buffer**.
- Postupně načti všechny plochy, neprůhledné zpracuj, průhledné si zapamatuj a odlož pro následné zpracování.

- (c) Po zpracování neprůhledných ploch setřídí průhledné plochy podle vzdálenosti.
- (d) Zpracuj průhledné plochy s použitím **alfa míchání**.

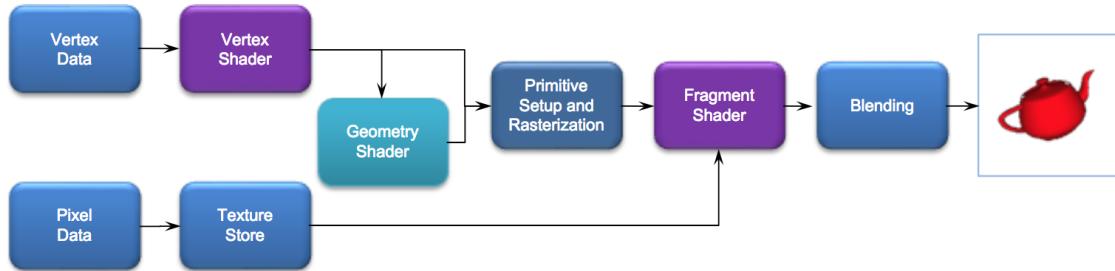
5.6 Grafický standard OpenGL: stručná charakteristika

- OpenGL je **multiplatformní** standard poskytující rozhraní (**API**) pro zobrazování 2D a 3D objektů. Je podporován takřka všemi výrobci grafických karet.
- Používá se pro tvorbu **PC Her**, **CAD programů**, aplikací **virtuální reality** či pro **vědecko-technické vizualizace**.
- Veškerá činnost OpenGL se řídí vykonáváním příkazů pomocí **volání funkcí** a procedur (kterých je v OpenGL cca 250).
- Byla vytvořena tak, aby byla **nezávislá** na použitém **operačním systému** nebo **programovacím jazyce**. Specifikace OpenGL tedy **neříká nic** o řízení kontextu platformy, správě a vytváření **oken**, **audio** a **vstupu**. Řešení této problematiky je necháno na **podpůrných systémech** (které jsou většinou platformě specifické) a OpenGL se zabývá pouze vykreslováním.
- Z programátorského hlediska se OpenGL chová jako **stavový automat**. To znamená že pokud změníme nějaké nastavení, toto nastavení zůstane až do doby než jej znovu změníme. To se hodí např. kdy jediným příkazem jsme schopni přepnout program do kreslení ve „wireframe módu“.
- 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.

5.6.1 Core-profile vs Immediate mode

Ve starších verzích OpenGL (před 3.2, kde se stal deprecated) se vyvíjely aplikace v tzv. **immediate módu** (non-shader mód). Tento mód byl značně **jednodušší** než core-profile, jelikož zakrýval (abstrahoval) většinu funkcionality OpenGL pod danou knihovnu, za to však zaplatil svou **neefektivitou**. To je způsobeno (mimo jiné) tím, že při každé specifikaci vertexu je jeho pozice odeslána na GPU, což představuje bottleneck v komunikaci mezi CPU a GPU. Tento mód lze stále použít v moderním OpenGL pokud využíváme tzv. **compatibility profile**.

Moderní OpenGL tedy nutí uživatele využívat moderní praktiky, přičemž pokud se snaží využívat starých metod, OpenGL vyhodí chybu a přestane vykreslovat. Přestože je core-profile na první pohled značně **složitější**, je **efektivnější**, umožňuje **větší flexibilitu** a **lepší porozumění grafického programování**. Core-profile přináší **buffer objecty** a je založen převážně na použití **shaderů**, které jsou vždy zkompilované do malých podprogramů na CPU a **nahrány na GPU pouze jednou**.



Od verze 4.1 přibyla ještě mezi vertex a geometry shader fáze Tessalace.

```
// ...: Initialization code :: ...
// 1. bind Vertex Array Object
glBindVertexArray(VAO);
// 2. copy our vertices array in a vertex buffer for OpenGL to use
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
// 3. copy our index array in a element buffer for OpenGL to use
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);
// 4. then set the vertex attributes pointers
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
 glEnableVertexAttribArray(0);

[...]

// ...: Drawing code (in render loop) :: ...
glUseProgram(shaderProgram);
glBindVertexArray(VAO);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0)
glBindVertexArray(0);
```

5.6.2 GLSL (OpenGL Shading Language)

Je součástí OpenGL 2.0 a vyšší. Jedná se o jazyk pro programování shaderů, který je syntaxí velice podobný C. Shadery jsou malé programy, které běží na GPU. Jeho vlastnosti:

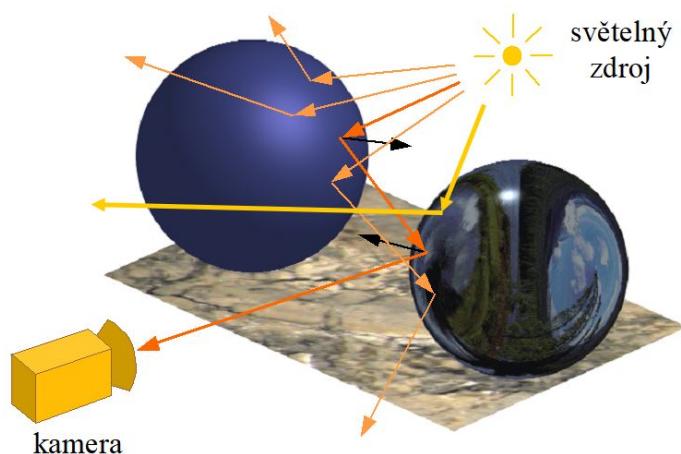
- Definuje **speciální proměnné** `gl_Position` (výstupní pozice z vertex shader) a `gl_FragColor` (výstupní barva z fragment shaderu).
- **Datové typy**: vektory, matice, int, float, bool, C-like structures.
- Předávání hodnot mezi vertex/geometry/fragment shadery lze pomocí **vstupních a výstupních proměnných** `in`, `out`, `inout`.
- Možnost vytváření vlastních **funkcí** (default funkce musí být `main()`).

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.

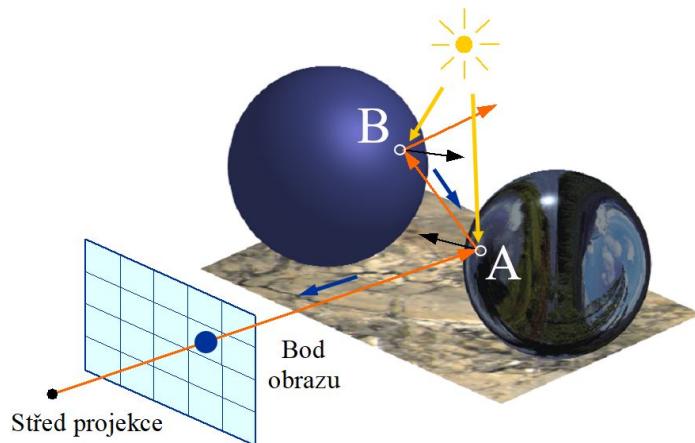
6.1 Sledování paprsku - ray tracing

- 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 (obdobně jako světlo v reálném světě).
- 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.



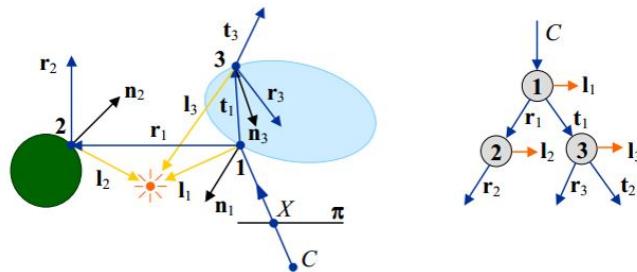
6.2 Zpětné sledování paprsku

- Funguje stejně jako běžné sledování paprsku, ovšem paprsky jsou **vysílány z kamery do scény**.
- Jakmile paprsky dosáhnou **ukončovacího kritéria** (jdou mimo scénu/maximální počet odrazů), sledujeme zpět jejich pohyb (navrácení z rekurze) a vypočítáváme osvětlení.
- Tím se eliminuje možnost, že by paprsek nepřinesl žádný přínos výslednému obrazu a značně se tak urychluje celý proces ray tracingu.



6.3 Rekurzivní sledování paprsku

- Metoda vyšetřuje „běh“ světelných paprsků ve scéně.
- Světlo je reprezentováno **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 znova 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 S_i představuje viditelnost i-tého zdroje světla v daném bodě.

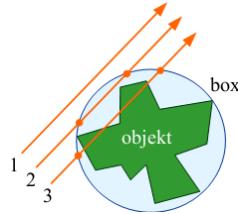
6.4 Urychlování trasování

Největším problémem je hledání průsečíků paprsků s objekty scény.

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

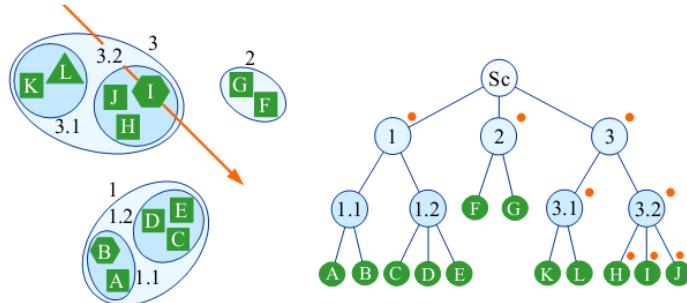
- objekt leží **celý uvnitř** ohraničující plochy, ale plocha jej obepíná co **nejtěsněji**,
- průsečíky paprsků s plochou musí jít spočítat, co nejjednodušším výpočtem,
- plochu musí být možné pro jednotlivé objekty dostatečně jednoduše nalézt.

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



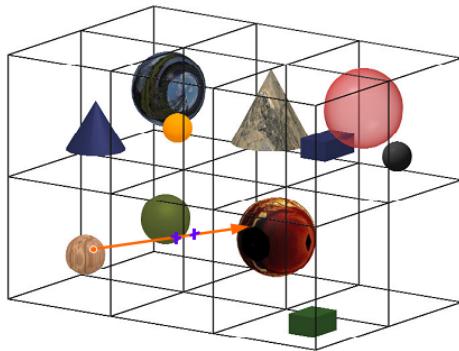
Princip ohranučujících ploch spočívá v tom, že pokud paprsek neprotne ohraničující plochu, pak neprotne ani těleso uvnitř (velmi časté). Odhalením této situace dojde ke značnému zrychlení. Pokud je to naopak, hledají se průsečíky s ohraničeným tělesem.

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 takovou strukturu automatizovaně nalézt.



