

KATEDRA POČÍTAČOVÉ GRAFIKY A INTERAKCE

Materiály, Textury, Shadery

Jiří Bittner

Obsah přednášky

- Výpočet osvětlení – Úvod GAE 10.1.2.1-2
- Osvětlovací modely GAE 10.1.3.1-2
- Světelné zdroje GAE 10.1.3.3
- Textury GAE 10.1.2.5
- Shadery GAE 10.2.1-10.2.7

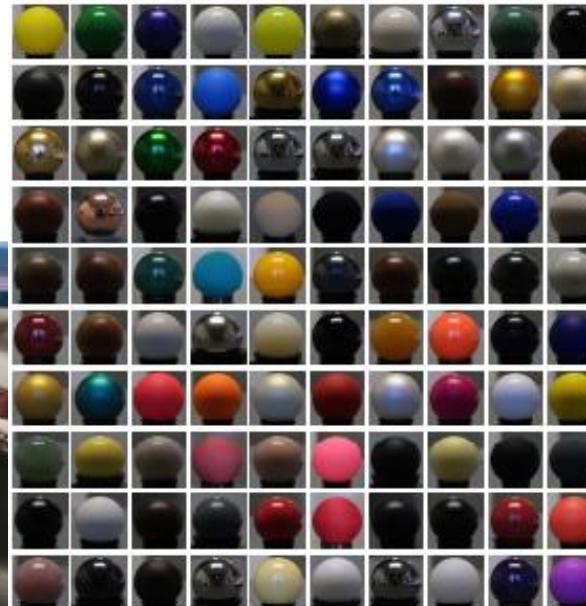
Shading - Motivace



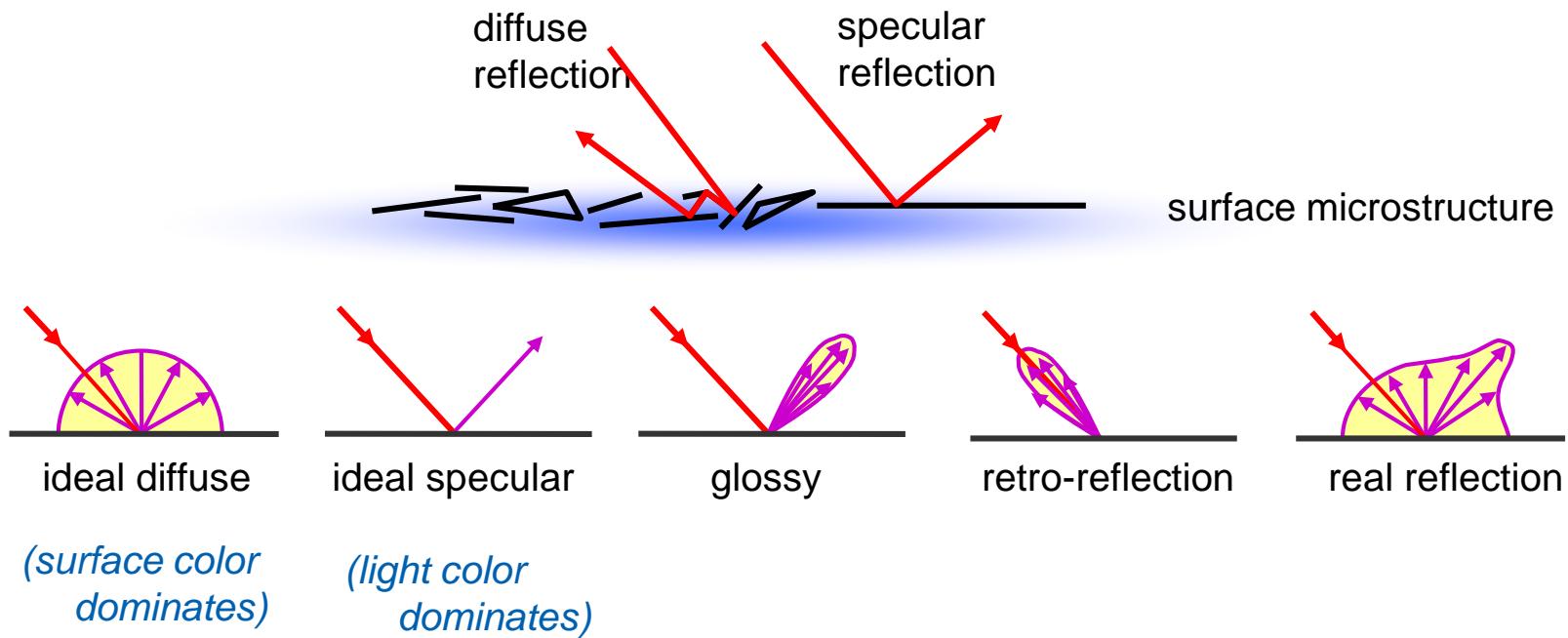
Image with post-processing (depth of field, bloom, motion blur, colorize, ILR)

Vzhled povrchu objektů

- Vzhled povrchu objektů
 - Geometrie (tvar objektů)
 - Materiály (odrazivost povrchů)
 - Světelné zdroje



Odraz světla na povrchu tělesa



Odraz světla na povrchu tělesa

- BRDF (bidirectional reflectance distribution function) = pravděpodobnostní dvousměrová funkce odrazu

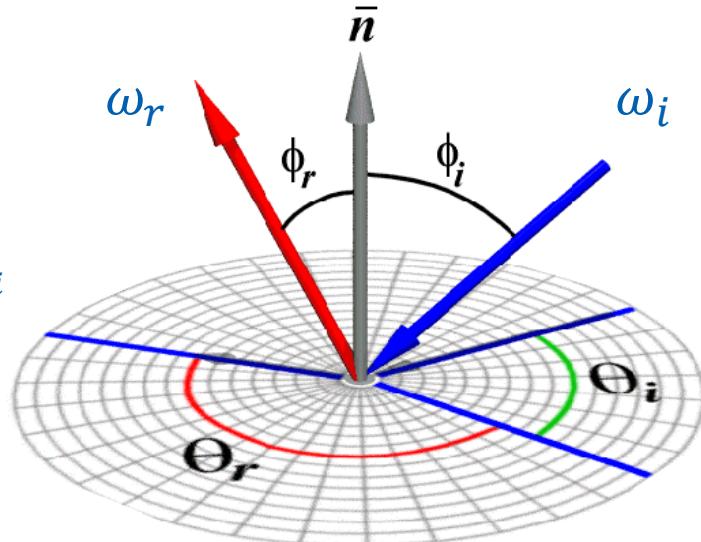
$$f_r(\omega_i, \omega_r) = \frac{dL_r(\omega_r)}{L_i(\omega_i) \cos \Phi_i d\omega_i}$$

$$dL_r(\omega_r) = f_r(\omega_i, \omega_r) L_i(\omega_i) \cos \Phi_i d\omega_i$$

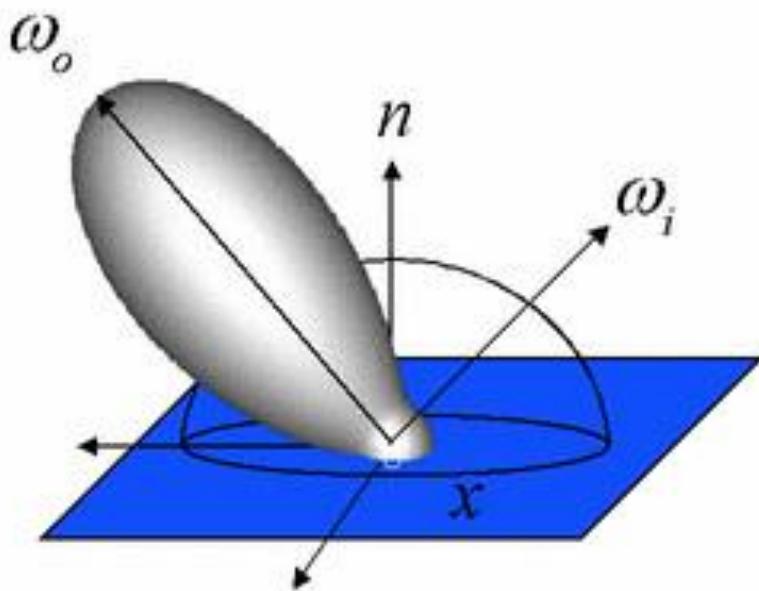
$$L_r(\omega_r) = \int_{\Omega} f_r(\omega_i, \omega_r) L_i(\omega_i) \cos \Phi_i d\omega_i$$

$L_i(\omega_i)$... příchozí radiance ze směru ω_i $[\frac{W}{m^2 sr^{-1}}]$

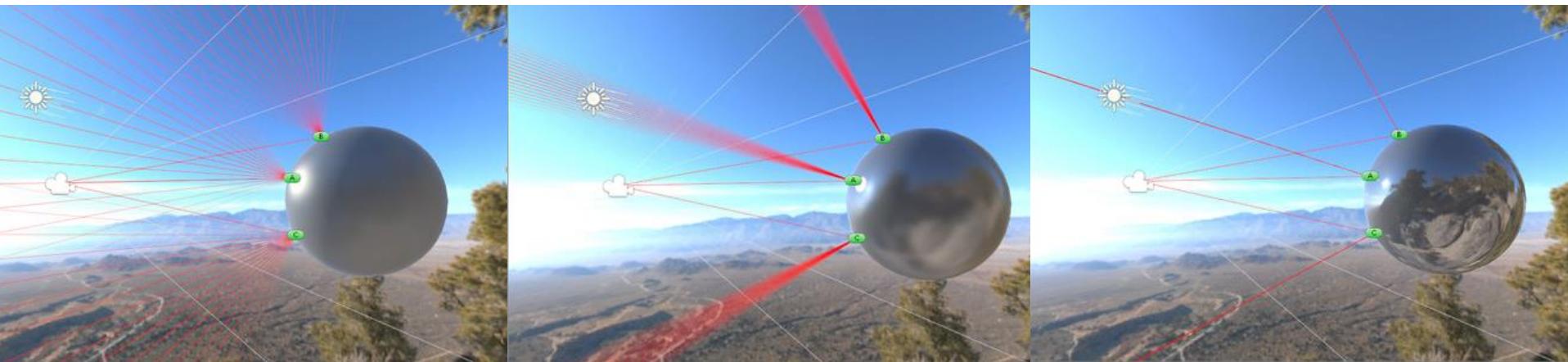
$L_r(\omega_r)$... odražená radiance do směru ω_r $[\frac{W}{m^2 sr^{-1}}]$



BRDF – příklad: lesklý povrch



BRDF příklad: Odraz na lesklé kouli



Osvětlovací modely

- Osvětlovací (BRDF) model = modelování BRDF
- Empirické
 - Phong, Blinn, Lafortune, Ward, ...
- Fyzikálně založené
 - Cook-Torrance, Torrance-Sparrow, Oren-Nayar, ...
- Naměřené BRDF hodnoty
 - pro použití v aplikaci nutno komprimovat

Phongův osvětlovací model

- Bui-Tuong Phong, dizertace 1973

$$I = I_A + I_D + I_S$$

- I_A okolní světlo (ambient)
- I_D difúzní odraz (diffuse)
- I_S zrcadlový odraz (specular)
- Výpočty intenzity (barvy) po složkách r, g, b

