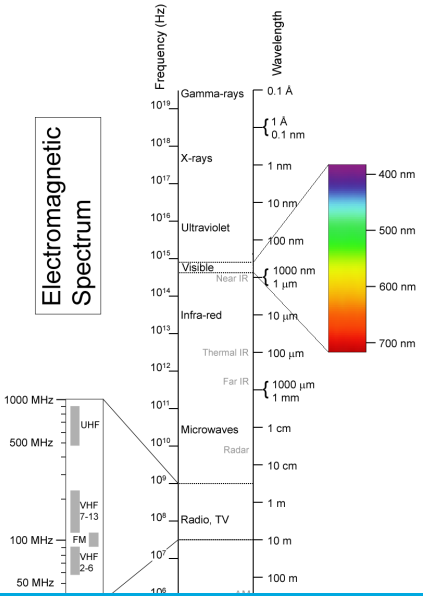
PGR - Fragment Shader, Light, Color / Světlo a Barvy

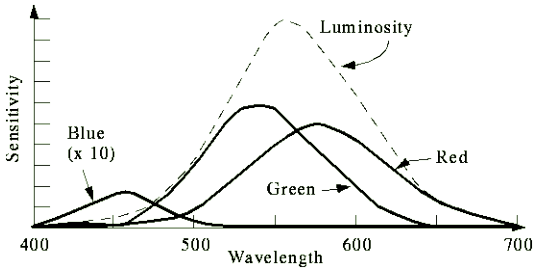
Tomáš Milet

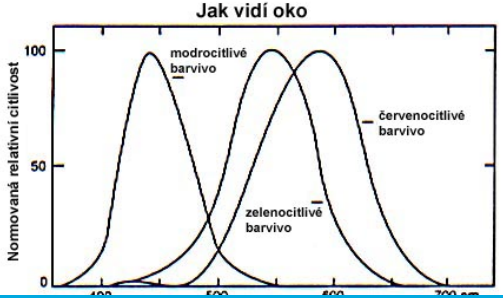
Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2. 612 66 Brno - Královo Pole
imilet@fit.vutbr.cz

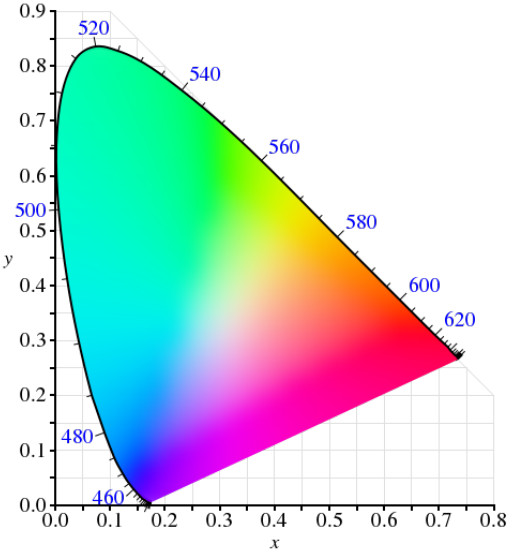
23. prosince 2022

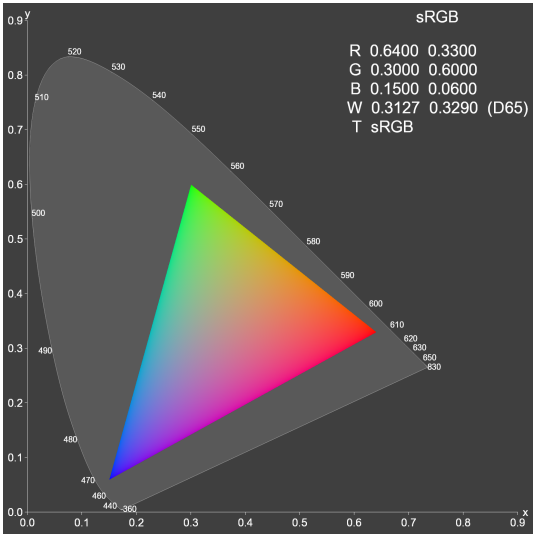
Color / Barva


Electromagnetic Spectrum
Frequency (Hz)
Wavelength
Gamma-rays
0.1 Å
1 Å
0.1 nm
X-rays
1 nm
10 nm
Ultraviolet
100 nm
Visible
Near IR
1000 nm
1 μm
Infra-red
10 μm
Thermal IR
100 μm
Far IR
1000 μm
1 mm
Microwaves
1 cm
Radar
10 cm
1 m
Radio, TV
10 m
100 m
400 nm
500 nm
600 nm
700 nm
1000 MHz
500 MHz
UHF
VHF 7-13
100 MHz
FM
VHF 2-6
50 MHz


Luminosity
Sensitivity
Blue
(x 10)
Red
Green
400
500
600
700
Wavelength


Jak vidí oko
Normovaná relativní citlivost
100
50
0
modrocitlivé barvivo
červenocitlivé barvivo
zelenocitlivé barvivo


520
540
560
580
600
620
500
480
460
y
x
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


sRGB
R 0.6400 0.3300
G 0.3000 0.6000
B 0.1500 0.0600
W 0.3127 0.3290 (D65)
T sRGB
y
x

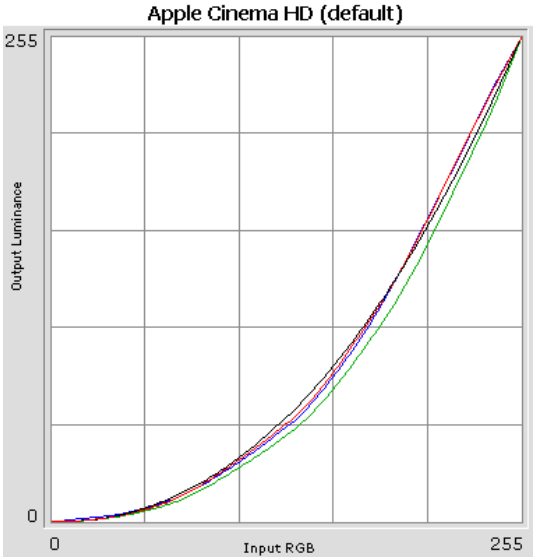
- CIE 1931

Logarithmic sensitivity of human eye
Logaritmická citlivost oka

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

$$a = 0.055$$

$$t = 0.0031308$$



Colors in OpenGL / Barva v OpenGL

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.