3. Další metodou je **Dělení prostoru scény na podprostory**. Obykle se dělí rovinami souřadné soustavy xy , xz , yz → 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 zasahuje do více elementů, 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 metodou 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:

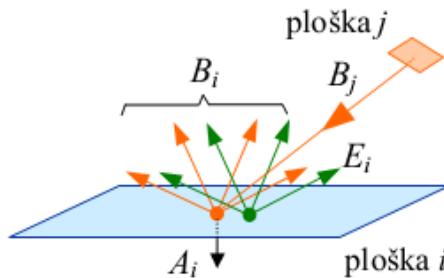
- Vypočítá se, jak jsou osvětlena jednotlivá místa scény.
- Podle toho se povrchy těles pokryjí sítí (v místech kde je komplikovaný průběh osvětlení je síť hustá – zlom světla a stínu).
- Pro každou plošku jsou spočítány hodnoty RGB – vyzařování je konstantní na celém povrchu plošky.
 - **PROBLÉM:** 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 síti (zprůměrování intenzit 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 absorbovaný = výkon na plošku dopadající od jiných ploch + výkon, který ploška sama vyzařuje**

$$B_i = E_i + p_i \sum_{j=1}^n B_j F_{j \rightarrow i} \frac{A_j}{A_i}.$$

kde:

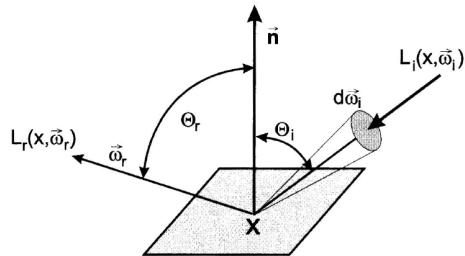
- B_j – výkon vyzářený ploškou j ,
- p_i – míra odrazu (optické vlastnosti materiálu),
- E_i – hodnota výkonu vlastního vyzařování plošky,
- A_i, A_j – velikost plošek (plošný obsah),
- $F_{j \rightarrow i}$ – konfigurační koeficient říká, jaká část výkonu vyzářeného ploškou j dopadne na plošku i (jedná se o $\int \langle 0, 1 \rangle$, 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 více než je celková přijatá energie
- **Odravivost** $p(x) = \frac{d\Phi_r(x)}{d\Phi_i(x)}$; $d\Phi_r(x)$ je **odražený světelný tok**, $d\Phi_i(x)$ je **dopadající světelný tok**.

- Obor hodnot odrazivosti je na intervalu $<0, 1>$, $1 = \text{plný odraz}$.
- **PRINCIP:** Vysílá se mnoho paprsků v každém bodě s různými offsety, některé padnou do zdroje světla, některé ne. Výsledná hodnota pixelu je poté průměrem všech hodnot paprsků (čím více paprsků vyšleme, čím lepší je výsledek (méně zrní)).

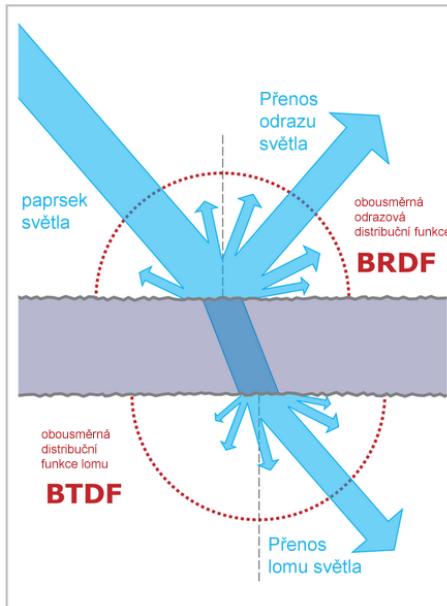


6.7 BTDF (Bidirectional Transmittance Distribution Function)

Dvousměrná distribuční funkce lomu. Popisuje **průchod světla povrchem**.

6.8 BSDF (Bidirectional Scattering Distribution Function)

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



6.9 Renderovací rovnice

Rekurzivní diferenciální rovnice

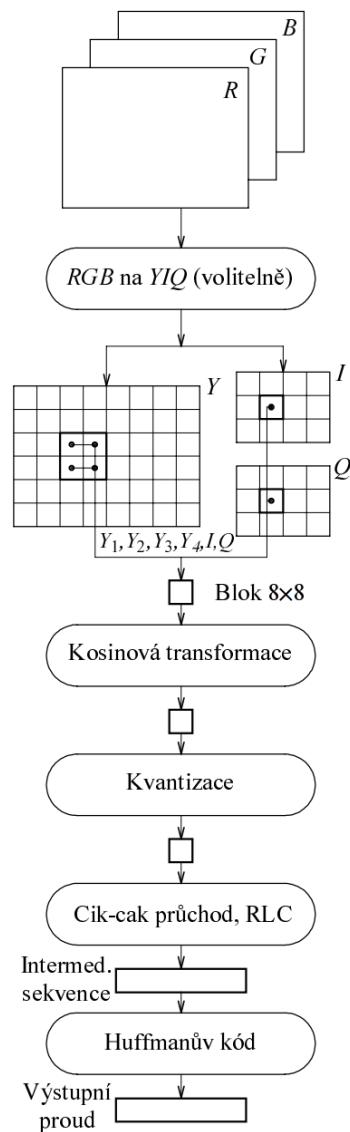
$$L(x, \omega_0) = L_e(x, \omega_0) + \int_{\Omega} L(r(x, \omega_i) - \omega_i) \cdot BRDF(\omega_i, x, \omega_0) \cos \theta_i d\omega_i$$

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

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

7.1 JPEG

JPEG představuje jeden z nejpoužívanějších obrazových formátů (fotografie), jedná se o **ztrátový** formát obrázků. **Kompreze JPEG** pracuje v několika krocích, principem je **redukce vysokofrekvenčních dat** v obrázků při zachování co nejvíce informací o nízkofrekvenčních datech. To je založeno na tom, že lidské oko se zaměřuje spíše na nízké frekvence (malou změnu jasu na ploše) a vysoké frekvence vnímá hůře (hrany). Můžeme si je tedy dovolit redukovat bez znatelné ztráty kvality obrazu. Zároveň se využívá **podvzorkování barev** → znova z toho důvodu, že lidské oko vnímá více jas než barvy.



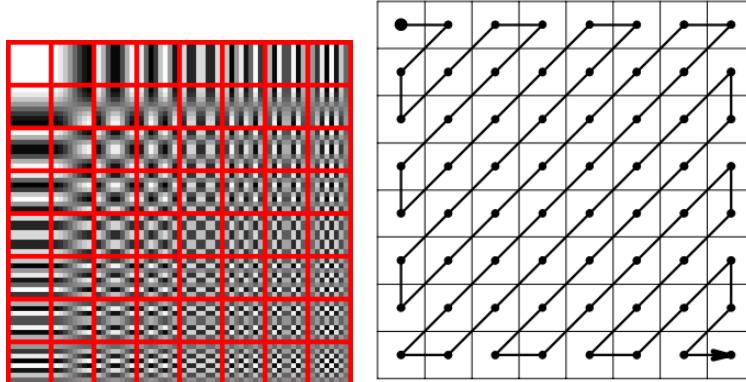
7.1.1 Komprese JPEG

1. **Převod RGB na YCbCr** – vstupní obraz se převede na YCbCr, což jej rozdělí na **jasovou složku** (Y) a **chrominanční** složky (barvonosné Cb a Cr). To nám umožňuje následnou redukci barvonosné složky v jednom ze 3 módů: 4:4:4 (nepodvzorkovává se), 4:2:2 (horizontálně na polovinu, pro 2 jasové 1 barva), 4:2:0 (horizontálně i vertikálně na polovinu, pro 4 jasové 1 barva).
2. **Provedení DCT** – obraz se následně rozdělí na bloky o velikosti 8×8 . Pro každý blok se provede 2D DCT (diskrétní kosinova transformace), ta narozdíl od DFT produkuje pouze reálné komponenty. Výsledek DCT nám vrátí počet jednotlivých frekvencí, které se v obraze nachází, přičemž **nízké frekvence** se koncentrují vlevo nahore a **vysoké frekvence** vpravo dole (viz obrázek).

$$F(k, l) = \frac{1}{4} c(k)c(l) \sum_{m=0}^7 \sum_{n=0}^7 f(m, n) \cos \frac{(2m+1)k\pi}{16} \cos \frac{(2n+1)l\pi}{16}$$

kde,

$$c(k), c(l) = \begin{cases} 1/\sqrt{2}, & k, l = 0 \\ 1, & \text{jinak} \end{cases}$$



3. **Kvantizace** – jednotlivé bloky jsou **vydeleny kvantizační maticí a zaokrouhleny**. Koeficienty matice definují **míru komprese** (čím větší hodnoty tím větší komprese a horší obraz). Většinou po aplikaci kvantizace v každém bloku zůstane pouze několik hodnot **vlevo nahore** to vyplývá z toho co jsme zmínil výše → více se redukuje vysoké frekvence, které si můžeme dovolit vynechat. Výsledek před a po kvantizaci je vidět obrázku níže.

139	144	149	153	155	155	155	155
144	151	153	156	159	156	156	156
150	155	160	163	158	156	156	156
159	161	162	160	160	159	159	159
159	160	161	162	162	155	155	155
161	161	161	161	160	157	157	157
162	162	161	163	162	157	157	157
162	162	161	161	163	158	158	158

a) Blok obsahující vzorky jasů.

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

c) Kvantizační matice pro složku Y .

235.6	-1.0	-12.1	-5.2	2.1	-1.7	-2.7	1.3
-22.6	-17.5	-6.2	-3.2	-2.9	-0.1	0.4	-1.2
-10.9	-9.3	-1.6	1.5	0.2	-0.9	-0.6	-0.1
-7.1	-1.9	0.2	1.5	0.9	-0.1	0.0	0.3
-0.6	-0.8	1.5	1.6	-0.1	-0.7	0.6	1.3
1.8	-0.2	1.6	-0.3	-0.8	1.5	1.0	-1.0
-1.3	-0.4	-0.3	-1.5	-0.5	1.7	1.1	-0.8
-2.6	1.6	-3.8	-1.8	1.9	1.2	-0.6	-0.4

b) Výsledek kosinové transformace.

d) Spektrum po kvantizaci.

4. **Zig-zag intermediální sekvence** – obraz po kvantizaci se projde **zig-zagem** a z daných čísel se vytvoří intermediální sekvence (**dvojice** ve tvaru **Symbol-1, Symbol-2**) pomocí **Run-Length encoding**. RLE kóduje pouze AC složky (1 - 64), DC složka (px na souřadnicích [0, 0] každého bloku) se kóduje **diferenciálně** vzhledem k DC složce v předchozím bloku. Intermediální sekvence vypadá následovně:

- **Symbol-2** – (AMPLITUDE) značí hodnotu na daném pixelu po kvantizaci.
 - **Symbol-1** – (RUNLENGTH, SIZE), kde RUNLENGTH značí počet nul, které danému prvku předchází v zig-zag sekvenci (pokud je počet větší než 15, zapíše se 15) a SIZE je počet bitů nutných k reprezentaci AMPLITUDE. **Speciální hodnota** (0, 0) říká, že předchozí nenulová hodnota byla poslední.
 - **Kódování** – **DC** složky se kódují odlišně (SIZE, AMPLITUDE), **AC** složky ((AMPLITUDE), (RUNLENGTH, SIZE)).