Ambientní složka

- Vše směrové osvětlení (světelny šum)
 - Konstantní pro celou scénu
 - Napodobuje globální osvětlení

$$I_A = C_A \cdot C_D \cdot k_A$$

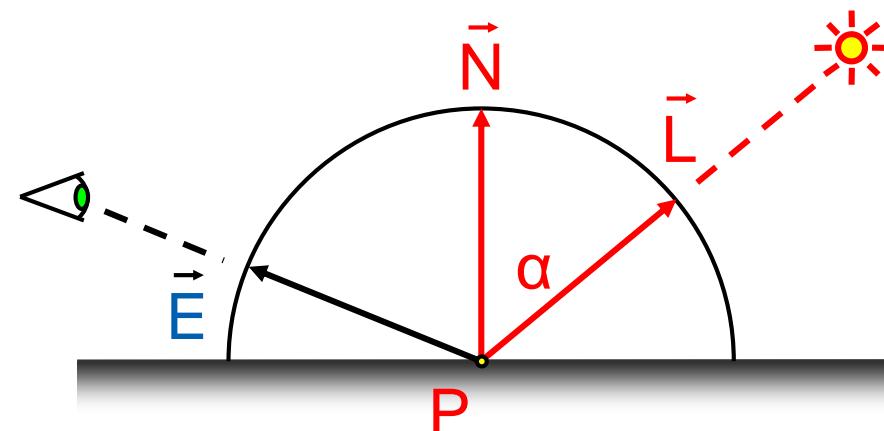
- C_A – barva ambientního světla
- C_D – barva povrchu (shodná i pro difúzní složku)
- $k_A \in \langle 0,1 \rangle$ koef. ambientního odrazu

Difúzní složka

- Odpovídá ideálně matnému tělesu
- Závisí na úhlu mezi L a N

$$I_D = C_L \cdot C_D \cdot k_D \cdot \cos(\alpha)$$

- C_L barva světla
- C_D barva povrchu
- $k_D \in <0,1>$ koef. difuzního odrazu
- $\cos(\alpha) =$ skalárni součin L a N

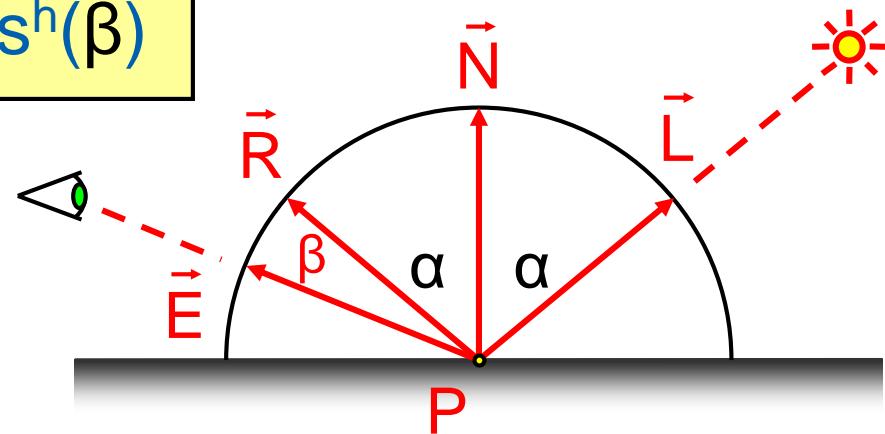


Zrcadlová složka

- Odpovídá ideálně odrazivému tělesu
- Závisí na úhlu mezi E a R

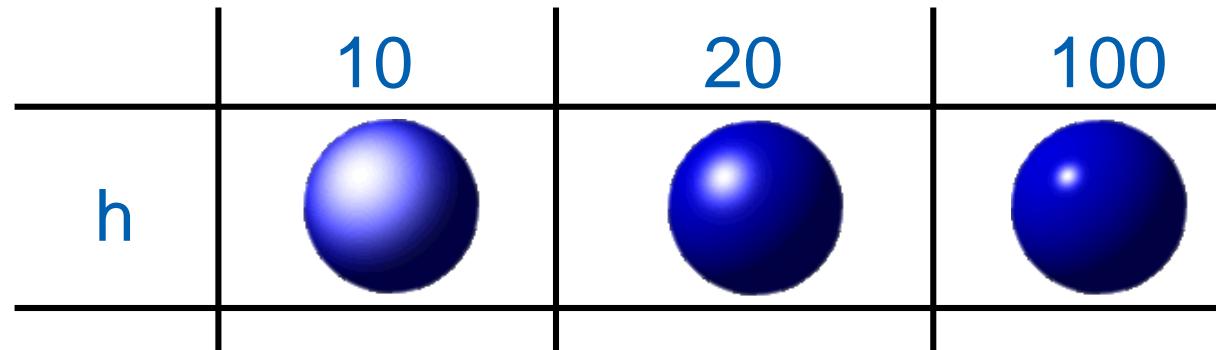
$$I_S = C_L \cdot C_S \cdot k_S \cdot \cos^h(\beta)$$

- barva lesklého povrchu C_S
- $k_S \in <0,1>$
- $h \in <1, \infty>$, ostrost odrazu (shininess)
- $\cos(\beta) = \text{skalární součin } E \text{ a } R$



$$R = 2(LN)N-L$$

Shininess - Phong

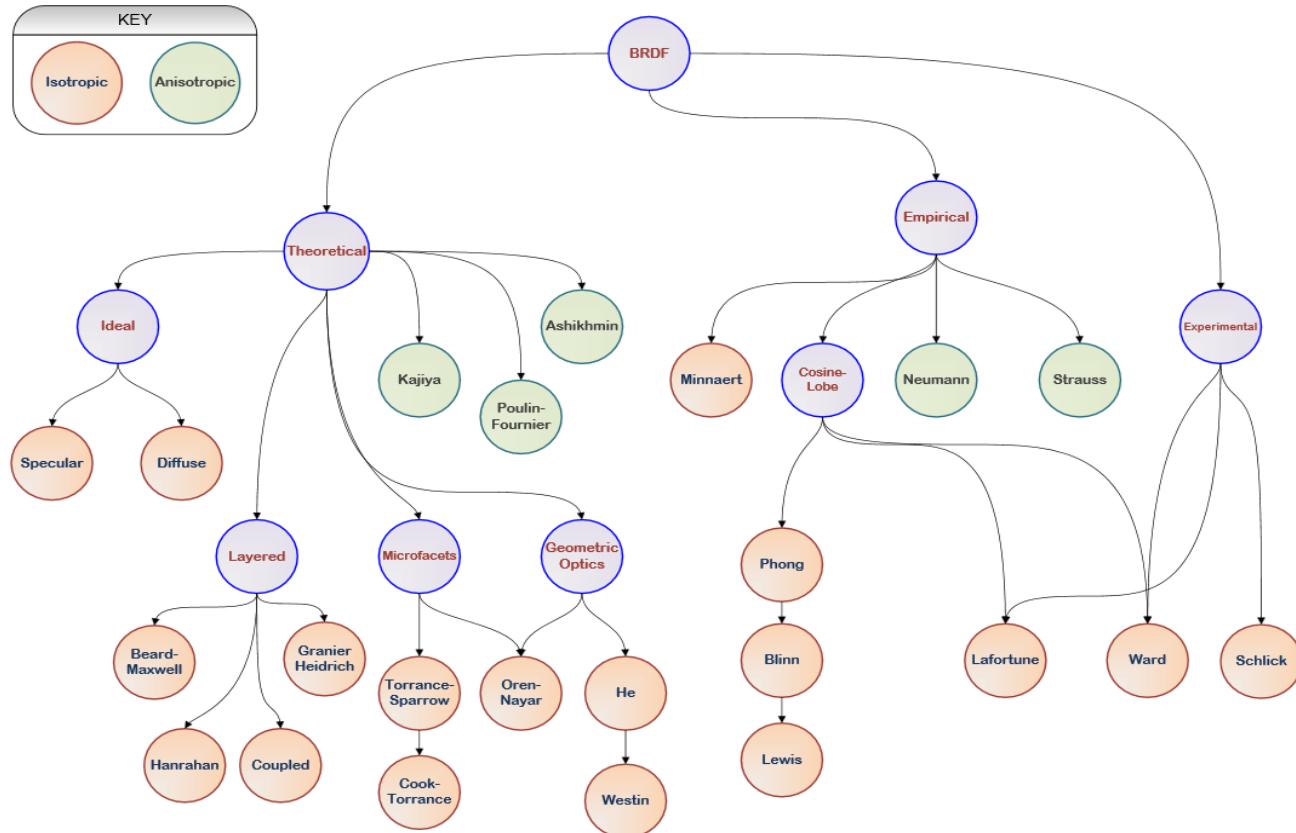


$$k_A = 0.5$$

$$k_D = 0.5$$

$$k_S = 1$$

Jiné BRDF modely



PBR Model – Unity Standard Shader

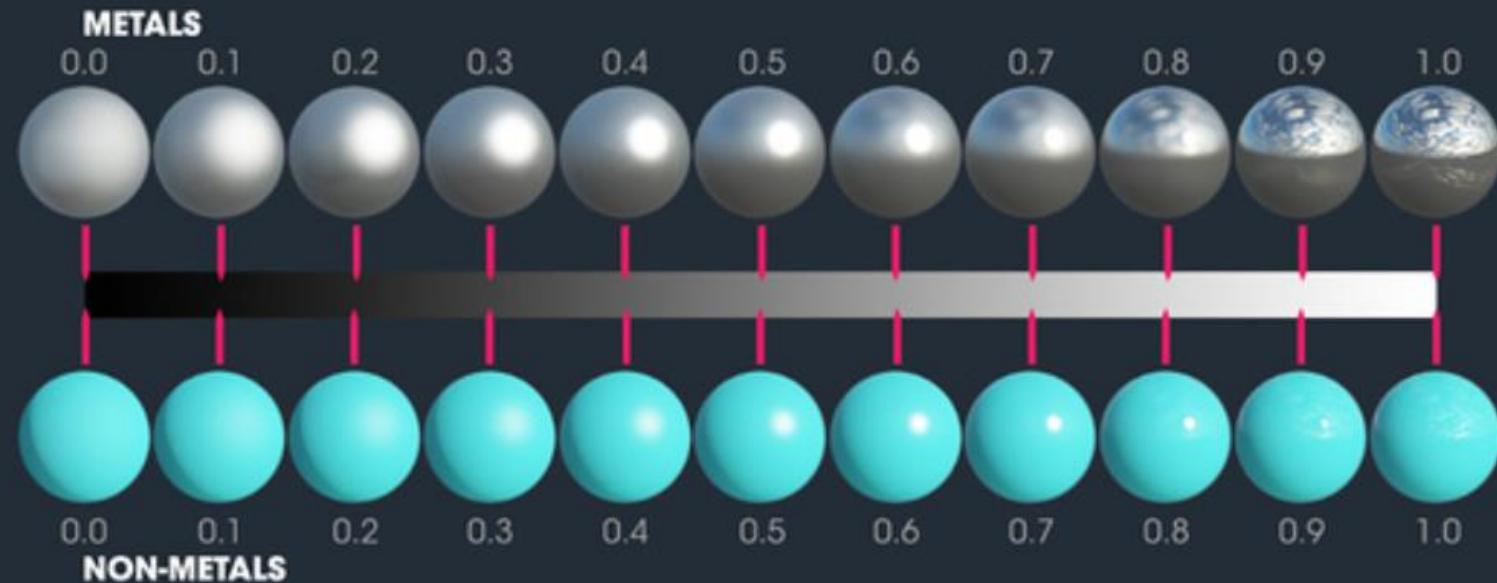
- Albedo (Barva)
 - Metallic (Kovovost)
 - Smoothness (Hladkost)
 - Emission (Vyzařování)
-
- Normal map, height map, occlusion



