

# PGR - Fragment Shader, Světlo, Barvy,

Tomáš Milet

Brno University of Technology, Faculty of Information Technology

Božetěchova 1/2. 612 66 Brno - Královo Pole

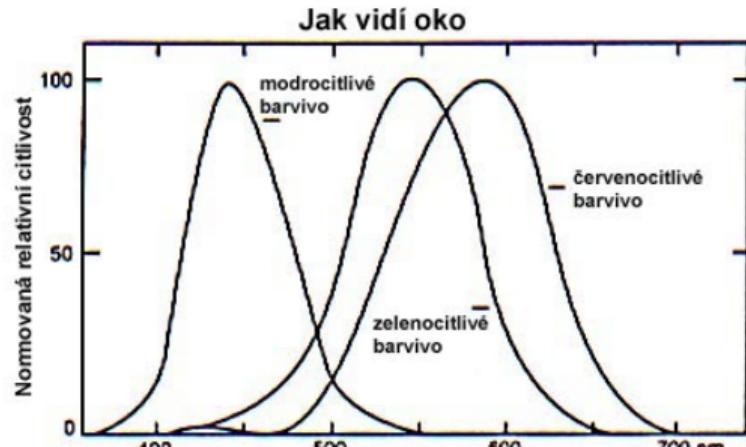
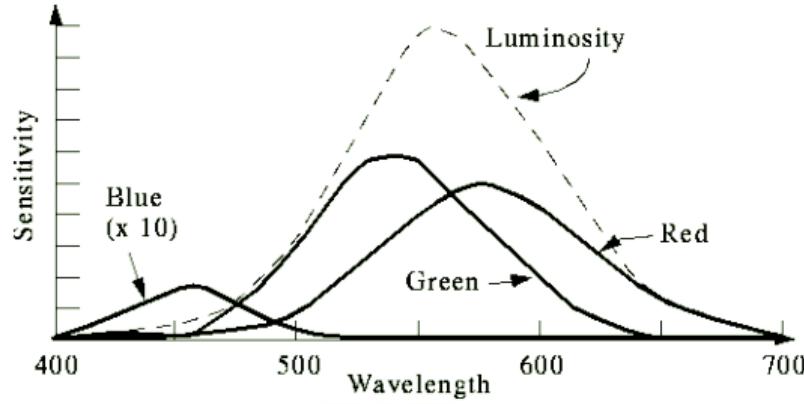
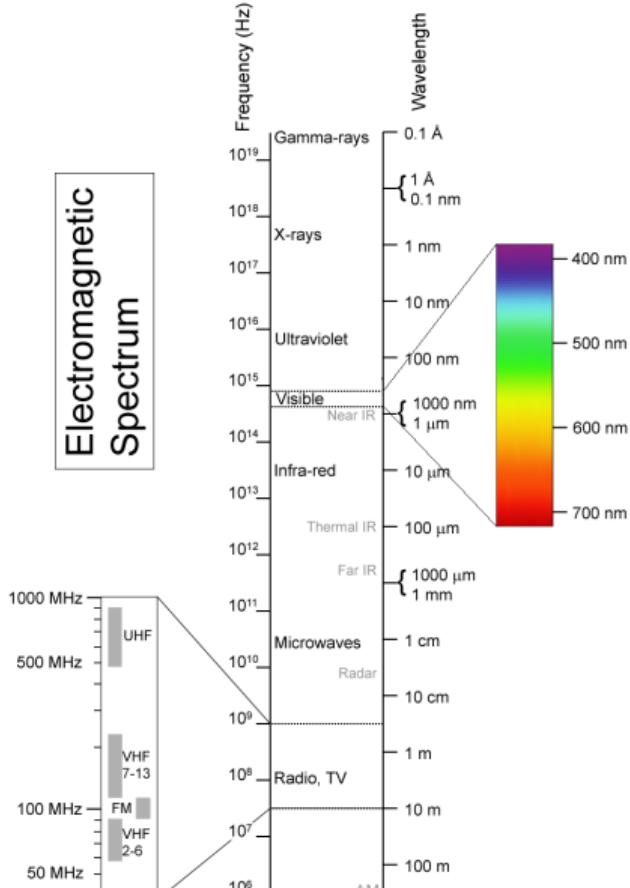
[imilet@fit.vutbr.cz](mailto:imilet@fit.vutbr.cz)

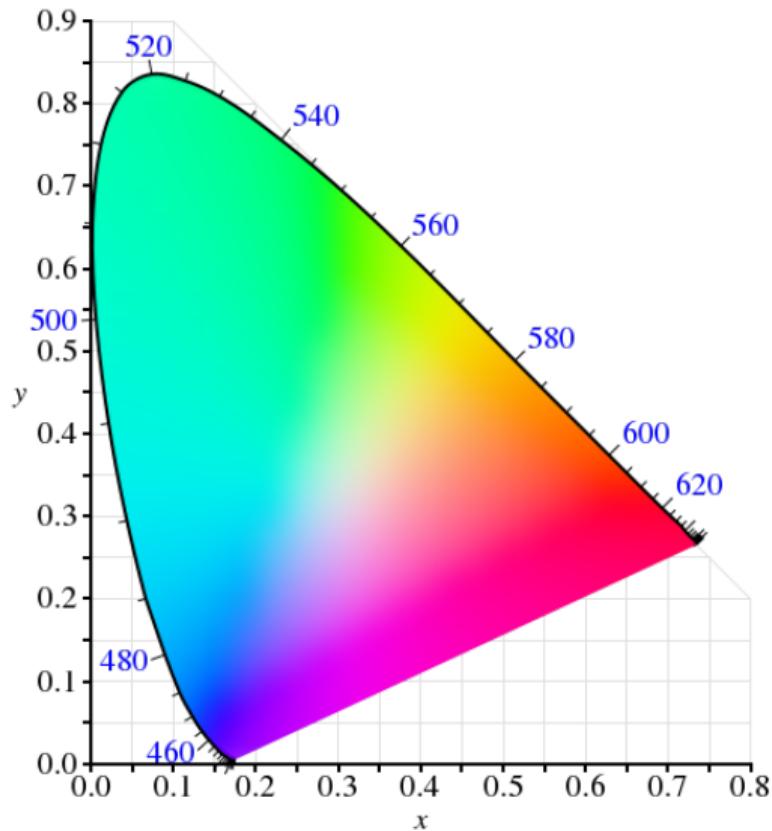


11. listopadu 2022

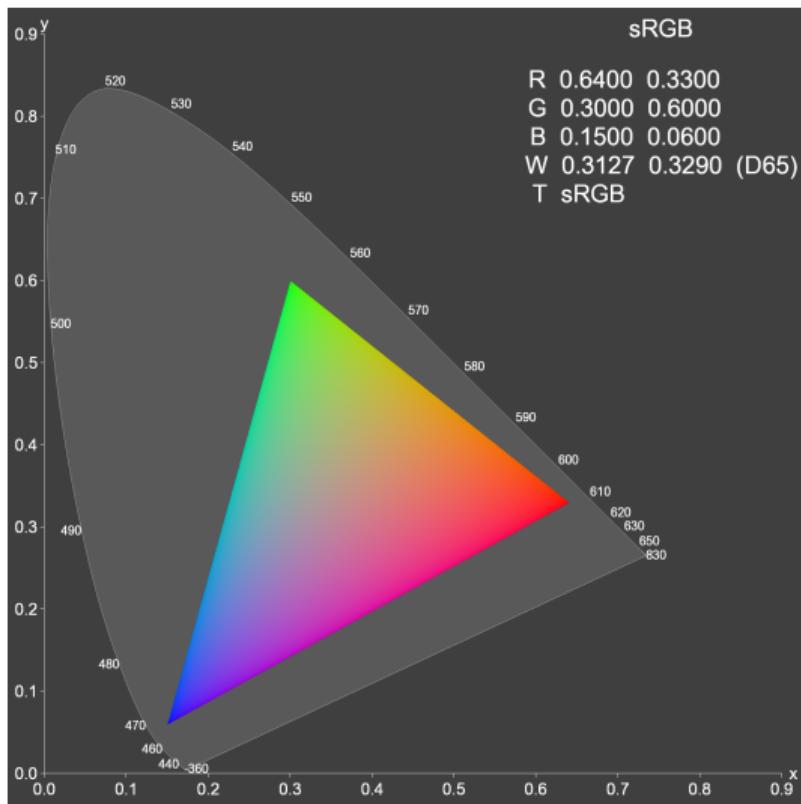
# Color / Barva

## Electromagnetic Spectrum





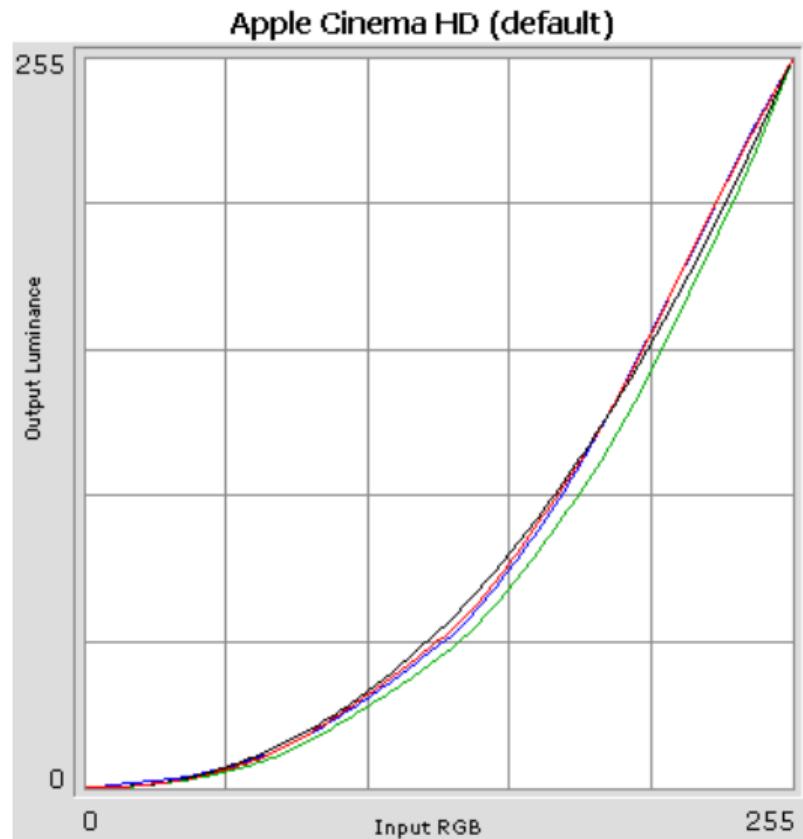
• CIE 1931



Logarithmic sensitivity of human eye  
Logaritmická citlivost oka

$$C_{\text{srgb}} = \begin{cases} 12.92C_l, & C_l \leq t \\ (1 + a)C_l^{\frac{1}{2.4}} - a, & C_l > t \end{cases}$$

$a = 0.055$   
 $t = 0.0031308$



Pipeline :

- float RGB(A) (vec3)
- (0,0,0) (1,0,0) (0,1,0) (0,0,1)  
! Linear Color space, interpolation, blending.