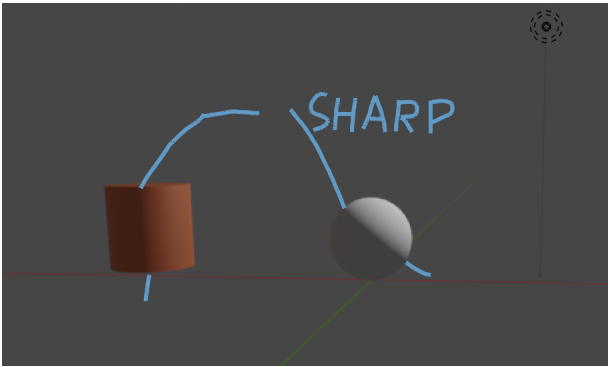
Inputs and framebuffer / Vstupy a framebuffer :

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

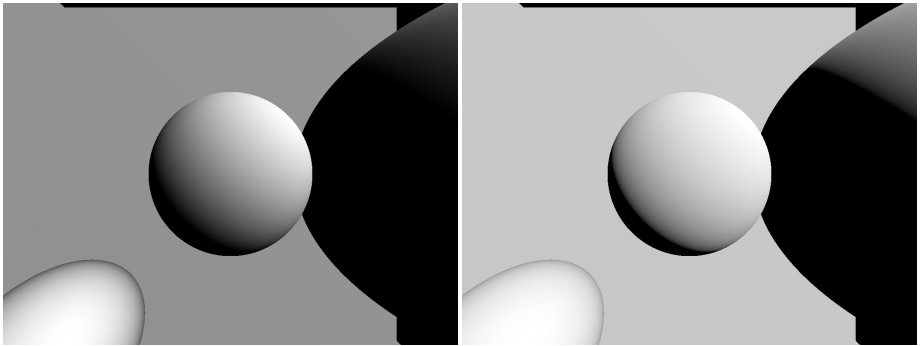
```
//default framebuffer
SDL_GL_SetAttribute(SDL_GL_FRAMEBUFFER_SRGB_CAPABLE,1);
...
//enabling SRGB in OpenGL
glEnable(GL_FRAMEBUFFER_SRGB);
```

Gamma Correct

- 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.



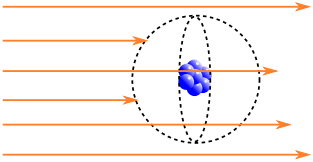
- Left incorrect, right correct.
- Vlevo bez korekce, vpravo s korekcí.

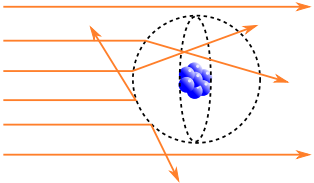


Light-Matter Interaction / Interakce světla a hmoty

Absoption, scattering

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



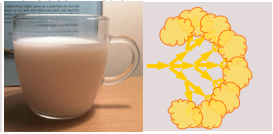


Clear media

Clear absorbent media

Cloudy media

Translucent media

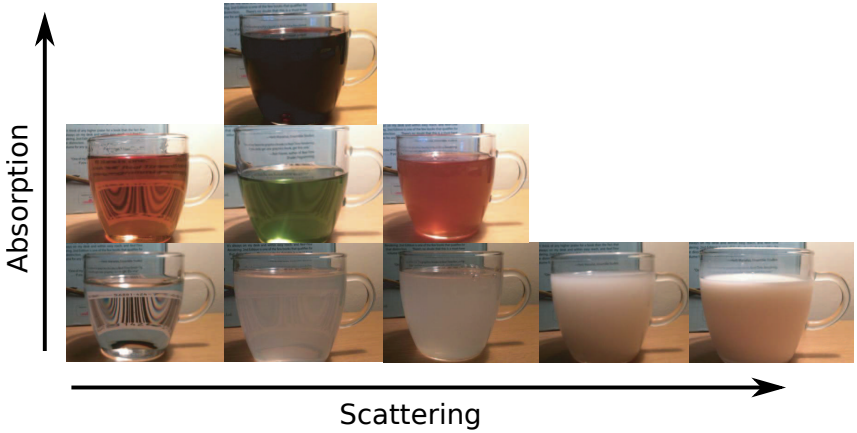


Water absorbent over large distances

Air scattering over large distances

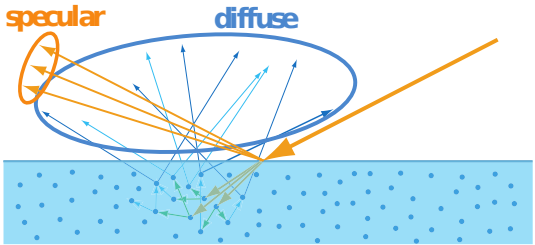
Absoption, scattering

- Absorption and scattering are semi independent values.
- But fotons that are absorbed cannot be scattered.
- And fotons that are scattered cannot be absorbed.
- Absorpce a odraz jsou více méně nezávislé hodnoty.
- Ale fotony, které jsou absorbovány nemůžou být odraženy a naopak.



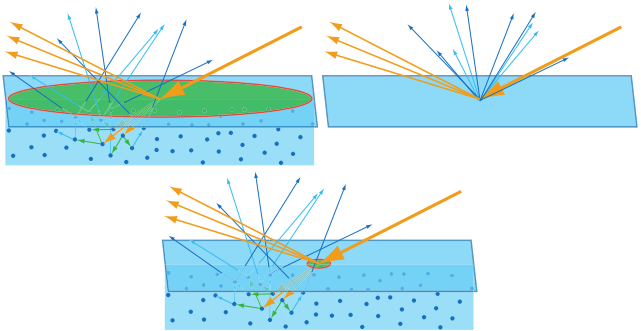
Specular, diffuse component

- 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 zalomilo uvnitř materiálů. Vstup světla a výstup světla z materiálu musí být k sobě blíž než je velikost pixelu.



