3D počítačová grafika na PC

http://cgg.ms.mff.cuni.cz/

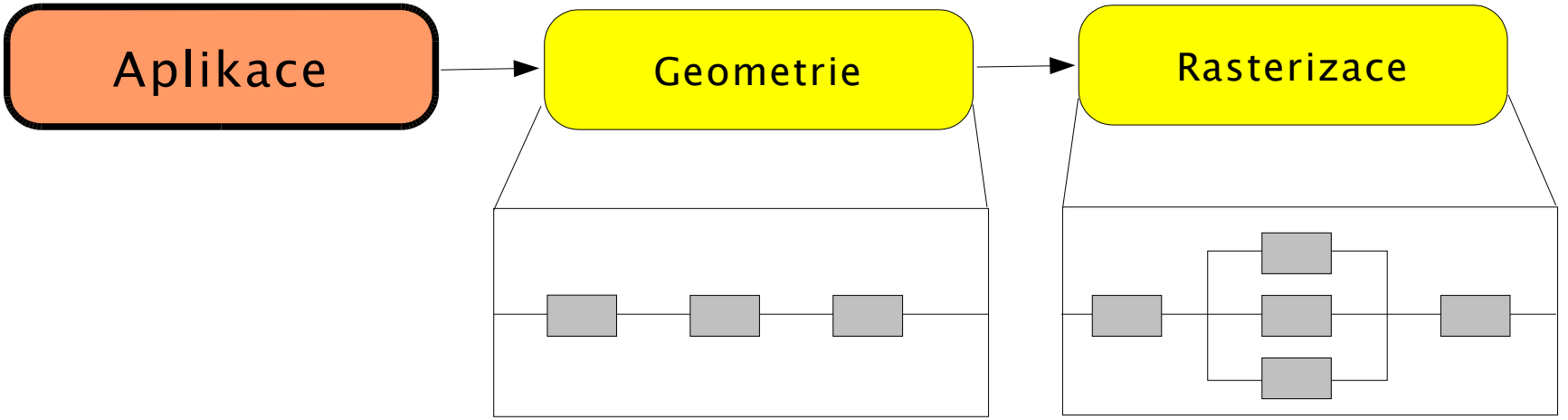
Pokroky v hardware

- **3D akcelerace** běžná i v konzumním sektoru
- zaměření na **hry**, **multimedia**
- **vzhled** – kvalita prezentace
  - velmi důmyslné techniky texturování
  - kombinace mnoha textur, modularita zpracování
- vysoký **výkon**
  - nejmodernější čipové technologie pro výrobu GPU (0.13 μm), **masivní paralelismus**
  - velmi rychlé **paměti** (dvoucestný přístup, DDR)
  - výjimečná sběrnice mezi GPU a CPU – **AGP**

Pokroky v software

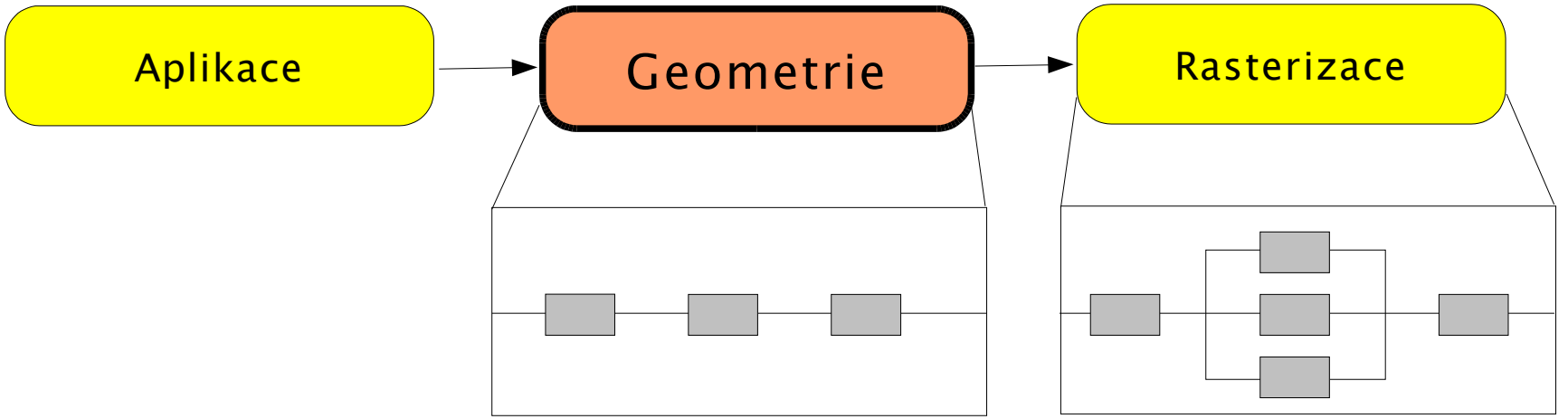
- dvě hlavní **knihovny** pro 3D grafiku
  - **OpenGL** (SGI) a **Direct3D** (Microsoft)
  - přístup je podobný, API je velmi ovlivněno hardwarem
- nastavování parametrů a **přenos dat** do GPU
  - maximální sdílení společných datových polí
- **programování grafického řetězce !**
  - revoluce v programování 3D grafiky
  - „***vertex-shaders***": zpracování vrcholů sítě
  - „***fragment-shaders***" („*pixel-shaders*"): zpracování jednotlivých pixelů před vykreslením

Řetězec zpracování 3D grafiky I



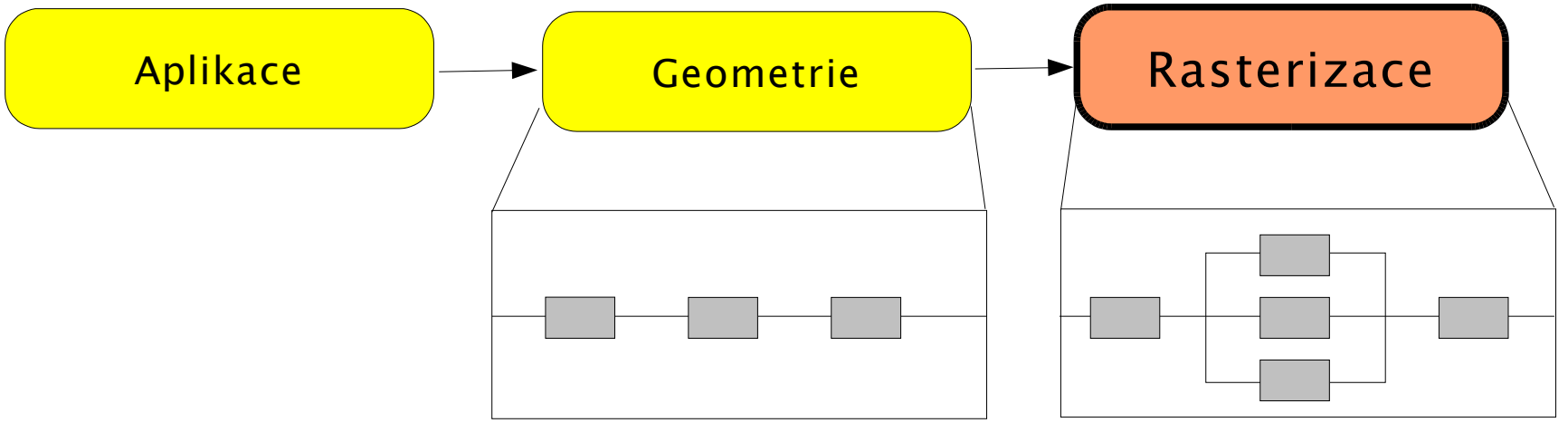
- **Aplikace**
  - **reprezentace 3D dat** (virtuální uložení na disku i v paměti), parametrizace, šablony, ..
  - **chování objektů**: fyzikální simulace, umělá inteligence
  - **interakce** mezi objekty: kolize, deformace, ..

Řetězec zpracování 3D grafiky II



- **Geometrie** („*HW T&L*”)
  - **modelové transformace** (pomáhají aplikační vrstvě)
  - **projekční transformace** (perspektiva), **ořezávání**
  - **výpočet osvětlení** (příp. jen příprava jistých vektorů..)
  - dlouhý řetězec (pipeline), řetězců může být více

Řetězec zpracování 3D grafiky III

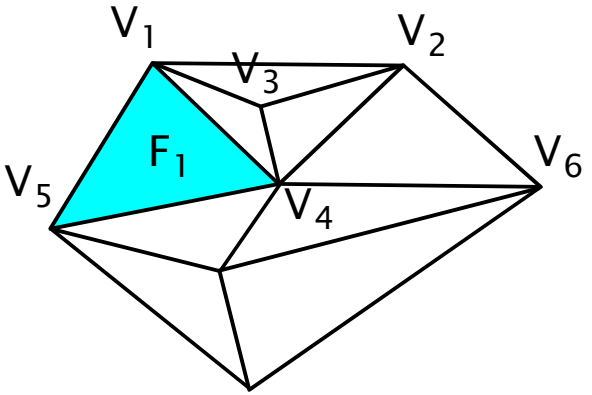


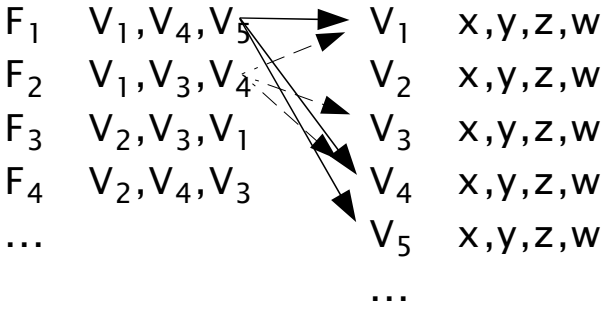
- **Rasterizace** (vykreslení)
  - převod objektů scény do jednotlivých **pixelů**
  - výpočet viditelnosti („*depth-buffer*"), mapování textur a jejich kombinace, interpolace barev, průhlednost, mlha
  - vysoký **paralelismus** (nezávislé zpracování)

Reprezentace 3D dat

- v počítačové grafice existuje několik odlišných systémů reprezentace 3D dat
- nejběžnější je **povrchová reprezentace** (**B-rep**, „*boundary representation*")
  - pouze informace o **povrchu těles**
  - nejjednodušší formát: povrch je tvořen rovinnými ploškami (**trojúhelníky**)
- **geometrie** + **topologie**
  - souřadnice vrcholů + napojení plošek (síť trojúhelníků)
  - **sdílení** sousedních vrcholů, hran apod. (úspory)

Síť trojúhelníků

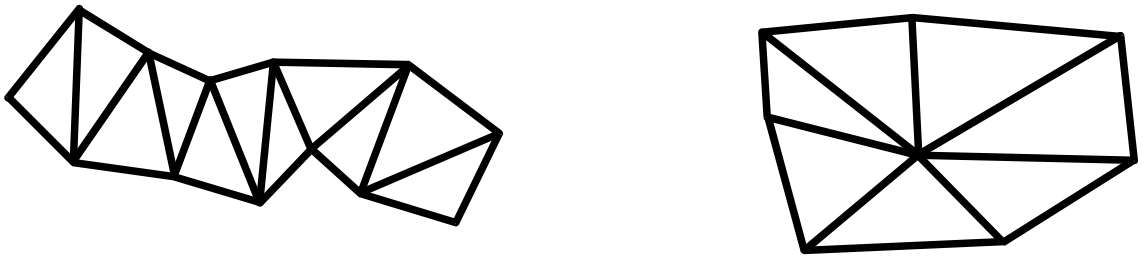




- podobnost s **relační databází**
  - tabulky **vrcholů** (V, „*vertex*"), **hran** (E, „*edge*") a **stěn** (F, „*face*") jsou navzájem propojeny
  - každá entita může obsahovat libovolný počet dalších **atributů** (vrchol: normála, barva; stěna: textura, ..)
  - odkazy pomocí **indexů** (místo ukazatelů)

Síť trojúhelníků

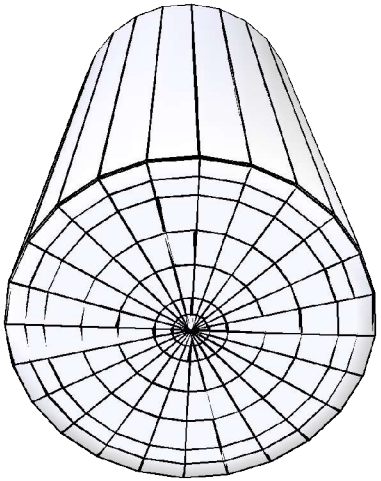
- **hrany** se u některých systémů používají, hardware s nimi však většinou nepočítá
- hardware umí přistupovat k datům **úsporně**:
  - zadá se kompletní **pole vrcholů**
  - jednotlivé stěny (trojúhelníky) se pak už jenom odkazují do pole vrcholů pomocí **indexů**
  - nejrychlejší konstrukce: **proužky trojúhelníků** („*triangle strip*”) nebo **vějíře** („*triangle fan*”)