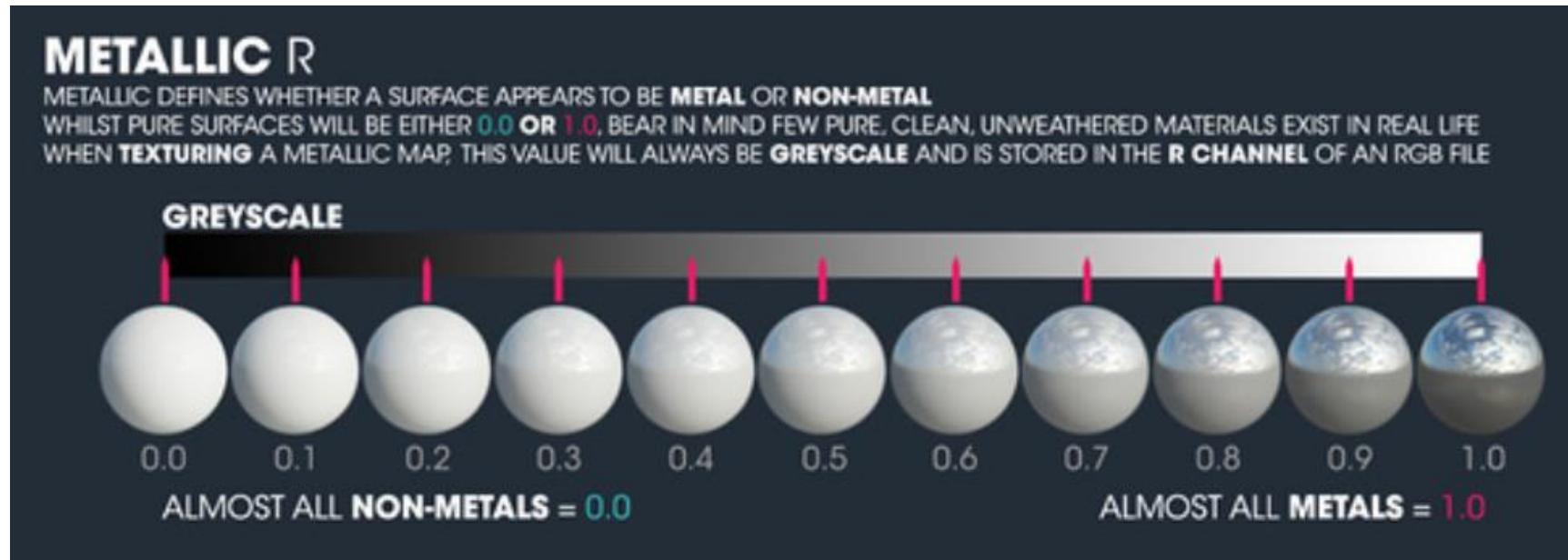
PBR Model – Unity Standard Shader

SMOOTHNESS A

SMOOTHNESS DEFINES THE PERCEIVED **GLOSSINESS** OR **ROUGHNESS** OF A SURFACE FOR TEXTURES, THIS IS STORED AS THE ALPHA CHANNEL OF THE **METALLIC MAP**



PBR Model – Unity Standard Shader

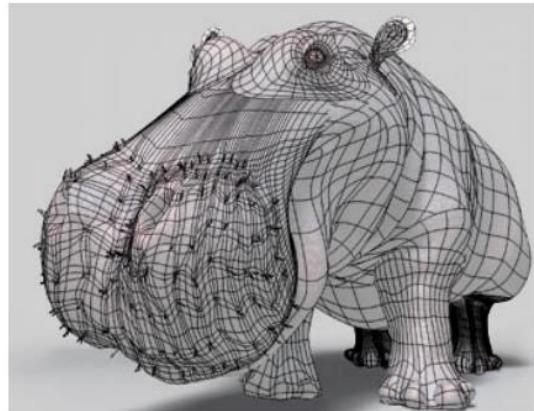


- 0: difúzní odraz + nezabarvený spekulární odraz s malou zákl. odrazivostí ($F_0 \sim [0.04, 0.04, 0.04]$)
- 1: pouze spekulární odraz, kde albedo je základní odrazivost F_0

Obsah přednášky

- Výpočet osvětlení – Úvod GAE 10.1.2.1-2
- Osvětlovací modely GAE 10.1.3.1-2
- Světelné zdroje GAE 10.1.3.3
- Textury GAE 10.1.2.5
- Shadery GAE 10.2.1-10.2.7

Proč textury?



model



shading



shading+texture

Model courtesy of Jeremy Birn

<http://www.3drender.com/jbirn/hippo/index.html>

Textury - úvod

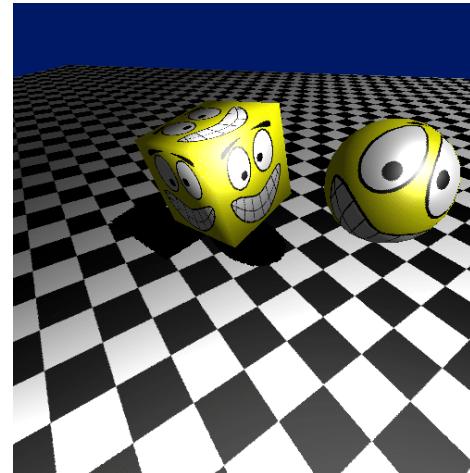
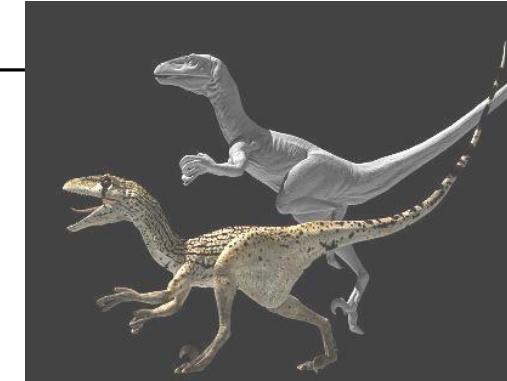
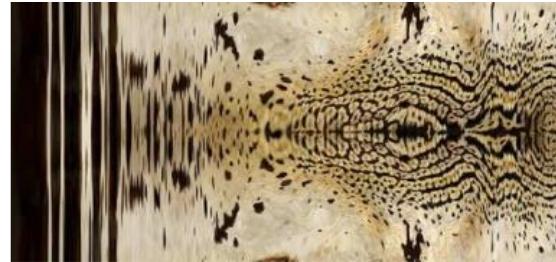
- Levný způsob zvýšení vizuální kvality 3D modelu
- Model makrostruktury povrchu

Dílčí úlohy:

1. Definice textury
 2. Aplikace (nanesení) textury
 - Mapování na objekt
- Použití textury
 - Modulace barvy, normály, ...

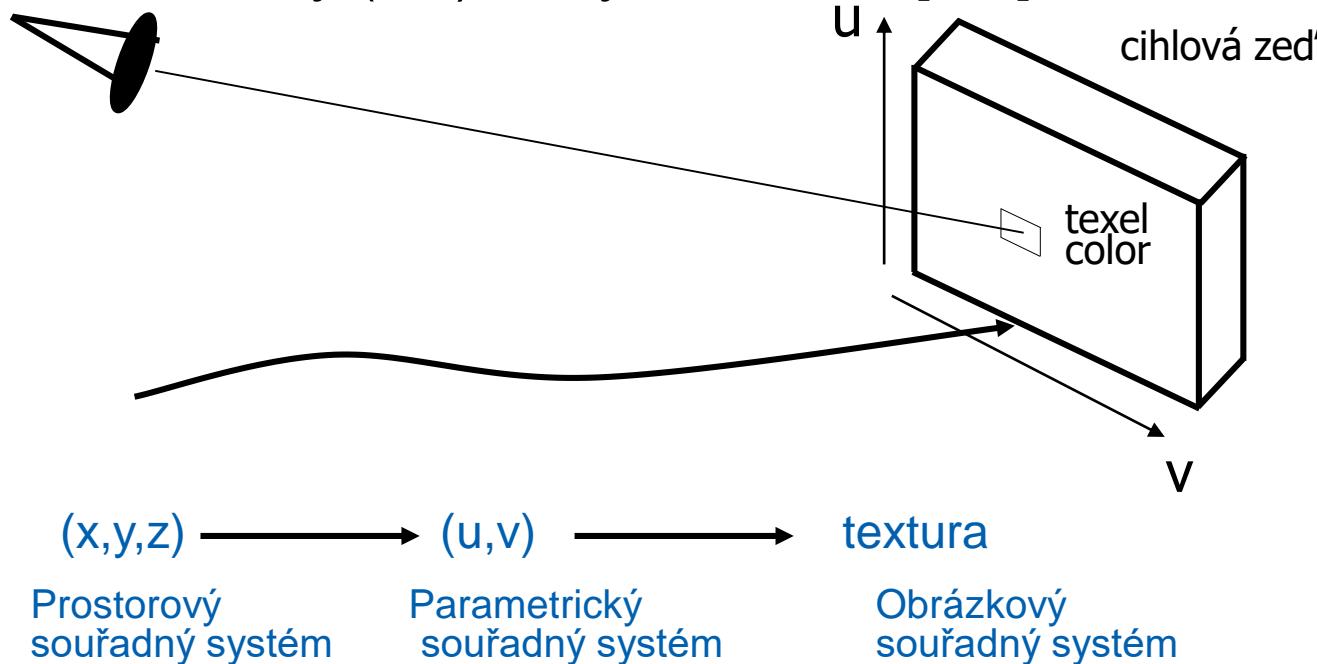
Zdroj dat textury

- Obrázek
 - Data uložená ve 2D matici
- Procedurální textury
 - Jednoduché funkce (např. šachovnice, šrafování)
 - Šum (noise functions)