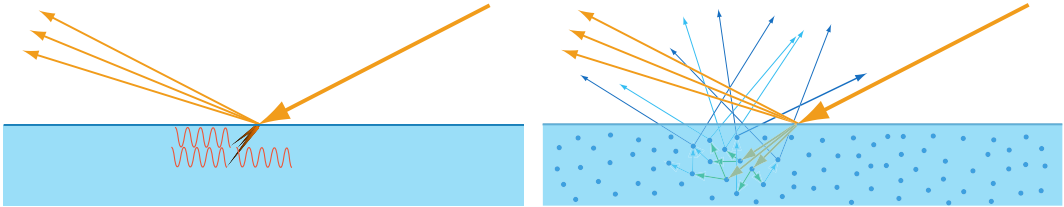
Subsurface scattering

- 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, ...



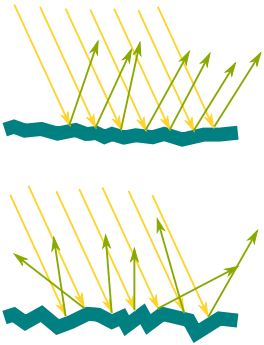
Metals / Lom světla kovů

- 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 (teplo).
- Vlevo: kov, vpravo: nekov

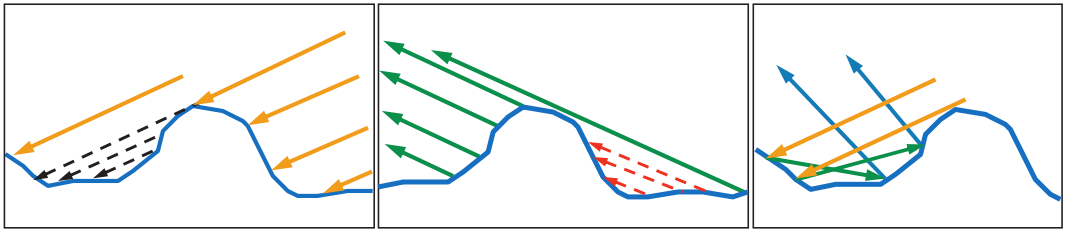


Surface roughness and microfacets / Drsnost povrchu a mikroplošky

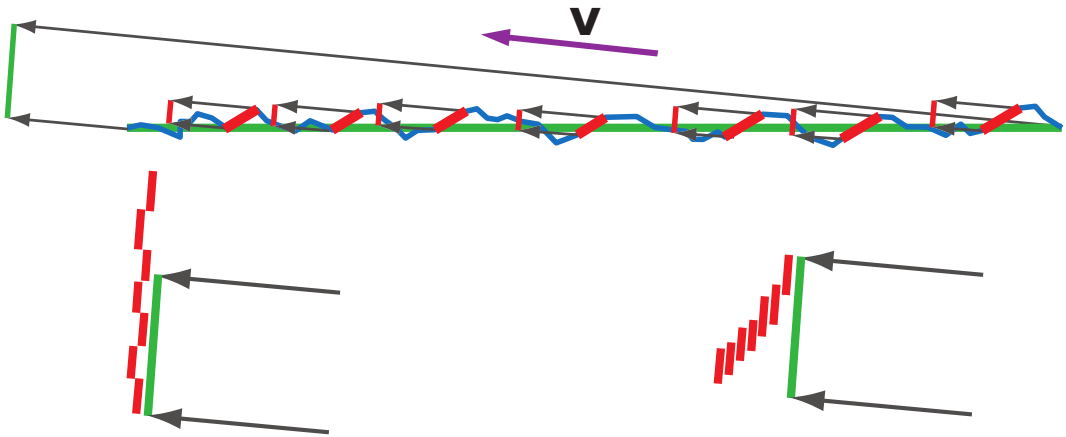
- Microscopic structures determine coherency of reflected light rays.
- Microfacet distribution determines the clarity of mirror images.
- Mikroskopické struktury určují koherentnost odražených paprsků.
- Distribuce mikroplošek určuje čistost zrcadlového odrazu.



Microfacet shadowing / Zastínění mikroplošek



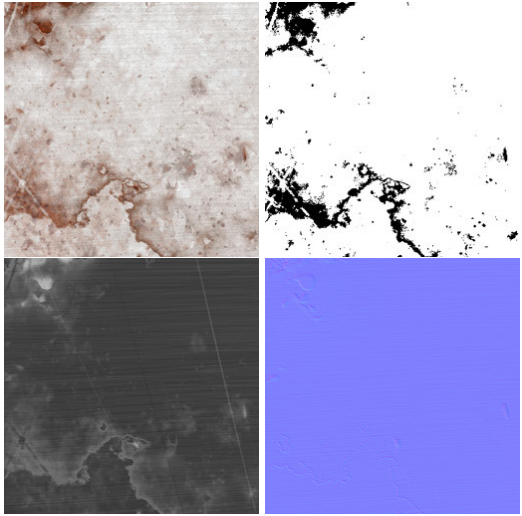
Microfacet shadowing / Zastínění mikroplošek




physically based rendering

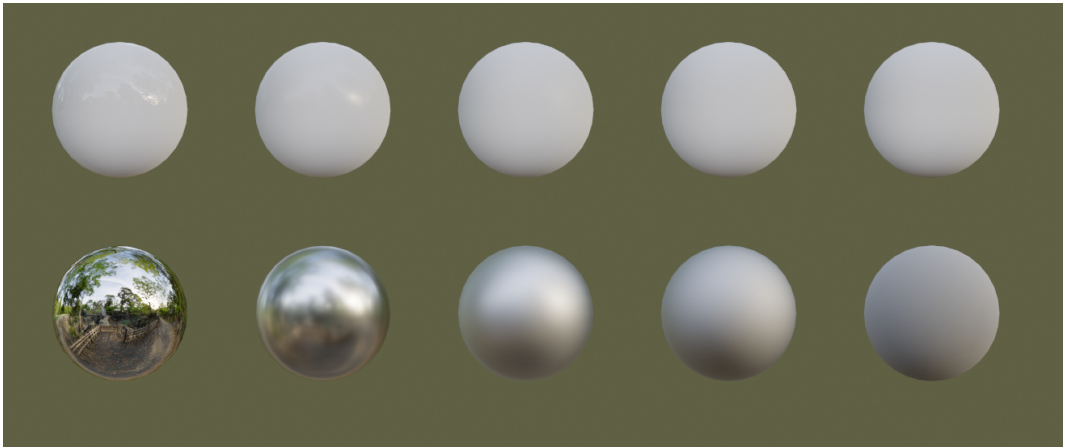
Metalic workflow

- Metalic 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) předtavuje distribuci mikroplošek.