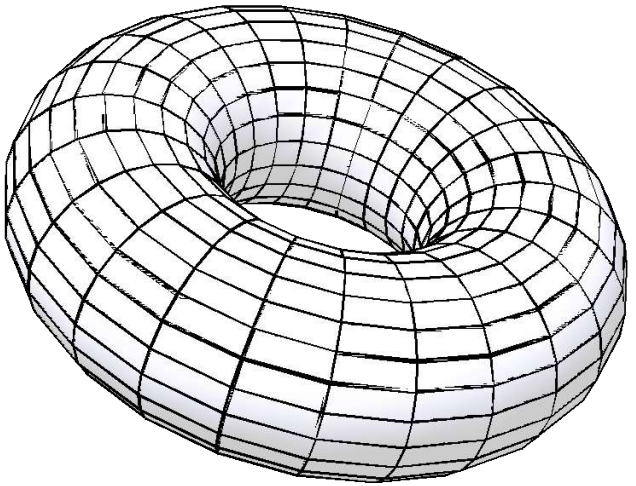


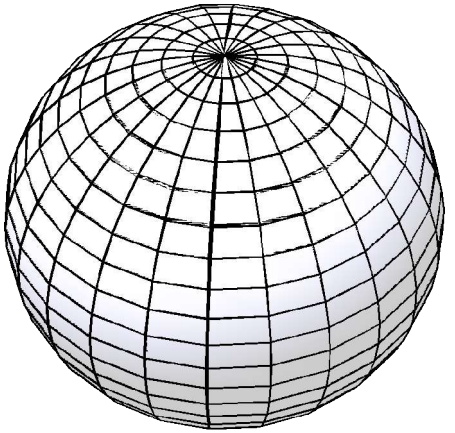
Aproximace hladkého tělesa

- aby byl vzhled co nejlepší, musí se použít **velký počet trojúhelníků**
  - pro kulovou plochu několik stovek, pro postavu člověka až desítky tisíc trojúhelníků
- hladkému vzhledu napomáhá některá ze **stínovacích technik**
  - interpolace barvy nebo normály, viz dále
- v plné přesnosti se nemusí kreslit objekty značně **vzdálené od pozorovatele**
  - úroveň detailu (LoD, „*Level of Detail*”), viz dále

Aproximace hladkého tělesa







- pro realistický výsledek ani taková přesnost nemusí stačit
  - viz **obrysy** „hladkých” těles

Geometrické transformace v 3D

- vektor 3D souřadnic [x, y, z]
- transformace násobením maticí 3×3
  - řádkový vektor se násobí zprava

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

- transformační matice 3×3 mají podstatné omezení – **nelze** je použít pro **posunutí** (translaci)

Homogenní souřadnice

- vektor **homogenních souřadnic** [x, y, z, w]
- transformace násobením maticí 4×4

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

- homogenní maticí lze i **posunovat** (translace) a provádět **perspektivní projekci**

Převod homogenních souřadnic

- homogenní souřadnice [x, y, z, w] se převádějí na běžné kartézské souřadnice vydělením (je-li $w \neq 0$) **[x/w, y/w, z/w]**
- souřadnice **[x, y, z, 0]** neodpovídají žádnému vlastnímu bodu v prostoru
  - lze je chápat jako reprezentaci **směrového vektoru** (bod v nekonečnu)
- převod z obyčejných souřadnic do homogenních je jednoduchý: **[x, y, z] ... [x, y, z, 1]**

Elementární transformace

Nejběžnější jsou afinní transformace:

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

- levá horní podmatice **[a_{11} až a_{33}]** vyjadřuje rozměr a orientaci, případně zkosení
- vektor **[t_1, t_2, t_3]** udává posunutí (translaci)
  - posunutí je až poslední operace

Kombinace několika transformací

- díky vlastnostem maticového násobení (asociativita) lze matice sdružovat:

$$(((([x, y, z, w] \cdot M_1) \cdot M_2) \cdot M_3) = [x, y, z, w] \cdot (M_1 \cdot M_2 \cdot M_3)$$

- **matice složené transformace** se připraví předem a při hromadném výpočtu se jednotlivé souřadné vektory násobí již jen touto maticí

Otočení

- matice **otočení** kolem osy „**z**” o úhel „**α**”:

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

- matice otáčení okolo ostatních souřadných os jsou analogické
- místo zadávání úhlu otočení se často používá přímo **sinus** a **cosinus** (aby se nemusela počítat goniometrie)

Posunutí (translace)

- matice **posunutí** o vektor **[t_1, t_2, t_3]**:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_1 & t_2 & t_3 & 1 \end{bmatrix}$$

- otočení kolem libovolné osy **rovnoběžné** s osou „**z**" lze realizovat jako: posunutí, otočení kolem „**z**" a posunutí zpět

$$R_{C\alpha} = T_{-C} \cdot R_{z\alpha} \cdot T_C$$

Změna měřítka

- **zvětšení** celého objektu se středem v počátku:

$$\begin{bmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \simeq \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1/s \end{bmatrix}$$

- **protažení** pouze ve směru osy „**z**”:

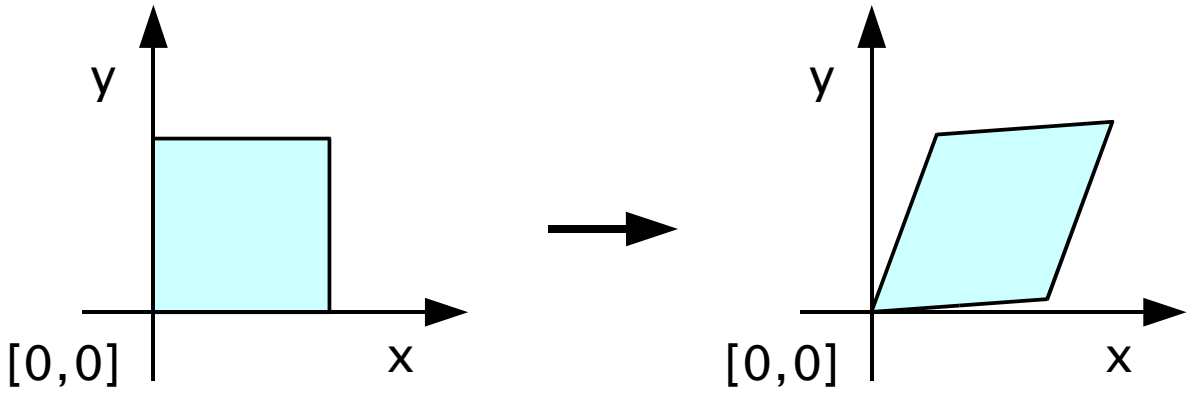
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Zkosení

- matice obecného **zkosení**:

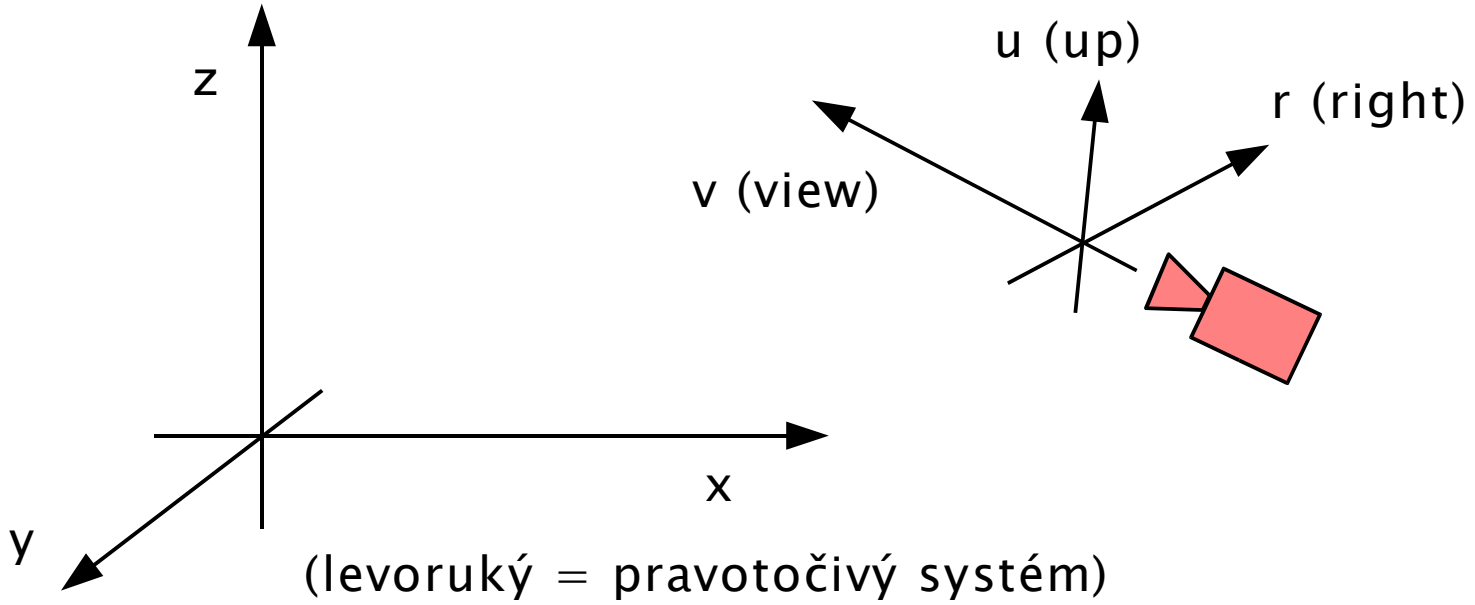
$$\begin{bmatrix} 1 & a & b & 0 \\ c & 1 & d & 0 \\ e & f & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- ukázka v rovině:



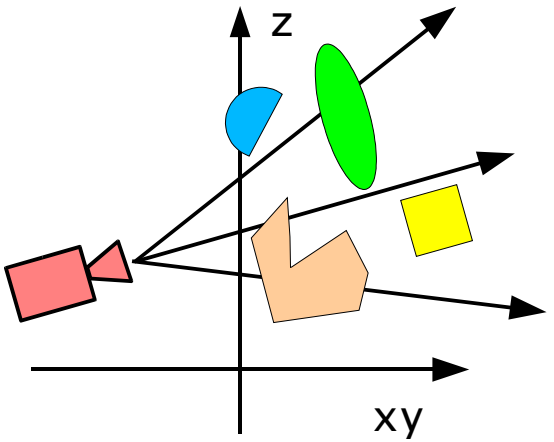
Transformace tuhého tělesa

- zachovává **tvar těles**, mění pouze jejich **orientaci**
  - skládá se jenom z posunutí a otáčení
  - často se používá k **převodu mezi souřadnicovými systémy** (např. mezi „světovými" souřadnicemi a systémem spojeným s pozorovatelem)

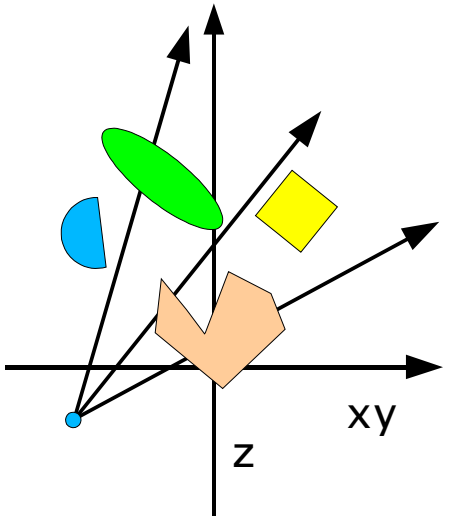


Projekce

- **rovnoběžné promítání**
  - nereálný pohled (jako z nekonečné vzdálenosti)
- **perspektivní projekce**
  - přirozený pohled (oko, fotoaparát, kamera)
  - implementace potřebuje operaci dělení

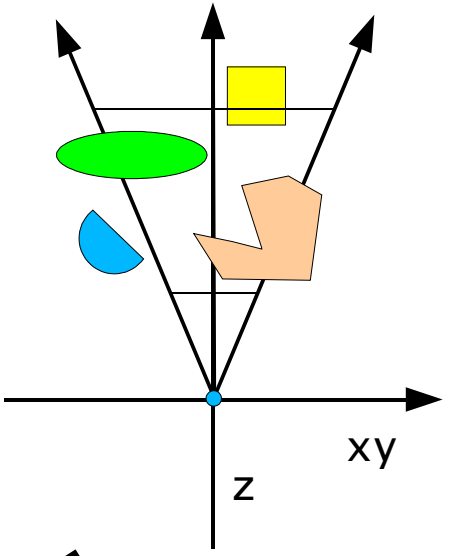


Perspektivní projekce



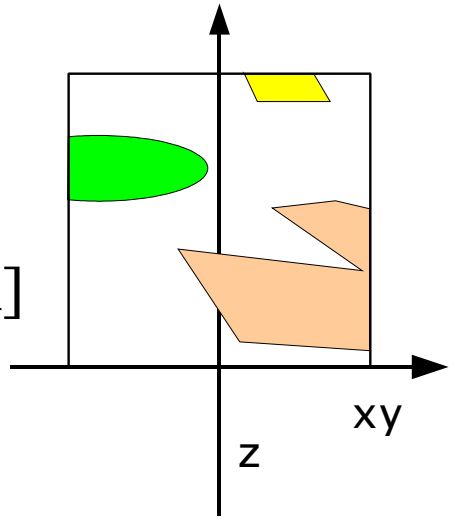
pohledová transformace

přenos soustavy kamery [v,r,u] do [z,x,y]



projekční transformace
a ořezání

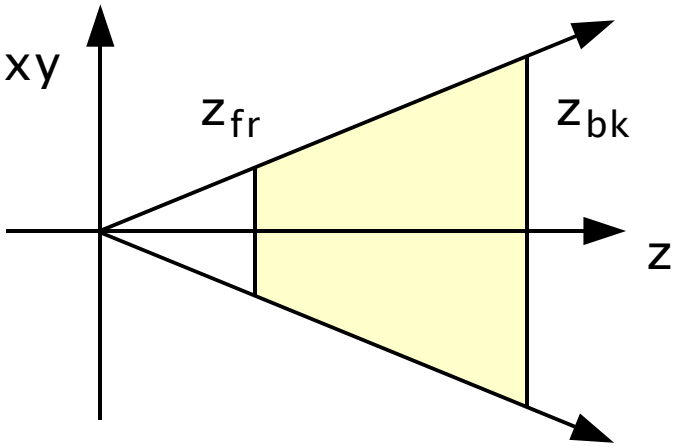
cílový box:
[-1,-1,0] až [1,1,1]