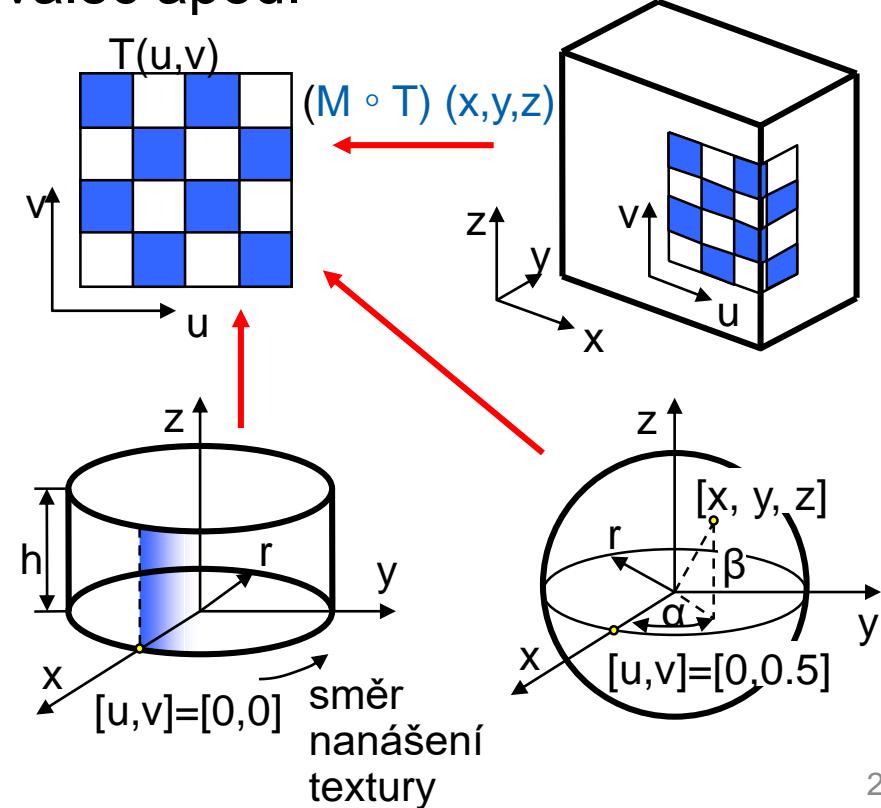
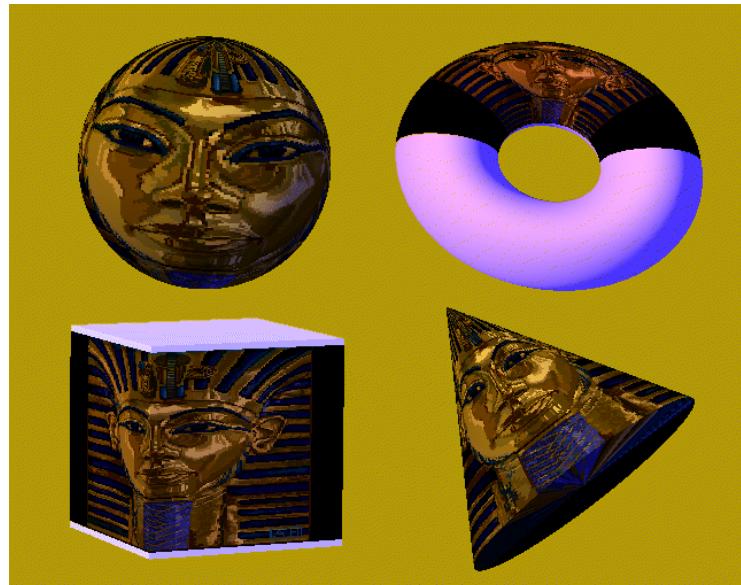
Typické použití textury

- Souřadnice textury (u,v) vždy v rozsahu $[0-1]^2$



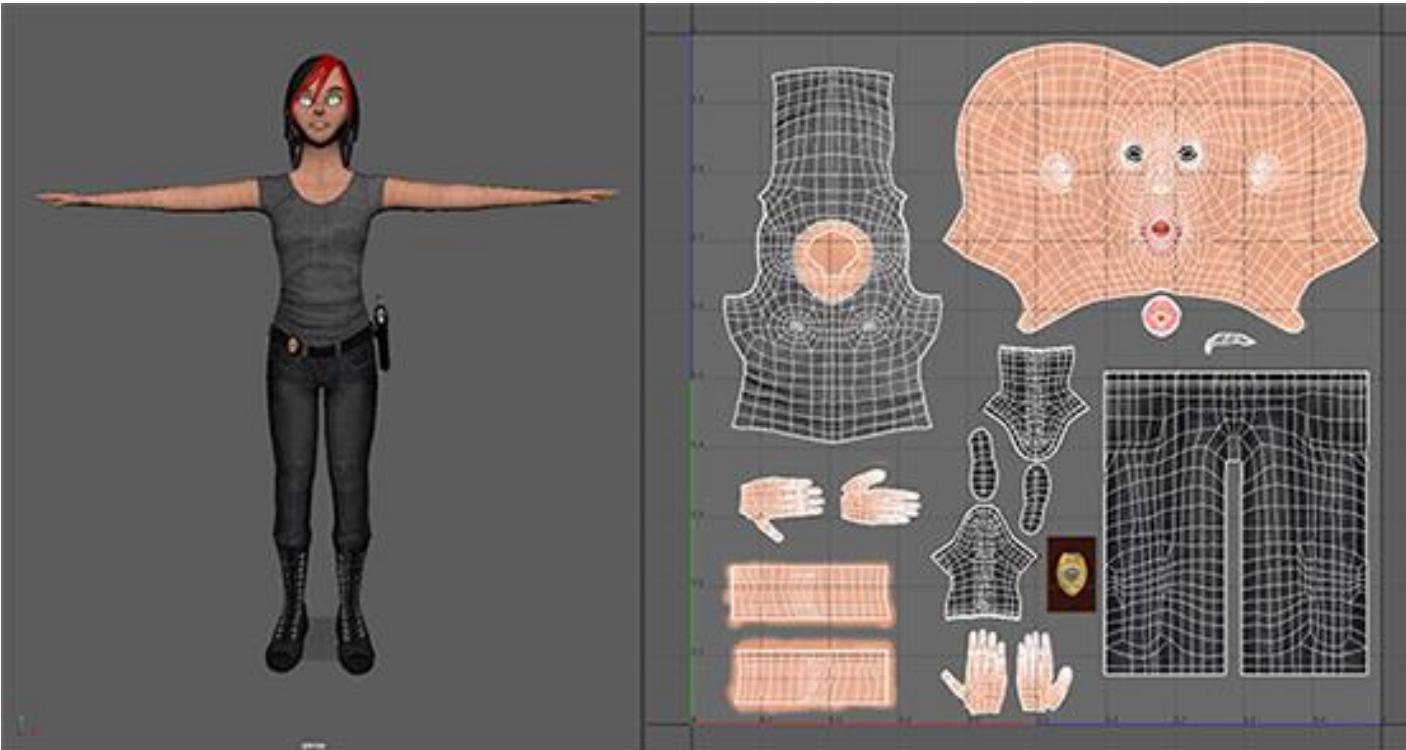
Mapování textury

- Koule, toroid, krychle, kužel, válec apod.

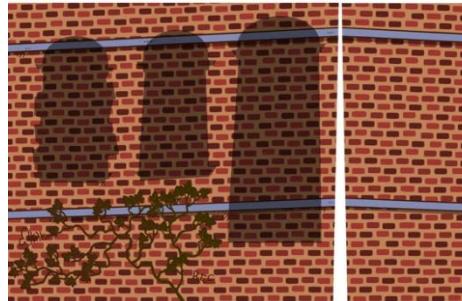
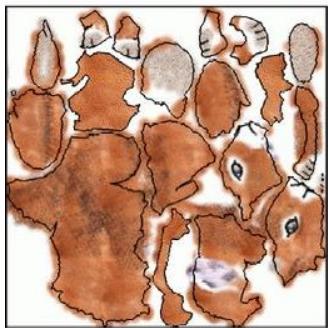


Parametrizace trojúhelníkové sítě

- Unwrap: mesh parametrization → UV map



Atlas textur



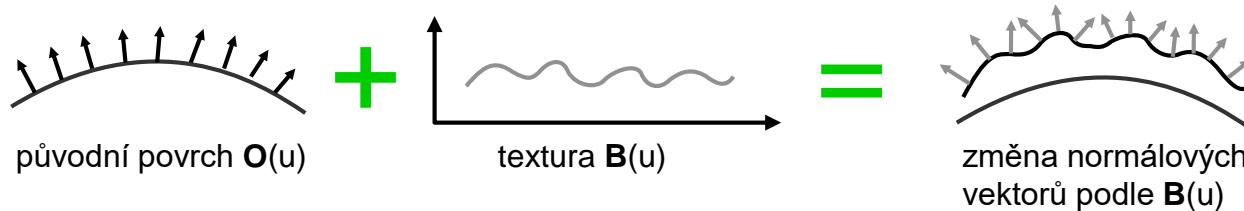
Aplikace textury - Modulace

- Barva - difúzní odraz
- Barva - spekulární odraz (*gloss mapping*)
- **Směr normál** (*bump mapping*)
- Tvar povrchu (*displacement mapping*)
- **Přicházející světlo** (*reflection mapping, environment mapping*)
- Průhlednost (*alpha mapping*)

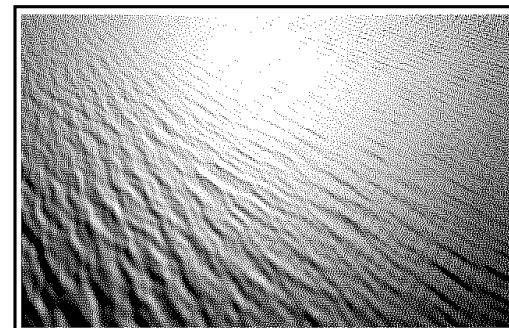
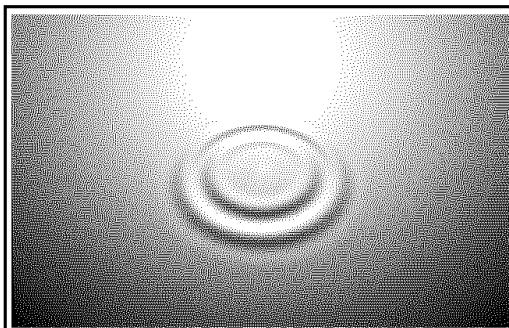


Modulace normál

■ Hrbolaté textury (*bump mapping*)

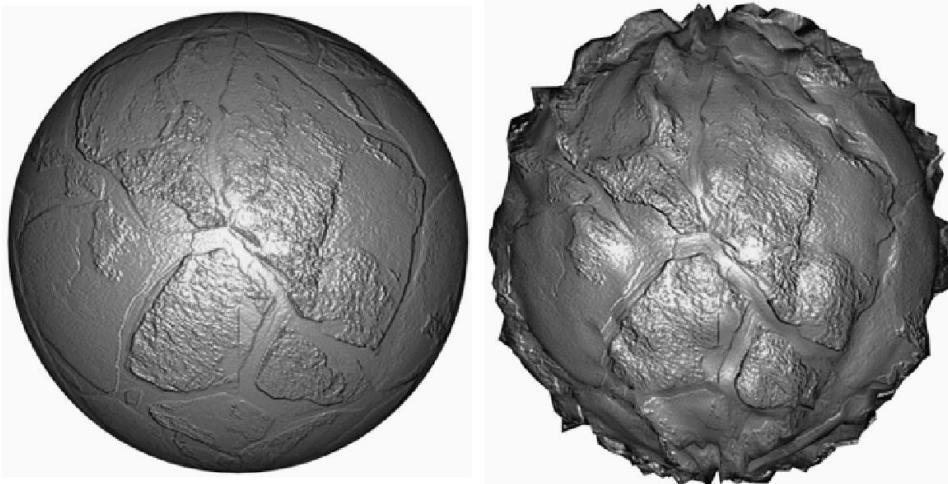


- Vstup: obraz v šedi (heightmap) => derivací směry normál
- Vstup: RGB (normal map) => odchylka normál