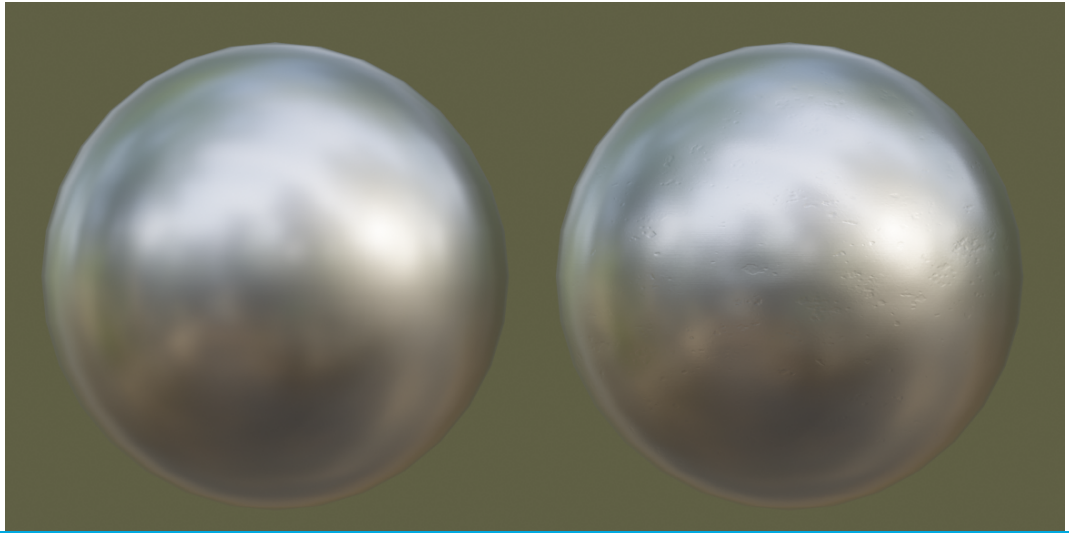
Metalness and roughness

- 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.

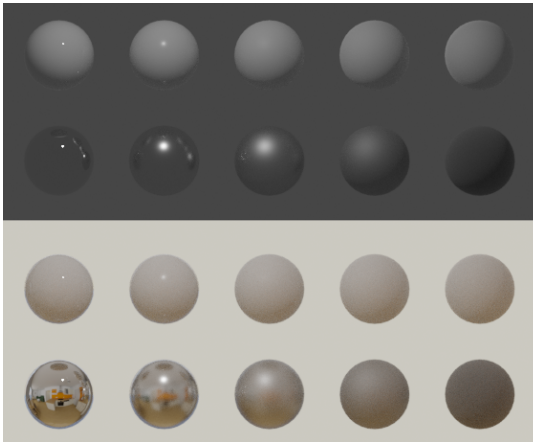


Normal map

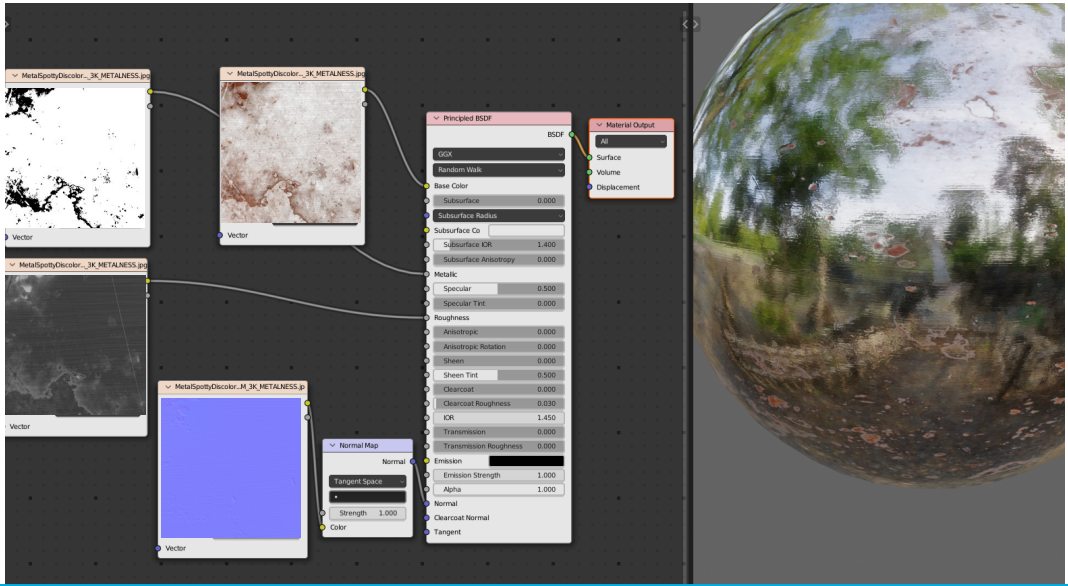
- Left: without normal map, Right: with normal map
- Vlevo: bez normálové mapy, vpravo s normálovou mapou



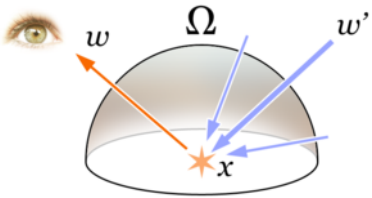
Environment mapping



Metalic workflow



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



Siggraph 1986 :

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

- Light reflection in "one point"
- Differential scene surface.
- Odraz světla v "jednom bodu".
- Diferenciální ploška scény.

- Light reflection in "one point"
- Differential scene surface.
- Odraz světla v "jednom bodu".
- Diferenciální ploška scény.

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

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

- Light reflection in "one point"
- Differential scene surface.
- 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.
- Almost like probability density function (pdf).
- Bidirectional Reflectance Distribution Function.
- Hustota pravděpodobnosti (skoro).

- Bidirectional Reflectance Distribution Function.
- Almost like probability density function (pdf).
- Bidirectional Reflectance Distribution Function.
- Hustota pravděpodobnosti (skoro).