Projekční transformace

- posunutí **přední ořezávací roviny** do počátku
- **perspektivní projekce** s faktorem „z + z_{fr}”

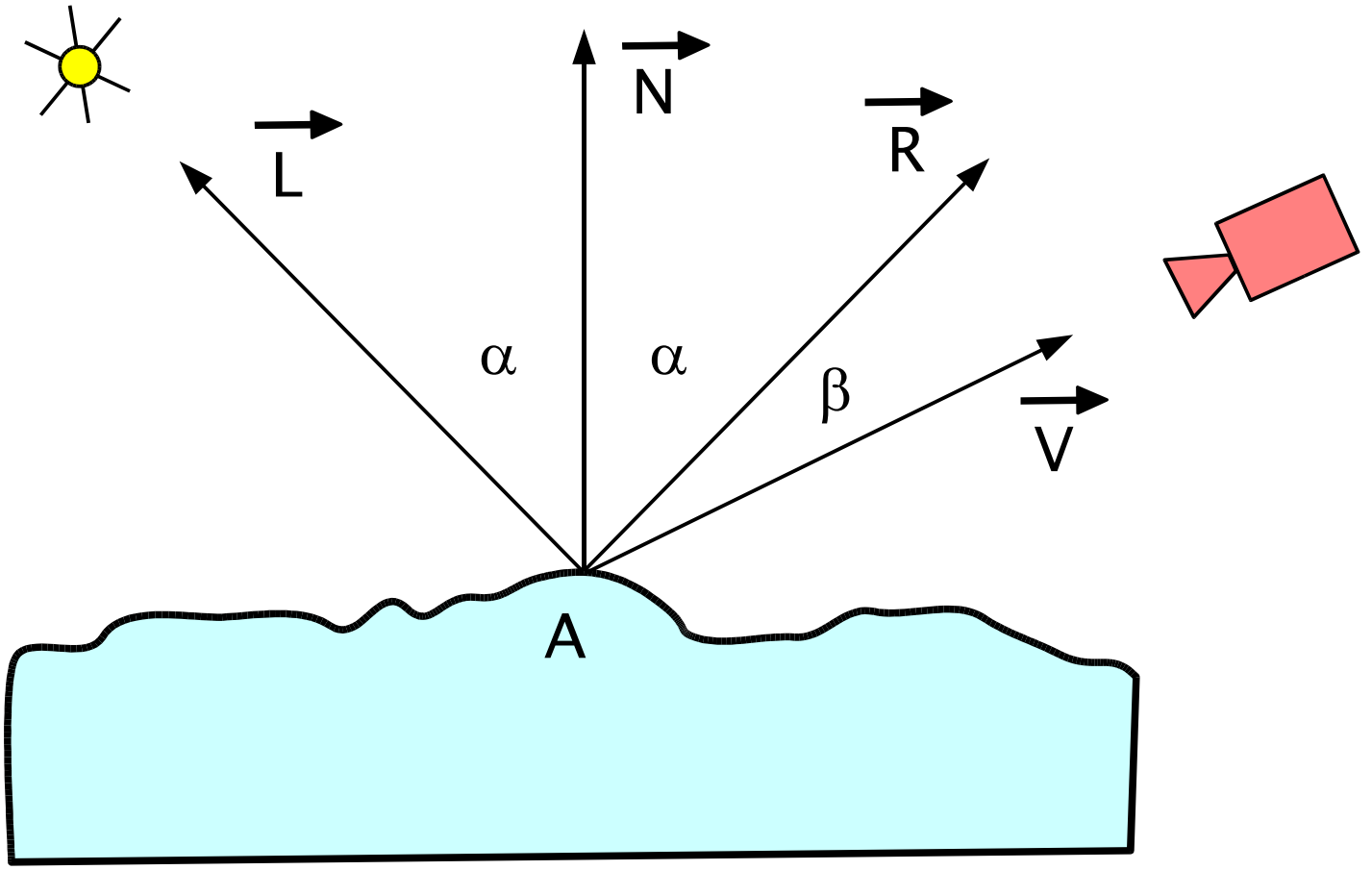
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -z_{fr} & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/z_{fr} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1/z_{fr} \\ 0 & 0 & -z_{fr} & 0 \end{bmatrix}$$



Stínování

- pozorovaný **odstín** na povrchu plošky je závislý na její orientaci v prostoru + na poloze světelných zdrojů
- existuje mnoho **osvětlovacích modelů**
  - lokální výpočet odrazu světla na povrchu tělesa
  - ideálem je přesné napodobení přírody
- **Phongův světelný model**
  - jednoduchý výpočet
  - alespoň kvalitativně souhlasí s fyzikální realitou
  - kvůli lesklé složce je nejlepší počítat ho v každém pixelu s interpolací normály

Situace



Phongův model osvětlení

- světlo se skládá ze tří složek:
  - **zbytkové světlo** („*ambient*”) – náhrada za nepočítané sekundární odrazy
  - **difusní odraz** („*diffuse*”) – ideálně matné těleso
  - **lesklý odraz** („*specular*”) – neostrý odraz, odlesk

$$L_a = C \cdot k_a$$

$$L_d = (C \cdot C_L) \cdot k_d \cdot \cos\alpha$$

$$L_s = C_L \cdot k_s \cdot \cos^h \beta$$

Parametry Phongova modelu

- optické vlastnosti povrchu tělesa („**materiál**”):
  - vlastní barva tělesa „**C**” (při osvětlení bílým světlem)
  - koeficienty „$\mathbf{k_a}$”, „$\mathbf{k_d}$” a „$\mathbf{k_s}$” (charakter povrchu: matný, lesklý, ..)
  - exponent odlesku „**h**” (čím větší, tím je odlesk ostřejší)
- vlastnosti **zdroje světla**:
  - barva a intenzita „$\mathbf{C_L}$” (třísložkový vektor, s barvou zdroje se násobí po složkách)
- **více** zdrojů světla:

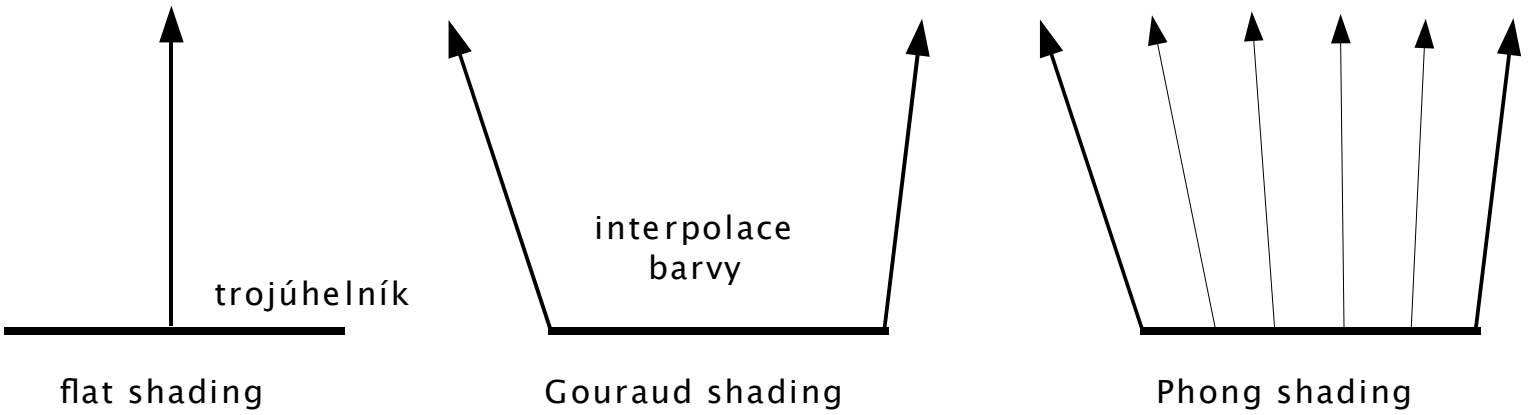
$$L = L_a + \sum L_d + L_s$$

Spojité stínování

Tři techniky použití osvětlovacího modelu:

- **konstantní stínování** („*flat shading*”)
  - celá ploška se vybarví stejným odstínem
- **Gouraudova interpolace barvy**
  - osvětlení se počítá ve vrcholech, uvnitř plošek se barva lineárně interpoluje
- **Phongova interpolace normály**
  - pro každý pixel plošky se normálový vektor lineárně interpoluje a spočítá světelný model
  - nejdokonalejší metoda, vyžaduje spolupráci HW

Spojité stínování



- **konstantní stínování** je adekvátní u hranatých těles
- **interpolace barvy** potlačuje „hranatý" vzhled aproximovaných těles
  - ostré odlesky se však nekreslí korektně
- **interpolace normály** je vhodná i pro vysoký lesk

Mlha

- předchozí osvětlovací model počítal pouze interakci světla s **povrchem předmětů**
- vliv **prostředí** (atmosféry) na šíření paprsku
  - nejjednodušší a nejčastější je výpočet **mlhy** (umí i HW)
  - přesný výpočet pohlcení/rozptylu paprsku v kouři nebo aerosolu je mnohem složitější
- **barva mlhy** „$\mathbf{C_f}$” (obvykle bílá) se mísí s barvou tělesa
  - **lineární mlha** (koeficient se mění lineárně se vzdáleností)
  - **exponenciální mlha** (exponenciální závislost koeficientu na vzdálenosti předmětu od pozorovatele)

Mlha

- míchání mlhy s barvou tělesa
  - „$\mathbf{C_f}$" je barva mlhy, „$\mathbf{C_s}$" barva tělesa
  - „**f**" je koeficient

$$C = f \cdot C_s + (1 - f) \cdot C_f$$

- **lineární mlha**
  - „$\mathbf{z_{end}}$" a „$\mathbf{z_{start}}$" jsou vzdálenosti začátku a konce mlhy
  - „**z**" je vzdálenost vykreslovaného bodu od pozorovatele
- **exponenciální mlha**
  - „**D**" je hustota mlhy (čím je větší, tím je mlha hustší)

$$f = \frac{z_{end} - z}{z_{end} - z_{start}} \qquad f = e^{-D \cdot z}$$

Úroveň detailu (LoD)

Optimální **efektivita vykreslování**:

- **vzdálené detaily** (velikostí srovnatelné s rozměrem pixelu) se už nemusí vykreslovat
- naopak – **nejbližší předměty** (na které se dívá uživatel) si zasluhují co možná nejlepší vizuální kvalitu

- **dynamická úroveň detailu** („dynamic Level of Detail”)
  - program automaticky přizpůsobuje jemnost dat
  - lze implementovat globální doladďování (např. zadání celkového počtu trojúhelníků, které se mají vykreslovat)
  - náročná příprava dat: předem se musí připravit několik úrovní dat (nebo algoritmus spojitého zjemňování)

Hierarchické modelování

- scéna se skládá z objektů
  - objekty se skládají z komponent
    - komponenty se skládají ze součástek
      - » součástky se skládají z ...

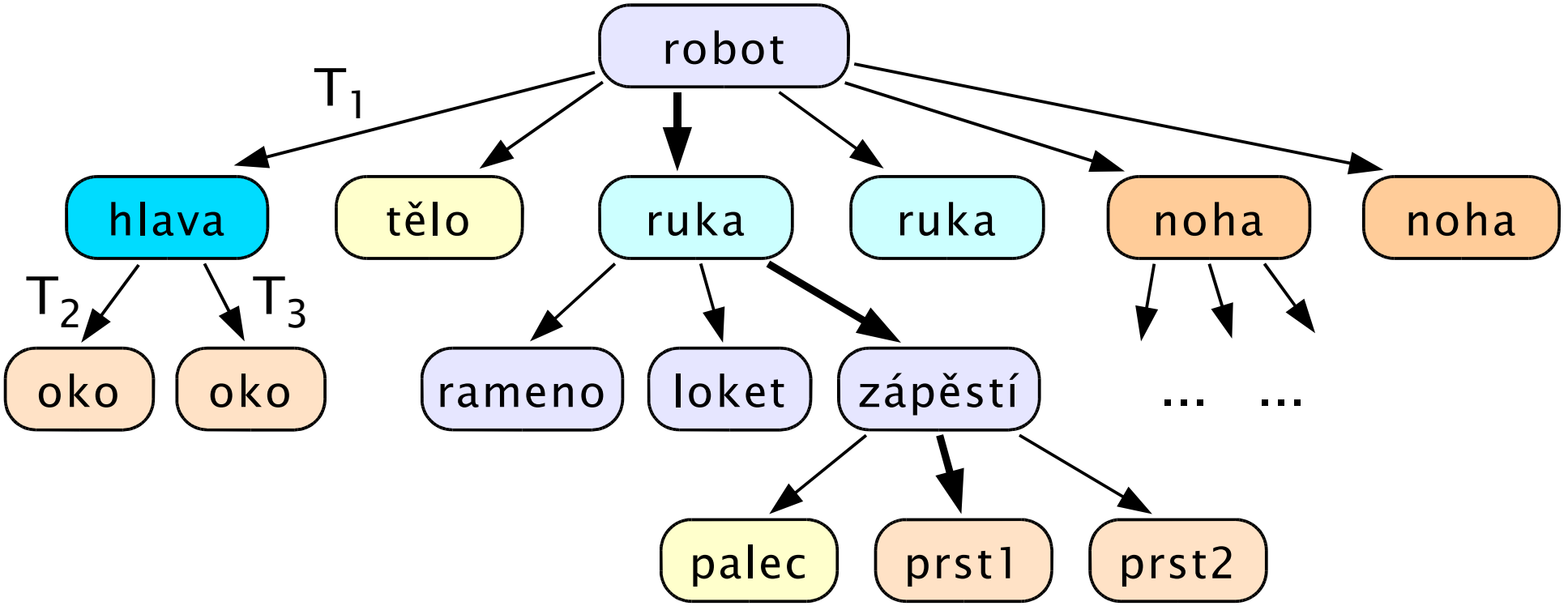
Hierarchické modelování je přirozené a efektivní