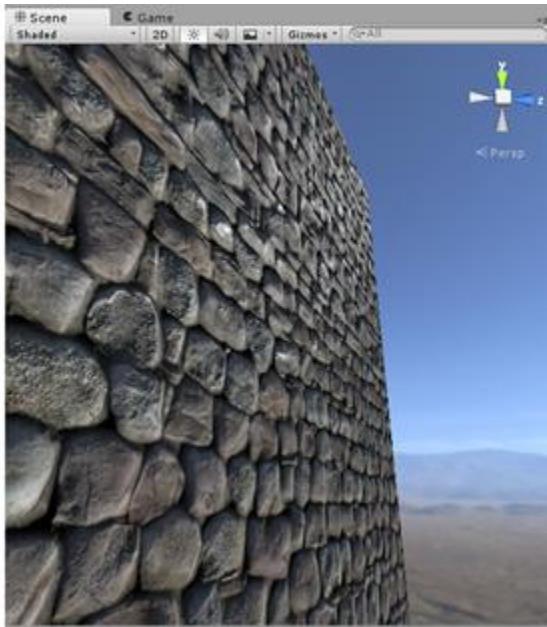


Displacement mapping

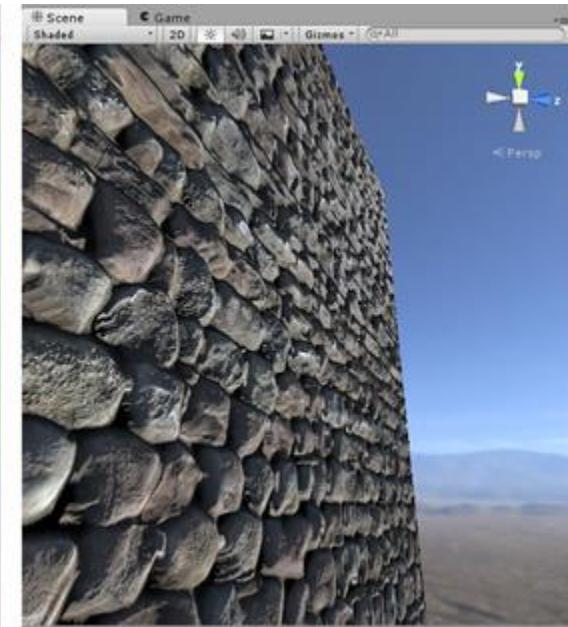
- Změna povrchu proti původnímu podle normály



Unity – normal map, height map



normal map

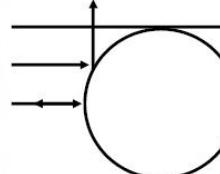


height map
(parallax mapping)

Mapa okolí (environment mapping)

- Environment map, sky box, light probe, reflection probe,...
- Používá směr odraženého paprsku
 - Levná alternativa k vrhání paprsku
- Dvě fáze
 - Vytvoření mapy okolí (dle očekávané pozice pozorovatele)
 - Použití mapy okolí při syntéze obrazu
- Typ pomocného objektu
 - koule, krychle, duální paraiboloid

Miller and Hoffman, 1984



Mapa okolí vs. sledování paprsku

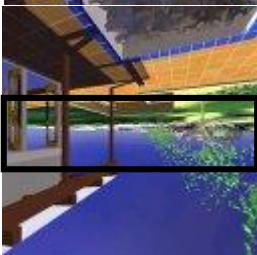


Ray Traced



Environment Map

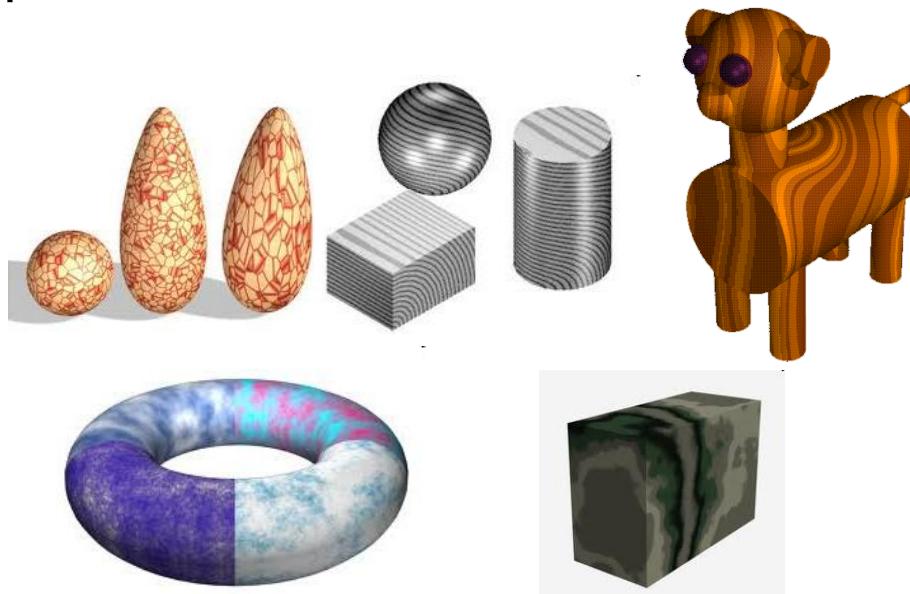
Cubemap – dynamické generování



- Reflection probes

3D textury

- Reprezentována 3D mřížkou či funkcí
- Zachycuje vnitřní materiál (dřevo, mramor)
- Jednoduché mapování

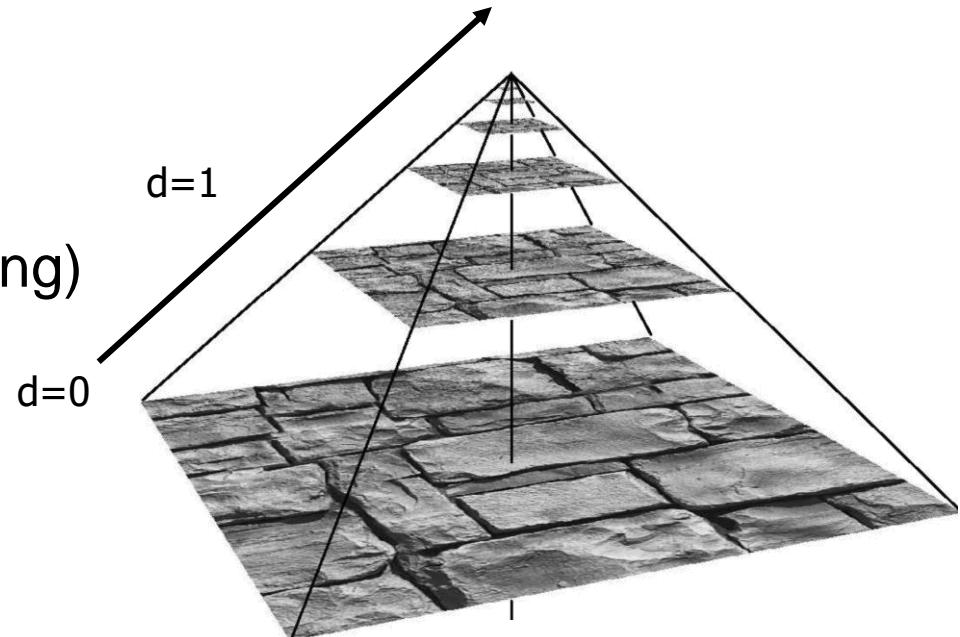


Vzorkování a filtrování textur

- Zvětšení rozlišení (magnification)
 - Jeden texel se promítá na více pixelů
 - A) Nearest neighbor
 - B) Bilineární interpolace
 - C) Bikubická interpolace
- Zmenšení rozlišení (minification)
 - Více texelů na jeden pixel výsledného obrázku)
 - *Mipmapa a trilineární interpolace*

Mipmapping

- *Mipmapping* : více rozlišení v jednom obrazu (*multum in parvo*)
 - Pyramida textur
 - Výběr úrovně podle vzdálenosti
 - Použití trilineární interpolace
- Výhody:
 1. Vyhlazuje textury (antialiasing)
 2. Urychljuje vykreslování
 - Koherentní přístup do paměti
- Nárůst paměti ~ 33%



Obsah přednášky

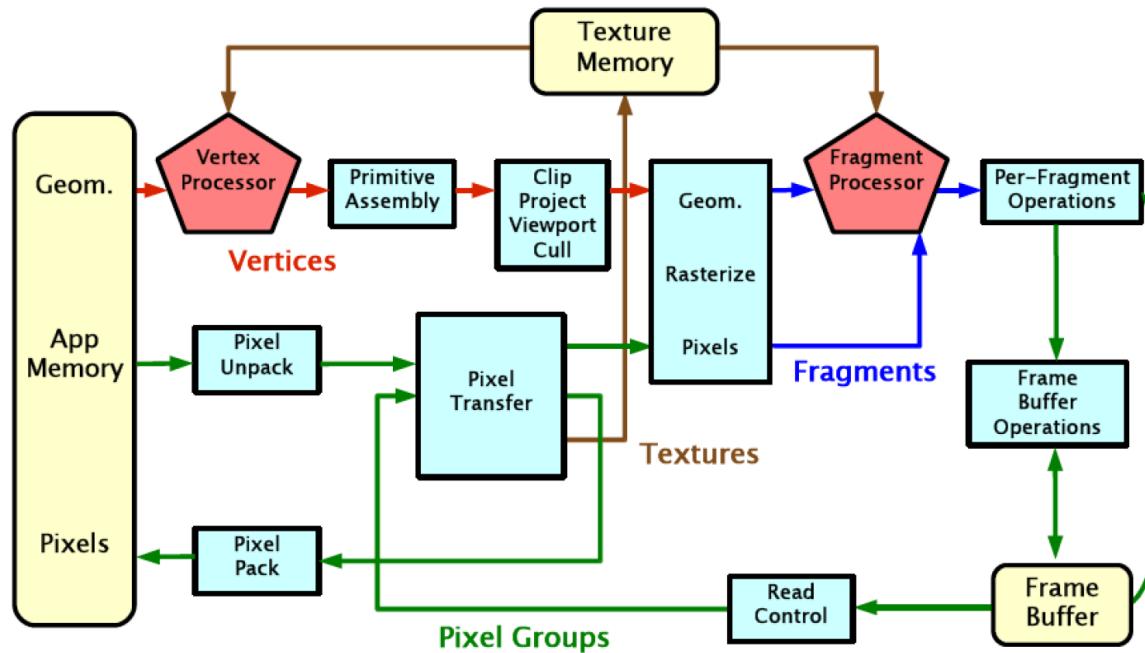
- Výpočet osvětlení – Úvod *GAE 10.1.2.1-2*
- Osvětlovací modely *GAE 10.1.3.1-2*
- Světelné zdroje *GAE 10.1.3.3*
- Textury *GAE 10.1.2.5*
- Shadery *GAE 10.2.1-10.2.7*

Shadery

- Specializovaný program pro výpočet osvětlení
 - Pixar: RenderMan, 1988
- Shadery používány v řadě rendererů, modelářů
- Shadery mohou realizovat více než shading
 - geometrické transformace, teselace, ray tracing, ...
- Hardwarová podpora
 - Programovatelné GPU
 - GLSL (OpenGL)
 - HLSL (DirectX)
 - Cg (NVIDIA, OpenGL i DirectX)

Programovatelný zobrazovací řetězec

- Vertex + Fragment shaders



Vertex shader

- Programovatelné zpracování vrcholů
 - Transformace vrcholů a normál
 - Výpočet/transformace texturovacích souřadnic
 - Výpočet osvětlení (per-vertex lighting)
 - Nastavení materiálových konstant
- Může libovolně měnit polohu vrcholu, ale
 - Nemá informaci o primitivu
 - neví nic o případných sousedech
 - Nemůže rušit ani přidávat vrcholy
- Výstup VS je vstupem FS