Pipeline :

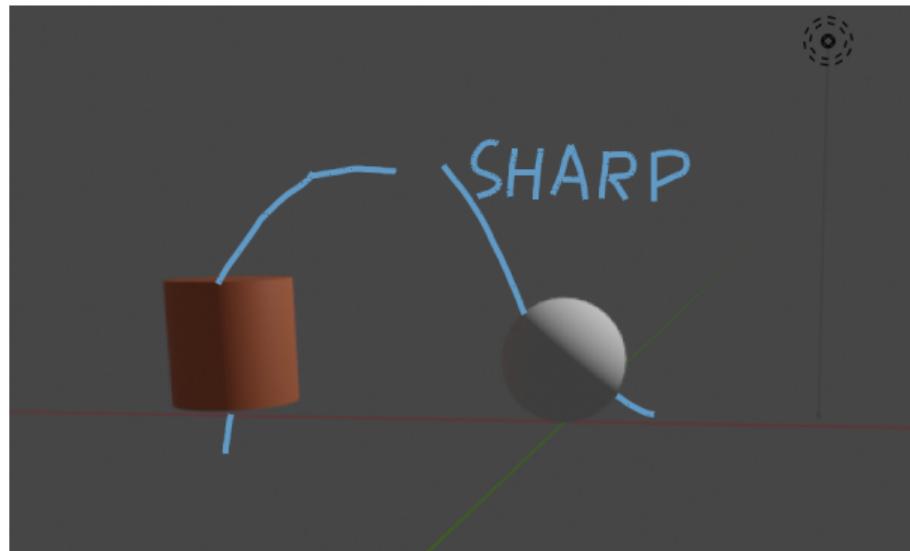
- float RGB(A) (vec3)
- (0,0,0) (1,0,0) (0,1,0) (0,0,1)  
! Linear Color space, interpolation, blending.

Vstupy a framebuffer :

- The most common RGB8
- Others : RGB16, RGB565, ...
- Linear or sRGB
- Conversion by hand?

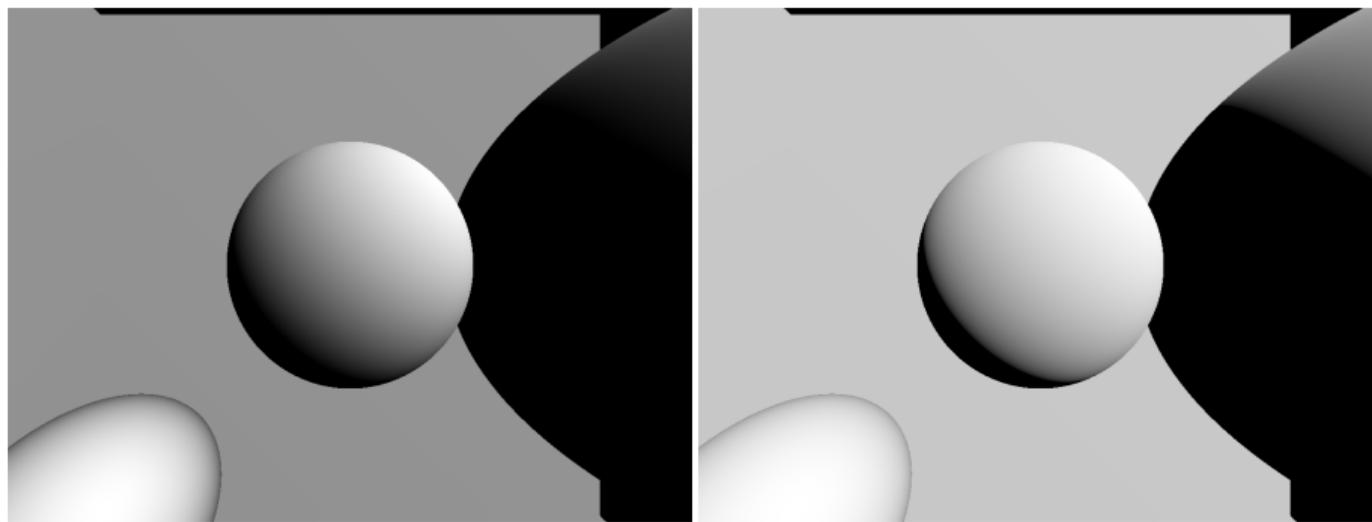
```
//default framebuffer
SDL_GL_SetAttribute(SDL_GL_FRAMEBUFFER_SRGB_CAPABLE, 1);
...
//enabling SRGB in OpenGL
 glEnable(GL_FRAMEBUFFER_SRGB);
```

- The light computation has to be done in linear space.
- After the computation is done, it has to be gamma corrected.
- Sharp transition from lit and shadowed parts.
- Výpočet osvětlení musí být proveden v lineárním prostoru.
- Po výpočtu osvětlení je nutné obrázek upravit gamma korekcí.
- Ostrý přechod ze světla do stínu je správně.





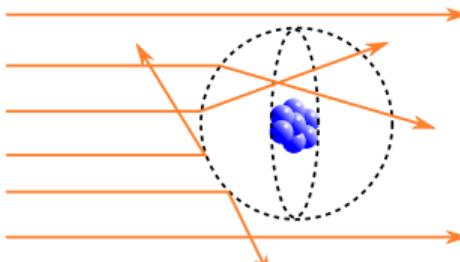
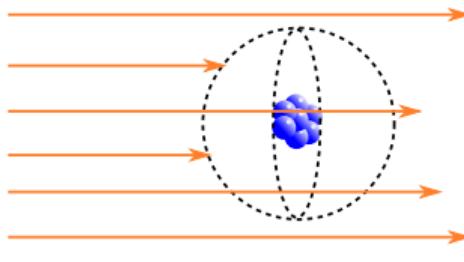
- Left incorrect, right correct.
- Vlevo bez korekce, vpravo s korekcí.



# Light-Matter Interaction / Interakce světla a hmoty

# Absorption, scattering

- Three kinds of interactions: Absorption, Scattering, Emission.
- Tři druhy interakcí: Pohlcení, odraz/lom, vyzáření.



Clear media



Clear absorbent media



Cloudy media



Translucent media



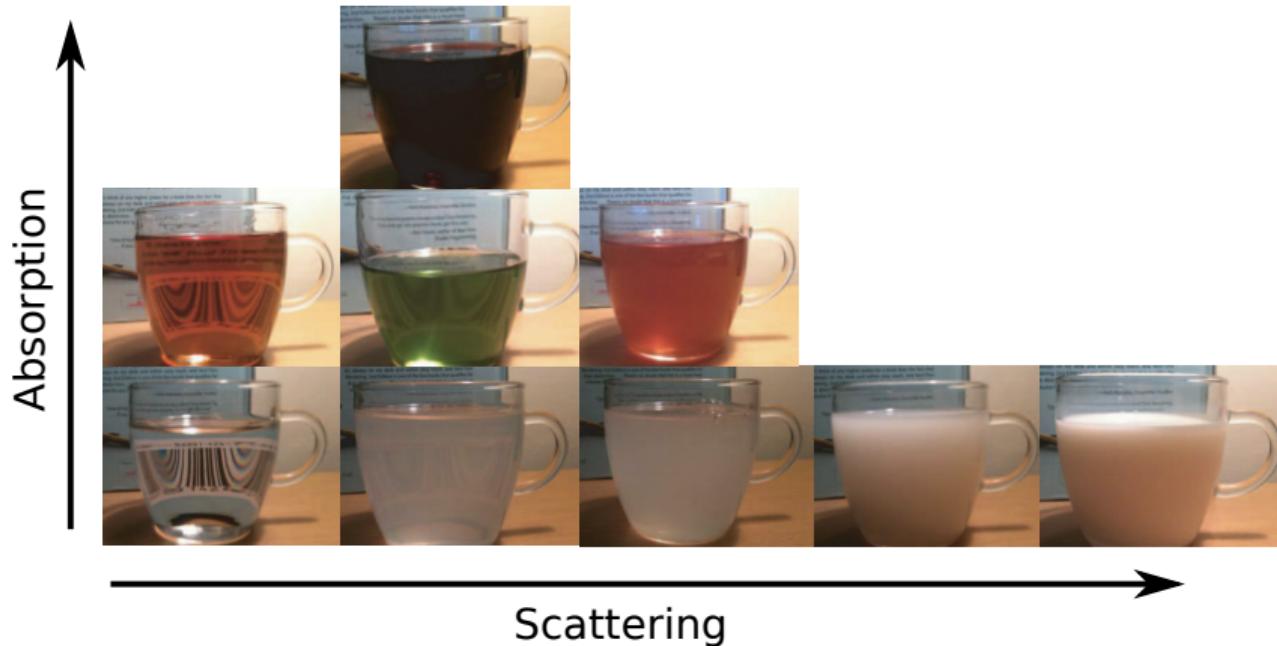
Water absorbent over large distances



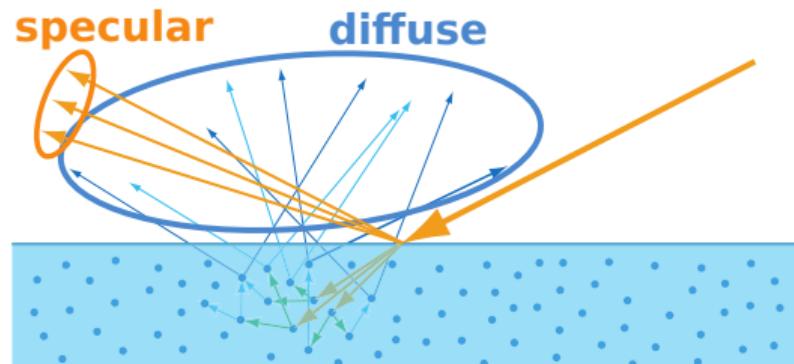
Air scattering over large distances



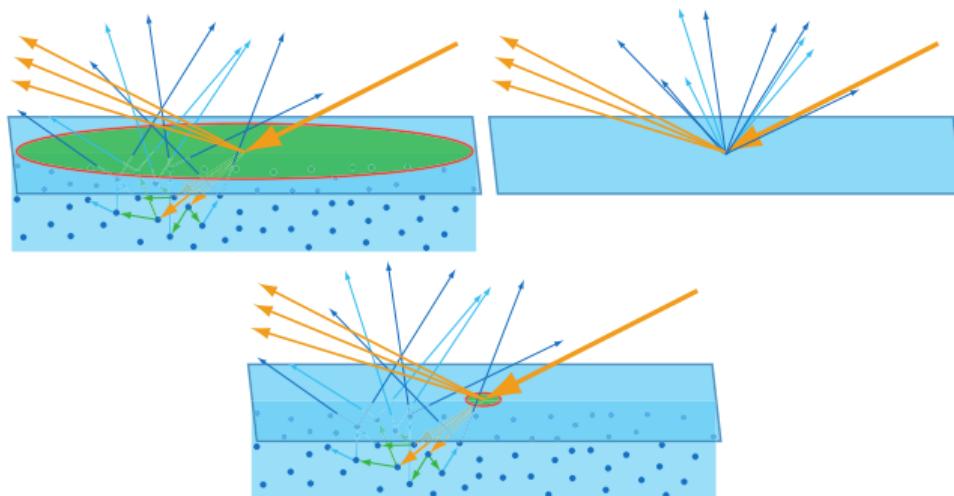
- Absorption and scattering are semi independent values.
- But fotons that are absorbed cannot be scattered.
- And photons that are scattered cannot be absorbed.
- Absorption and reflection are more or less independent values.
- However, photons that are absorbed can also be reflected and vice versa.