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

Strom scény

Strom scény je uložen na disku

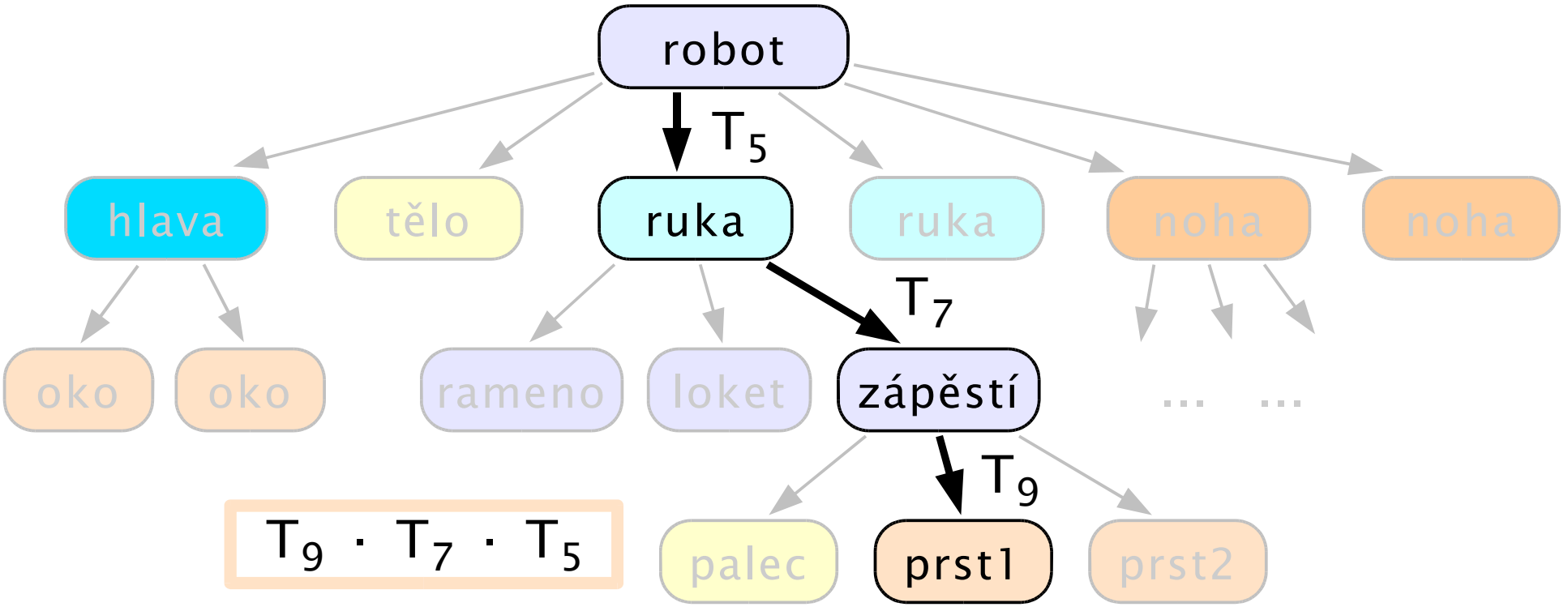
- kvůli vícenásobným odkazům se někdy v paměti rozvíjí



Relativní transformace

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

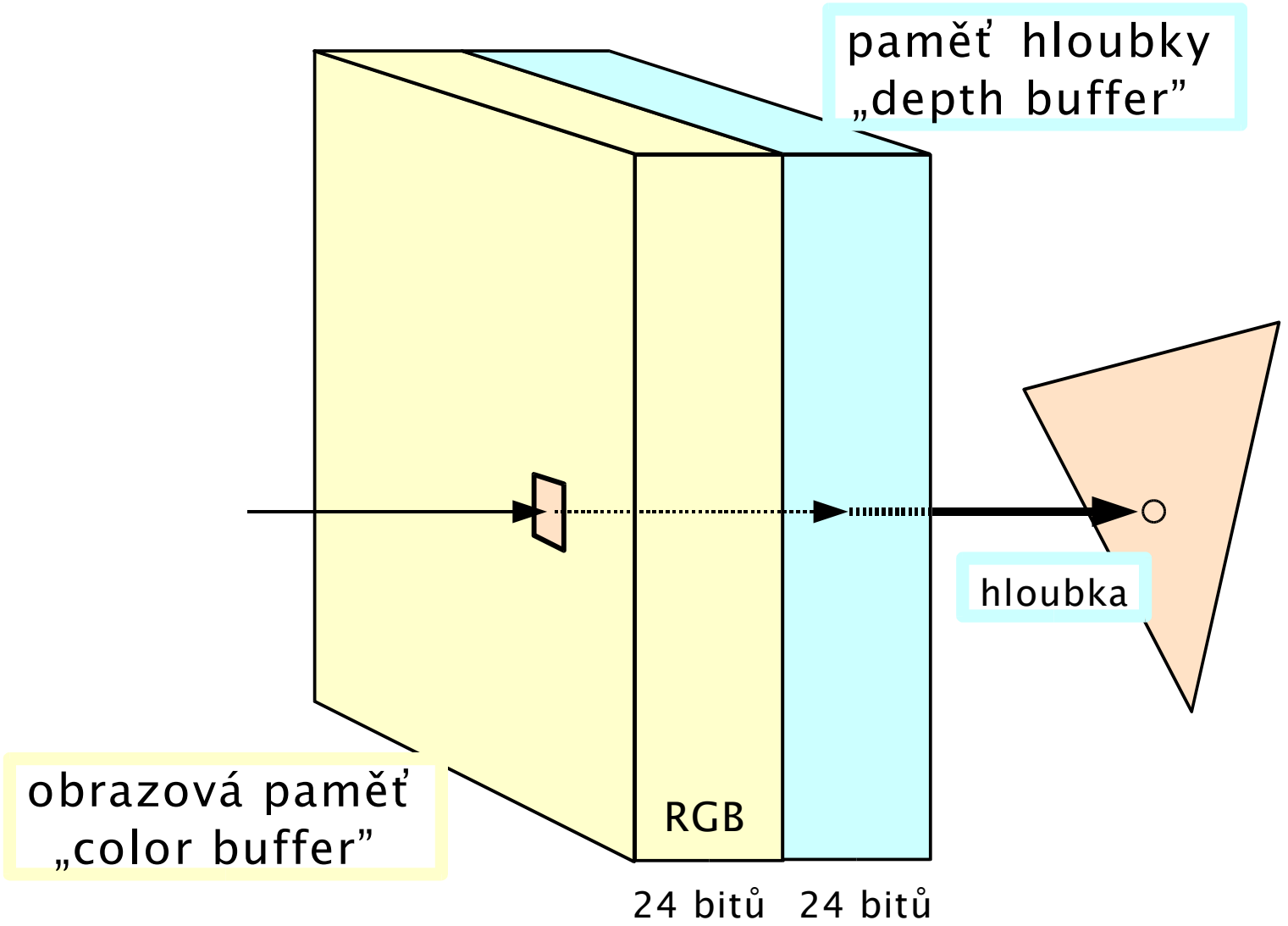
- složené transformace může omezeně počítat až **GPU**



Výpočet viditelnosti

- historicky první komponenta přenesená na hardware (Silicon Graphics)
- algoritmus: **paměť hloubky** („*depth-buffer*”)
  - viditelnost se řeší na úrovni **jednotlivých pixelů**
  - **pro každý pixel** je v bufferu uložena vzdálenost (hloubka) nejbližšího dosud nakresleného bodu
  - **hloubka** je 16- nebo 24-bitové číslo (pevná des. čárka)
  - souřadnice „**z**” nebo „**w**” (homogenní složka před vydělením, výhodnější pro interpolaci i kvůli přesnosti)
  - při kreslení se musí grafická primitiva **rozkládat** na jednotlivé pixely („**rasterizace**”, řádkový rozklad)

Paměť hloubky



Paměť hloubky

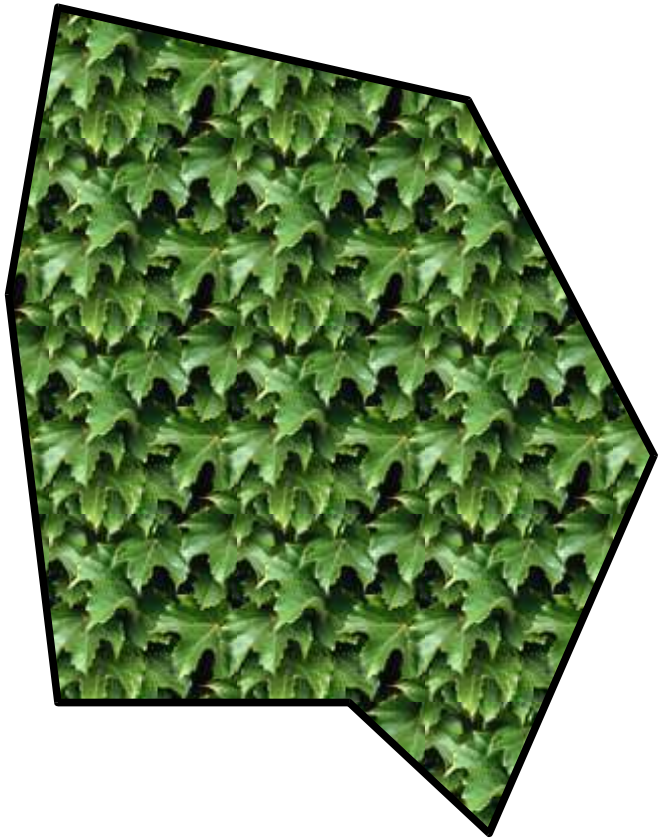
Výhody:

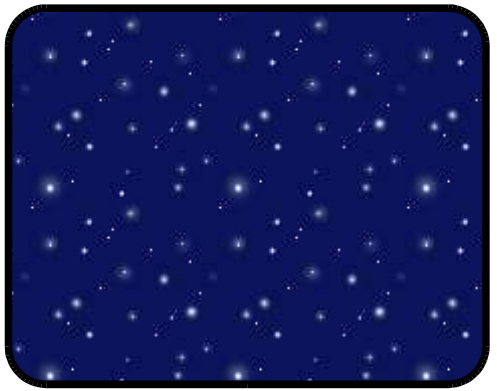
- vykreslování objektů v **libovolném pořadí**
- správně vyřeší všechny problematické situace (cykly)
- jednoduchost ⇒ snadná **implementace v hardware**, masivní paralelismus

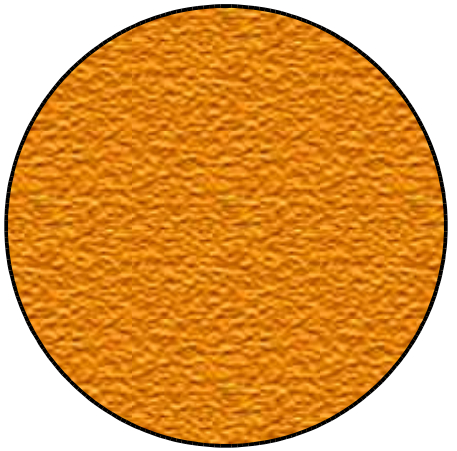
Nevýhody:

- **omezená přesnost** hloubky „**z**" (místo ní „**w**")
- pozor na „***Z-fighting***" (polygony v jedné rovině)
- výpočet je závislý na směru pohledu
- poloprůhledné plochy se musí třídit

Textury







Textury

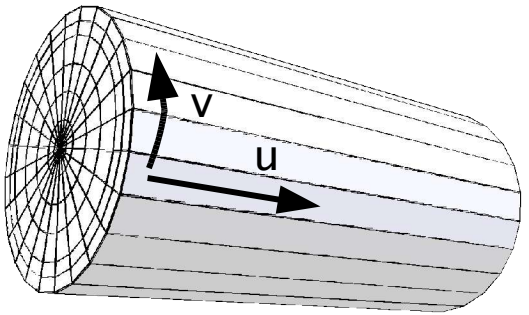
- **vylepšují vzhled** povrchu těles
  - modifikace **barvy** („bitmapa”)
  - dojem **hrbolatého povrchu** („*bump-texture*”)
  - příp. modulace **dalších parametrů**: průhlednost, odrazivost, lesk

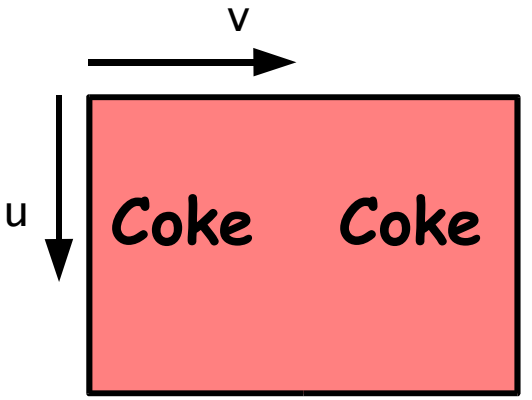
Definice textury:

- **2D datovým polem** (rastr, „bitmapa”) – nejčastější, umí to každý HW
- **3D datovým polem** („*volume texture*”)
- **procedurálně** – algoritmem v každém pixelu (programovatelné GPU)

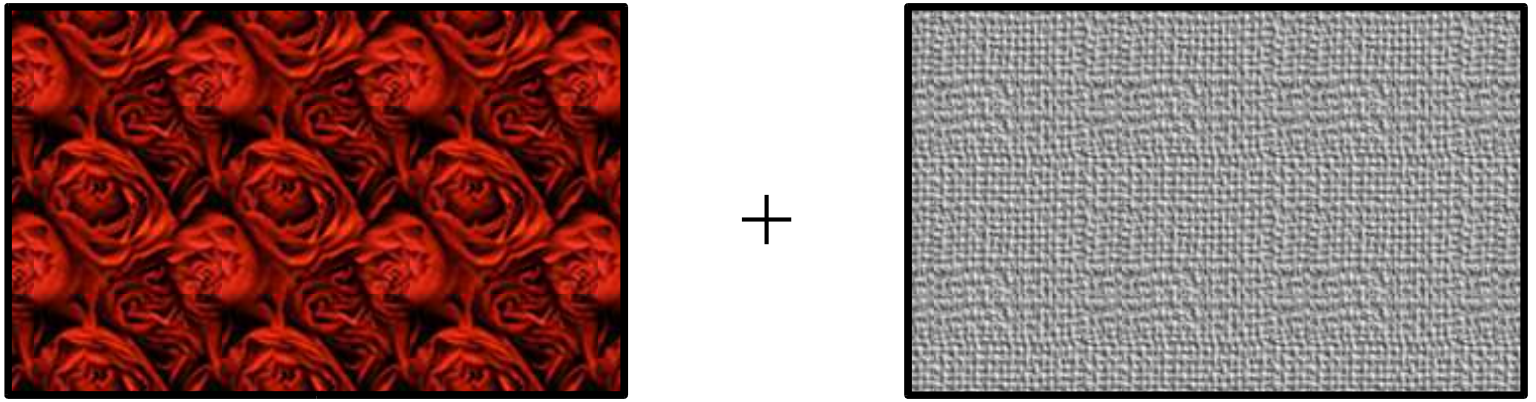
Mapování textur

- **2D textury** se musí na povrch tělesa **mapovat**
  - **texturové souřadnice [u, v]** se zadávají v každém vrcholu
  - grafický akcelerátor je interpoluje do jednotlivých pixelů
  - v bitmapě se musí intepolovat (mezi sousedními pixely textury = „**texely**”)





Kombinace textur



- moderní GPU umějí v jednom pixelu **kombinovat** několik textur
  - globální (pomalu se měnící) **základ** + **detailní** textura
  - předem spočítané **osvětlení** (globální osvětlení)
  - „***environment maps***" – odlesk zbytku scény apod.

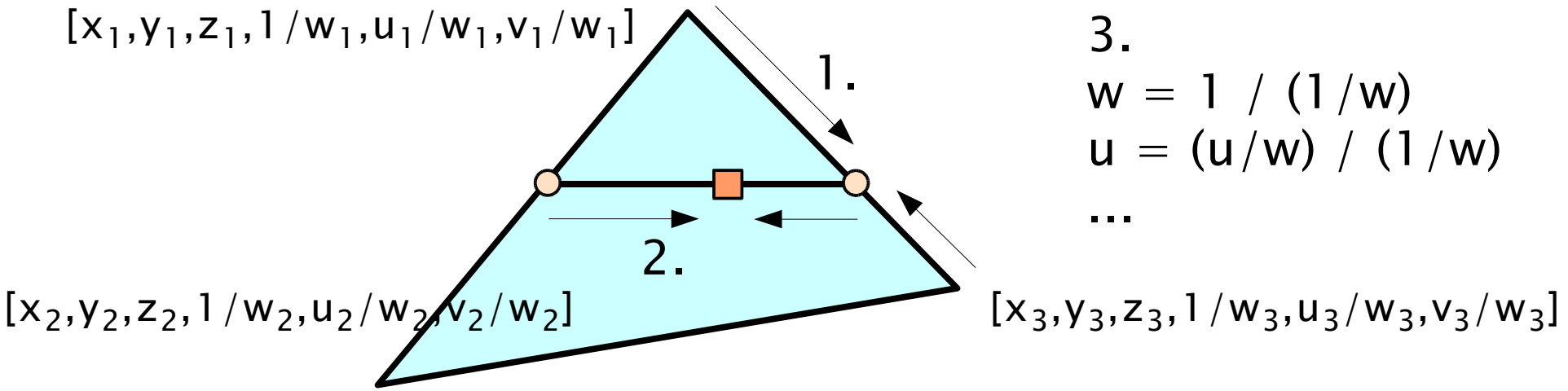
Zpracování pixelů

Rasterizace – rozklad primitiv (trojúhelníků) na pixely

- algoritmus řádkového rozkladu („*scanline*”)
- implementace v hardwaru
- už několik let se uvažuje o **složitějších** primitivech (Bèzierovy plochy, NURBS, „subdivision” plochy)

- zpracování pixelů – finální fáze **Rasterizace**
  - **interpolace** barvy, hloubky a texturových souřadnic
  - **test hloubky** („*depth test*”)
  - případně též: **test šablony** („*stencil test*”) a u viditelného pixelu **výpočet průhlednosti**

Interpolace v pixelech



3.
$w = 1 / (1/w)$
$u = (u/w) / (1/w)$
...

Snahou je používat co nejvíce **lineární interpolaci**

- **hloubku** „**z**" lze interpolovat lineárně
- pro **texturové souřadnice** nebo „**w**" se musí implementovat perspektivně-korektní interpolace:
- **lineárně** interpoluji „**1/w**", „**u/w**" i „**v/w**", **[u, v]** pak dopočítám dělením (**hyperbolická interpolace**)

Průhlednost

- poloprůhledné **plochy** nebo **textury**
  - další atribut pixelu – kanál „**α**" („*alpha channel*") definuje **neprůhlednost**
  - 0 … absolutně průhledný povrch
  - 1 … neprůhledný materiál (za ním už nic neprosvítá)
  - v praxi se používá stejné rozlišení jako u R,G,B (8 až 16 bitů) – barevný prostor se rozšiřuje na **[R, G, B, α]**
  - další užívaný formát: **[Rα, Gα, Bα, α]**
- poloprůhlednost a grafické karty
  - při vykreslování poloprůhledné scény je potřeba plošky **třídit** odzadu dopředu (nebo bude kresba nekorektní)

Šablony („stencils“)

Další buffer velikosti obrazovky

- není předem dáno, pro jaké účely se má použít
- jeho použití je třeba **programovat** nebo alespoň **konfigurovat** (k dispozici jsou: boolovské I/O operace, testy)
- běžně se ke každému pixelu dá připojit **8 bitů** šablon
- do šablony se **zapisuje** (a/nebo se **testuje**) v jedné z posledních fází zpracování pixelu

Aplikace:

- zejména při **víceprůchodovém zpracování** (viz)
- omezené vykreslování, selektivní hloubkový test, výpočet vržených stínů, ...

Programování GPU

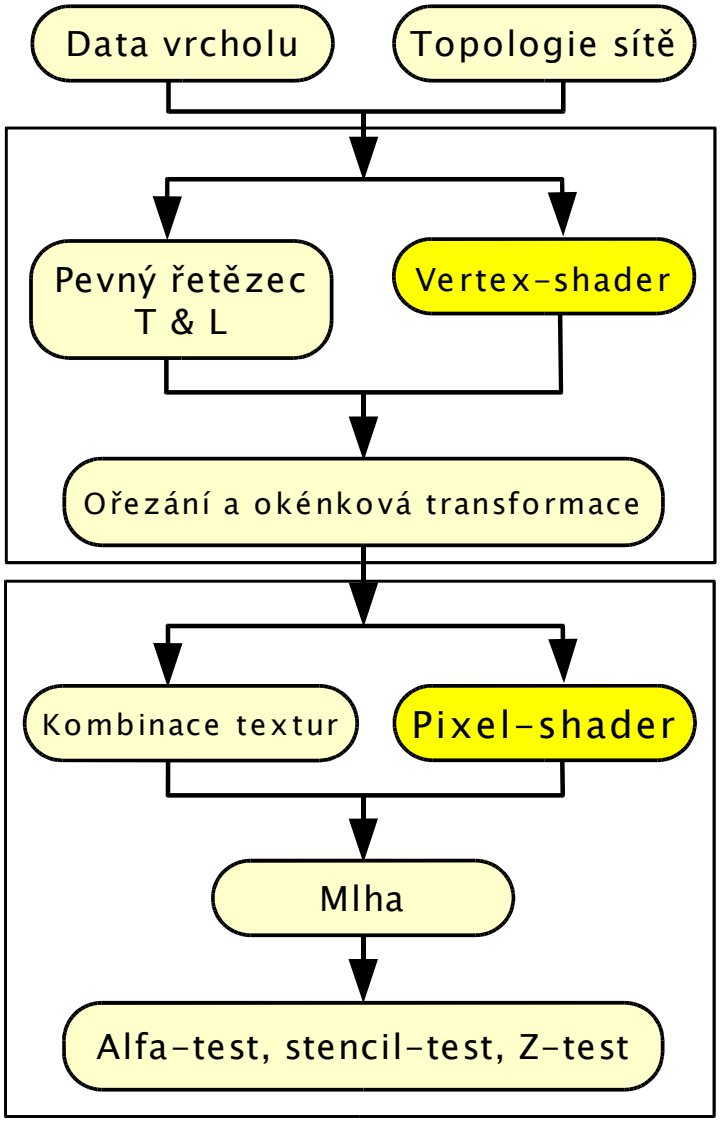
- revoluční (ale logický) krok kupředu
  - **NVidia** a **Microsoft – DirectX 8.0**: „*vertex shaders*" VS a „*pixel shaders*" PS (2000)
  - **OpenGL** – NVidia „*extensions*" (2001)
  - **DirectX 8.1** – PS verze 1.2 až 1.4 /jen **ATI**/ (2001)
  - **DirectX 9.0** – VS 2.0, PS 2.0 (2002)
  - velmi silné specifikace VS 3.0 a PS 3.0 zatím většina akcelerátorů simuluje softwarově
  - „***High Level Shading Language***" **HLSL**: pro člověka čitelnější, strukturované příkazy, definice typů (pod tím je třeba stále vidět hardwarová omezení!)

Shaders

Princip programování (konfigurace) GPU:

- některé části zobrazovacího řetězce zůstávají **neměnné** (algoritmy ořezávání, okénková transformace, výpočty mlhy, průhlednosti, testy hloubky a šablony)
- dvě nejdůležitější fáze se mohou programovat zvenčí:
  - zpracování jednoho **vrcholu trojúhelníka**
  - zpracování jednoho **pixelu trojúhelníka** (nebo celého segmentu – **fragmentu**)
- „***vertex shader***" se blíží programu v assembleru běžných CPU (počet instrukcí i zprac. dat jsou omezeny)
- „***pixel shader***" je velmi omezený, málo instrukcí, „skript" popisující konfiguraci jistých etap rasterizace

Programovatelný grafický řetězec



HW T&L („*Hardware Transform & Lighting*")

- od Nvidia GeForce 256 (1999)

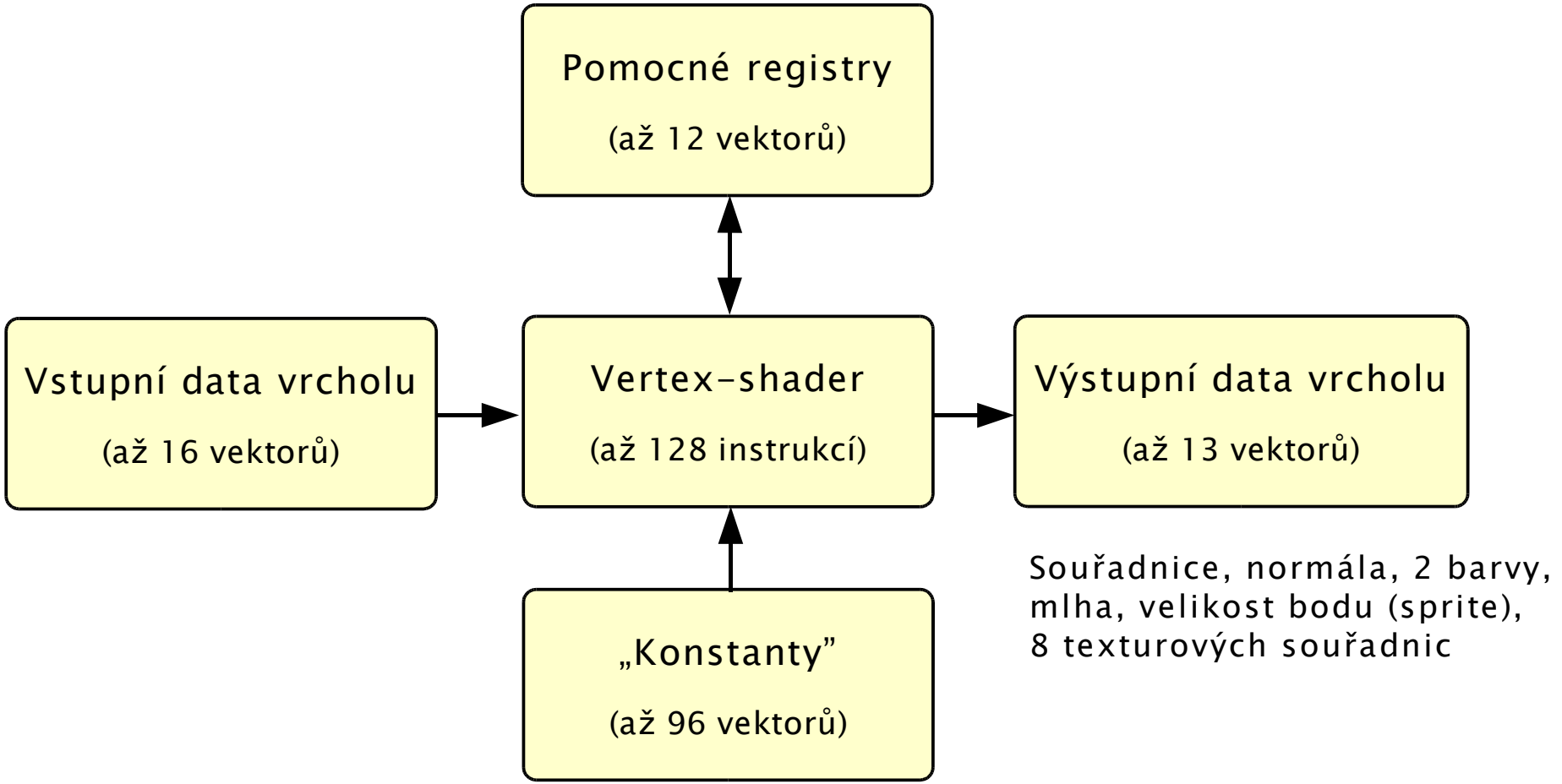
Rasterizace se na čipu dělá již dlouho

- stanice Silicon Graphics
- API: IrisGL pod.
- **multi-texturování**: až druhá polovina 90. let