Fragment shader

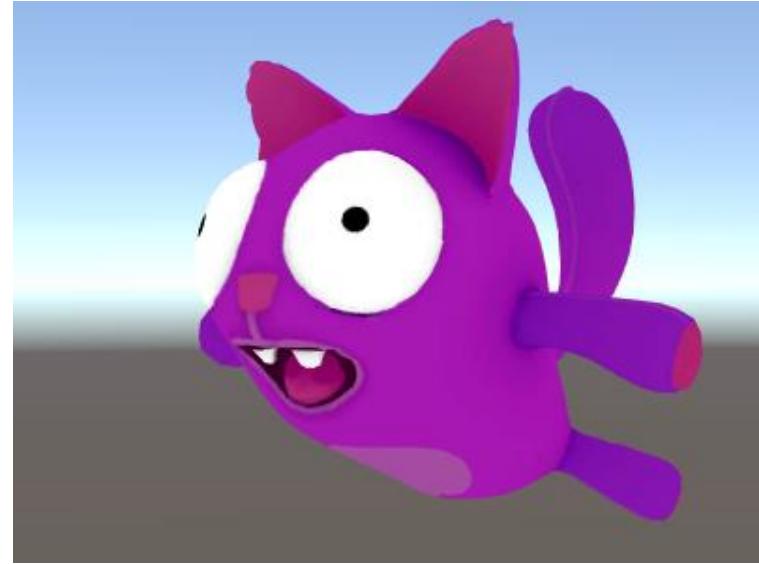
- Rasterizátor generuje fragmenty
- Programovatelné zpracování fragmentů
 - Musí vypočítat finální barvu fragmentu (pixelu)
 - Čtení z textur, výpočet mlhy, osvětlovacího modelu
- Nemůže měnit (x,y) souřadnice fragmentu!
- Může měnit hloubku fragmentu (depth)
 - ale pozor, může to mít dopad na efektivitu!
- Může fragment zahodit (discard)

Spolupráce VS a FS

- Vertex shader (VS)
 - Povinně produkuje pouze 3D souřadnice vrcholu (v clip space)
 - aby bylo možné 3D primitiva rasterizovat
 - Ostatní data jsou nepovinná (texturovací souřadnice, barva, ...)
- Rasterizace
 - Data se automaticky interpolují do fragmentů
 - Perspektivně korektní interpolace
 - FS je dostává na vstupu a dále zpracovává
- Fragment shader (FS)
 - Čte interpolovaná data
 - Vypočítá výslednou barvu fragmentu

Unity Shaders – Příklad: vertex / fragment shader

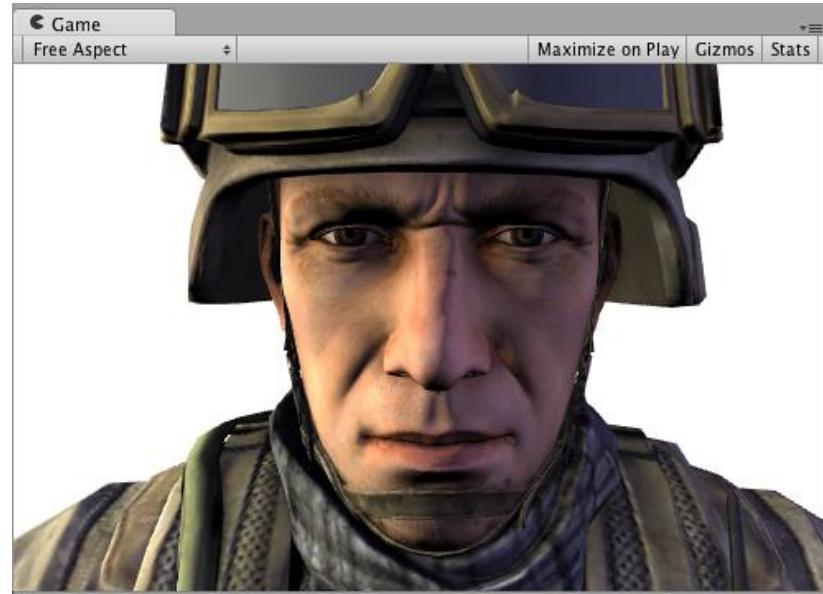
```
Shader "Example/Textured Colored" {
    Properties {
        _Color ("Main Color", Color) = (1,1,1,0.5)
        _MainTex ("Texture", 2D) = "white" {}
    }
    SubShader {
        Pass {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag
            #include "UnityCG.cginc"
            fixed4 _Color;
            sampler2D _MainTex;
            struct v2f {
                float4 pos : SV_POSITION; float2 uv : TEXCOORD0;
            };
            float4 _MainTex_ST;
            v2f vert (appdata_base v) {
                v2f o;
                o.pos = UnityObjectToClipPos(v.vertex); o.uv = TRANSFORM_TEX (v.texcoord, _MainTex); return o;
            }
            fixed4 frag (v2f i) : SV_Target {
                fixed4 texcol = tex2D (_MainTex, i.uv); return texcol * _Color;
            }
        ENDCG
    }
}
```



Explicitně počítá výsledek,
t.j. barvu fragmentu!

Unity Shaders – Příklad: surface shader

```
Shader "Example/Diffuse Bump" {
    Properties {
        _MainTex ("Texture", 2D) = "white" {}
        _BumpMap ("Bumpmap", 2D) = "bump" {}
    }
    SubShader {
        Tags { "RenderType" = "Opaque" }
        CGPROGRAM
        #pragma surface surf Lambert
        struct Input {
            float2 uv_MainTex;
            float2 uv_BumpMap;
        };
        sampler2D _MainTex;
        sampler2D _BumpMap;
        void surf (Input IN, inout SurfaceOutput o) {
            o.Albedo = tex2D (_MainTex, IN.uv_MainTex).rgb;
            o.Normal = UnpackNormal (tex2D (_BumpMap, IN.uv_BumpMap));
        }
        ENDCG
    }
    Fallback "Diffuse"
}
```



Mění parametry BRDF modelu!

Unity Shaders – Příklad: vertex + surface shader

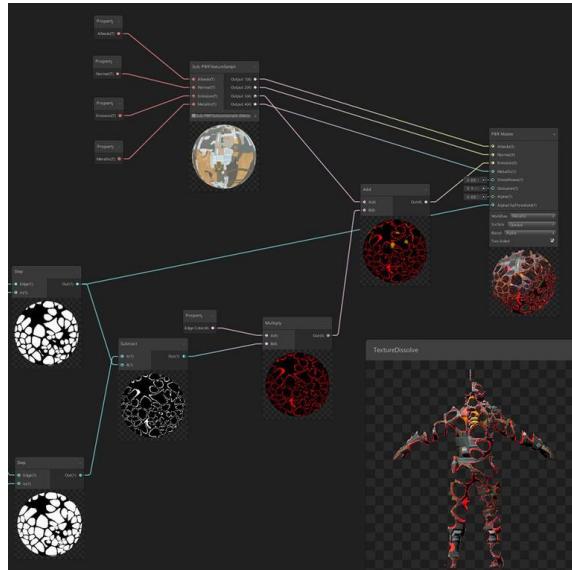
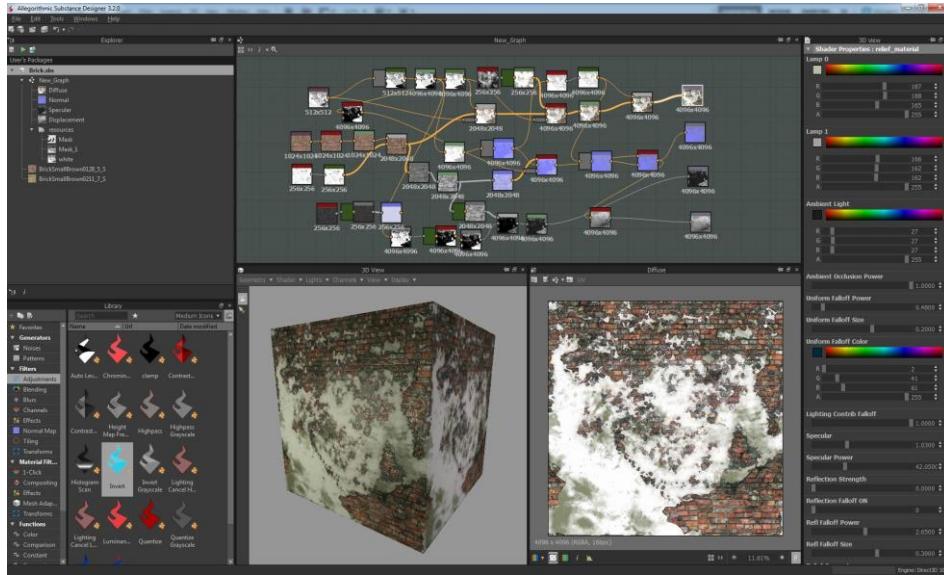
```
Shader "Example/Normal Extrusion" {
    Properties {
        _MainTex ("Texture", 2D) = "white" {}
        _Amount ("Extrusion Amount", Range(-1,1)) = 0.5
    }
    SubShader {
        Tags { "RenderType" = "Opaque" }
        CGPROGRAM
        #pragma surface surf Lambert vertex:vert
        struct Input {
            float2 uv_MainTex;
        };
        float _Amount;
        void vert (inout appdata_full v) {
            v.vertex.xyz += v.normal * _Amount;
        }
        sampler2D _MainTex;
        void surf (Input IN, inout SurfaceOutput o) {
            o.Albedo = tex2D (_MainTex, IN.uv_MainTex).rgb;
        }
    ENDCG
}
Fallback "Diffuse"
}
```



Mění geometrii + parametry BRDF modelu!

Vizuální programování shaderů

- Shader networks / graphs
 - Znovupoužitelnost / modulárnost, srozumitelné i pro „umělce“



Substance designer (www.allegorithmic.com/products/substance-designer) Unity Shader Graph

Způsoby vykreslování scény

- Dopředné vykreslování (forward rendering)
- Odložené vykreslování (deferred rendering)

Forward rendering

- Jednoprůchodové osvětlování (single-pass lighting)
 - Každému objektu spočítáno osvětlení od všech světel společným shaderem
 - Omezení počtu světel
- Vícepřechodové osvětlování (multi-pass lighting)
 - Několik vykreslovacích průchodů (skupiny světel)
 - Různé, ale jednodušší shadery
 - Násobné vykreslování geometrie, opakované výpočty
- Společná nevýhoda
 - Plýtvání výpočty na objektech, které nejsou vidět

Deferred rendering

- Postup
 - Vykreslení objektů bez osvětlení
 - Pro každý pixel data předpřipravená pro osvětlení (**G-buffer**)
 - **Shading jako postprocess s využitím G-bufferu**
- Výhody
 - Zredukovaný počet vykreslovacích příkazů
 - Výpočty osvětlení se provádějí pouze nad viditelnými pixely
 - Snadné další postprocess efekty
- Nevýhody
 - Nároky na paměť
 - Transparentní objekty nutno řešit zvlášť
 - Problematický antialiasing (multisampling)

Deferred rendering: G-buffer

- Pro každý pixel uložená data nutná pro výpočet osvětlení
- Data
 - barva (albedo)
 - koeficient a ostrost spekulárního odrazu
 - Normála (komprimovaná reprezentace)
 - hloubka

Deferred rendering: G-buffer



Deferred rendering: G-buffer



Deferred rendering: G-buffer



Deferred rendering: G-buffer



Deferred rendering: G-buffer



Celkový postup

- 1. Deferred rendering pro neprůhledné objekty
- 2. Forward rendering pro transparentní objekty
- 3. Post-process
 - Tone mapping (HDR, korekce barev)
 - Screen space motion blur / depth of field
 - FXAA (antialiasing jako postprocess s detekcí hran)
 - Exposure, glow, gamma, etc.