Intermediální sekvence se poté zakóduje **Huffmanovým kódem**, kde se kódují pouze kombinace (RUNLENGTH, SIZE) u AC a (SIZE) u DC podle předem daných tabulek. AMPLITUDE se zapisuje pomocí **jednotkového doplňku** (způsob reprezentace záporných čísel).

Intermediální sekvence: $(2)(3), (1,2)(-2), (0,1)(-1), (0,1)(-1), (0,1)(-1), (2,1)(-1), (0,0)$
 (předpokládáme, že v předchozím bloku byla stejnosměrná složka 12).

Kódování hodnot Symbol-1: (2)→011, (0,0)→1010, (0,1)→00, (1,2)→11011, (2,1)→11100.

Kódování hodnot Symbol-2: $(-1) \rightarrow 0$, $(1) \rightarrow 1$, $(-3) \rightarrow 00$, $(-2) \rightarrow 01$, $(2) \rightarrow 10$, $(3) \rightarrow 11$.

Výsledný kód: 011 11 11011 01 00 0 00 0 000 11100 0 1010.

7.2 MPEG

MPEG je zkratkou pro Moving Picture Expert Group. Cílem práce této skupiny bylo standardizovat metody komprese videosignálu. Existuje několik standardů MPEG-(1, 2, 4, 7).

Profil Úroveň	Simple	Main	High
Low		4:2:0 352×288 4 Mb/s I,P,B	
Main	4:2:0 720×576 15 Mb/s I,P	4:2:0 720×576 15 Mb/s I,P,B	4:2:0, 4:2:2 720×576 20 Mb/s I,P,B
High		4:2:0 1920×1152 80 Mb/s I,P,B	4:2:0, 4:2:2 1920×1152 100 Mb/s I,P,B

Tab.7.2. MPEG-2: profily a úrovně.

- **MPEG-1** – první standard, dokončen v roce 1991. Navržen zejména pro práci s obrazy 352×288 pixelů, 25FPS (odvozeno od televizní normy PAL) nebo 352×240 , 30FPS (odvozeno od NTSC) při datovém toku **1.5 MBit/s** (optimální tok, mohl být i vyšší).
- **MPEG-2** – dokončen v roce 1994, mnohem velkorysejší implementace, snaží se být co nejuniverzálnější. Zavádí několik **profilů** (podmnožina z nejširší možné syntaxe) a **úrovní** (definuje parametry v rámci daného profilu).
- **MPEG-3** – práce na tomto standardu byly zastaveny (měl sloužit pro HDTV) později byl sloučen do MPEG-2.
- **MPEG-4** – metodou komprese se značně liší oproti MPEG-1,2, je určen pro **extrémně nízké datové toky**.
- **MPEG-7** – neříká nic o kódování, jedná se o standard pro popis dat (**metadata**) s multimediálním obsahem.

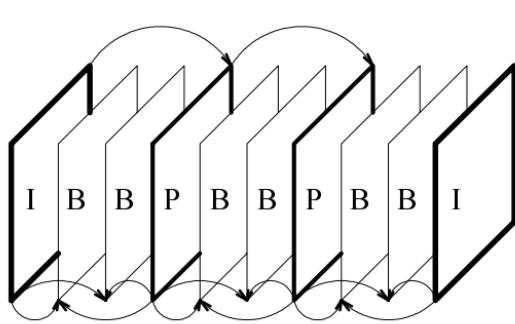
7.2.1 Kompresce MPEG

Standard MPEG, rozlišuje **tři typy rámčů (I, B, P)**, ve své podstatě pro kódování využívá stejných principů jako JPEG (**rámec I** se kóduje nezávisle). Oproti JPEG však využívá i **časové koherence**. Tedy k dosažení maximální komprese se předpokládá, že po sobě jdoucí rámce jsou s největší pravděpodobností dosti podobné. Počítá se ovšem s tím, že části obrazů se mohou přemístit, k tomu se využívá vložených rámčů P a B, které se kódují **závisle** vzhledem k ostatním.

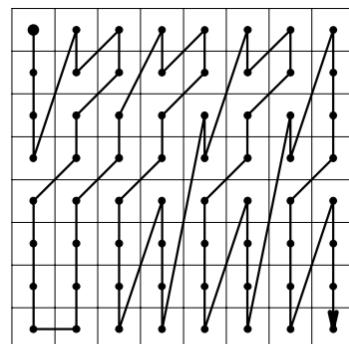
- **I** – jsou kódovány každý **zvlášť**, bez vazby na rámce předcházející či následující. Princip kódování (komprese) je stejný jako u standardu **JPEG** (i když v detailech existují

některé **odlišnosti**: jiná kvantovací tabulka, jiná struktura intermediální sekvence a jiný způsob **zig-zagu** (podle MPEG-2)). **Kvantizační tabulka** může být pro každý makroblok jiná – změnou měřítka lze **řídit tok dat** (některé aplikace mohou vyžadovat konstantní tok dat).

- **P (Predicted)** – tento rámec je kódován **vzhledem k jedinému předcházejícímu rámci typu I nebo P**.
- **B (Interpolated bi-directionally)** – tyto rámce jsou kódovány vzhledem k nejbližšímu **předchozímu** a **nejbližšímu** budoucímu rámci typu **I** nebo **P**. Jejich použití je nepovinné ale z hlediska dosahovaných kompresních poměrů výhodné. **Komplikace** z použití rámci B spočívá v uchovávání v paměti dva kotevní obrazy. Dále je nevyhnutelné jisté časové zpoždění, protože nejprve musí být k dispozici obraz **novější** a teprve potom může být kódován obraz starší.



Obr.7.5. Sekvence IBBPBBPBB...
v proudu MPEG rámci.

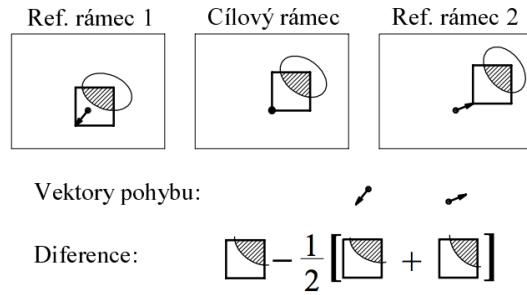


Obr. 7.4. Alternativní postup při kódování bloku v MPEG-2.

Velmi často používané řazení rámci je IBBPBBPBBI. Kódování rámci P a B taky probíhá po **makroblocích**. Pro každý makroblok v cílovém (tj. právě kódovaném rámci) jsou sestaveny **vektory pohybu** (pro každý makroblok v P **jeden** vektor, v B jsou vektory **dva**) vzhledem k referenčním rámci.

Vektor pohybu je definován takto: jestliže o uvedený vektor posuneme kódovaný makroblok a porovnáme s odpovídající částí referenčního obrazu, pak je dosaženo dobré shody. Vektory posunutí se stávají **součástí komprimované** sekvence. Po nalezení vektoru jsou **kódovány difference** – podobně jako u JPEGu mezi odpovídajícími makrobloky bude v dvou rámci **malý rozdíl** a výsledné data po kvantizaci vyžadují tak **malé sekvence** (komprese).

- Bloky **P** se kódují $T - R$, kde T je makroblok v cílovém rámci,
- bloky **B** se kódují $T - 0.5(R_1 + R_2)$, kde R_1 a R_2 jsou makrobloky v cílových rámci.

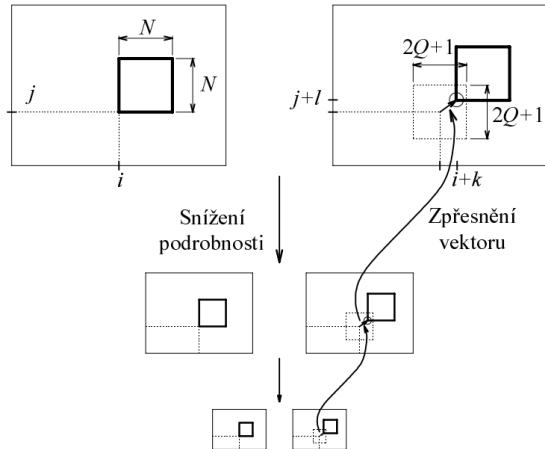


Obr.7.6. Kódování rámčů typu B.

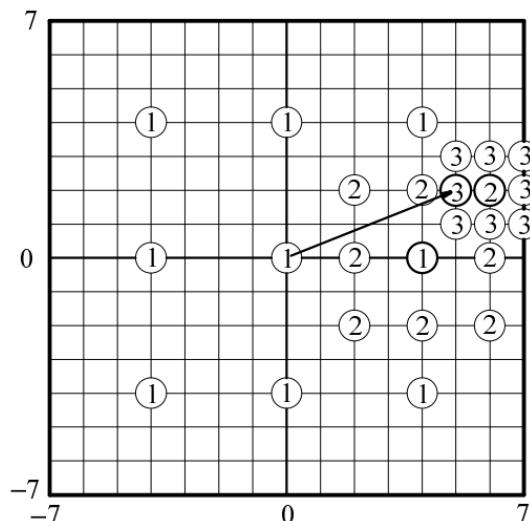
7.2.2 Stanovení pohybového vektoru

Stanovení pohybového vektoru MPEG norma nepředepisuje, jedná se však o jeden z **nejobtížnějších problémů**. Většinou se využívá pouze **jasové** (Y) složky, existuje několik metod:

1. **Porovnání makrobloků** – v referenčním rámci se nalezne makroblok, který nejvíce odpovídá zpracovávanému makrobloku. Pro určení shody lze použít např. SSD. Účinné, avšak velmi časově náročné.
2. **Logaritmické vyhledávání** – vylepšení předchozí metody, má logaritmickou časovou složitost. V prvním kroku algoritmus testuje 9 dvojic. V každém dalším kroku se testuje vždy po 8 dvojicích rozmístěných kolem předchozího bodu, který měl největší shodu. Rychlejší, ale nemusí být tak přesné jak předchozí metoda.
3. **Rekurzivní dělení** – rekurzivně dělíme obraz na menší a menší. Poté nalezneme prvotní odhad v nejmenším obrazu a se zvyšující se velikostí (návrat z rekurze) pozici vektoru jen upřesňujeme. Rychlé a spolehlivé.