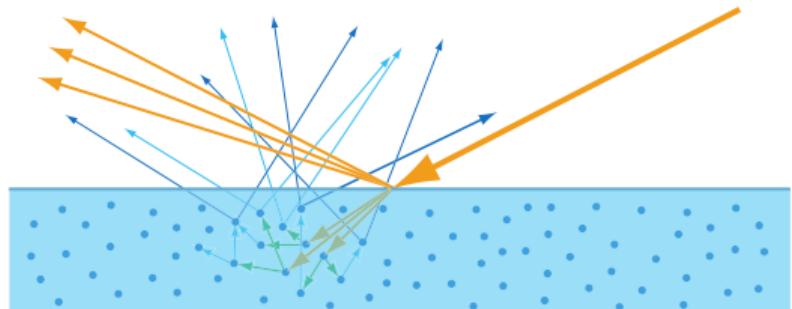
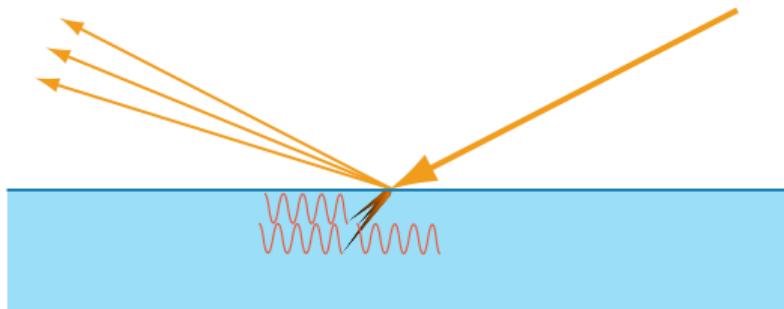
- Specular component is a light directly reflected from a surface.
- Specular component is dominant in metals.
- Diffuse component is dominant in dielectrics/isolants.
- Diffuse component is a light scattered inside material, when the entry and exit point of the light is closer than the size of screen pixel.
- Spekulární složka je odražené světlo přímo z povrchu.
- Je dominantní pro kovy.
- Difuzní složka je dominantní pro dielektrika / izolanty.
- Je to světlo, které se zlomilo uvnitř materiálů. Vstup světla a výstup světla z materiálu musí být k sobě blíž než je velikost pixelu.



- If the entry and exit point of the light is far away from each other, we are talking about subsurface scattering not about diffuse light.
- Wax, skin, ...
- Když jsou vstupní a výstupní body světla od sebe daleko, mluvíme od pod povrchovém lomení světla. Řeší se pomocí jiných metod.
- Vosk, kůže, ...

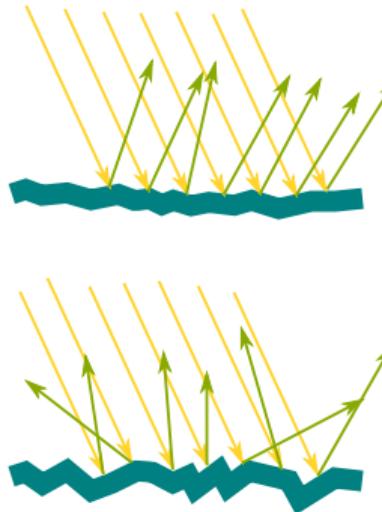


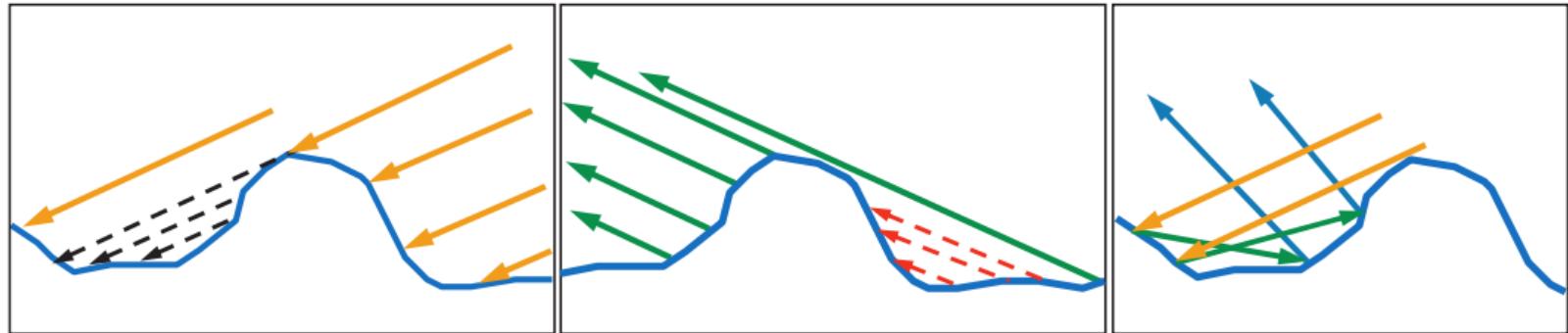
- Metals only reflect incoming light or convert the light to other forms of energy.
- Left: metal, right: non-metal
- Kovy světlo pouze odráží nebo přeměňují na jiný druh energie (teplota).
- Vlevo: kov, vpravo: nekov

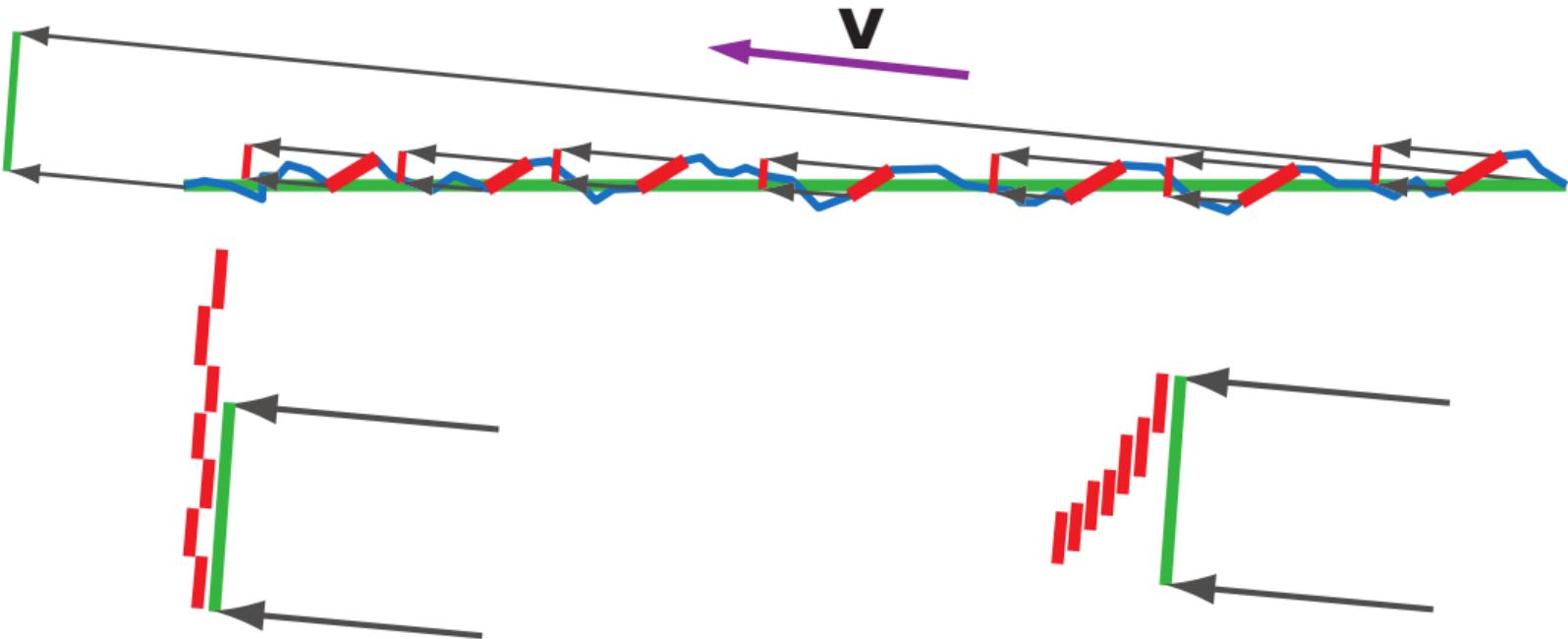


# Surface roughness and microfacets / Drsnost povrchu a mikroplošky

- Microscopic structures determine coherency of reflected light rays.
- Microfacet distribution determines the clarity of mirror images.
- Mikroskopické struktury určují koherentnost odražených paprsků.
- Distribuce mikroplošek určuje čistotu zrcadlového odrazu.