Reciprocity
Reciprocita

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

Conservation of energy
Zákon zachování energie

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

Izotropy x Anisotropy
Izotropie x Anizotropie

BRDF variants / Varianty BRDF

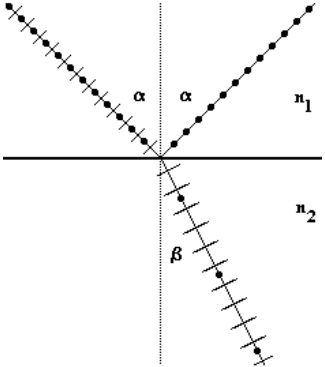
- Phong
- Blinn-Phong
- Cook-Torrance
- GGX (microfacet distribution)
- ...

BRDF variants / Varianty BRDF

- Phong
- Blinn-Phong
- Cook-Torrance
- GGX (microfacet distribution)
- ...

- Pragmatic x Physically Based
- Depends on Application.

Odraz a lom světla



- Reflection and refraction happens on interface between two mediums.
- Index of Refraction (IOR) (light speed in the medium).
- Reflection based light polarization.
- Odraz a lom na hranicích prostředí.
- Indexy lomu (rychlost v daném prostředí).
- Polarizace odrazem.

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

Fresnel equations /Fresnellovy vzorce

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

- Depend on polarization / Závislé na polarizaci.
- Impractical / Nepraktické.

Fresnel equations /Fresnellovy vzorce

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

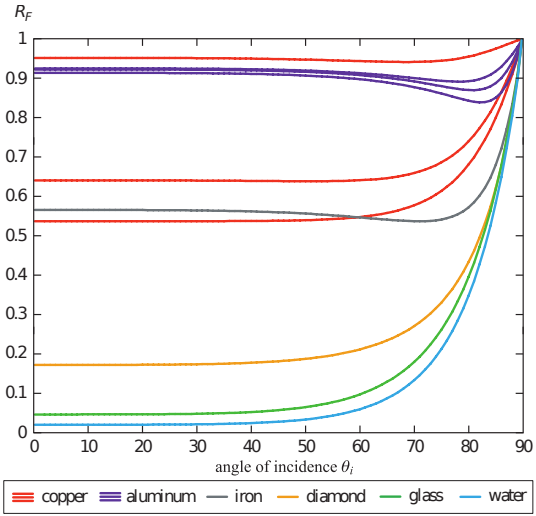
- Depend on polarization / Závislé na polarizaci.
- Impractical / Nepraktické.

Schlick approximation / Schlickova aproximace

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

- R_0 is material property. / R_0 je vlastnost materiálu.
- Isolants: / Pro izolanty: $R_0 \approx 0.04$.
- $\cos\theta \to 0$ ... $R \to 1$

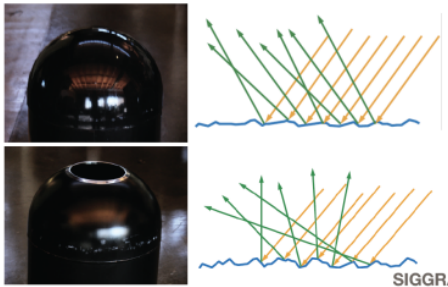
Fresnel reflectance

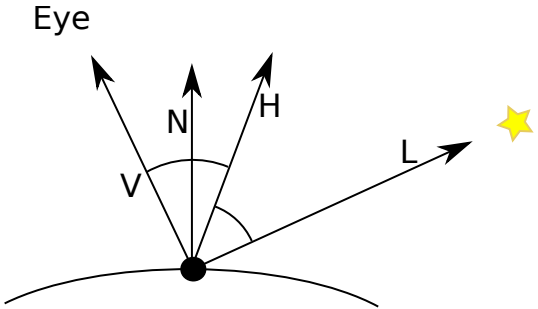


Fresnel reflectance

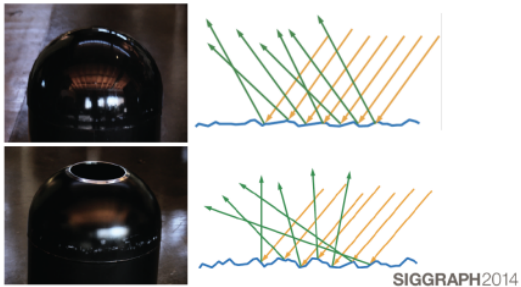
Material	$F(0^\circ)$ (Linear)	$F(0^\circ)$ (sRGB)	Color
Water	0.02,0.02,0.02	0.15,0.15,0.15	
Plastic / Glass (Low)	0.03,0.03,0.03	0.21,0.21,0.21	
Plastic High	0.05,0.05,0.05	0.24,0.24,0.24	
Glass (High) / Ruby	0.08,0.08,0.08	0.31,0.31,0.31	
Diamond	0.17,0.17,0.17	0.45,0.45,0.45	
Iron	0.56,0.57,0.58	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	

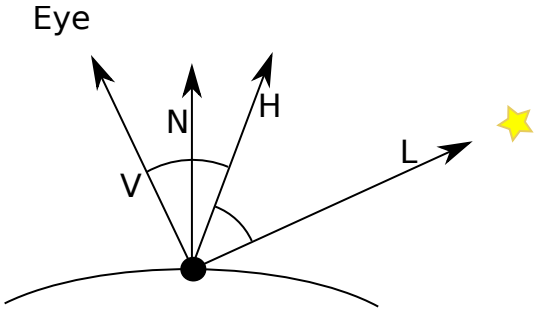
Imprefect surface / Nedokonalý povrch





Which microfacets have correct orientation?
Které plošky jsou správně orientované?





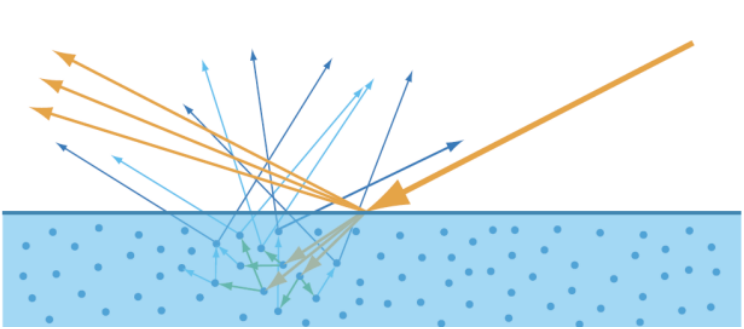
Which microfacets have correct orientation?
Které plošky jsou správně orientované?