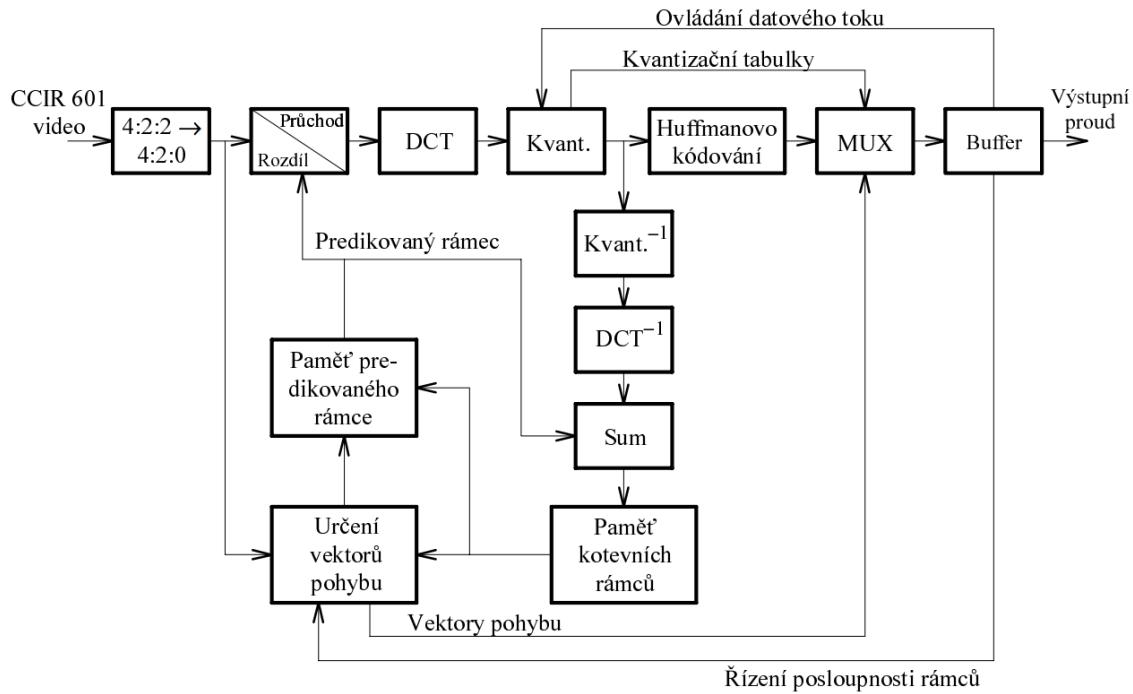


Obr. 7.8. Hledání vektoru pohybu s využitím obrazů s nižší podrobností.



Obr. 7.7. Stanovení pohybového vektoru logaritmickým vyhledáváním.

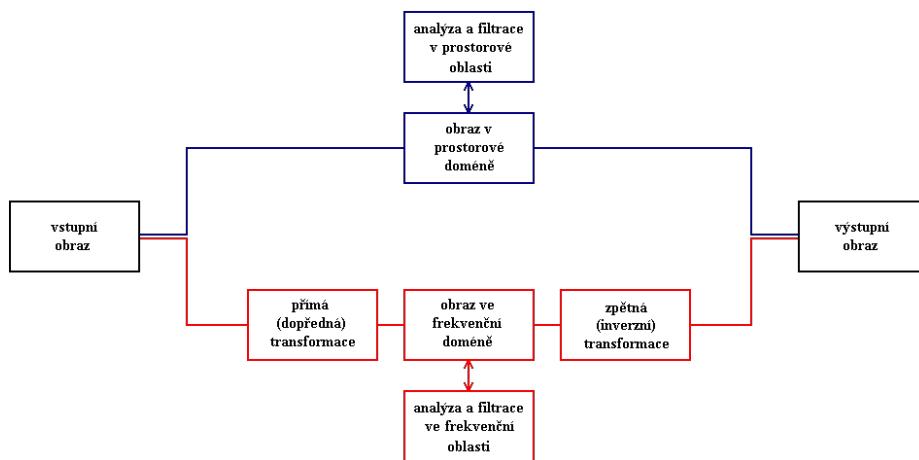
Kódování složek vektorů se provádí **inkrementálně** vzhledem k předchozí hodnotě. Zároveň může nastat situace kdy se vektor nepodaří nalézt, v tom případě je možné kódovat dané makrobloky nezávisle na ostatních (stejně jako I).



Obr. 7.10. Schéma MPEG enkodéru obrazu.

7.3 Principy úprav obrazu v prostorové a frekvenční doméně

Úpravy nad obrazy lze provádět v **prostorové** (obraz je reprezentován souřadnicemi x, y a hodnotou pixelu) nebo **frekvenční** (snímek je popsán harmonickými funkcemi (sin a cos) různé amplitudy, frekvence a fáze) doméně. Většina operací nad prostorem signálů je popsána **operátorem**, pro převod do frekvenční domény se poté používá (**Diskrétní Fourierova transformace**).



7.4 Operace nad prostorem signálů

- **Lineární operace** – úprava signálu je popsána operátorem \mathcal{O} , velmi často předpokládáme, že operátor \mathcal{O} má nějaké speciální vlastnosti. Platí: $\mathcal{O}\{af(x, y) + bg(x, y)\} = a\mathcal{O}\{f(x, y)\} + b\mathcal{O}\{g(x, y)\}$.
- **Operace invariantní vůči posuvu** – považujeme aby měl operátor kromě linearity ještě další vlastnosti (invarianci vůči posuvu). Operátor \mathcal{O} je invariantní vůči posuvu, pokud pro všechna $f(x, y)$ platí: $g(x - a, y - b) = \mathcal{O}\{f(x - a, y - b)\}$. Méně formálně: pokud provedeme operátor na dva vzájemně posunuté ale jinak shodné signály, jejich výsledek bude také vzájemně posunutý ale jinak **stejný**.
- **Dirakův impulz** – je impulzová funkce $\delta(x, y) = 0$, v bodě $(0, 0)$ je hodnota **nekonečno**, všude jinde nulovou, přičemž platí $\delta(-x, -y) = \delta(x, y)$, a **integrál** přes celou funkci je roven 1. Jinak řečeno se jedná o funkci, která je nekonečně vysoká a nekonečně úzká v bodě $(0, 0)$.

7.5 Konvoluce (lineární sumace bodových zdrojů)

Konvoluce představuje základní matematický **operátor**, který pracuje s dvěma funkcemi (značí se $*$). Umožňuje zpracování obrazových signálů jak v prostorové tak frekvenční doméně (konvoluční teorém). Spojitá konvoluce je definována takto:

$$f(x, y) * h(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(a, b) f(x-a, y-b) da db,$$

kde funkci h nazýváme **konvoluční jádro** (filtr), přesně řečeno se jedná o impulzovou charakteristiku filtru. Při výpočtu se funkce h v rovině x, y otočí o 180° (důsledek členů se zápornými znaménky). Konvoluční operátor má tyto **vlastnosti**:

- **Komutativní** – $f * g = g * f$.
- **Asociativní** – $f * (g * h) = (f * g) * h$.
- **Distributivní** – $f * (g + h) = (f * g) + (f * h)$.
- **Existence jednotky** – $f * \delta = \delta * f = f$, kde δ je dirakův impulz a $\delta(x) = 0, x \neq 0$.
- **Asociativita při násobení skalárem** – $a(f * g) = (af) * g = f * (ag)$
- **Konvoluční teorém** – $\mathcal{F}(f * g) = [\mathcal{F}(f)] \cdot [\mathcal{F}(g)] = F \cdot G$, kde $\mathcal{F}[f(x)]$ značí Fourierovu transformaci. Jinými slovy: Fourierovým obrazem konvoluce funkcí f, g je součin jejich Fourierových obrazů F a G .

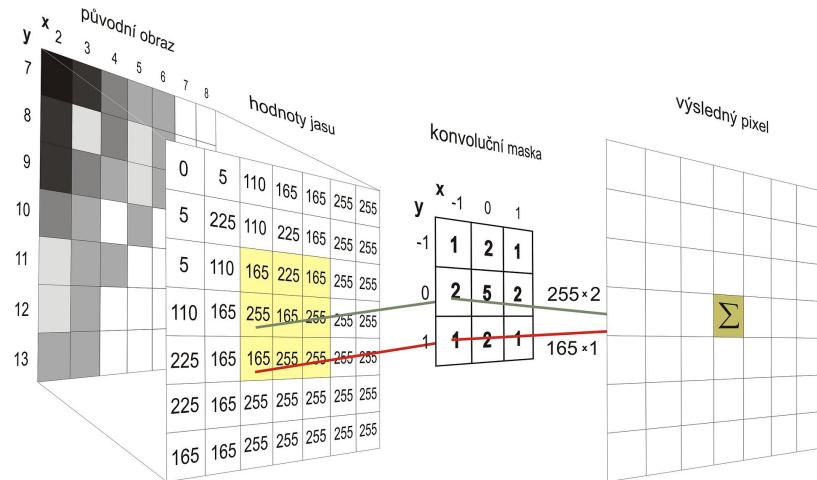
7.5.1 Diskrétní konvoluce

V počítačové grafice využíváme často pro zpracování dvourozměrného diskrétního signálu, je popsána tímto vzorcem:

$$(f * h)(x, y) = \sum_{i=-k}^k \sum_{j=-k}^k f(x-i, y-j) \cdot h(i, j)$$

V případě diskrétní konvoluce lze jádro chápat jako **tabulkou** (konvoluční masku), kterou přiložíme na příslušné místo obrazu. Každý pixel překrytý tabulkou vynásobíme koeficientem v příslušné buňce a hodnoty sečteme, ty uložíme na pozici středového pixelu. V případě, že koeficienty již nejsou normalizované, je nutné je **normalizovat** (zprůměrovat), např.: maska 3×3 pokud je všude 1 (uniformní filtr) tak výsledek je nutné vydělit 9.