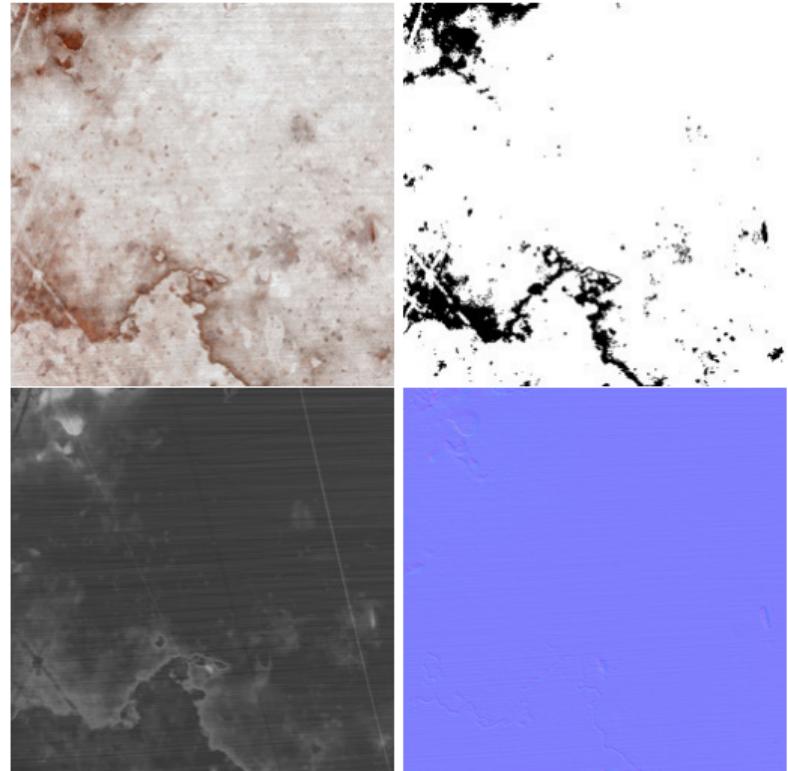




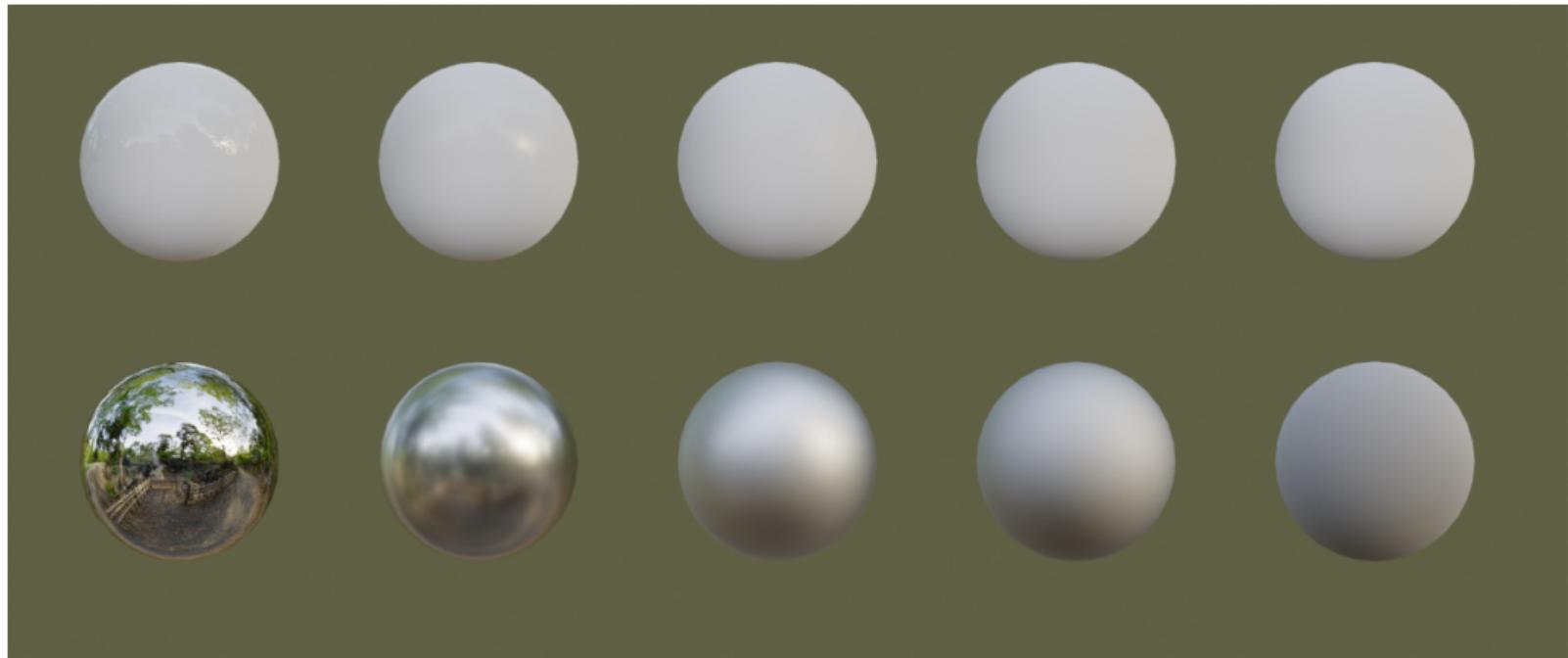




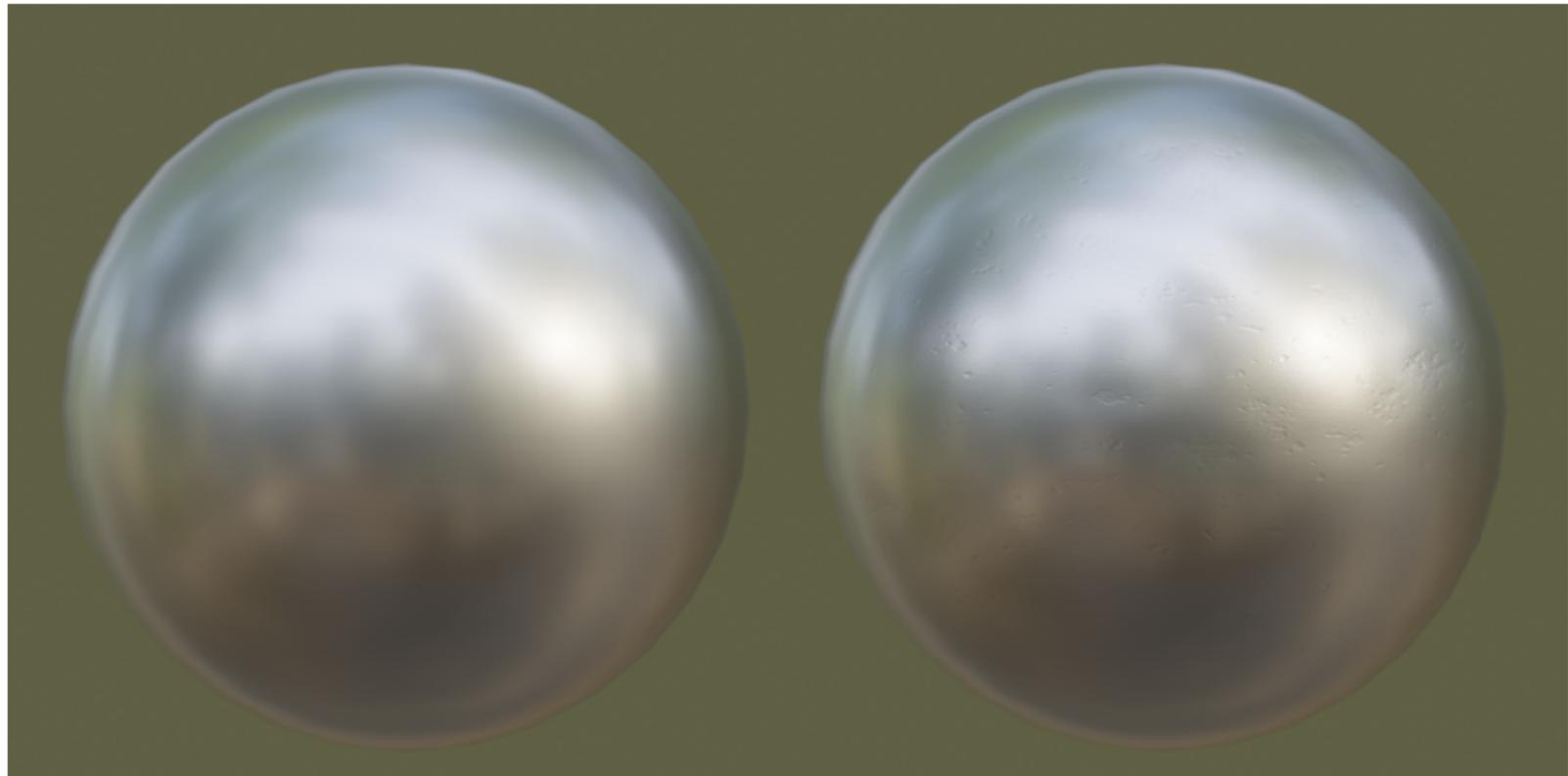
- Metallic workflow is intuitive way of representing different material properties.
- Color, metalness, roughness, normals.
- Metalness 1 or 0: metal or non-metal.
- Roughness represents microfacets.
- Metalická reprezentace materiálů je intuitivní.
- Barva, kovovost, drsnost, a normálová mapa.
- Metalness (kovovost) je buď jedna nebo nula.
- Roughness (drsnost) představuje distribuci mikroplošek.