Vertex-shaders

- **zpracovávaná data vrcholu** (dle VS 1.1):
  - souřadnice vrcholu
  - normálový vektor
  - barva povrchu
  - texturové souřadnice
  - další atributy si může zvolit programátor
- **výstup** (do zpracování pixelů/fragmentů) :
  - souřadnice vrcholu a normálový vektor
  - dvě barvy pro stínování (matná a lesklá složka)
  - barva mlhy
  - velikost bodu (pro bodová primitiva „*point sprite*”)
  - až 8 texturových souřadnic

Prostředí VS 1.1



Výpočet ve vrcholu

- **výsledek** pokračuje dál v cestě zobrazovacím řetězcem
  - většina hodnot je později automaticky interpolována v rasterizátoru do jednotlivých pixelů
  - některá data může využívat i „fragment-shader”
- **nelze přidat nový vrchol**!
- **programování**:
  - **assembler** nebo několik vyšších vývojových prostředí: **Cg**/NVidia, **RenderMonkey**/ATI, **HLSL**/Microsoft
  - 4-složkové vektory **32-bitových** čísel („float[4]”)
  - více vektorů za sebou tvoří **matice** 4×3 nebo 4×4, všechny instrukce umějí implicitně permutace složek, ..

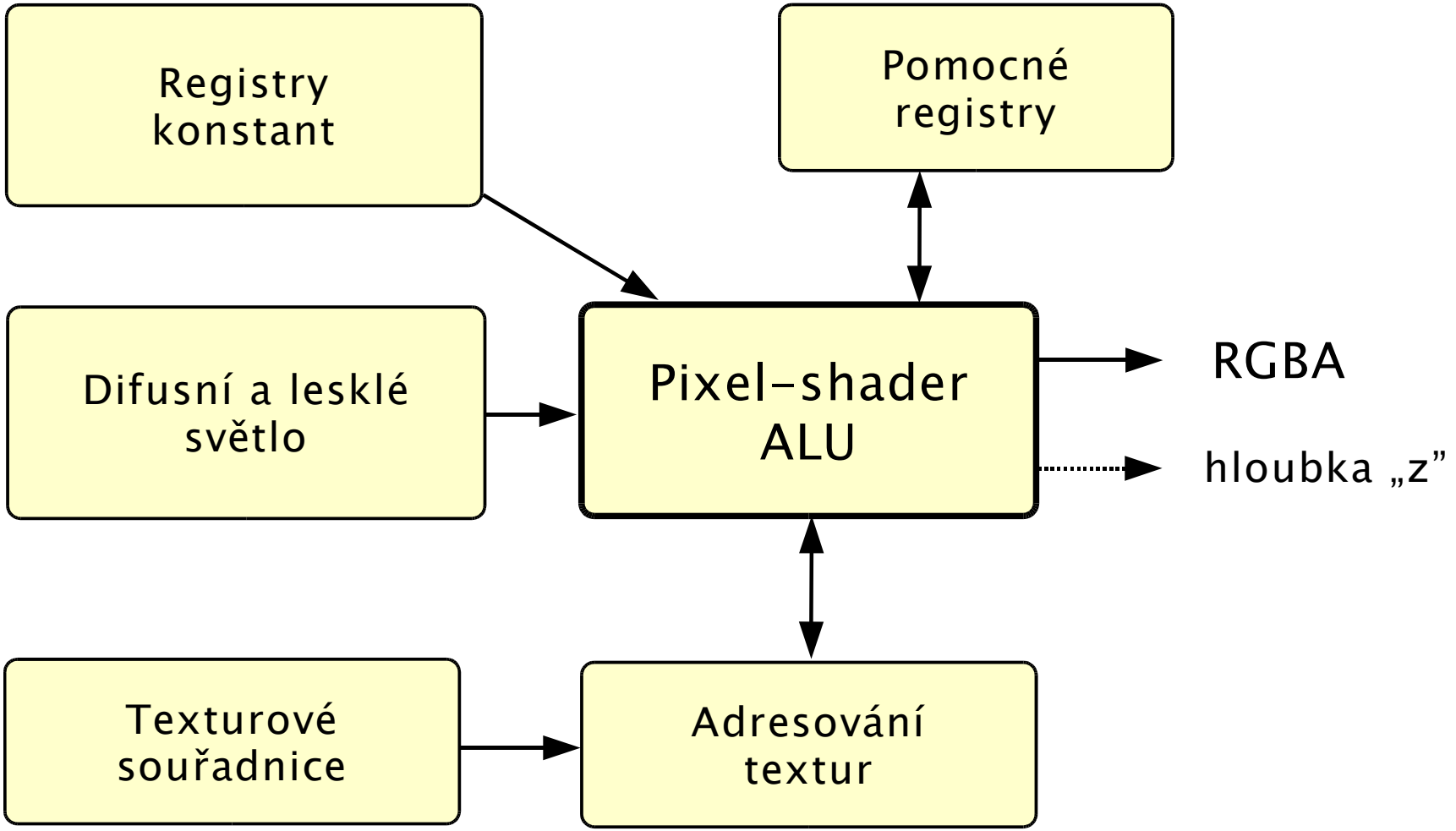
Příklad v HLSL

```
float4x4 World      : WORLD;
float4x4 View       : VIEW;
float4x4 Projection : PROJECTION;

struct VS_OUTPUT {
    float4 Pos  : POSITION;
    float4 Diff : COLOR0;
    float4 Spec : COLOR1;
    float2 Tex  : TEXCOORD0;
    };

VS_OUTPUT VS ( float3 Pos : POSITION, float3 Norm : NORMAL, float2 Tex : TEXCOORD0 )
{
    VS_OUTPUT Out = (VS_OUTPUT)0;
    float3 L = -lightDir;
    float4x4 WorldView = mul( World, View );

    float3 P = mul( float4(Pos, 1), (float4x3)WorldView );
               // position (view space)
    float3 N = normalize( mul( Norm, (float3x3)WorldView ) );
               // normal (view space)
    float3 R = normalize( 2 * dot(N, L) * N - L );
               // reflection vector (view space)
    float3 V = -normalize( P );
               // view direction (view space)

    Out.Pos  = mul( float4(P, 1), Projection );              // position (projected)
    Out.Diff = I_a * k_a + I_d * k_d * max( 0, dot(N, L) );  // diffuse + ambient
    Out.Spec = I_s * k_s * pow( max(0, dot(R, V) ), 8 );     // specular
    Out.Tex  = Tex;

    return Out;
}
```

Pixel-shaders (fragment-shaders)

- **vstupní data pro jeden pixel** (dle PS 1.x):
  - interpolovaná matná a lesklá barva (**RGB**)
  - interpolovaná průhlednost („**α**”)
  - osm konstant
  - interpolované texturové souřadnice (maximálně osm)
  - hloubka „**z**”
- **výstup**:
  - výsledná barva pixelu (**RGBα**)
  - případně změněná hloubka „**z**”

Prostředí PS



Výpočet v pixelu

- kód je velmi úzce svázán s **pixelovými interpolátory**
  - přísné limity na počet instrukcí a délku „*pipeline*”
  - výpočet musí běžet co nejrychleji, jinak by se mohl snižovat pixelový výkon („*fill-rate*”)
- **nelze modifikovat souřadnici pixelu**!
- PS lze chápat spíš jako **konfigurační skript** některých fází rasterizace
  - kombinace textur + výpočet výsledné barvy pixelu
  - někdy je dovoleno modifikovat hloubku „**z**” (např. pro „hypertextury”)

Příklad v HLSL

```
texture Tex0 < string name = "tiger.bmp"; >;

sampler Sampler = sampler_state {
    Texture   = (Tex0);
    MipFilter = LINEAR;
    MinFilter = LINEAR;
    MagFilter = LINEAR;
    };

float4 PS ( float4 Diff : COLOR0, float4 Spec : COLOR1, float2 Tex : TEXCOORD0 ) : COLOR
{
    return tex2D( Sampler, Tex ) * Diff + Spec;
}

technique TBothShaders
{
    pass P0
    {
            // both vertex & pixel shaders
        VertexShader = compile vs_1_1 VS();
        PixelShader  = compile ps_1_1 PS();
    }
}


// shader samples: © 2002 Microsoft Inc.
```

Pokročilé techniky

Nejdůležitější **pokročilé techniky** užívané v hardwarem podporované 3D grafice:

- „***anti-aliasing***" („vyhlazování")
- „***mip-mapping***" a **neizotropické filtrování**
- **víceprůchodové** zpracování
- výpočet **vržených stínů**
- „***bump-mapping***"
- „***environment-mapping***"

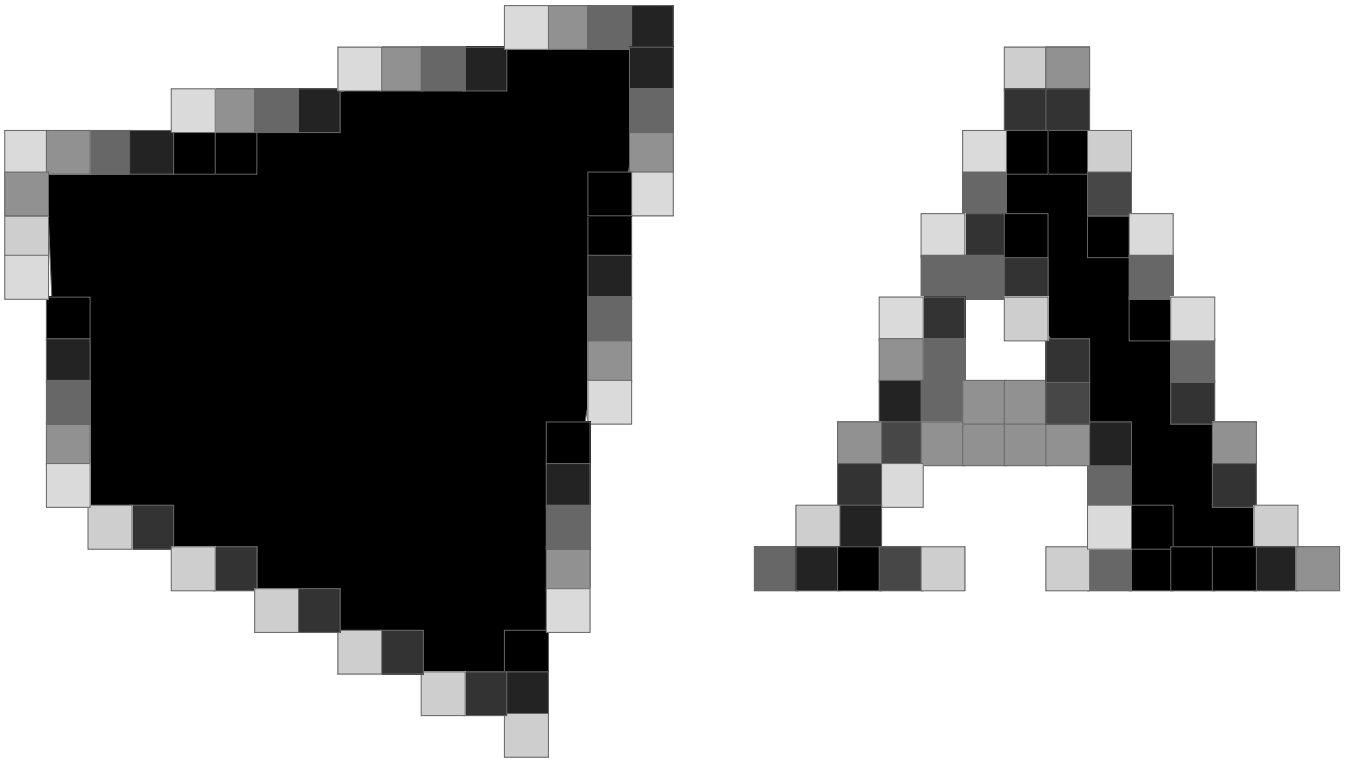
- často jsou to triky související s **mapováním textur**
  - ne vždy je potřeba programovatelná GPU