Hodnoty konvoluční masky mají vliv na výsledek operace, používají se masky pro: **rozostření, zosolení, detekci hran, atd.**

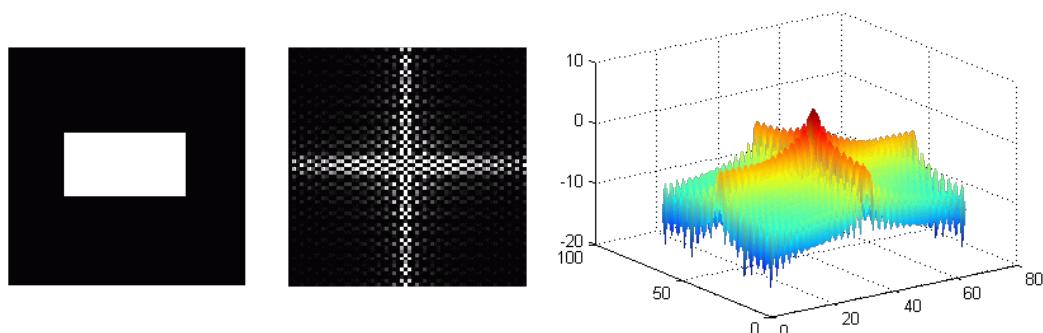


7.6 Fourierova transformace

Slouží pro převod obrazových signálů z prostorové do **frekvenční domény** (na komponenty složené ze sínů a cosínů). Pro analýzu **diskrétního** obrazu se využívá DFT, která je dáná vztahem:

$$F(k, l) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m, n) \frac{1}{\sqrt{MN}} \exp \left[-j2\pi \left(\frac{mk}{M} + \frac{nl}{N} \right) \right],$$

kde $k = 0, \dots, M-1$, $l = 0, \dots, N-1$. Symbol $F(k, l)$ značí frekvenční spektrum obrazu s souřadnicemi (k, l) , které nabývá hodnot od 0 - (M, N) . Frekvenční spektrum obsahuje **komplexní čísla**, proto se pro zpracování obrazového signálu ve frekvenční oblasti používají **amplitudy** komplexních čísel (amplitudová frekvenční charakteristika) nebo výkonová spektrální hustota (koeficienty násobené komplexně sdruženým číslem). Příklad amplitudové charakteristiky je na následujícím obrázku.



Tato frekvenční charakteristika obsahuje **koeficienty odpovídající různým frekvenčním složkám**. Analýzou a operacemi s koeficienty ve frekvenční oblasti lze **modifikovat obraz v prostorové oblasti** např. realizovat filtr typu **dolní propust pro vyhlazení obrazu**. Pro získání modifikovaného obrazu je třeba převést koeficienty zpět do prostorové oblasti. Tomuto procesu se říká zpětná nebo někdy **inverzní Fourierova transformace** IFT, jejíž diskrétní varianta IDFT je dána následujícím definičním vztahem:

$$f(m, n) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} F(k, l) \frac{1}{\sqrt{MN}} \exp \left[j2\pi \left(\frac{mk}{M} + \frac{nl}{N} \right) \right].$$

7.7 Filtrace ve frekvenční oblasti

Postup filtrace obrazů ve frekvenční doméně je následující:

1. aplikace diskrétní Fourierovy transformace (DFT) $f(x, y) \rightarrow F(u, v)$,
2. aplikace filtru $G(u, v) = H(u, v)F(u, v)$, kde:
 - $G(u, v)$... výslední snímek
 - $H(u, v)$... funkce filtrace (konvoluční maska?)
 - $F(u, v)$... původní snímek

inverzní Fourierova transformace (IDFT) $G(u, v) \rightarrow g(x, y)$.

7.7.1 Nízkofrekvenční, pásmový, vysokofrekvenční filtr

Bázové funkce jsou síný a cosíný s rostoucí frekvencí. Tedy $F(0, 0)$ (**střed obrázku**) reprezentuje DC složku, tedy **průměrný jas** napříč obrazem a $F(M - 1, N - 1)$ reprezentuje **nejvyšší frekvenci**, to lze pozorovat na amplitudové charakteristice výše.

Tedy **čím dále od středu, tím je frekvence vyšší**. Zároveň můžeme vidět, že **amplituda je menší pro vyšší frekvence**, nízké frekvence tedy obsahují více informací o obrazu než ty vyšší a obecně se jich ve většině obrazů nachází více (čím rychlejší změna jasu/gradient tím vyšší frekvence). Všech těchto znalostí můžeme využít k návrhu požadovaného filtru:

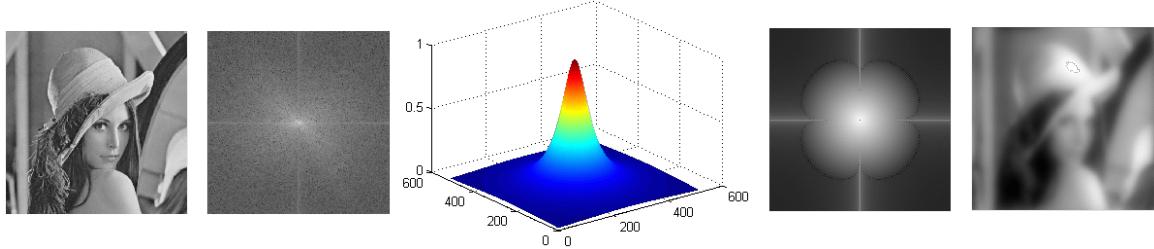
- **Nízkofrekvenční** (lowpass) filtr může například propouštět pouze frekvence kolem středu (tedy ty nízké) a ostatní vynulujem, tento filtr lze navrhnout pomocí binární masky (bílý kruh ve středu) → **minimalizujeme ostré hrany**.
- **Vysokofrekvenční** (highpass) filtr, postup je analogický k předchozímu, pouze tentokrát vynulujeme frekvence na středu a ty na krajích (vysoké) propustíme → **zvýrazníme hrany**.
- **Pásmový** filtr propouští pouze předem definované pásmo.

7.7.2 Butterworthův filtr

Často se pro filtraci ve frekvenční oblasti využívá Butterworthův filtr, který má ze všech běžných filtrů (Gaussian, Chebyshev, Bessel) **nejméně zvlněné frekvenční spektrum** a konverguje k nule u maximální frekvence. Je dán vztahem:

$$H(u, v) = \frac{1}{1 + \left(\frac{D(u, v)}{D_0}\right)^{\frac{2}{n}}}, \quad \text{kde } D(u, v) = \sqrt{(u - u_c)^2 + (v - v_c)^2}$$

kde $d(x, y)$ představuje běženě používanou L2-normu a n je řád filtru (nejčastěji 1 nebo 2).

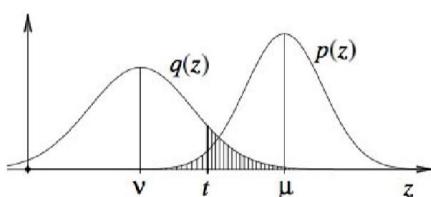


8 Základní metody úpravy a segmentace obrazu (filtrace, prahování, hrany).

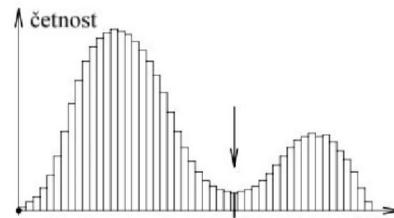
8.1 Prahování

- Cílem prahování je **oddělit pozadí od popředí** na základně stanoveného prahu (nějaká reálná hodnota). Výsledkem binární obraz (1 = objekt, 0 = pozadí).
- Práh může být buď **stejný** pro celý obrázek, anebo **adaptivní** pro jednotlivé části obrazu. Další možností je stanovit práh v **intervalu** a, b . Úspěšnost detekci oblastí závisí na správné hodnotě prahu.
- Pokud neznáme hodnotu prahu, snažíme se jí stanovit na základně informací získaných z obrazu, který má být **segmentovaný**.
- **Bimodální histogram** (dva kopce), **multimodální histogram** – práh určit jako **minimum histogramu** mezi vysokými hodnotami, pak lze dále rekurzivně dělit (předpokládáme, že v obrazu jsou převážně dva a více druhů pixelů).
- **Obraz lze rekurzivně dělit** na menší části, ve kterých se vypočte histogram a dle něho určí práh pro konkrétní část (pokud nelze práh určit, lze ho interpolovat pomocí sousedních prahů).
- **Minimalizace rizika chyby**:
 - stanovení prahu tak, aby se minimalizovala špatná detekce,
 - stanovení dle approximace normálních rozdělení popředí a pozadí

$$\varepsilon = \theta P(t) + (1 - \theta)[1 - Q(t)].$$
 - nejlepších výsledků lze dosáhnout v **extrému první derivace**
 - pokud je zastoupení pozadí a popředí stejné a má stejný rozptyl $(t - \mu)^2 = (t - v)^2$.



Obr.8.23. Stanovení prahu minimalizací chyby.



Obr.8.22. Bimodální histogram jasu.

- Na levém obrázku je práh označen t a vyšrafováná oblast značí chybu, která nastane při prahování, kdy bude špatně rozpoznané popředí/objekt $q(z)$ a pozadí $p(z)$ – minimalizace chyby.

8.2 Detekce hran

- Každá oblast je obklopena hranicí.

- Hranice se skládá z hran (případně také z jediné zakřivené hrany).
- Hrana se skládá z jednotlivých hranových bodů.
- Většinou se postupuje tak, že se obraz převede do stupně šedi a následně se naleznou jednotlivé body hran.
- Za bod hrany se často považuje místo, kde průběh jasu **vykazuje náhlou změnu**, případně **inflexní bod**.
- Po nalezení jsou jednotlivé nalezené body hran spojovány různými technikami do hran a celých hranic.

8.2.1 Detekce hran s využitím gradientu

- Hrana je v obrazu zastoupeny (prudkou) **změnou jasu**, lze ji tedy najít zkoumáním síly a směru gradientu v jednotlivých bodech.
- Pro určení směru gradientu či hrany (**směr gradientu je kolmý ke směru hrany**) je třeba provést **derivaci** (nejlépe v x i y), která je při výpočtu nahrazena diferencí.
- Diference může být buď **centrální** nebo **dopředná/zpětná**.

$$d_x = \frac{I(x-1, y) - I(x+1, y)}{2}, \quad d_y = \frac{I(x, y-1) - I(x, y+1)}{2}.$$

- Velikost hrany lze určit velikostí gradientu (norma), hrana je tam, kde $e >$ práh. (hrana je kolmá k gradientu)
- $$e(x, y) = \sqrt{(f_x(x, y)^2 + f_y(x, y)^2)}$$
- Směr hrany a gradientu lze určit (kde φ – směr gradientu, ψ – směr hrany)

$$\varphi(x, y) = \arctan \left[\frac{f_y(x, y)}{f_x(x, y)} \right], \quad \psi(x, y) = \varphi(x, y) + \frac{\pi}{2}.$$

- Výše uvedené derivace lze nahradit **konvolučními maskami**

- **Sobel** – vážený průměr (Prewittove dělá pouze normální)
- **Kirsch** – počítání hran v 8 směrech

- Robertsův operátor:

$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}.$$

- operátor Previttové:

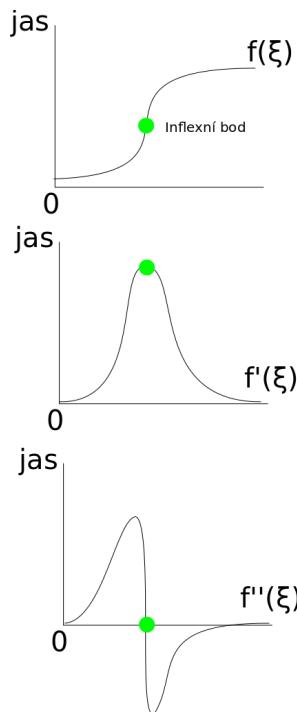
$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

- Sobelův operátor:

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

- Kirschův operátor:

$$\begin{bmatrix} -5 & -5 & -5 \\ 3 & 0 & 3 \\ 3 & 3 & 3 \end{bmatrix}.$$



Obrázek 2: Velikost gradientu a jeho první a druhá derivace

8.2.2 Detekce hran hledáním průchodu nulou

- První derivace obrazové funkce nabývá svého maxima v místě hrany.
- **Druhá derivace protíná v místě hrany nulovou hodnotu.**
- Spolehlivější metoda, než hledání maxima v první derivaci. **NE** v případě, že je obraz postižen šumem. V tomto případě selhává, jelikož druhá derivace ještě více zesílí šum.