- Reflects light toward viewer. / Odrážejí ze světla do pozorovatele.
- The normal vector is preciselly in between. / Mají normálu přesně mezi.

! Half vector.

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

Diffuse + Specular



Specular

- "Mirror" reflection / "Zrcadlový" odraz
- Dominant for metals / Dominantní pro kovy

Diffuse

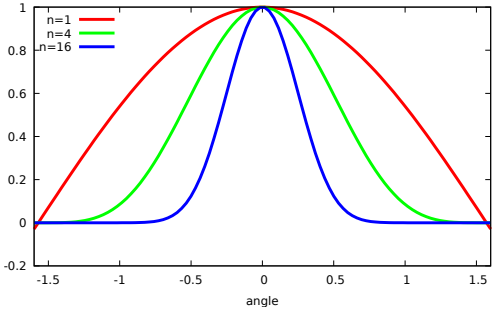
- Reflection inside materials / Odraz uvnitř materiálu.
- Dominant for isolants / Dominantní pro izolanty.

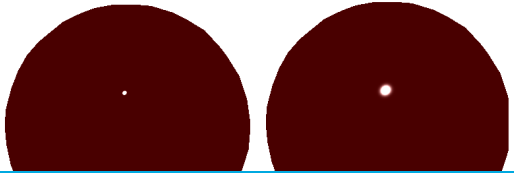
Specular reflection / Spekulární odraz

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

- F - Fresnell/Schlick
- D - Distribution function of microfacets / Distribuční funkce plošek
- G - Geometric shadowing of microfacests / Geometrické zastínění plošek

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





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

What is missing? / Co chybí?

- Color of materials (isolants) / Barva materiálu (izolantú). R_0 – "color of metals"/ "barva kovu".
- ! Refracted rays / Lomené paprsky.
- Conservation of energy / Zachování energie.

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

- Independent on viewer position / Nezáleží na poloze pozorovatele.
- $1/\pi$ – for energy conservation / pro zachování energie.

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

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

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

- $\mathbf{n} \cdot \mathbf{l}$ – from rendering eq. / z renderovací rovnice.
- Energy conservation for specular light / Zachování energie pro specular.
- See / Odvození na `http://www.farbrausch.de/~fg/stuff/phong.pdf`

Point light sources / Bodový zdroj světla

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

- Point light has zero sized area / Bod má nulovou plochu.
- Point lights are signularities. / Bodové zdroje jsou singularity.
- How to change rendering equation? L_i? / Jak upravit renderovací rovnici? Jaké L_i?

Point light sources / Bodový zdroj světla

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

- Point light has zero sized area / Bod má nulovou plochu.
- Point lights are signularities. / Bodové zdroje jsou singularity.
- How to change rendering equation? L_i? / Jak upravit renderovací rovnici? Jaké L_i?
- Can be set using color of ideal diffuse material. / Nastavíme ho přes barvu ideálního difúzního povrchu.

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

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

- Loop over light sources / Sčítáme přes světla.
- π is eliminated / se vykrátí.

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

- Loop over light sources / Sčítáme přes světla.
- π is eliminated / se vykrátí.

What is missing? / Co tam chybí?

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

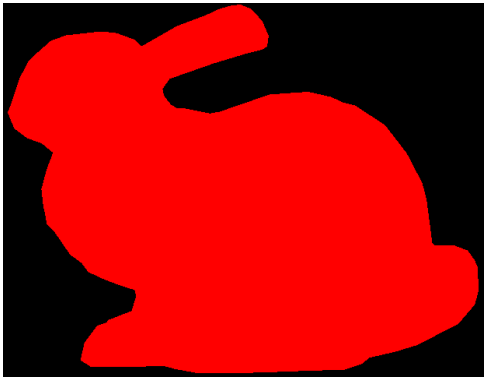
- Loop over light sources / Sčítáme přes světla.
- π is eliminated / se vykrátí.

What is missing? / Co tam chybí?

- Distance to the light. / Vzdálenost od světla $L_i = f(|\mathbf{p}_i - \mathbf{x}|^2)$
- Light cone. / Světelný kužel.
- Multi bounce reflection of light. / Vícenásobný odraz světla.

$$L_o = \sum_i k_a L_i$$

$$k_a = k_d$$



- Crude approximation of multi bounced refleced light. / Hrubá aproximace

- Perpendicular to the surface / Kolmé k povrchu
- $|N| = 1$ (normalized / normalizované)
- 1 per triangles or 1 per vertex / 1 na trojúhelník nebo 1 na vrchol

- $\vec{N_{\text{face}}} = \vec{AB} \times \vec{AC}$
- $\vec{N_{\text{vertex}}} = \mathbf{normalize}\left(\sum_{i=1}^{n} N_{\text{face}i}\right)$
- Triangle area weighted average / Průměr vážený plochou trojúhelníků

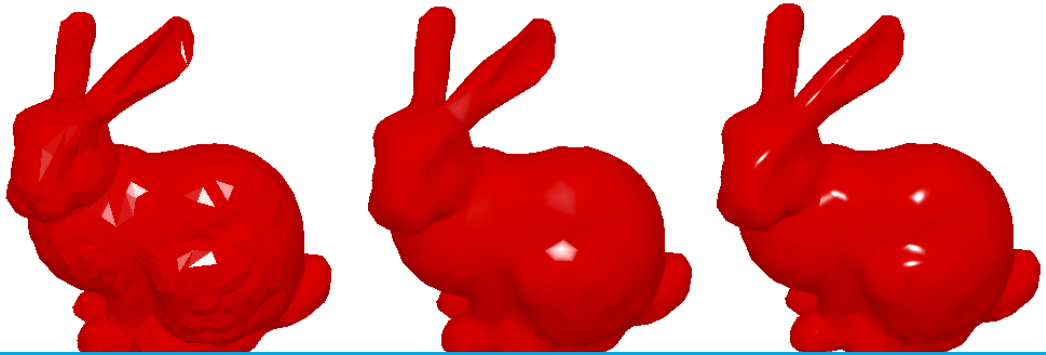
- Transformation / Transformace :
  - Roration + translation / Rotace + posun - $N_{\text{eye}} = \mathit{ModelView} \cdot N_{\text{model}}$
  - Scale - $N_{\text{eye}} = (\mathit{ModelView}^T)^{-1} \cdot N_{\text{model}}$
  - ! Vector needs to be renormalized after transformation using scale! / Při scale je nutné vektory po transformaci znovu normalizovat!