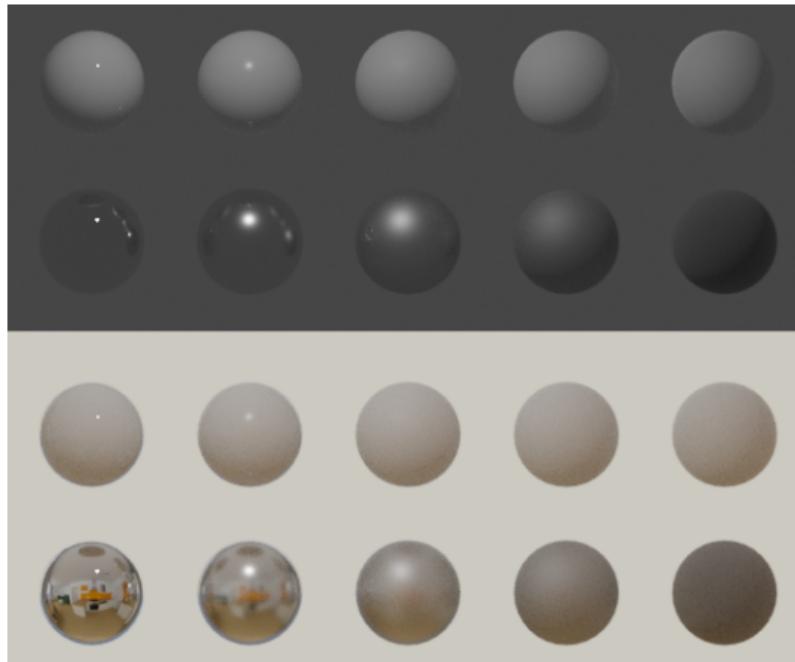


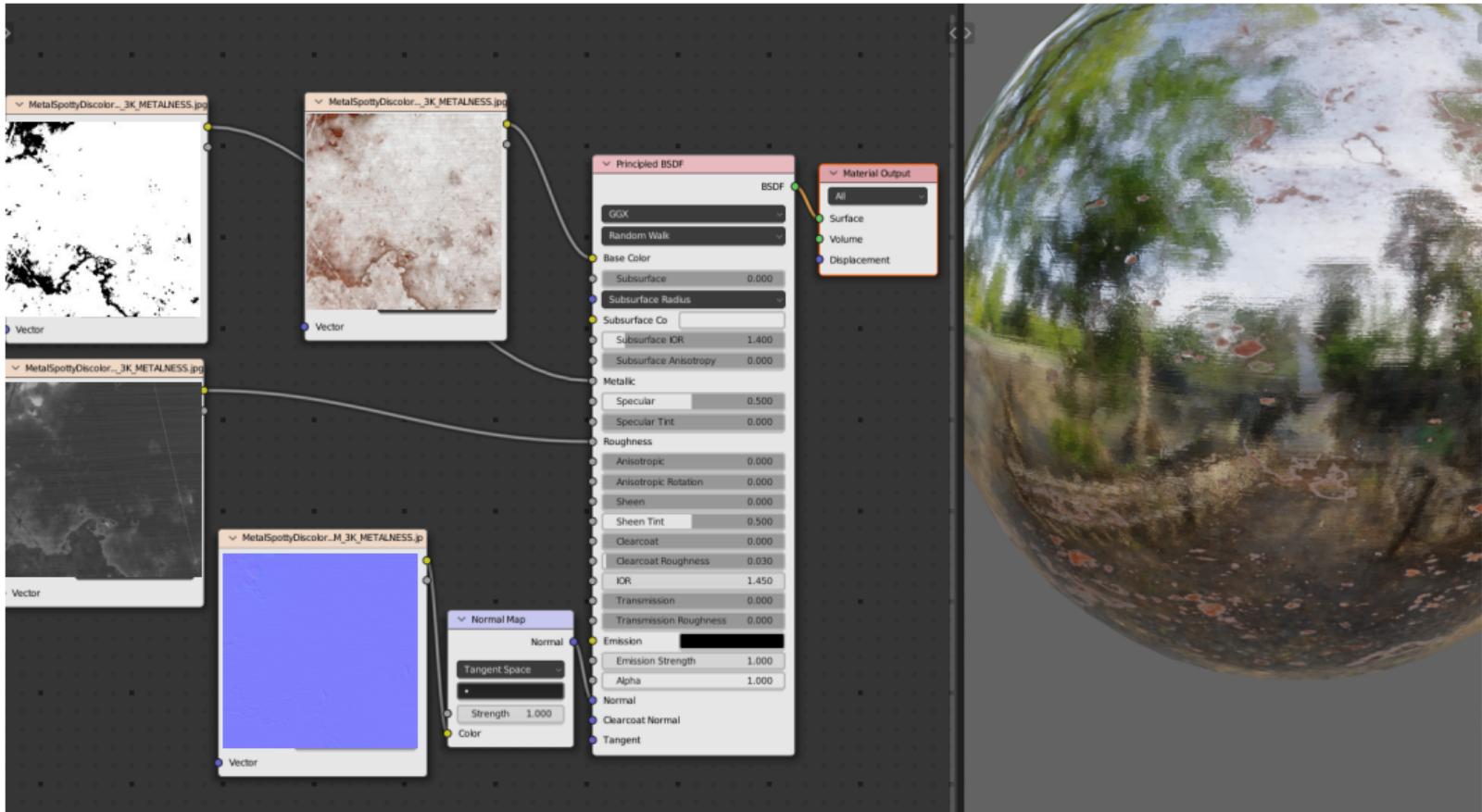
- Top row: Non-metal, bottom row: metal.
- From left to right: roughness from 0 to 1
- Horní řádek nekov, dolní kov.
- Zleva do prava, zvyšující se drsnost od 0 po 1.



- Left: without normal map, Right: with normal map
- Vlevo: bez normálové mapy, vpravo s normálovou mapou

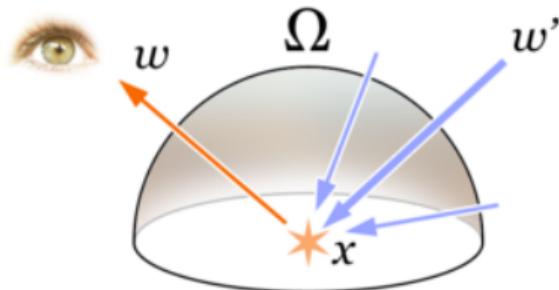








$$L_o(\mathbf{x}, \omega, \lambda, t) = L_e(\mathbf{x}, \omega, \lambda, t) + \int_{\Omega} f_r(\mathbf{x}, \omega', \omega, \lambda, t) L_i(\mathbf{x}, \omega', \lambda, t) (-\omega' \cdot \mathbf{n}) d\omega'$$



Siggraph 1986 :

$$I(x, x') = g(x, x') \left[ e(x, x') + \int_S p(x, x', x'') I(x', x'') dx'' \right]$$

- Odraz světla v "jednom bodu".
- Diferenciální ploška scény.

- Odraz světla v "jednom bodu".
- Diferenciální ploška scény.

$$L_{\{o,e,i\}}(\mathbf{x}, \omega, \lambda, t)$$

- Radiance
- $W \cdot sr^{-1} \cdot m^{-2}(\cdot m^{-1})$

- Odraz světla v "jednom bodu".
- Diferenciální ploška scény.

$$L_{\{o,e,i\}}(\mathbf{x}, \omega, \lambda, t)$$

- Radiance
- $W \cdot sr^{-1} \cdot m^{-2}(\cdot m^{-1})$

$$f_r(\mathbf{x}, \omega', \omega, \lambda, t)$$

- BRDF
- $sr^{-1}$

- Bidirectional Reflectance Distribution Function.
- Hustota pravděpodobnosti (skoro).

- Bidirectional Reflectance Distribution Function.
- Hustota pravděpodobnosti (skoro).

Reciprocity

$$f_r(\mathbf{x}, \omega', \omega) = f_r(\mathbf{x}, \omega, \omega')$$

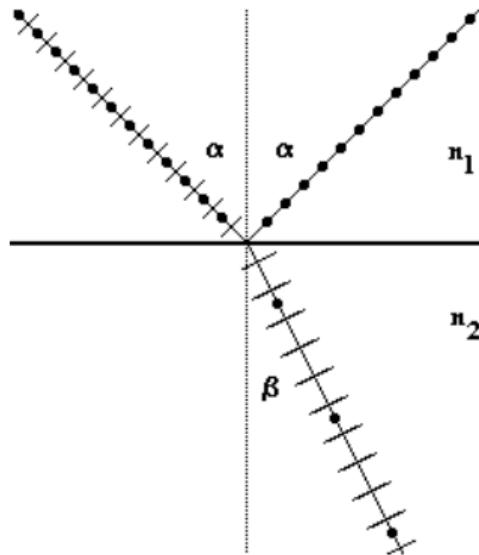
Zákon zachování energie

$$\int_{\Omega} f_r(\mathbf{x}, \omega', \omega) (\omega \cdot \mathbf{n}) d\omega \leq 1$$

Izotropie x Anizotropie

- Phong
- Blinn-Phong
- Cook-Torrance
- GGX
- ...

- Phong
  - Blinn-Phong
  - Cook-Torrance
  - GGX
  - ...
- 
- Pragmatické x Fyzikální
  - Výběr podle aplikace.



- Odraz a lom na hranicích prostředí.
- Indexy lomu (rychlosť v daném prostředí).
- Polarizace odrazem.

$$\frac{v_1}{v_2} = \frac{\sin \alpha}{\sin \beta}$$

## Fresnellovy vzorce

$$R_s = \left| \frac{Z_2 \cos \theta_i - Z_1 \cos \theta_t}{Z_2 \cos \theta_i + Z_1 \cos \theta_t} \right|^2$$

- Závislé na polarizaci.
- Nepraktické.

## Fresnellovy vzorce

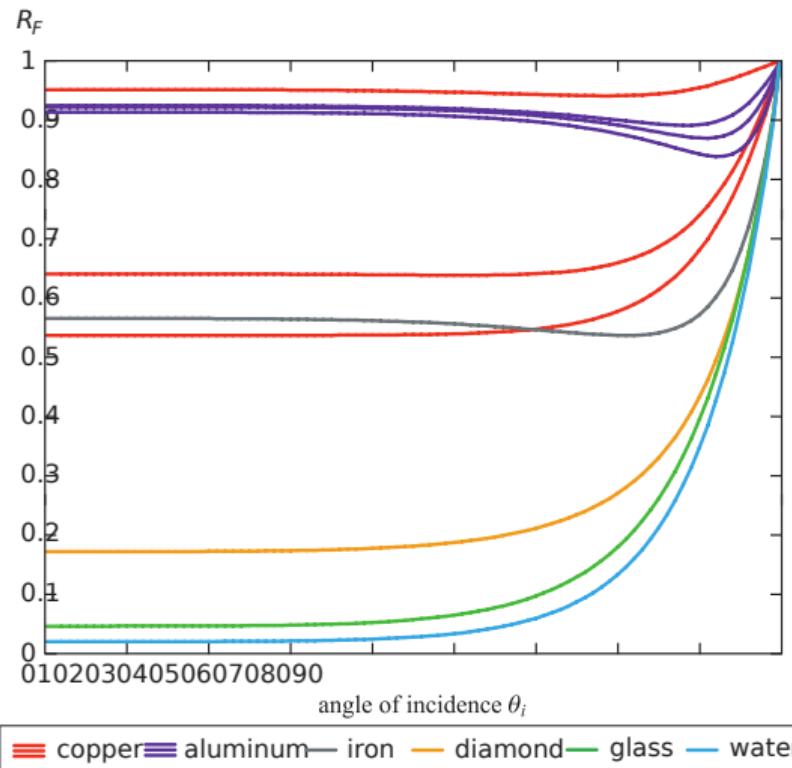
$$R_s = \left| \frac{Z_2 \cos \theta_i - Z_1 \cos \theta_t}{Z_2 \cos \theta_i + Z_1 \cos \theta_t} \right|^2$$

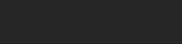
- Závislé na polarizaci.
- Nepraktické.

## Schlickova aproximace

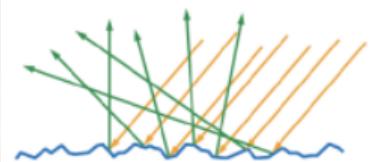
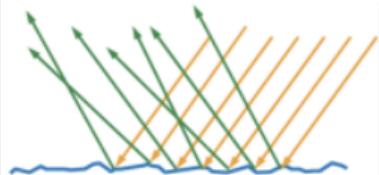
$$R(\theta) = R_0 + (1 - R_0)(1 - \cos \theta)^5$$

- $R_0$  je vlastnost materiálu.
- Pro izolanty  $R_0 \approx 0.04$ .
- $\cos \theta \rightarrow 0 \dots R \rightarrow 1$



Material	$F(0^\circ)$ (Linear)	$F(0^\circ)$ (sRGB)	Color
Water	0.02,0.02,0.02	0.15,0.15,0.15	
Plastic/Glass(Low)	0.03,0.03,0.03	0.21,0.21,0.21	
PlasticHigh	0.05,0.05,0.05	0.24,0.24,0.24	
Glass(High)/Ruby	0.08,0.08,0.08	0.31,0.31,0.31	
Diamond	0.17,0.17,0.17	0.45,0.45,0.45	
Iron	0.56,0.57,0.58	0.77,0.78,0.78	
Copper	0.95,0.64,0.54	0.98,0.82,0.76	
Gold	1.00,0.71,0.29	1.00,0.86,0.57	
Aluminum	0.91,0.92,0.92	0.96,0.96,0.97	
Silver	0.95,0.93,0.88	0.98,0.97,0.95	

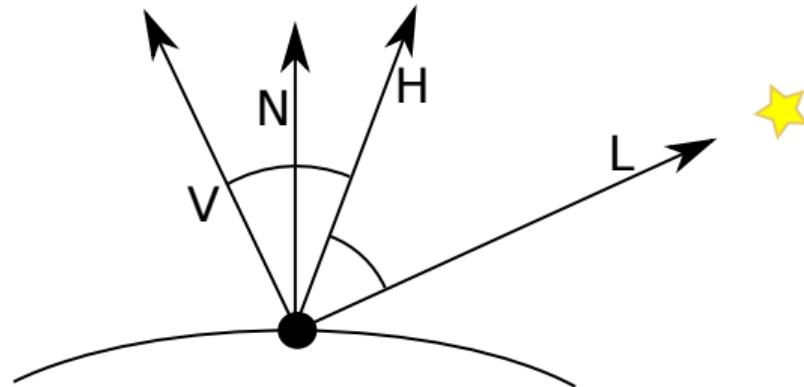


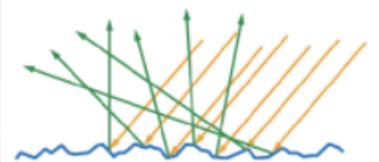
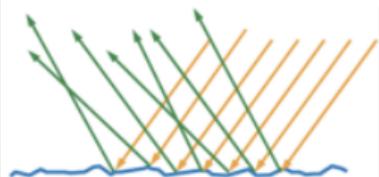


SIGGRAPH2014

Které plošky jsou správně orientované?