Laplaceův operátor (druhá derivace gradientu)

- Pro výpočet se používá symetrická difference nebo konvoluční masky (na krajích je maska ořezaná)

$$d_x = I(x-1, y) - 2I(x, y) + I(x+1, y),$$

$$d_y = I(x, y-1) - 2I(x, y) + I(x, y+1).$$

0	1	0
1	-4	1
0	1	0

a)

0	1	0
1	-3	1
0	0	0

a)

Obr.8.6. Konvoluční masky pro výpočet Laplaceova operátoru.

0	1	0
0	-2	1
0	0	0

b)

Obr.8.7. Příklady masek pro body na krajích a v rozích obrazu.

- Hrana je detekována jako **změna znaménka v průchodu mezi dvěma extrémy**.
- Je **více citlivý na šum než první derivace** (i při malém šumu je detekováno množství falešných hran).
- Pro redukci šumu a zahlazení vysokých frekvencí lze použít **Gaussův operátor**:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 1 & 2 \\ 1 & 2 & 1 \end{bmatrix}.$$

8.2.3 Cannyho detekce hran

Canny první stanovil požadavky, které by měl detektor splňovat a následně navrhl detektor. Požadavky:

- **Minimalizovat** pravděpodobnost **chybné detekce**.
- Najít polohu hrany, co **nejpřesněji**.
- Bod hrany identifikovat **jednoznačně**.

Postup:

1. Eliminace šumu Gaussovým filtrem.
2. Velikost a směr gradientu – nejčastěji Sobelův operátor (nebo centrální derivace).
3. Nalezení lokálních maxim a stanovení interpolace v osmi okolí. **Redukce na hranu velikosti 1 px**.
4. Eliminace nevýznamných hran (**double thresholding**)
 - Všechny body, kde je velikost hrany $\leq t_{high}$ – „jistá“ hrana
 - Pak ty, které jsou $> t_{low}$ a sousedí s hranou – „jistá“ hrana

8.3 Filtrace

Rozlišujeme 2 základní druhy filtrů **rekurzivní** a nerekurzivní. Nerekurzivní i rekurzivní filtrace obrazu je lineární filtrace.

8.3.1 Nerekurzivní filtry

- Vytváří výstupní signál v každém bodě jako **lineární kombinaci** vzorků vstupního signálu.
- Jsou to vlastně **konvoluční filtry**. Jedná se o úpravu signálu (prostorová doména) pomocí předpisu:

$$g(m, n) = f(m, n) * h(m, n),$$

kde $f(m, n)$ je původní signál a $h(m, n)$ je **impulsová charakteristika filtru**.

- Přepis při Fourierové transformaci – **frekvenční doména**: $G(k, l) = F(k, l) \cdot H(k, l)$.
- Tyto filtrace obrazu jsou **stabilní**.
- Gaussův filtr, uniformní, atd.

8.3.2 Rekurzivní filtry

- U rekurzivní filtrace je výstupní signál g **svázán se vstupním signálem** f prostřednictvím vztahu:

$$b(m, n) * g(m, n) = a(m, n) * f(m, n),$$

kde $a(m, n)$ a $b(m, n)$ jsou **diskrétní funkce**, které popisují rekurzivní filtr.

- U rekurzivní filtrace obrazu **bývá nižší časová náročnost** než u nerekurzivní filtrace, ale **nebývá vždy stabilní**, proto je potřeba stabilitu sledovat.
- Když je filtr nestabilní, mohou se šířit čím dál vyšší zaokrouhlovací **chyby** a **šumy**.
- S iteračním výpočtem můžeme využít následujícího předpisu:

$$gi(m, n) = a(m, n) * f(m, n) + c(m, n) * gi - 1(m, n),$$

kde: $c(0, 0) = 0$ a $c(m, n) = -b(m, n)$.

8.3.3 Inverzní filtr

- Jedná se prakticky o pouhou **dekonvoluci**. Máme-li rozmazený (symetricky, v daném směru, ...) obraz bez šumu, pak lze obraz rekonstruovat zpět do podoby před rozmazením.
- Nevhodné pokud byl obraz postižen šumem, **šum se inverzním filtrem zvýrazní**.
- Inverzní filtr **lze použít i pro detekci šumu**. Pracuje ve **frekvenční doméně** (po DFT).

- Pokud je signál zkreslen signálem $H(x)$ pak stačí jen hodnotu převrátit a získáme původní hodnotu $F(x) : F(x) = \frac{1}{H(x)}$.
- Obnovený obraz inverzním filtrem má rozmazané původně ostré hrany.



8.3.4 Wienerův filtr

- Jedná se v podstatě o inverzní filtr, který lze použít i na **zašumělý obraz**.
- Slouží pro rekonstrukci/opravu obrazu, který je poničen šumem, který **má odhadnutelné statistické vlastnosti** (tedy známe přibližně šum nebo konvoluci, kterou byl obraz poničen).
- Většinou neznáme cílový obraz abychom si mohli stanovit střední hodnotu.
- Úloha se vyjádří jako **optimalizace řešením předurčené soustavy lineárních stochastických rovnic** → **minimalizace chyby**.

$$e^2 = \epsilon \{(f(i,j) - \hat{f}(i,j)^2\},$$

kde ϵ označuje operátor střední hodnoty.

- **Praktické použití:** Hubbelův teleskop (v době kdy měl poškozené zrcadlo), Rozpoznaní automobilových značek (rozmazání pohybu – známá rychlosť).



Vlevo: Obraz rozmazaný pohybem o 5 pixelů ve směru osy x .

Vpravo: Výsledek restaurace Wienerovým filtrem.

9 Základní metody rozpoznávání objektů (příznakové rozpoznávání).

9.1 Příznakové rozpoznávání

- Založeno na tom, že každý objekt lze popsat množinou vhodných číselných hodnot = **příznaků**. Těch může být i více a potom tvoří **vektor příznaků**.
- Vektor příznaků nese všechny podstatné informace o objektu a je **jedinou informací** pro následné **rozpoznání**.
- Počet příznaků a jejich typy bývá **obtížné** určit, je nutné provést experimenty. **Velikost** vektoru příznaků by měla být rozumná (ne příliš velká).
- Obecně by zvolené příznaky měli **splňovat** následující:
 - hodnoty příznaků jsou podobné pro objekty stejných tříd a odlišné pro objekty jiných tříd,
 - měly by být pokud možno nezávislé na hodnotách jiných příznaků.

9.1.1 Momenty

Momenty **různého stupně** patří k často používaným příznakům. Důvodem je, že jejich schopnost od sebe rozlišit objekty různých tříd často vyhovuje a jejich **výpočet je snadný**. Moment vztažený k souřadné soustavě obrazu:

$$m_{p,q} = \int \int_{\Omega} x^p y^q f(x, y) dx dy,$$

kde $f(x, y)$ je obrazová funkce, p, q udávají **stupeň** momentu a Ω je ta část obrazu, kterou považujeme za rozpoznávaný objekt. U diskrétních obrazů se nahrazuje **sumou**. Hodnoty p, q lze zvolit $p \geq 0, q \geq 0$, přičemž: $m_{0,0}$ – **plocha objektu** vážená hodnotou jasu. Máme 3 základní typy momentů:

- **invariantní vůči posunu** (nezávislá na poloze objektu) - $m_{p,q} \rightarrow \mu_{p,q}$,
- **invariantní vůči měřítku** (nezávislá na velikosti objektu, normalizace) - $m_{p,q} \rightarrow v_{p,q}$,
- **invariantní vůči rotaci** dána momentovými invarianty - sestavení θ_n ,

Výpočet polohy těžiště (nezávislost na poloze)

K výpočtu polohy těžiště objektů lze použít momentů $m_{0,0}, m_{1,0}, m_{0,1}$, což dává:

$$x_t = \frac{m_{1,0}}{m_{0,0}}, \quad y_t = \frac{m_{0,1}}{m_{0,0}}.$$

Také **poloha objektu** reprezentována jeho těžištěm může být sama o sobě použita jako **příznak** (x_t, y_t). Momenty vztažené vzhledem k těžišti, které je činí **nezávislé na poloze objektu** avšak stále závislé na **velikosti a rotaci** objektu, lze vypočítat:

$$\mu_{p,q} = \int \int_{\Omega} (x - x_t)^p (y - y_t)^q f(x, y) dx dy.$$

Momenty nezávislé na velikosti

Pokud není pro rozpoznání důležitá velikost objektu, je možné použít **normalizovaných momentů** $v_{p,q}$ (normalizace pomocí součtu jasových hodnot objektu):

$$v_{p,q} = \frac{\mu_{p,q}}{(m_{0,0})^\gamma}, \quad \text{kde } \gamma = \frac{p+q}{2} + 1.$$

Momenty nezávislé na natočení

Pokud nemá rozpoznání záležet ani na orientaci (natočení), je nutné momenty počítat k hlavním osám objektu. Hlavní osy vytvářejí souřadnou soustavu, která má počátek v těžišti rozpoznávaného objektu a vzhledem k souřadné soustavě obrazu je pootočena o úhel Θ . Pootočení θ je dáno podmínkou, aby momenty $\mu'_{2,0}$ a $\mu'_{0,2}$ vypočítány k hlavním osám byly extrémy. Opět lze použít i samotný úhel pootočení jako **příznak**.

9.1.2 Pravoúhllost a podlouhlost

Hranici objektu **postupně rotujeme** v rozsahu $0 - 90^\circ$. Krok úhlu rotace zvolíme v jednotkách stupňů (např. 5°). Po provedení rotace opíšeme kolem objektu pravoúhelník, jehož strany jsou rovnoběžné se stranami obrazu (po provedení rotace stačí nalézt extrémy hranice). Ze všech pravoúhelníků vybereme ten, který má **nejmenší plochu** A_O , a délky jeho stran značíme postupně A_R , a , b . Pravoúhllost (R) a podlouhlost (S) je definována následovně:

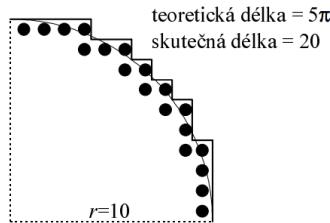
$$R = \frac{A_O}{A_R}, \quad S = \frac{a}{b}.$$

9.1.3 Kruhovost

Nechť P , A označují délku hranice objektu a jeho plochu, kruhovost je pak definována vztahem:

$$C = \frac{P^2}{A}.$$

Pro kruh je $C = 4\pi$, pro čtverec $C = 16$, pro objekty nepravidelného tvaru jsou hodnoty vyšší. Tyto hodnoty jsou však teoretické a skutečná hodnota se může lišit (viz obr).