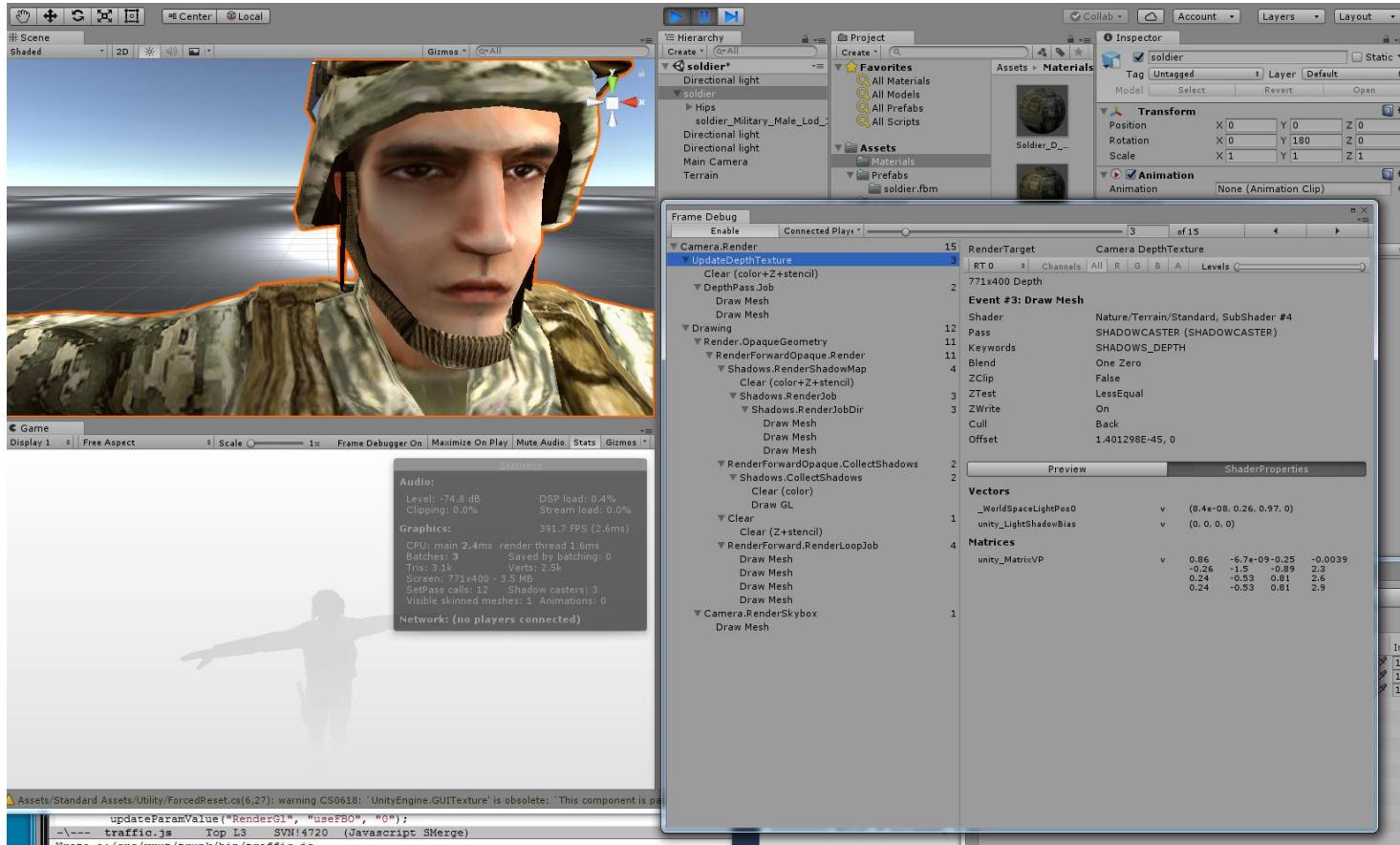
Post-process - příklad





Image with post-processing (depth of field, bloom, motion blur, colorize, ILR)

Unity: Frame Debugger



Efekt: SSAO

- Screen space ambient occlusion
- Aproximace zastínění okolím



VAO++: Practical Volumetric Ambient Occlusion for Games
<https://projectwilberforce.github.io/vaopaper/>

Efekty: Inverse Diffuse

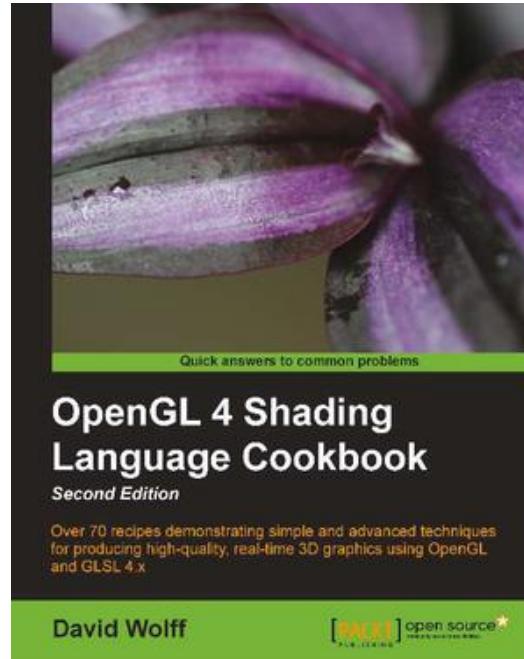
- Vypadá jako hologram
- Jednoduché zvýraznění hran

$$ID = 1 - N \cdot L$$



GLSL Shadery - Příklady

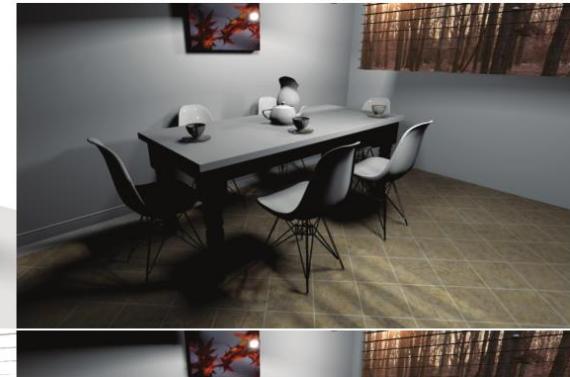
- David Wolff. OpenGL Shading Language Cookbook. Packt publishing 2013.
- 70 příkladů různých shaderů
- Zdrojové kódy
 - <https://github.com/daw42/glslcookbook>



Stíny

■ Stíny

- Části scény neosvětlené zdrojem světla
- Ostré stíny
- Měkké stíny



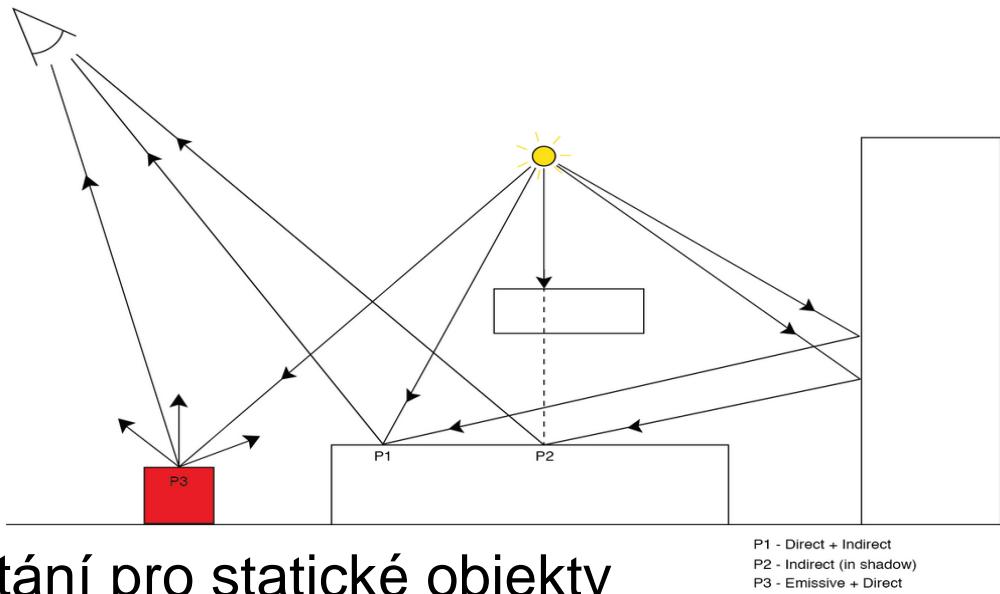
■ Varianty

- Předpočítané (baked)
- Stínové mapy
- Ray tracing



Globální osvětlení

- Simulace vícenásobného odrazu světla



- Předpočítání pro statické objekty
 - Textury (baking)
- Aproximace možné v reálném čase

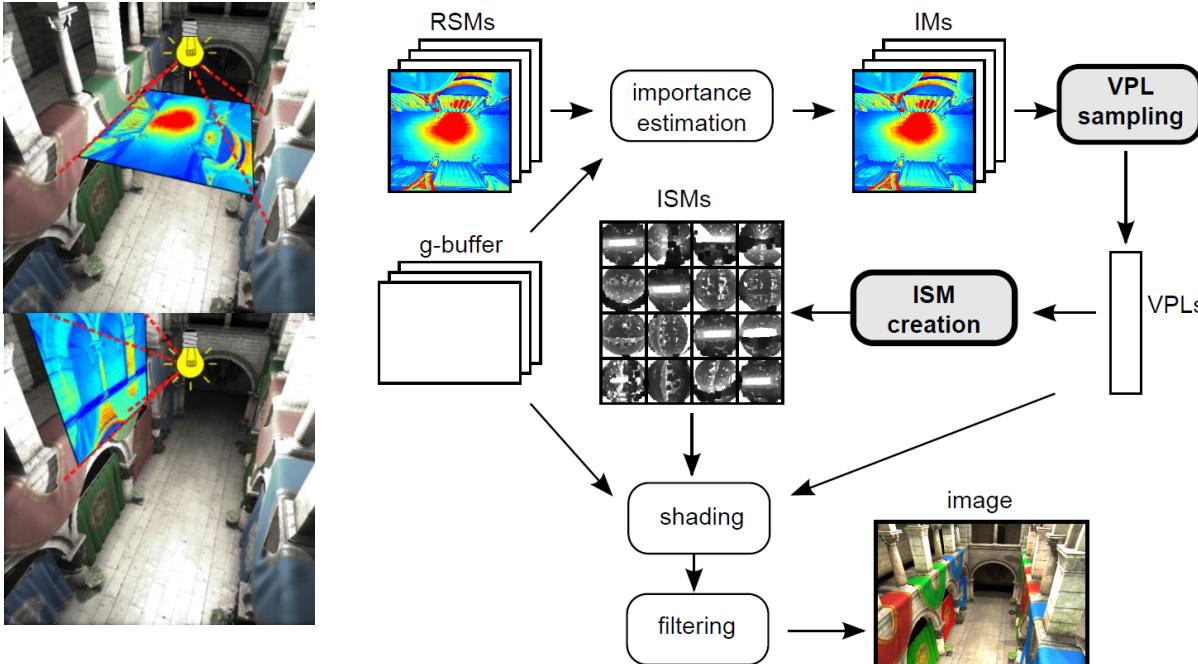
Globální osvětlení

- Radiosity (1988)
 - Instant radiosity (1997)
 - Cascaded light propagation volumes (2010)
 - Voxel cone tracing (2011)
 - Light Field probes (2017)
 - Ray Tracing / Path tracing
-
- Detaily v APG / RSO / PGR2