Shadinig / Stínování

"Flat"shading / stínování
- Illumination is computed once per triangles. / Osvětlují se celé trojúhelníky.
- Nothing is interpolated. / Nic se neinterpoluje.

Gouraudovo shading / stínování - "per vertex lighting"
- Illumination is computed once per vertex. / Osvětlení se počítá pro vrcholy.
- Illumination is interpolated. / Interpolují se barvy.
- We do not need it nowdays. / Dnes už nemá smysl.

Phong shading / stínování - "per fragment lighting"
- Illumination is computed once per fragment. / Osvětlení se počítá pro fragmenty.
- Normal vectors are interpolated. / Interpolují se normálové vektory.



Textures / Textury

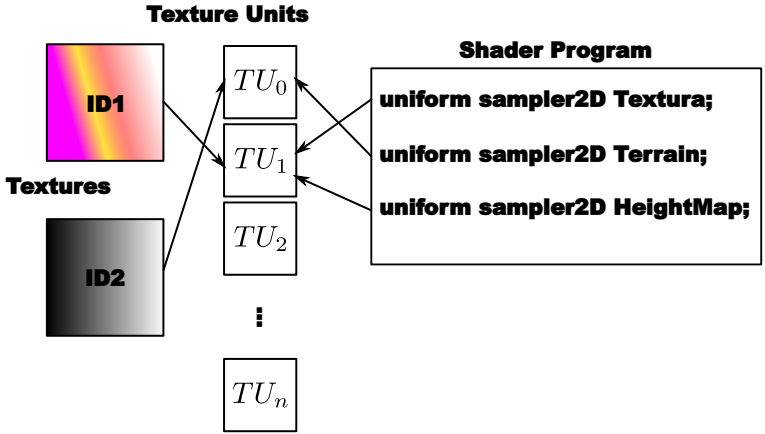
Textures - uploading of raster data / nahrání rastrových dat

- A texture is raster image mapped to the geometry.
- It is an object in OpenGL.
- Textura je rastrový obrázek, který se namapuje na geometrii.
- V OpenGL je to objekt.

```
//create texture identifier
glCreateTextures(GL_TEXTURE_2D,1,&tColor);

//set texture parameters
//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);

//allocation and data upload
glTextureImage2DEXT(tColor,GL_TEXTURE_2D,0,//allocate
  GL_RGBA32F,//internal format on GPU
  WIDTH,HEIGHT,0,GL_RGBA,//format on CPU
  GL_UNSIGNED_BYTE,//type of data
  Data);//pointer to data on CPU
```

Usage of textures / Nastavení textur

- Shader can access textures using sampler uniform variable.
- Textures is bound to texture unit.
- Uniform variable is bound to texture unit.
- 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:

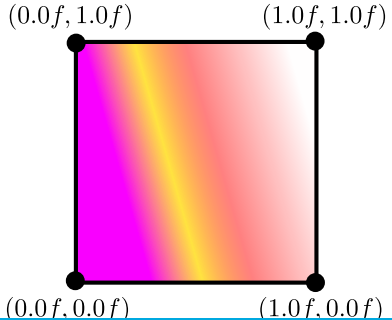
```
#version 330
layout(binding=0)uniform sampler2D Textura;
layout(binding=1)uniform sampler2D Terrain;
layout(binding=1)uniform sampler2D HeightMap;
void main(){
  texture(Texture,coords);
  //...
```

Application:

```
//bind textures to texture units
glBindTextureUnit(0,ID2);
glBindTextureUnit(1,ID1);
```

Texture space /Adresování textur

- Texture coordinates
- Size of texture in pixels does not matter.
- Left bottom corner - (0.0f,0.0f)
- Right top corner - (1.0f,1.0f)
- Textura se adresuje texturovacími koordináty.
- Velikost Textury nehraje roli.
- Levý dolní roh 2D textury má souřadnice (0.0f,0.0f)
- Pravý horní roh 2D textury má souřadnice (1.0f,1.0f)



Texture atlas / Atlas textur

- Multiple images in one textures.
- Reduces number of texture units.
- Reduces texture switching.
- Více obrázků v jedné textuře.
- Stačí jedna texturovací jednotka.
- Zamezíme přepínání.

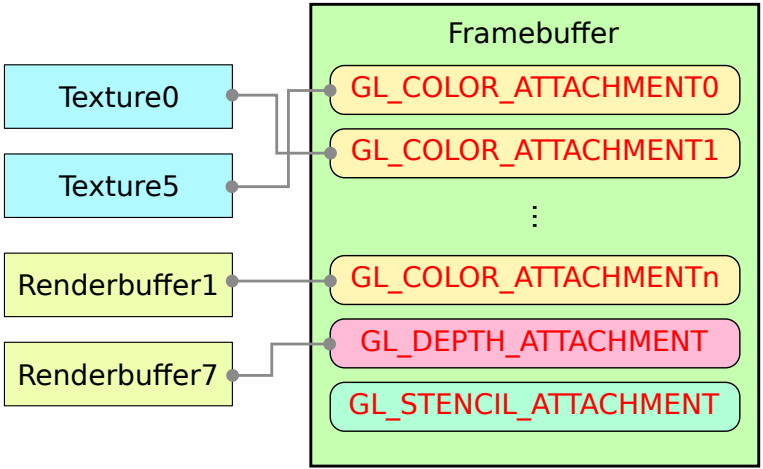
http://www.scriptspot.com/3ds-max/scripts/texture-atlas-generator

Framebuffer

Frame Buffer Object (FBO)

- Usually, a scene is rendered into default framebuffer.
- From OpenGL 3.3, it is possible to create custom framebuffer and draw the scene into it.
- Framebuffer is composed of multiple 2D arrays (depth, stencil and few color buffers).
- FBO supports rendering into textures.
- It is possible to render other informations as well (normals, positions, depths, ..., Multiple Render Targets).
- FBO is building step for many graphical effects (water, SSAO, ...)
- Supports deferred shading.
- Layered rendering.