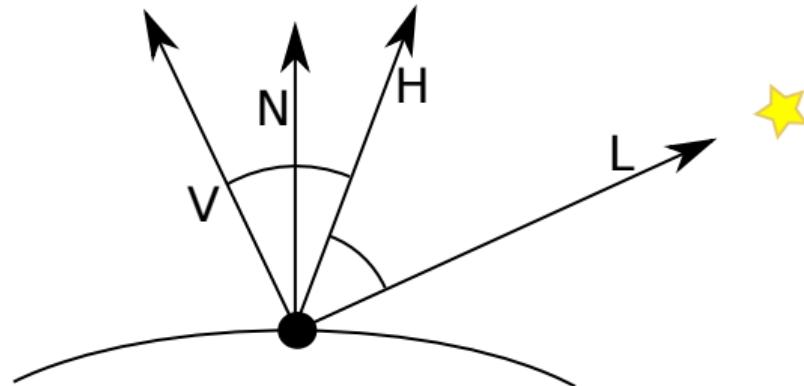
Eye





SIGGRAPH2014

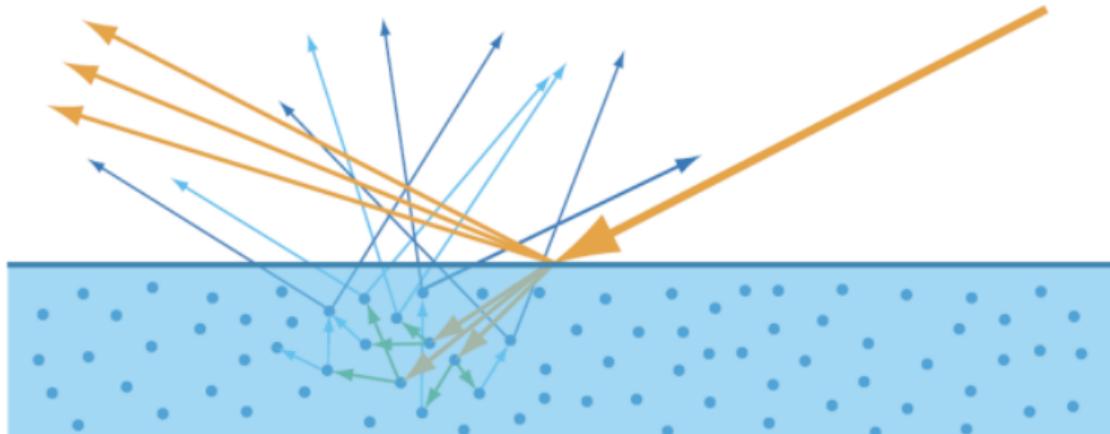
Eye



Které plošky jsou správně orientované?

- Odrážejí ze světla do pozorovatele.
- Mají normálu přesně mezi.
- ! Half vector.

$$\mathbf{h} = \frac{\mathbf{l} + \mathbf{v}}{2}$$



## Specular

- "Zrcadlový" odraz
- Dominantní pro kovy

## Diffuse

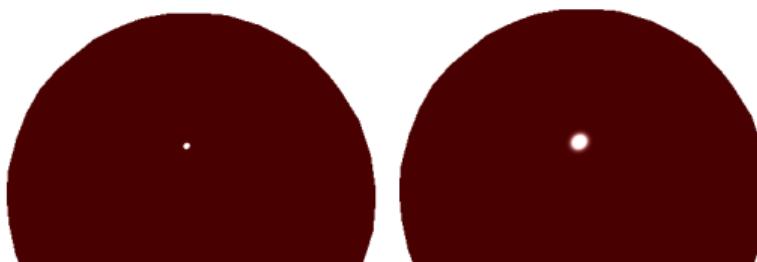
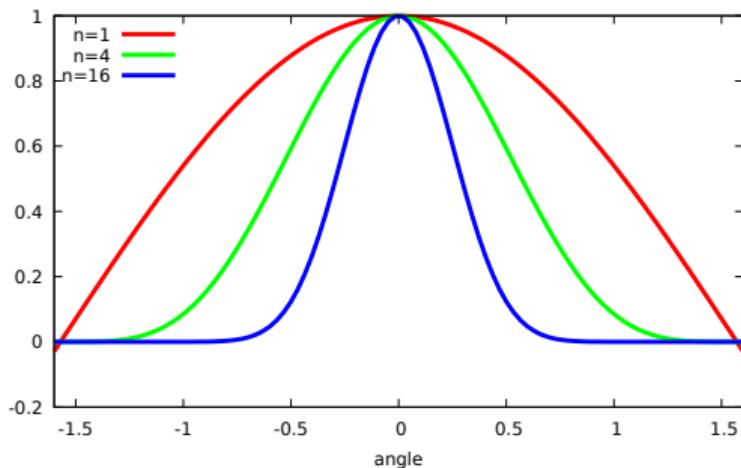
- Odraz uvnitř materiálu.
- Dominantní pro izolanty.

## Spekulární odraz

$$f_r(\mathbf{x}, \mathbf{l}, \mathbf{v}) = \frac{F(\mathbf{l}, \mathbf{h}) G(\mathbf{l}, \mathbf{v}, \mathbf{h}) D(\mathbf{h})}{4(\mathbf{n} \cdot \mathbf{l})(\mathbf{n} \cdot \mathbf{v})}$$

- F - Fresnell/Schlick
- D - Distribuční funkce plošek
- G - Geometrické zastínění plošek

$$D(\mathbf{h}) = (\mathbf{n} \cdot \mathbf{h})^{\text{shininess}}$$



$$f_r(\mathbf{x}, \mathbf{l}, \mathbf{v}) = (R_0 + (1 - R_0)(1 - (\mathbf{l} \cdot \mathbf{h}))^5)(\mathbf{n} \cdot \mathbf{h})^{\text{shininess}}$$

Co chybí?

- Barva materiálu (izolantú).  $R_0$  je "barva kovu".
- ! Lomené paprsky.
- Zachování energie.

$$f_r(\mathbf{x}, \mathbf{l}, \mathbf{v}) = \frac{k_d}{\pi}$$

- Nezáleží na poloze pozorovatele.
- $1/\pi$  je zachování energie.

$$\int_{\Omega} 1/\pi d\omega = 1$$

$$\left( \frac{k_s}{\pi} + k_n(R_0 + (1 - R_0)(1 - (\mathbf{l} \cdot \mathbf{h}))^5)(\mathbf{n} \cdot \mathbf{h})^{\text{shininess}} \right) (\mathbf{n} \cdot \mathbf{l})$$

$$k_n = \frac{(\text{shininess} + 2)(\text{shininess} + 4)}{8\pi(2^{-\frac{\text{shininess}}{2}} + \text{shininess})}$$

- $\mathbf{n} \cdot \mathbf{l}$  z renderovací rovnice.
- Zachování energie pro specular.
- Odvození na <http://www.farbrausch.de/~fg/stuff/phong.pdf>

$$L_o(\mathbf{x}, \omega, \lambda, t) = L_e(\mathbf{x}, \omega, \lambda, t) + \int_{\Omega} f_r(\mathbf{x}, \omega', \omega, \lambda, t) L_i(\mathbf{x}, \omega', \lambda, t) (-\omega' \cdot \mathbf{n}) d\omega'$$

- Bod má nulovou plochu.
- Bodové zdroje jsou singularity.
- Jak upravit renderovací rovnici? Jaké  $L_i$ ?

$$L_o(\mathbf{x}, \omega, \lambda, t) = L_e(\mathbf{x}, \omega, \lambda, t) + \int_{\Omega} f_r(\mathbf{x}, \omega', \omega, \lambda, t) L_i(\mathbf{x}, \omega', \lambda, t) (-\omega' \cdot \mathbf{n}) d\omega'$$

- Bod má nulovou plochu.
- Bodové zdroje jsou singularity.
- Jak upravit renderovací rovnici? Jaké  $L_i$ ?
- Nastavíme ho přes barvu ideálního difúzního povrchu.

$$L_o = \int_{\Omega} \frac{1}{\pi} L_i(-\omega' \cdot \mathbf{n}) d\omega'$$

$$L_i = \pi L_o$$

$$L = \sum_i \left( \frac{k_s}{\pi} + k_n F(R_0, \mathbf{l}_i, \mathbf{h}_i) (\mathbf{n} \cdot \mathbf{h}_i)^{\text{shininess}} \right) (\mathbf{n} \cdot \mathbf{l}_i) \pi L_i$$

- Sčítáme přes světla.
- $\pi$  se vykrátí.

$$L = \sum_i \left( \frac{k_s}{\pi} + k_n F(R_0, \mathbf{l}_i, \mathbf{h}_i) (\mathbf{n} \cdot \mathbf{h}_i)^{\text{shininess}} \right) (\mathbf{n} \cdot \mathbf{l}_i) \pi L_i$$

- Sčítáme přes světla.
- $\pi$  se vykrátí.

Co tam chybí?

$$L = \sum_i \left( \frac{k_s}{\pi} + k_n F(R_0, \mathbf{l}_i, \mathbf{h}_i) (\mathbf{n} \cdot \mathbf{h}_i)^{\text{shininess}} \right) (\mathbf{n} \cdot \mathbf{l}_i) \pi L_i$$

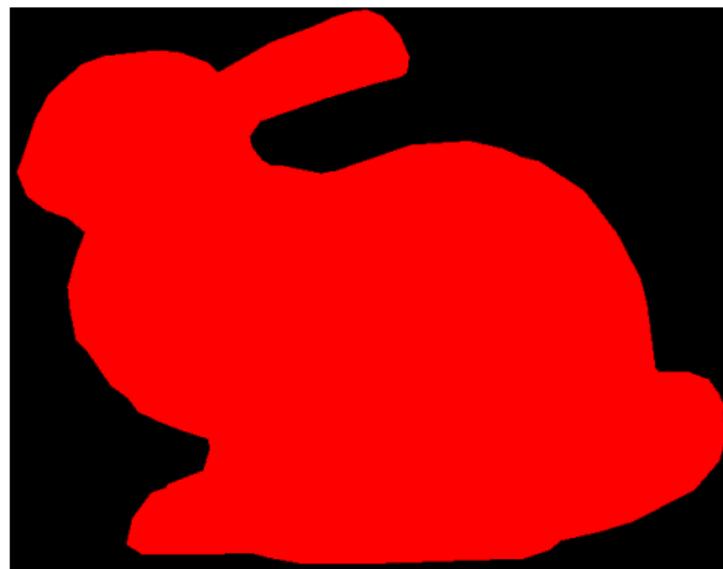
- Sčítáme přes světla.
- $\pi$  se vykrátí.