Temporally Coherent VPL Sampling

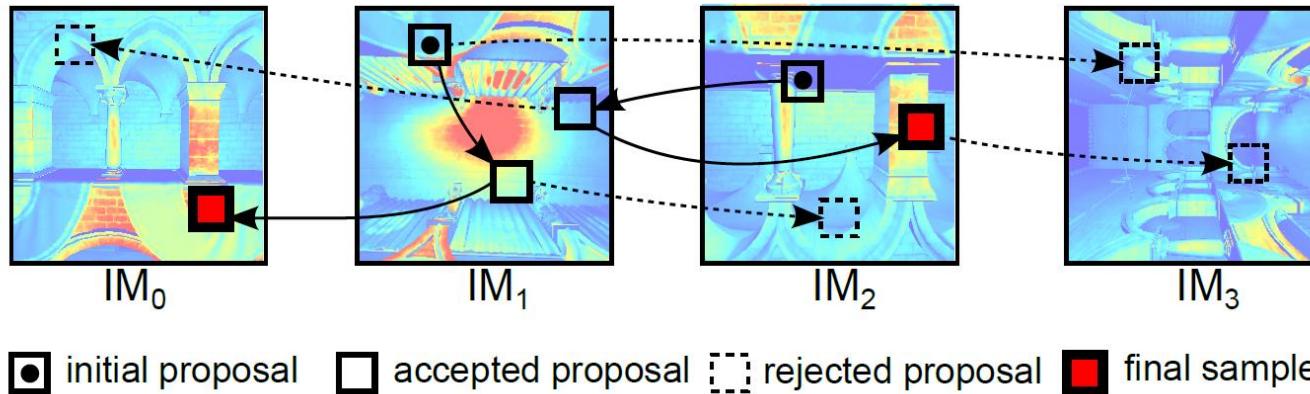
- Global illumination using instant radiosity (many VPLs)
- Improve stability of adaptive VPL sampling



[Temporally Coherent Adaptive Sampling for Imperfect Shadow Maps (2013)]

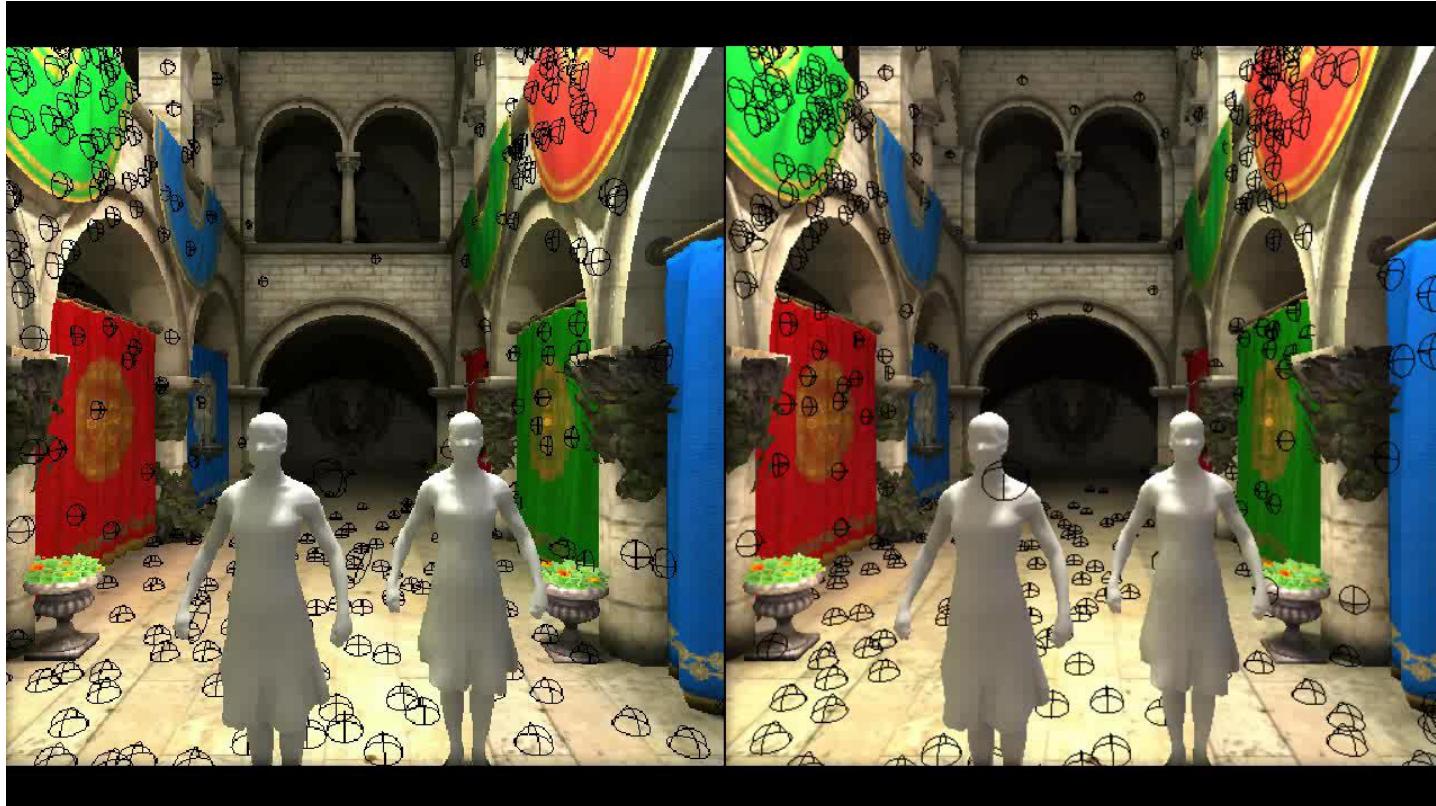
Temporally Coherent VPLs

- Metropolis-Hastings sampling
- Independent Markov chain



[Temporally Coherent Adaptive Sampling for Imperfect Shadow Maps (2013)]

Temporally Coherent VPLs

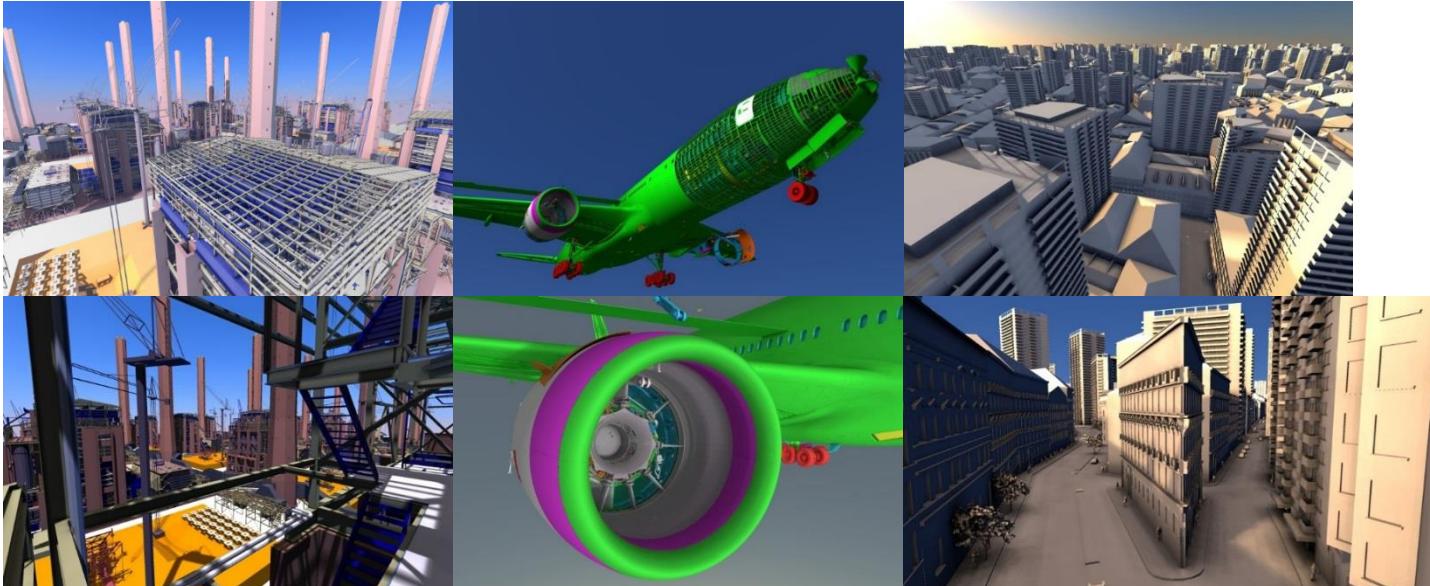


CDF sampling

Our method

GLSL Ray Tracing

- Interactive GPU Ray Tracing of Massive Models
- CHC+RT: Coherent Hierarchical Culling for Ray Tracing (2015)

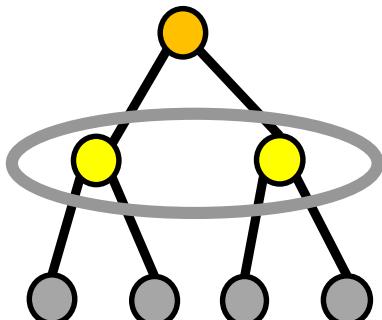


[CHC+RT: Coherent Hierarchical Culling for Ray Tracing (2015)]

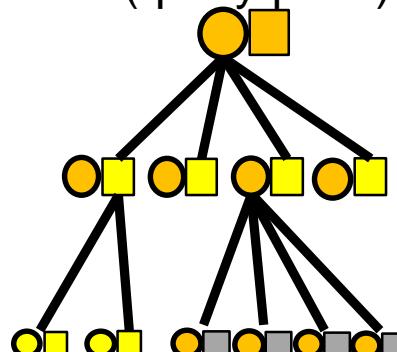
CHC+RT: Coherent Hierarchical Culling for Ray Tracing

- Ray tracing in OpenGL shaders
- HW occlusion queries for scheduling & out-of-core

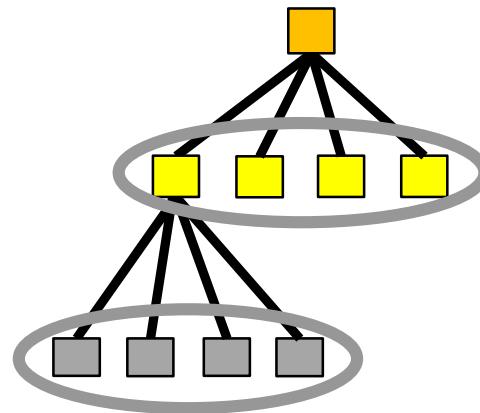
Object hierarchy
(BVH nodes)



Intersection hierarchy
(query pairs)



Ray hierarchy
(screen tiles)



[CHC+RT: Coherent Hierarchical Culling for Ray Tracing (2015)]

CHC+RT: Coherent Hierarchical Culling for Ray Tracing

O. Mattausch^{1,2} J. Bittner³ A. Jaspe⁴ E. Gobbetti⁴ M. Wimmer⁵ R. Pajarola¹



Eurographics 2015



1

Universität
Zürich^{UZH}



2



3



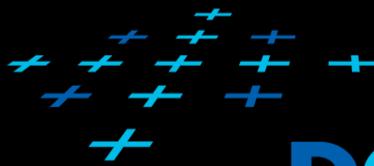
4



5

Obsah přednášky

- Výpočet osvětlení – Úvod GAE 10.1.2.1-2
- Osvětlovací modely GAE 10.1.3.1-2
- Světelné zdroje GAE 10.1.3.3
- Textury GAE 10.1.2.5
- Shadery GAE 10.2.1-10.2.7



DCGI

KATEDRA POČÍTAČOVÉ GRAFIKY A INTERAKCE

Otázky?