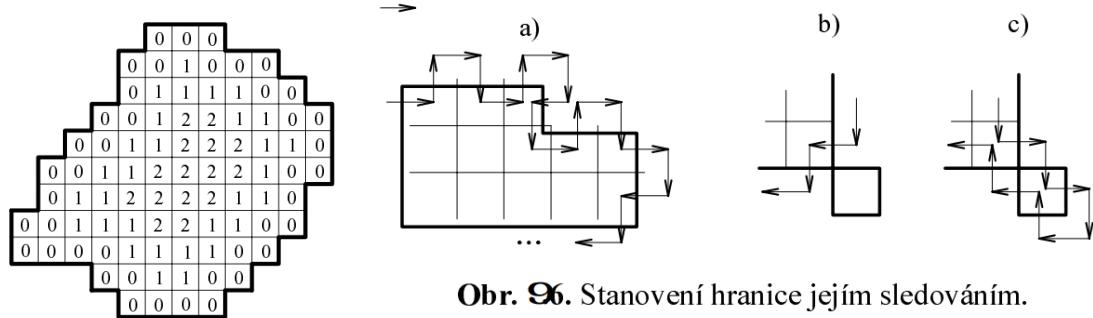


9.1.4 Průměrná vzdálenost pixelu od hranice

Nechť d_i je vzdálenost i -tého pixelu od hranice objektu, jehož plocha je A (vzdáleností rozumíme, kvůli rychlosti, pouze počet řad pixelů, které leží mezi pixelem a hranicí). Průměrnou

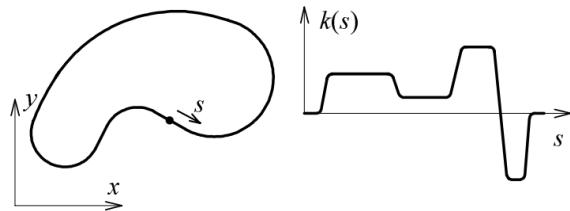
vzdálenost μ_d a koncentrovanou informaci o tvaru objektu S získáme jako:

$$\mu_d = \frac{1}{N} \sum_{i=0}^N d_i, \quad S = \frac{A}{\mu_d^2}.$$



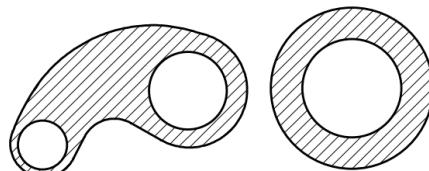
Obr. 9.6. Stanovení hranice jejím sledováním.

9.1.5 Průběh křivosti hranice objektu



9.1.6 Eulerovo číslo

Nechť C je počet navzájem nesouvislých částí rozpoznávaného objektu a H je počet dér v objektu pak Eulerovo číslo je rozdíl $C - H$, jelikož však většinou detekujeme jeden souvislý objekt, rovnice bude rovna $1 - H$.



Obr. 9.4. Objekt se dvěma nesouvislými částmi a třemi děrami ($C=2, H=3$).

9.1.7 Atributy odvozené z histogramu jasu

Možné použití, pokud rozpoznáváme objekty, pro něž je charakteristické jisté rozložení jasu po ploše objektu, nebo pro objekty pokryté texturou. Pokud je N počet pixelů plochy objektu, potom normalizovaný histogram (vydělíme počtem pixelů plochy), v podstatě představuje **pravděpodobnost** toho, že se na daném místě nachází pixel s danou hodnotou. S

využitím pozorování, že funkci $p(b)$ (pravděpodobnost, že tam bude pixel) lze interpretovat jako hustotu pravděpodobnosti, lze jako **příznaky** využít **střední hodnotu, varianci, šíkmost, energie, entropie**.

9.1.8 Atributy odvozené z frekvenčního spektra jasu

Hodí se např. když má objekt nějakou **texturu**. Provede se furierova transformace nad oblastí, kterou objekt zaujímá (pokud je objekt třeba kulatý, vyplníme jej menšími obdelníky a na každý uděláme furiera). Jednotlivá data frekvenčního spektra mohou sloužit jako příznaky, je však žádoucí je nějakým způsobem redukovat – zapisujeme počet frekvencí, která se nachází v nějakém definovaném **rozsahu**.

9.2 Příznakové metody analýzy obrazu (klasifikátory)

Klasifikátor nazveme zobrazení $d : X \rightarrow \Omega$, která každému vektoru příznaků (X) přiřadí identifikátor třídy, do které dané objekt náleží. Rozlišujeme několik základních druhů klasifikace:

1. **Klasifikace pomocí diskriminačních funkcí** – předpokládáme že existuje n reálných diskriminačních funkcí $g_1(x), g_2(x), \dots$ definovaných nad X . Rozpoznání je v tomto případě založeno na stanovení funkce dávající pro zadaný vektor příznaků **máximální hodnotu**. **Problém**: nalezení těchto funkcí →, lze pomocí rozdělení hustoty pravděpodobnosti.

Nejjednodušším tvarem diskriminační funkce je funkce lineární, která definuje váhy (a) pro každý příznak:

$$g_r(x) = a_{r,0} + a_{r,1}x_1 + a_{r,2}x_2 + \dots + a_{r,m}x_m$$

2. **Klasifikace pomocí minimální vzdálenosti (etanoly)** – etanol e_r je vektor, který reprezentuje nějakou třídu příznaků. Tento vektor lze nejjednodušeji získat jako **průměr** všech ostatních příznaků dané třídy trénovací množiny. Klasifikaci objektu do dané třídy pak lze získat vypočítáním a nalezením **nejmenší Euklidovské vzdálenosti** mezi vektorem příznaků objektu a jednotlivými etanoly.

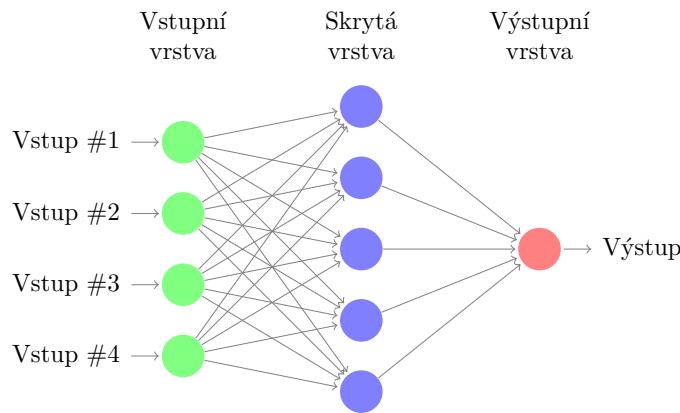
9.2.1 KNN - K-nearest neighbours

- velmi rychlý, jednoduchý (lze použít i jiné shlukovací algoritmy pro vytváření jednotlivých **etanolů**),
- pomocí **k nejbližších sousedů** určíme label dotazovaného ,
- učení je velmi rychlé (jen se uloží data do struktur),
- určení se zpomaluje se zvyšujícím se k.

9.2.2 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 třídou**, 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 3.



Obrázek 3: 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 $[-1, 1]$.

Učení:

1. Jednotlivé neurony jsou navzájem propojeny synapsemi (**váhami**), které na začátku náhodně nastřelíme (v intervalu $\angle -0.5, 0.5$).

2. Poté neuronové sítě podáváme trénovací data (vstupní příznaky a odpovídající výstup), která postupně **upravuje váhy** jednotlivých synapsí, aby dávala správný výsledek a minimalizovala chybu.
3. Jednou z metod minimalizace chyby je učení „**back propagation**“, což představuje hledání minima **gradientní metodou**.

Další vlastnosti:

- 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 předešlé vrstvy, sečteny s určitými váhami a hodnotou zkreslení. (Suma těchto hodnot je transformována pomocí aktivační funkce, může se lišit pro různé neurony)

9.3 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 (tentokrát může být přeskoven, dle Dalal a Triggs → předzpracování má malý vliv na výkon)
2. obraz se **rozdělí** na malé prostorové oblasti (buňky, například 8×8 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 (binách) po 20° ; rozsah 0° – 180°) → výjde nám vektor o velikosti 9
4. buňky spojíme do větších **propojených bloků** 16×16 z důvodu normalizace osvětlení a kontrastu. Pro chodce se používá L2-norm normalizace, dle vztahu (6) → vznikne vektor velikosti $9 \times 4 = 36$ (čtyři 8×8 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). Rozdelení do rozsahu 0 – 180° proto, že se jedná o bezznaménkové gradienty (unsigned)

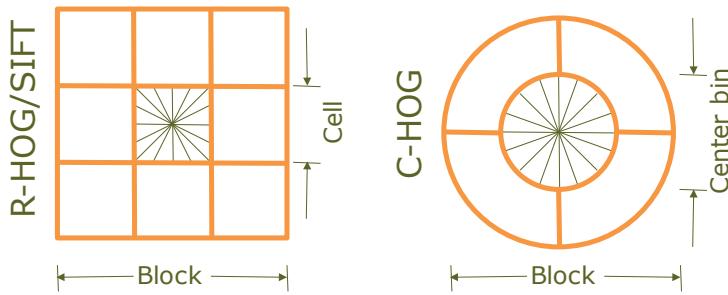
a bylo dokázáno, že fungují lépe než znaménkové (signed) $0\text{--}360^\circ$. Některé implementace HOG umožní určit, zda chceme používat signed gradienty.

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

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 rozdelenou 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 4: Varianty geometrie spojení bloků

9.4 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 různými urovněními 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

9.5 Haarový 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 7), kde každý bod x představuje součet hodnot předchozích pixelů doleva a nahoru. Spodní pravý bod obsahuje součet všech pixelů v obrazu. 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 5: Vstupní obraz

1	2	3
2	4	6
3	6	9

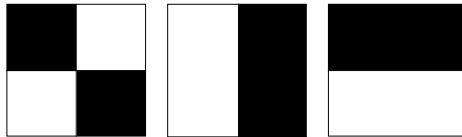
Obrázek 6: Integrální obraz

Obrázek 7: 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:

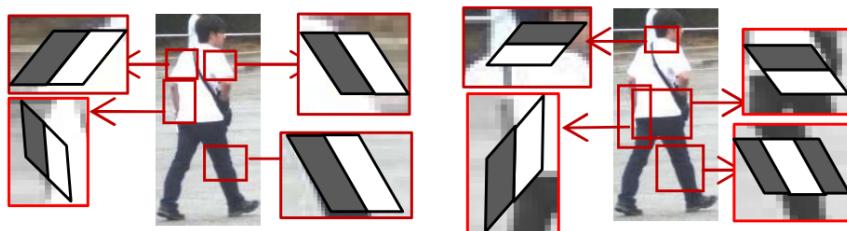
- Oční oblast je tmavší než oblast nosního mostu,
- hlava člověka je tmavší než její okolí,
- oblast mezi dolními končetinami je světlejší než samotné nohy.