Co tam chybí?

- Vzdálenost od světla  $L_i = f(|\mathbf{p}_i - \mathbf{x}|^2)$
- Světelný kužel.
- Vícenásobný odraz světla.

$$L_o = \sum_i k_a L_i$$

$$k_a = k_d$$



- Hrubá aproximace mnohonásobného odrazu světla.

- Kolmé k povrchu
- $|N| = 1$  (normalizované)
- 1 na trojúhelník/1 na vrchol
- $\vec{N}_{\text{face}} = \vec{AB} \times \vec{AC}$
- $\vec{N}_{\text{vertex}} = \text{normalize} \left( \sum_{i=1}^n N_{\text{face}i} \right)$
- Průměr vážený plochou trojúhelníků
- Transformace :
  - Rotace + posun -  $N_{\text{eye}} = \text{ModelView} \cdot N_{\text{model}}$
  - Scale -  $N_{\text{eye}} = (\text{ModelView}^T)^{-1} \cdot N_{\text{model}}$
- ! Při scale je nutné vektory po transformaci znova normalizovat!

### "Flat" stínování

- Osvětlují se celé trojúhelníky.
- Nic se neinterpoluje.

### Gouraudovo stínování - "per vertex lighting"

- Osvětlení se počítá pro vrcholy.
- Interpolují se barvy.
- Dnes už nemá smysl.

### Phongovo stínování - "per fragment lighting"

- Osvětlení se počítá pro fragmenty.
- Interpolují se normálové vektory.



# Textury

- Textura je rastrový obrázek, který se namapuje na geometrii.
- V OpenGL je to objekt.

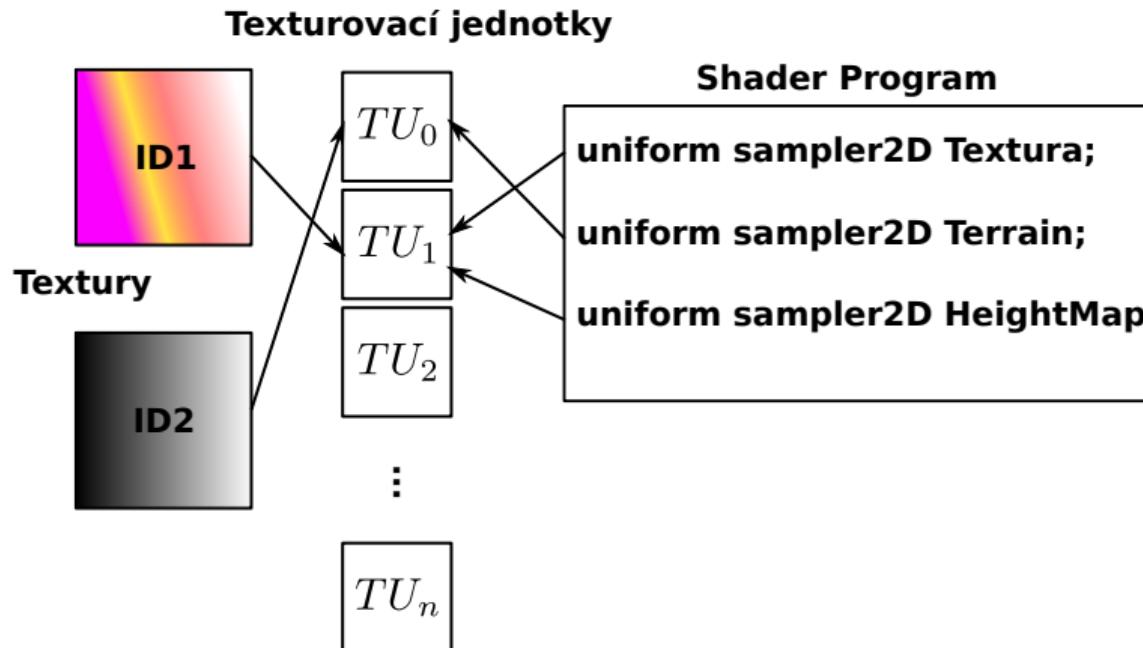
```
//vytvoreni jmena textury
glCreateTextures(GL_TEXTURE_2D,1,&tColor); //vygenerovani jmeno textury pro barvu

//nastaveni parametru textury
//filtering
glTextureParameteri(tColor,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
glTextureParameteri(tColor,GL_TEXTURE_MIN_FILTER,GL_NEAREST);

//wrapping
glTextureParameteri(tColor,GL_TEXTURE_WRAP_S,GL_CLAMP_TO_EDGE);
glTextureParameteri(tColor,GL_TEXTURE_WRAP_T,GL_REPEAT);

//alokace a nahrani dat na GPU
glTextureImage2DEXT(tColor,GL_TEXTURE_2D,0,//alokujeme misto
    GL_RGBA32F,//format ulozeni na GPU (4*float pro pixel)
    WIDTH,HEIGHT,0,GL_RGBA,//format ulozeni dan na CPU
    GL_UNSIGNED_BYTE,//typ dat na CPU
    Data); //data na CPU
```

- V Shader Programu se k nim přistupuje přes uniformní proměnné.
- Textura se naváže k texturovací jednotce.
- Uniformní proměnná se naváže na texturovací jednotku.



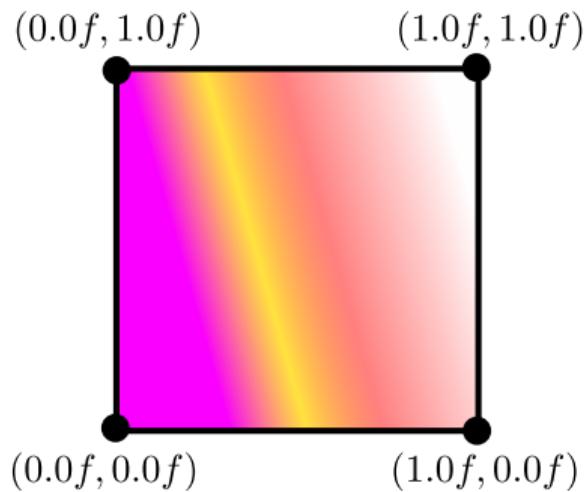
## Shader Program:

```
#version 330
layout(binding=0)uniform sampler2D Textura;
layout(binding=1)uniform sampler2D Terrain;
layout(binding=1)uniform sampler2D HeightMap;
void main(){
    texture(Texture,coords);
    //...
```

## Aplikace:

```
//pripojemni textur k texurovacim jednotkam
glBindTextureUnit(0, ID2);
glBindTextureUnit(1, ID1);
```

- Textura se adresuje texturovacími koordináty.
- Velikost Textury nehráje roli.
- Levý dolní roh 2D textury má souřadnice  $(0.0f, 0.0f)$
- Pravý horní roh 2D textury má souřadnice  $(1.0f, 1.0f)$



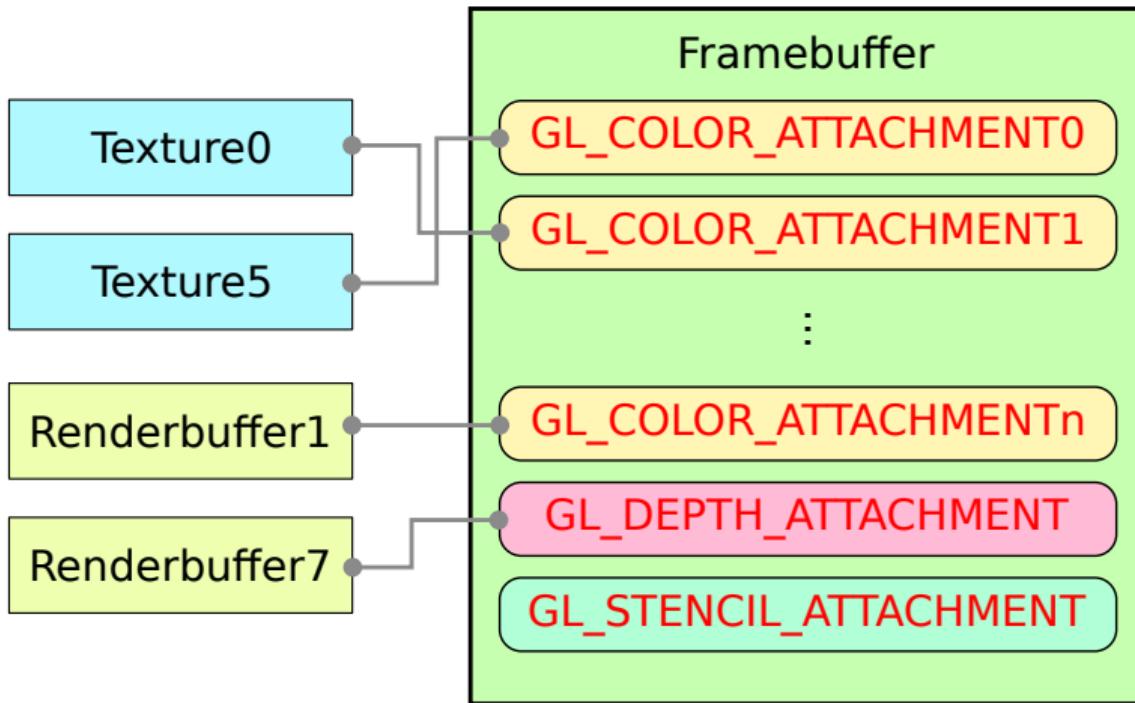
- Více obrázků v jedné textuře.
- Staří jedna texturovací jednotka.
- Zamezíme přepínání.



<http://www.scriptspot.com/3ds-max/scripts/texture-atlas-generator>

# Framebuffer

- Obvykle se scéna renderuje do defaultního framebufferu obrazovky.
- Od verze OpenGL 3.3 je umožněno vytvoření vlastního framebufferu a vykreslovat scénu do něj.
- Framebuffer se skládá z několika 2D polí (hloubkový buffer, stencil buffer, několik barevných bufferů).
- FBO umožňují renderování do textur.
- Umožňují renderovat uživatelské informace (normály, pozici, hloubku, ...) (Multiple Render Targets).
- Jsou základem pro různé grafické efekty např. vody, SSAO.
- Umožňují odložené stínování (deferred shading).
- Layered rendering



- 1 Získání jména FBO.
- 2 Aktivování FBO.
- 3 Připojení textur/renderbufferů k attachmentům.
- 4 Nastavení seznamu attachmentů.
- 5 Deaktivace FBO.
- 6 ...
- 7 Aktivování FBO.
- 8 Vykreslení scény.
- 9 Deaktivace FBO.
- 10 Zpracování vyrenderovaných textur.