Anti-aliasing

- při každém pravidelném bodovém vzorkování spojitě definované funkce vzniká „***alias***”
  - zubaté obrysy hladkých předmětů
  - „zrnění” a přeskakování u vzdálených textur
  - interference pravidelného vzorku v dálce
- „***anti-aliasing***” je označení technik potlačujících alias
  - „***supersampling***” – metoda hrubé síly
    - obraz se rasterizuje v několikrát zvětšeném rozlišení
    - finální výstup se získá **průměrováním** tohoto mezivýsledku (konfigurace faktoru zvětšení i filtrů…)

Anti-aliasing

- místo zubatých okrajů se na obrysu objeví pixely s přechodovými barvami:



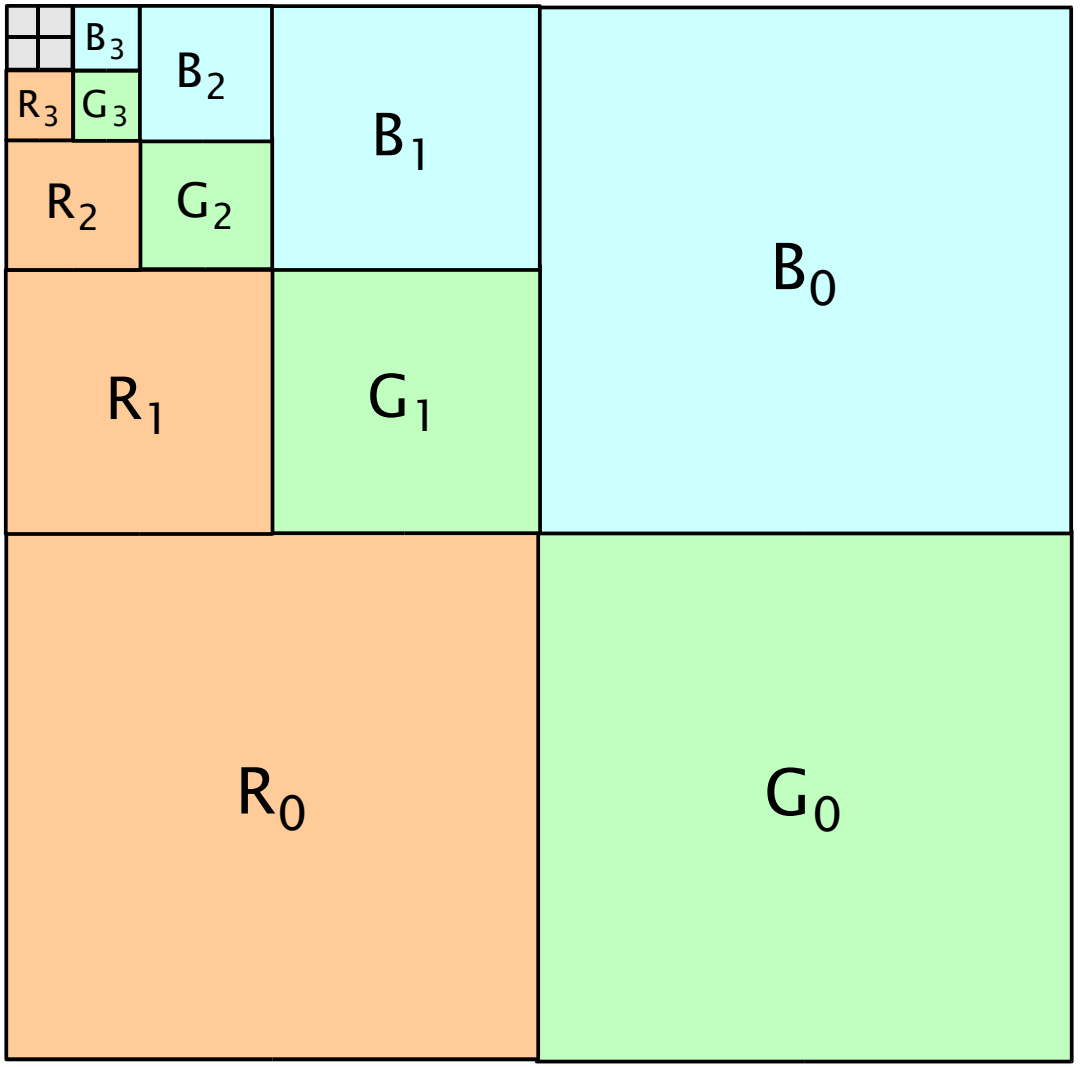
Filtrování textur

- **textury pozorované z velké dálky** se také musejí filtrovat (průměrovat při zmenšování)
  - jinak by se projevil „***alias***" – zrnění při pohybu kamery
- existuje několik technik, jak si zmenšené textury **předem připravit**:
  - „**MIP-map**" („multum in parvo") je asi nejznámější
  - „**ripmap**" (Hewlett-Packard) obsahuje i neizotropní zmenšeniny
  - **neizotropní filtrace** – dynamicky se počítá, v jakém poměru se má textura zmenšit, sčítá se jen pár texelů
  - **součtové tabulky** – předem sečtené LU obdélníky

MIP–mapping

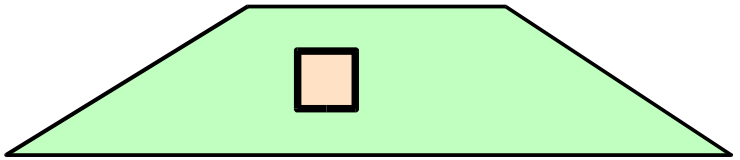
- předem se textura zmenší na **binární zlomky** (1/4, 1/16, apod.)
  - při zmenšování se použije **průměrování barev** v sousedních pixelech
  - pro **3-kanálovou barvu** (např. RGB) existuje šikovné uspořádání MIP-mapy – zmenšená čtvrtina textury se v paměti uloží místo chybějící čtvrté složky
- použití MIP-mapy
  - určí se úroveň a z ní se přečte jediná barva (rychlost)
  - interpoluje se **mezi dvěma úrovněmi** MIP-mapy, případně ještě mezi čtyřmi sousedy v jemnější úrovni

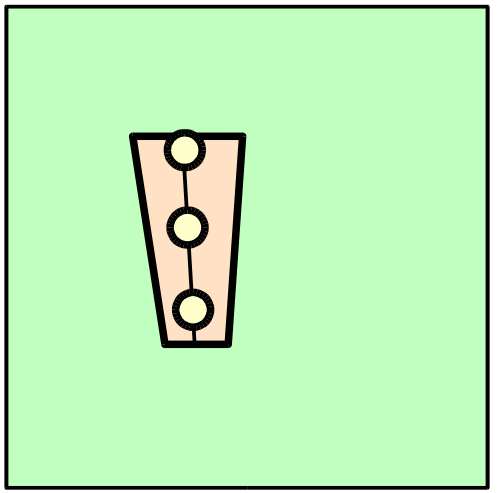
MIP-mapa



Neizotropické filtrování

- **zpětně promítnutý pixel** má na textuře tvar deformovaného čtverce
  - z MIP-mapy se vybere taková úroveň, aby měla **kratší strana** toho čtyřúhelníka délku cca jednoho texelu
  - podél **delší strany** čtyřúhelníka se průměruje





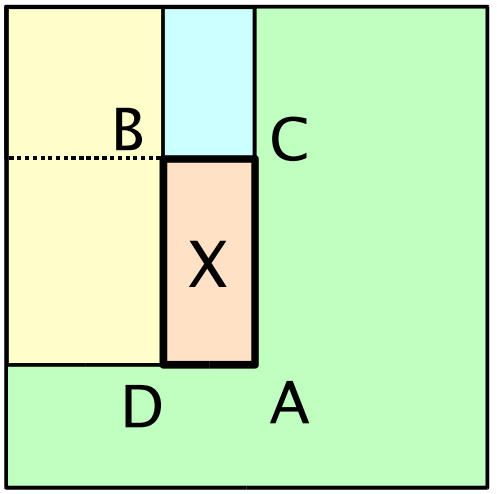
zobrazovaný pixel

přístup do MIP-mapy

Součtové tabulky

$X = A + B - C - D$

A, B, C, D ... levé horní součty



- pro **rychlé počítání součtu** (a tedy i průměru) libovolného obdélníka
  - předem se připraví součty všech obdélníků začínajících v levém horním rohu textury
  - je potřeba větší přesnost (minimálně 16 bitů místo 8)

Více průchodů

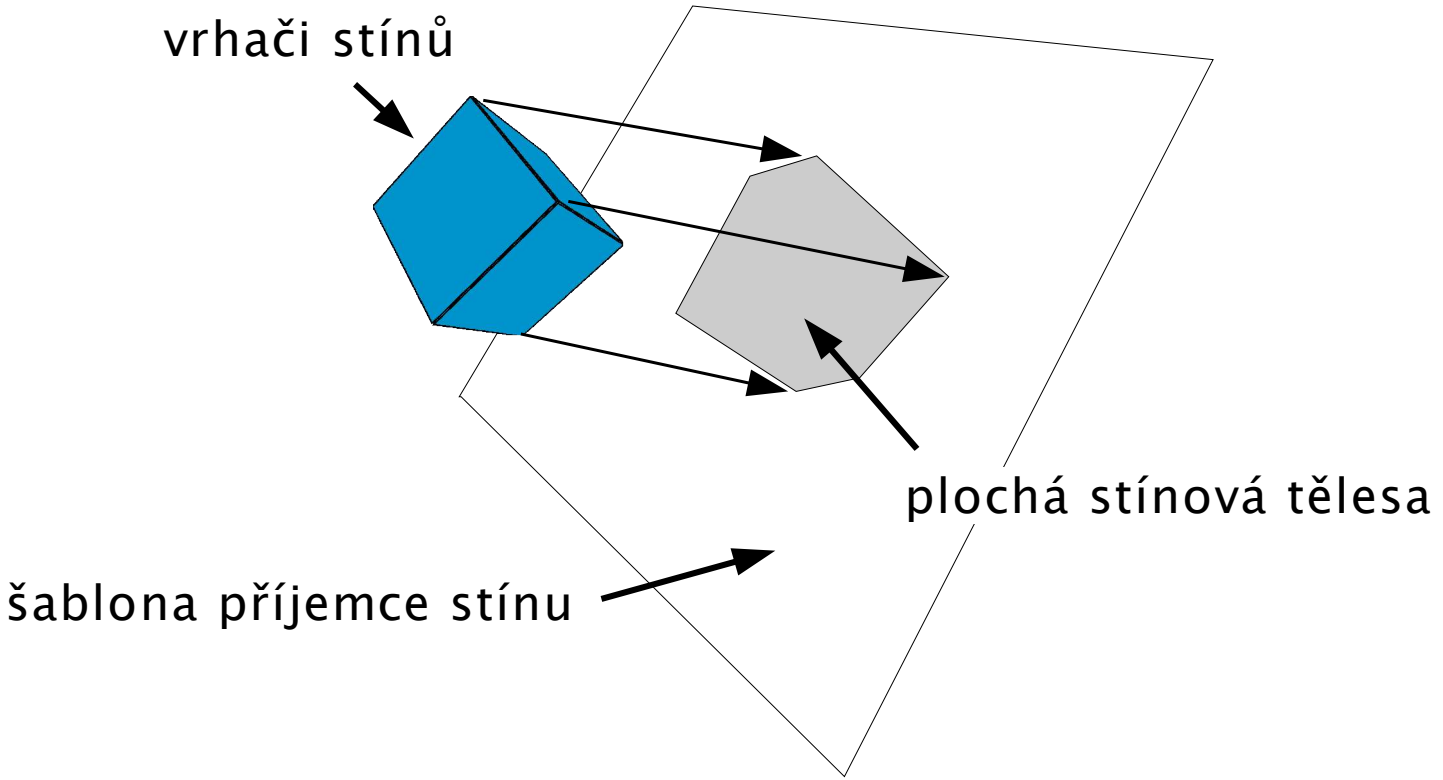
- některé efekty se bez více průchodů scénou neobejdou
  - použití **šablon**, **akumulačního bufferu**
- mezi jednotlivými průchody se **zachovává** (může zachovat – dle potřeby):
  - **šablony**
  - **buffer hloubky**
  - **buffer barvy** (výsledná bitmapa)
  - **akumulační buffer** (do něj lze sčítat barevné buffery)
- topologie scény se mezi průchody obyčejně nemění
  - **geometrie** a **transformace** se měnit mohou (pro měkké stíny nebo rozmazání pohybem)

Výpočet vržených stínů

- výpočet stínů, které vznikají při osvětlení scény ostrým zdrojem světla
- učebnicový příklad použití **šablony** („*stencil*”) a **více průchodů** scénou
  - šablona maskuje plochy, na které má stín dopadat, a zařídí, aby se stíny nezdvojovaly
- jednoduchý algoritmus:
  - stíny se vrhají na **jedinou rovinu** („rovina příjemce”)
  - obraz stínu může být **neprůhledný** (ve stínu zaniká původní barva/textura příjemce) nebo **průhledný** (stín jen snižuje množství světla)

Stíny vrhané do roviny

- jednoduchý algoritmus, stíny se vrhají do jediné roviny
  - **projekční matice** z 3D prostoru do roviny příjemce