Sada Haarových vlnek je na obrázku 8, jedná se pouze o základní sadu příznaků.



Obrázek 8: 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 9 je příklad detekce pomocí Haar-like vlnek. **Hodnota příznaku je rozdíl mezi sumou hodnot pixelů v bílé a černé oblasti Haarových vlnek.**



Obrázek 9: Použití Haar-like příznaků na chodcích

9.6 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-dimensionální vektor příznaků)

Matematicky lze LPB vyjádřit jako:

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

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 13. 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

1	0	0
1		0
1	1	1

1	2	4
128		8
64	32	16

Obrázek 10: *

Vstupní buňka

Obrázek 11: *

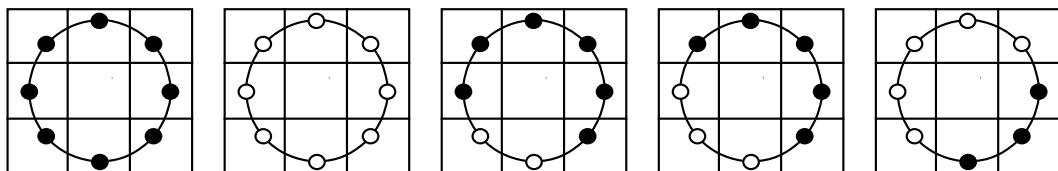
Prahové hodnoty

Obrázek 12: *

Pixely ohodnoceny váhou

Obrázek 13: 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 19). Pro parametr $P = 8$, získáme 59 vzorů.



Obrázek 14: *

Bod

Obrázek 15: *

Bod/Plocha

Obrázek 16: *

Křivka

Obrázek 17: *

Roh

Obrázek 18: *

Hrana

Obrázek 19: Lokální okolí LBP metody

9.7 Klasifikátory

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

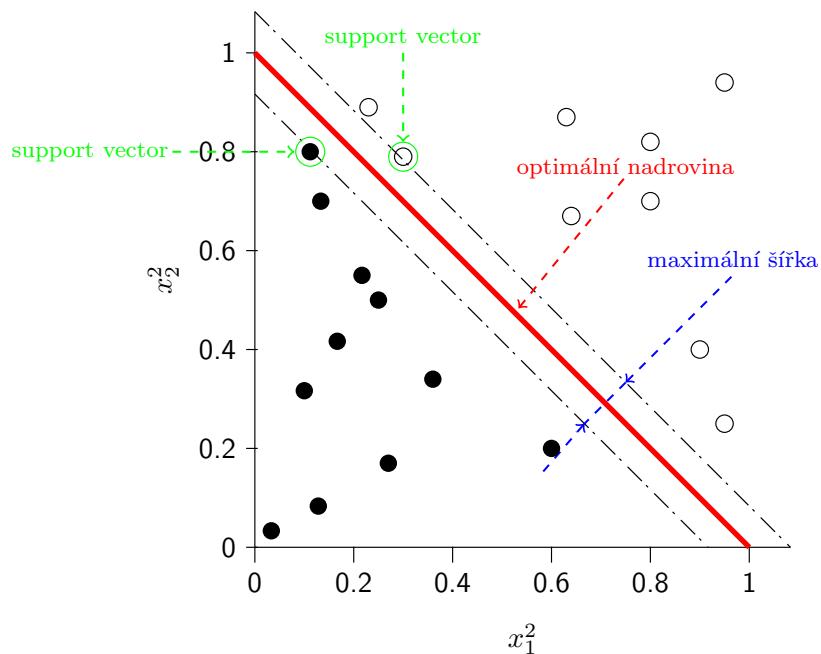
SVM Support vector machines

- 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 20. 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 20: 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 oddelení tříd pro n -tříd ($n > 2$) s postihovým multiplikátorem C , pro odlehle hodnoty ($C > 0$).
- **ν -Support vektorová klasifikace (ν -SVC)** – n -třídní klasifikace s možností nedokonalé separace. Tato klasifikace přidává nový parametr $\nu \in (0; 1)$, 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.
- **ε -Support vektorová regrese (ε -SVR)** – Vzdálenost mezi vektory příznaků a rozdělovací nadrovinou musí být menší než mez tolerance ε . Pro odlehle hodnoty opět použijeme multiplikátor C . Musí tedy platit: $C > 0$ a $\varepsilon > 0$.
- **ν -Support vektorová regrese (ν -SVR)** – Tato klasifikace je podobná jako ε -SVR. Na místo ε se použije parametr $\nu \in (0; 1)$.

Účinnost SVM závisí na výběru správného jádra a jeho parametrů. Často se používá Gaussovo jádro s jedním parametrem γ . 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 nadrovinu 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 χ^2 je podobné RBF jádrum a využívá se převážně na histogramy

$$K(x_i, x_j) = e^{-\gamma \chi^2(x_i, x_j)}, \chi^2(x_i, x_j) = \frac{(x_i - x_j)^2}{(x_i + x_j)}, \gamma > 0,$$

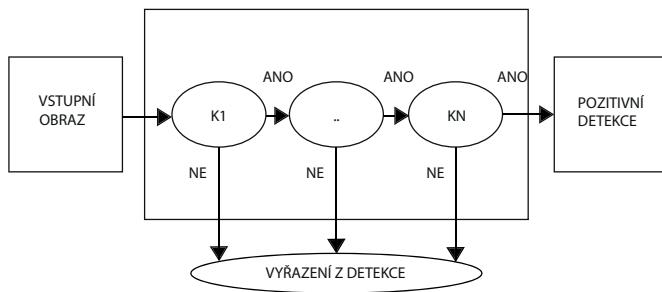
- **Jádro histogramu průsečíků** – Toto jádro je také známé jako *Min Kernel*, jedná se o nejnovější jádro v této knihovně a je velmi rychlé a užitečné při klasifikaci

$$K(x_i, x_j) = \min(x_i, x_j).$$

9.7.1 Kaskádové klasifikátory

- Skládá z více slabších klasifikátorů umístěných v **kaskádách** za sebou.
- Požadavky na tento druh klasifikátoru byly **rychlosť** detekce, aby mohl být implementován na procesorech s nižším výkonem. (v kamerách, v telefonech..)
- Klasifikátory si mezi sebou **předávají všechny** informace o vstupním obrazu (může se redukovat čas, nutný pro detekci v daném obrazu).
- 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 21, kde K_1-K_N je klasifikátor první až n -té vrstvy.



Obrázek 21: Ukázka pipeline kaskádového klasifikátoru

9.7.2 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. Vahou nula jsou ohodnoceny klasifikátory, které mají přesnost 50%. Pouze ty, které mají přesnost vyšší než 50%, jsou přínosné do této kombinace a můžeme hovořit o zesílení (boosting) klasifikace.

9.8 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 zdůvodnění nízké odolnosti proti variabilitě Změna velikosti, pozice nebo tvaru objektu významně ovlivňuje výsledky metody.

9.8.1 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.

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

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.

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

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.

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

V reálných aplikacích se však tato metoda většinou nepoužívá. Přestože je relativně výpočetně nenáročná, tak vzhledem k její nespolehlivosti v případech, kdy se v ob-

rázku 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.

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

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 nedostatkami. Při vyhledávání jsou výskyty vzorků ve vstupním obrazu nuteny zachovat orientaci referenčního obrázku. Zároveň je velice neefektivní a časově náročné počítat korelací mezi templatem a vstupním obrazem pro obrazy středních a vyšších rozlišení.

9.9 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.

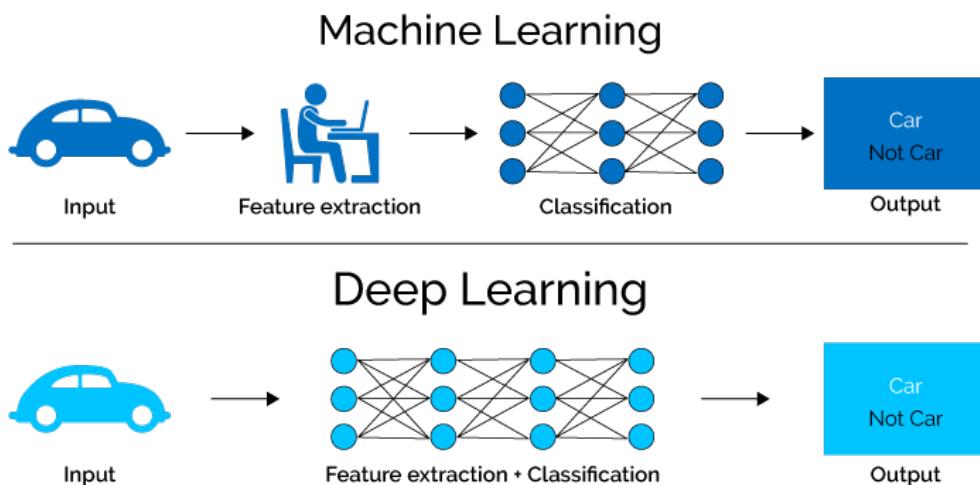
Hluboké strukturované učení může být:

- **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. Při 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 22) 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.



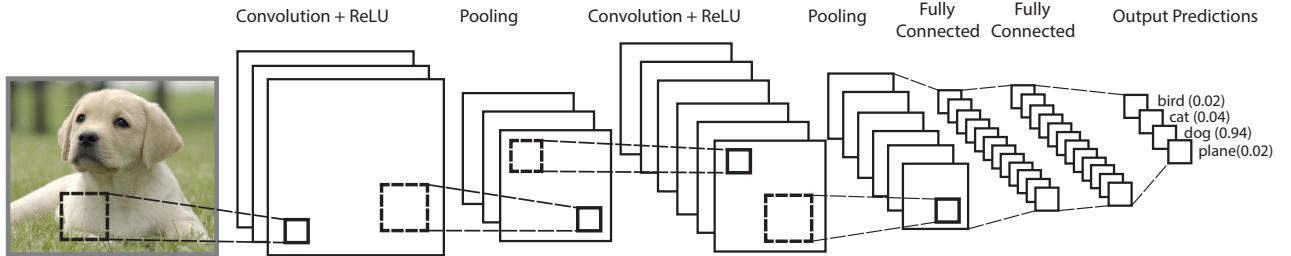
Obrázek 22: Hlavním rozdílem mezi strojovým a hlubokým učením je ten, že u strojového se příznaky musí extrahouvat manuálně.

9.9.1 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.
- **Nelinearity (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 23: Řetězec LeNet konvoluční neuronové sítě

9.10 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