- Vytvoření VBO identifikátorů:

```
void glCreateFramebuffers(GLsizei n, GLuint * buffers);
```

- Uvolnění VBO identifikátorů:

```
void glDeleteFramebuffers(GLsizei n, GLuint * buffers);
```

- Pro ověření stavu FBO použijeme tuto funkci

```
GLenum glCheckNamedFramebufferStatus(GLuint fbo, GLenum target);
```

- Pokud funkce vraci **GL\_FRAMEBUFFER\_COMPLETE** je vše v pořádku.

- Aktivování/deaktivování FBO:

```
void glBindFramebuffer(GLenum target, GLuint framebuffer);
```

**target** GL\_FRAMEBUFFER stejný buffer pro čtení i zápis,  
GL\_READ\_FRAMEBUFFER, GL\_DRAW\_FRAMEBUFFER.

**framebuffer** Jméno FBO získané [glCreateFramebuffers](#). Pokud nastaveno na 0  
deaktivuje buffer.

- Attachment představuje jeden podbuffer FBO - například hloubkový buffer. Připojení textur k attachmentům:

```
void glNamedFramebufferTexture(GLuint fbo, GLenum attachment,  
    GLuint texture, GLint level);
```

**fbo** framebuffer

**attachment** Které informace budeme zapisovat do textury.

GL\_DEPTH\_ATTACHMENT - hloubkový buffer.

GL\_STENCIL\_ATTACHMENT - stencil buffer. GL\_COLOR\_BUFFERx - barva, nebo jiná informace. Specifikováno pomocí fragment shaderu (layout).

**texture** Identifikátor textury.

**level** Stupeň mipmappingu.

- Layered rendering

```
void glNamedFramebufferTextureLayer(GLuint fbo, GLenum attachment,  
    GLuint texture, GLint level, GLint layer);
```

- Nastavením seznamu attachmentů definujeme, do kterých barevných bufferů se bude kreslit.

```
void glNamedFramebufferDrawBuffers(GLuint id, GLsizei n, const GLenum * bufs);
```

**id** framebuffer

**n** Počet bufferů, do kterých budeme kreslit.

**bufs** Seznam GL\_COLOR\_BUFFERx. Pořadí specifikuje, ke kterému layout ve fragment shaderu bude buffer navázán.

Do textur budeme vykreslovat barvu, normálu a hloubku. Fragment shader:

```
#version 430
layout(location=0)out vec4 fragColor;//jeden výstup - drawbuffer0
layout(location=1)out vec3 fragNormal;//druhy výstup - drawbuffer1
//...
void main(){
    //...
    fragColor=vec4(fCol,1); //zapis barvy
    fragNormal=(fNor+1)/2; //zapis normaly
}
```

## Aplikace - inicializace:

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8F, WIDTH, HEIGHT, 0, GL_RGBA,
    GL_UNSIGNED_BYTE, NULL); //alokujeme misto
//...
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB32F, WIDTH, HEIGHT, 0, GL_RGBA,
    GL_UNSIGNED_BYTE, NULL); //alokujeme misto
//...
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT24,
    WIDTH, HEIGHT, 0, GL_DEPTH_COMPONENT, GL_UNSIGNED_BYTE, NULL);

glCreateFramebuffers(1, &fbo); //vygenerujeme nazev pro FBO

//navazani textur
glNamedFramebufferTexture(fbo, GL_DEPTH_ATTACHMENT,
    tDepth, 0); //navazeme texturu pro hloubku
glNamedFramebufferTexture(fbo, GL_COLOR_ATTACHMENT3,
    tColor, 0); //navazeme texturu pro barvu
glNamedFramebufferTexture(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENTS,
    tNormal, 0); //navazeme texturu pro normalu

GLenum drawBuffers[]={
    GL_COLOR_ATTACHMENT3, //layout(location=0)out vec4 fragColor;
    GL_COLOR_ATTACHMENT5 //layout(location=1)out vec3 fragNormal;
};
glNamedFramebufferDrawBuffers(fbo, 2, drawBuffers); //nastavime seznam ciliu

if(glCheckNamedFramebufferStatus(fbo, GL_FRAMEBUFFER) != GL_FRAMEBUFFER_COMPLETE)
    std::cerr<<"chyba\n";
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

## Aplikace - kreslení:

```
glBindFramebuffer(GL_FRAMEBUFFER, FBO); //aktivujeme FBO
glDrawArrays(...);
glBindFramebuffer(GL_FRAMEBUFFER, 0); //deaktivování FBO
```

# Hierarchický depth buffer

## GLSL - CreateHierarchy:

```
//vertex shader///////////////
#version 430
void main(){
    gl_Position=vec4(0);
}
//geometry shader///////////////
#version 430
layout(points) in;
layout(triangle_strip,max_vertices=4) out;
void main(){
    gl_Position=vec4(-1,-1,0,1);EmitVertex();
    gl_Position=vec4(+1,-1,0,1);EmitVertex();
    gl_Position=vec4(-1,+1,0,1);EmitVertex();
    gl_Position=vec4(+1,+1,0,1);EmitVertex();
}
//fragment shader///////////////
#version 430
layout(binding=0) uniform sampler2D Last;//last mipmap
layout(location=0) out vec2 fDepth;//output depth
ivec2 Coord=ivec2(gl_FragCoord.xy); //coordinates
void main(void){
    vec2 A=texelFetch(Last,Coord*2+ivec2(0,0),0).xy;
    vec2 B=texelFetch(Last,Coord*2+ivec2(1,0),0).xy;
    vec2 C=texelFetch(Last,Coord*2+ivec2(0,1),0).xy;
    vec2 D=texelFetch(Last,Coord*2+ivec2(1,1),0).xy;
    fDepth=vec2(min(min(A.x,B.x),min(C.x,D.x)),max(max(A.y,B.y),max(C.y,D.y)));
}
```

## Aplikace - inicializace:

```
glCreateTextures(GL_TEXTURE_2D,1,&Depth);
//textura obsahujici minimalni a maximalni hloubku
glBindTexture(GL_TEXTURE_2D,Depth);
glTexParameteri(Depth,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
glTexParameteri(Depth,GL_TEXTURE_MIN_FILTER,GL_NEAREST_MIPMAP_NEAREST);
int Size=WSize;
int Level=0;
while(Size>0){//smycka pres lebely
    glTextureImage2DEXT(Depth,GL_TEXTURE_2D,Level++,GL_RG32F,Size,Size,0,GL_RG,
        GL_FLOAT,NULL); //alokace vrstvy
    Size/=2;
}
//render buffer s hloubkou
glCreateRenderbuffers(1,&RBO_Depth);
glNamedRenderbufferStorage(RBO_Depth,GL_DEPTH_COMPONENT,WSize,WSize);

glCreateFramebuffers(1,&FBO); //vygenerujeme nazev pro FBO
```

## Aplikace - render2 - tvorba hierarchie:

```
glUseProgram(CreateHierarchy);
int Level=1;
int ActSize=WSize/2;
glBindFramebuffer(GL_FRAMEBUFFER,HFBO); //bind framebuffer
glBindTextureUnit(0,Depth); //bind depth texture to tex. unit 0
while(ActSize>0){ //while there are
    glViewport(0,0,ActSize,ActSize); //set viewport
    glTexParameteri(Depth,GL_TEXTURE_BASE_LEVEL,Level-1); //starting mipmap level
    glTexParameteri(Depth,GL_TEXTURE_MAX_LEVEL,Level-1); //max mipmap level
    glNamedFramebufferTexture(FBO,GL_COLOR_ATTACHMENT0,Depth,Level);
    GLenum DrawBuffers[]={GL_COLOR_ATTACHMENT0};
    glNamedFramebufferDrawBuffers(1,DrawBuffers);
    glDrawArrays(GL_POINTS,0,1);
    Level++; //increment level of mipmap
    ActSize/=2; //actual size of mipmap
}
glBindFramebuffer(GL_FRAMEBUFFER,0); //unbind framebuffer
glTexParameteri(Depth,GL_TEXTURE_BASE_LEVEL,0); //starting mipmap level
glTexParameteri(Depth,GL_TEXTURE_MAX_LEVEL,Level-1); //max mipmap level
glViewport(0,0,WSize,WSize); //reset viewport
```

# Layered Rendering

## Cube map rendering - CPU

```
GLuint cubeMap;
glCreateTextures(GL_TEXTURE_CUBE_MAP, 1, &cubeMap);
for(size_t i=0;i<6;++i)
    glTextureImage2DEXT(cubeMap, GL_TEXTURE_CUBE_MAP_POSITIVE_X+i, 0,
        GL_RGBA8, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, nullptr);
//it will attach all six sides of cube map
glNamedFramebufferTexture(cubeMap, GL_COLOR_ATTACHMENT0, cubeMap, 0);
```

## Cube map rendering - Vertex Shader

```
layout(location=0) in vec3 position;
uniform float near = 0.1;
uniform float far = 1000;
uniform vec4 lightPosition = vec4(0,0,0,1);
out int vInstanceID;
void main() {
    const mat4 views[6] = {
        mat4(vec4(+0,+0,-1,0), vec4(+0,-1,+0,0), vec4(-1,+0,+0,0), vec4(0,0,0,1)),
        mat4(vec4(+0,+0,+1,0), vec4(+0,-1,+0,0), vec4(+1,+0,+0,0), vec4(0,0,0,1)),
        mat4(vec4(+1,+0,+0,0), vec4(+0,+0,-1,0), vec4(+0,+1,+0,0), vec4(0,0,0,1)),
        mat4(vec4(+1,+0,+0,0), vec4(+0,+0,+1,0), vec4(+0,-1,+0,0), vec4(0,0,0,1)),
        mat4(vec4(+1,+0,+0,0), vec4(+0,-1,+0,0), vec4(+0,+0,-1,0), vec4(0,0,0,1)),
        mat4(vec4(-1,+0,+0,0), vec4(+0,-1,+0,0), vec4(+0,+0,+1,0), vec4(0,0,0,1))
    };

    mat4 projection = mat4(
        vec4(1,0,0,0),
        vec4(0,1,0,0),
        vec4(0,0,-(far+near)/(far-near),-1),
        vec4(0,0,-2*far*near/(far-near),0));
    gl_Position = projection*views[gl_InstanceID]*vec4(position-lightPosition.xyz,1);
    vInstanceID = gl_InstanceID;
}
```

## Cube map rendering - Geometry shader

```
layout(triangles) in;
layout(triangle_strip, max_vertices=3) out;
in int vInstanceID[];
void main(){
    gl_Layer = vInstanceID[0];
    gl_Position = gl_in[0].gl_Position; EmitVertex();
    gl_Position = gl_in[1].gl_Position; EmitVertex();
    gl_Position = gl_in[2].gl_Position; EmitVertex();
    EndPrimitive();
}
```

Thank you for your attention! Questions?