Stíny vrhané do roviny

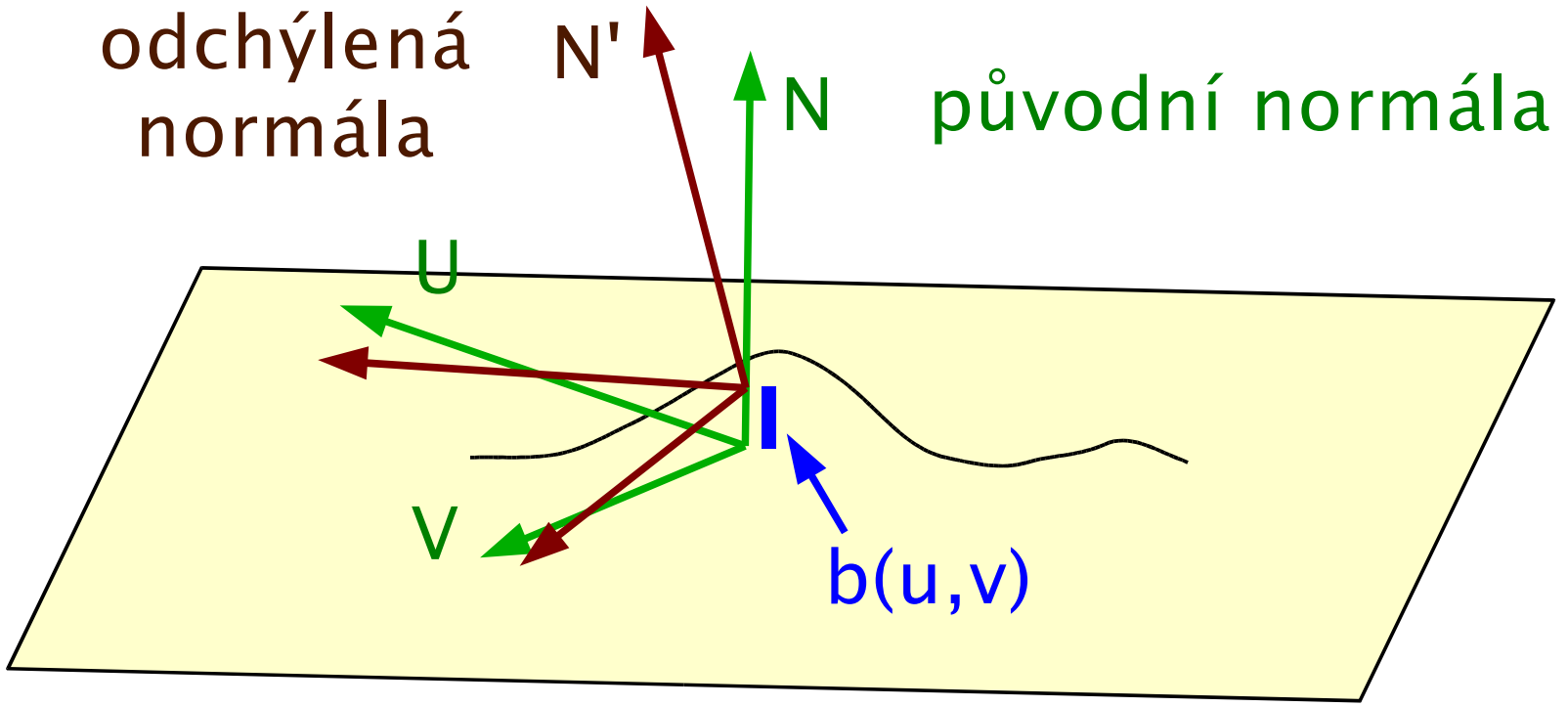
- postup vykreslování:
  1. celá 3D scéna se vykreslí v **běžném promítání**
     - příjemce **nastavuje** daný bit šablony
     - všechny ostatní plochy tento bit **nulují**
  2. s **vypnutým testem hloubky** se všichni potenciální vrhači stínu promítnou **do roviny příjemce**
     - musí se nastavit **speciální promítací matice**
     - stínové plošky se kreslí pouze na místa, kde je **nastaven** daný bit šablony (z prvního průchodu)
     - používají-li se **poloprůhledné stínové plošky**, je nežádoucí, aby se dvě překreslily přes sebe – i tady pomůže **šablona** (první stínová ploška ji zpátky vynuluje)

Bump-mapping

- speciální texturovací technika – vytváří **dojem nerovného povrchu**
  - nahrazuje velmi komplikovanou mikro-geometrii
  - modifikace (modulace) **normálového vektoru**
  - pozorovatel získává většinu informace o **struktuře povrchu těles** ze stínování (odlesků)
- implementace předpokládá spolupráci rasterizačního hardware
  - alespoň kombinace více textur
  - lepší je počítat složky osvětlení přímo **v jednotlivých pixelech** – spolupráce s Phongovým stínováním

Situace

- zadání: diferenční funkce **b(u,v)** definující odchylku simulovaného povrchu od jeho jednodušší aproximace



Výpočet odchýlené normály

- simulovaný bod na povrchu hrbolatého tělesa

$$P'(u,v) = P(u,v) + b(u,v) \cdot \frac{N}{|N|}$$

- odchýlení normálového vektoru

$$N' = N - \frac{\frac{\partial b}{\partial u}(u,v) \cdot U + \frac{\partial b}{\partial v}(u,v) \cdot V}{|N|}$$

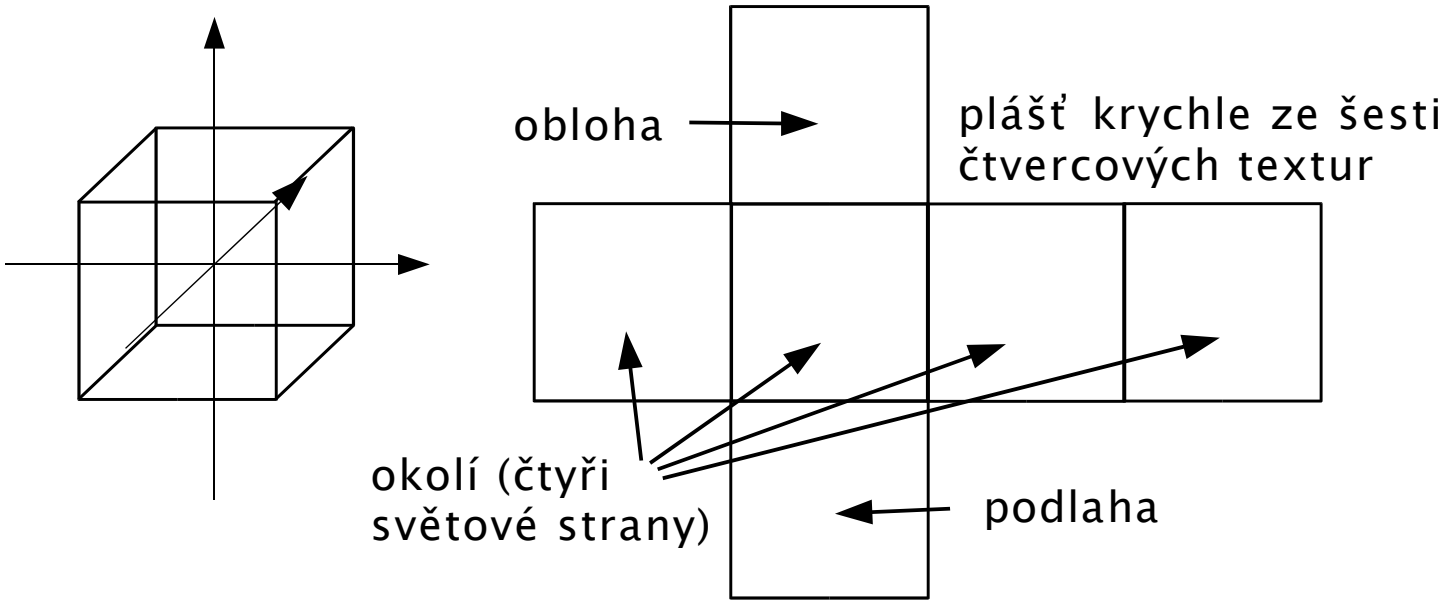
- v podstatě je potřeba znát pouze **parciální derivace** funkce **b(u,v)** – a ne její hodnotu ...

Další texturovací techniky

- co může být **definičním oborem textury** ?
  - dvojrozměrné souřadnice na povrchu tělesa **[u, v]** – klasický přístup
  - trojrozměrné souřadnice **[x, y, z]** – tzv. „**prostorová textura**”
    - vnitřní struktura materiálu (dřevo, mramor, ...)
  - **normálový vektor N** – „***environment mapping***”
    - předem spočítaná data (CPU i spolupráce s GPU) se do rasterizátoru předávají pomocí zvláštní „textury”
    - **odraz okolní scény** na lesklém povrchu tělesa
    - **mapa osvětlení** (dopředu spočítané světlo od více zdrojů, od plošných zdrojů, složitějším světelným modelem, ...)

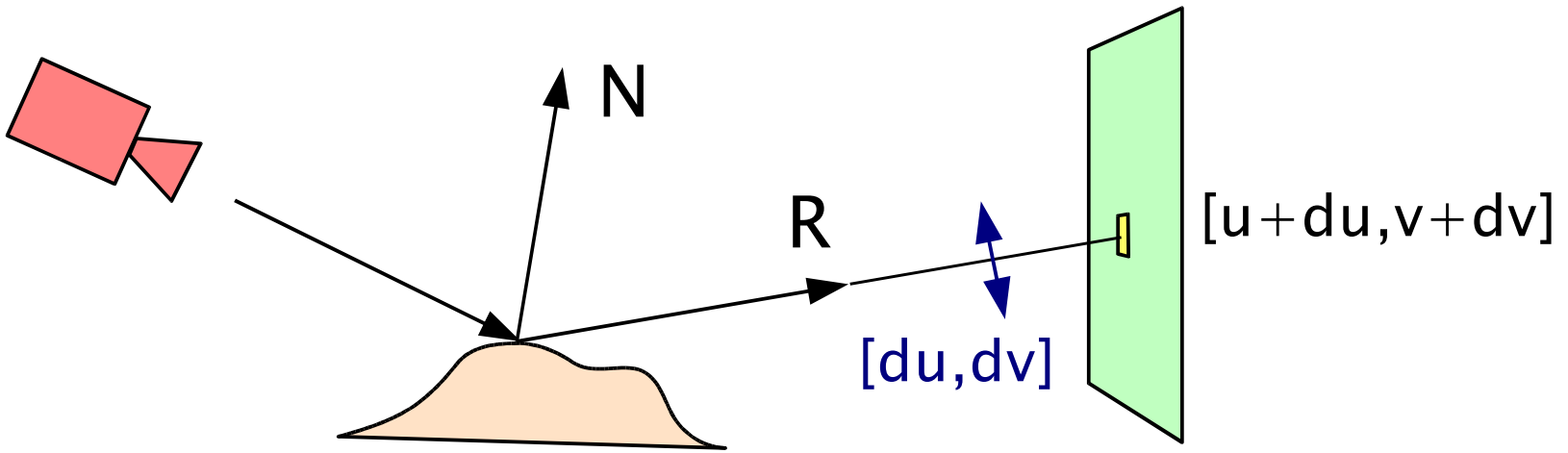
Environment-mapping (EM)

- normálový vektor **N** se převádí do
  - **sférických souřadnic** – složitější příprava dat
  - šesti čtvercových **stěn krychle** – „***cubic mapping***"
    - vstupní data se dají získat dynamicky pomocí GPU !



Environment map bump-mapping

- sférické i kubické mapování umí dělat současné GPU
- v **texturových souřadnicích** „*environment-mapy*" lze dělat elegantně též „***bump-mapping***"
  - místo změny normály se **modifikuje [u, v]** pro EM
  - také tuto techniku umí dnešní rasterizační jednotky



Akumulační buffer

- vedlejší barevný buffer, do kterého lze přičítat aktuální nakreslené barevné buffery
- používá se pro různé efekty **„rozmazání" obrazu**
  - **rozmazání** předmětu rychlým **pohybem**
    - napodobení reálné uzávěrky fotoaparátu/kamery
    - pohybující předměty se nakreslí **v několika časových okamžicích**, akumulační buffer obrázky zprůměruje
  - **hloubka ostrosti kamery**
    - skutečná optika kreslí úplně ostře jen v **rovině zaostření**
    - simulace: celá scéna se nakreslí několikrát, vždy s maličko modifikovaným úhlem pohledu (změna projekční matice)

Literatura

- Tomas Akenine-Möller, Eric Haines: ***Real-time rendering, 2nd edition***, A K Peters, 2002
- J.D. Foley, A. van Dam, S.K. Feiner, J.H. Hughes: *Computer Graphics: Principles and Practice, Second Edition in C*, Addison-Wesley, 1996
- Alan Watt, Mark Watt: Advanced Animation and Rendering Techniques – Theory and Practice, Addison-Wesley, 1992
- Jiří Žára, Bedřich Beneš, Petr Felkel: *Moderní počítačová grafika*, Computer press, 1998
- Peter Kovach: *Inside Direct3D*, Microsoft Press, 2000

On-line zdroje

- **NVidia** pro vývojáře: *http://developer.nvidia.com/*
- **ATI** pro vývojáře: *http://www.ati.com/developer/*
- **OpenGL** konsorcium: *http://www.opengl.org/*
- **Microsoft** o DirectX: *http://msdn.microsoft.com/directx/*
- e-zin pro vývojáře **her**: *http://www.gamasutra.com/*
- populární **citační databáze** (obsahuje často i plné verze článků): *http://citeseer.nj.nec.com/*