- 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


Framebuffer
Texture0
Texture5
Renderbuffer1
Renderbuffer7
GL_COLOR_ATTACHMENT0
GL_COLOR_ATTACHMENT1
GL_COLOR_ATTACHMENTn
GL_DEPTH_ATTACHMENT
GL_STENCIL_ATTACHMENT

Usage / Použití FBO

1. Reserve FBO integer id.
2. Attach textures/renderbuffers
3. Setup the list of attachments
4. ...
5. Bind FBO.
6. Draw scene.
7. Unbindg FBO.
8. Process rendered textures.

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

Creation/Deletion / Vytvoření/uvolnění

- Creation of FVO identifier / Vytvoření FBO identifikátorů:

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

- Deletion of FBO identifier / Uvolnění FBO identifikátorů:

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

Check the state / Ověření stavu FBO

- To check the state of the FBO:
- Pro ověření stavu FBO použijeme tuto funkci:

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

- GL_FRAMEBUFFER_COMPLETE – the state is OK.
- Pokud funkce vraci GL_FRAMEBUFFER_COMPLETE je vše v pořádku.

Bind/Unbind / Aktivování/deaktivování FBO

- To bind/unbind framebuffer:
- Aktivování/deaktivování FBO:

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

target	GL_FRAMEBUFFER == GL_READ_FRAMEBUFFER + GL_DRAW_FRAMEBUFFER.
framebuffer	0 - default framebuffer.

Attachment of textures / Připojení textur

- One attachment represents one sub buffer of FBO – for example depth buffer. Attachment of textures:
- 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	Attachment type / Které informace budeme zapisovat do textury. GL_DEPTH_ATTACHMENT - depth / hloubkový buffer. GL_STENCIL_ATTACHMENT - stencil / stencil buffer. GL_COLOR_BUFFERx - color, other informations / barva, nebo jiná informace. Fragment shader (layout(location=x) / Specifikováno pomocí fragment shaderu (layout).
texture	Texture indentifier. / Identifikátor textury.
level	Mipmap level / Stupeň mipmappingu.

- Layered rendering

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

List of attachments / Seznam attachmentů (MRT)

- List of attachments defines, which color buffer will be rendered.
- Nastavením seznamu attachmentů definujeme, do kterých barevných bufferů se bude kreslit.

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

id	framebuffer
n	Nof color buffers / Počet bufferů, do kterých budeme kreslit.
bufs	Seznam GL_COLOR_BUFFERx. The index *x* specifies the location in FS: layout(location=*x*) / Pořadí specifikuje, ke kterému layout ve fragment shaderu bude buffer navázán.

Full example / Kompletní příklad

We want to draw color, normal and depth. Fragment shader:
Do textur budeme vykreslovat barvu, normálu a hloubku. Fragment shader:

```
#version 430
layout(location=0)out vec4 fragColor;//first output - drawbuffer0
layout(location=1)out vec3 fragNormal;//second output - drawbuffer1
//...
void main(){
  //...
  fragColor=vec4(fCol,1);//color
  fragNormal=(fNor+1)/2;//normal
}
```

Full example / Kompletní příklad

Application – initialization / Aplikace - inicializace:

```
glTexImage2D(GL_TEXTURE_2D,0,GL_RGBA8F,WIDTH,HEIGHT,0,GL_RGBA,GL_UNSIGNED_BYTE,NULL);//color texture
glTexImage2D(GL_TEXTURE_2D,0,GL_RGB32F,WIDTH,HEIGHT,0,GL_RGBA,GL_UNSIGNED_BYTE,NULL);//normal texture
glTexImage2D(GL_TEXTURE_2D,0,GL_DEPTH_COMPONENT24,WIDTH,HEIGHT,0,GL_DEPTH_COMPONENT,GL_UNSIGNED_BYTE,NULL);//depth texture

glCreateFramebuffers(1,&fbo);//reserve FBO identifier

//attach textures
glNamedFramebufferTexture(fbo,GL_DEPTH_ATTACHMENT,tDepth,0);
glNamedFramebufferTexture(fbo,GL_COLOR_ATTACHMENT3,tColor,0);
glNamedFramebufferTexture(GL_FRAMEBUFFER,GL_COLOR_ATTACHMENT5,tNormal,0);

//setup the list of color attachments
GLenum drawBuffers[]={
  GL_COLOR_ATTACHMENT3,//layout(location=0)out vec4 fragColor;
  GL_COLOR_ATTACHMENT5 //layout(location=1)out vec3 fragNormal;
};
glNamedFramebufferDrawBuffers(fbo,2,drawBuffers);

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

Full example / Kompletní příklad

Drawing / Kresleni:

```
glBindFramebuffer(GL_FRAMEBUFFER,FBO);//bind framebuffer
glDrawArrays(...);
glBindFramebuffer(GL_FRAMEBUFFER,0);//bind default framebuffer
```

Hierarchical depth buffer

Full example

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)));
}
```

Kompletní příklad tvorba hier. z-bufferu

Initialization / inicializace:

```
glCreateTextures(GL_TEXTURE_2D,1,&Depth);
//texture containing minimal and maximal depth
glBindTexture(GL_TEXTURE_2D,Depth);
glTextureParameteri(Depth,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
glTextureParameteri(Depth,GL_TEXTURE_MIN_FILTER,GL_NEAREST_MIPMAP_NEAREST);
int Size=WSize;
int Level=0;
while(Size>0){//loop over levels
  glTextureImage2DEXT(Depth,GL_TEXTURE_2D,Level++,GL_RG32F,Size,Size,0,GL_RG,
    GL_FLOAT,NULL);//allocate mipmap level
  Size/=2;
}
//render buffer with depth
glCreateRenderbuffers(1,&RBO_Depth);
glNamedRenderbufferStorage(RBO_Depth,GL_DEPTH_COMPONENT,WSize,WSize);

glCreateFramebuffers(1,&FBO);//create FBO identifier
```

Kompletní příklad tvorba hier. z-bufferu

Computing hierarchical depth buffer / 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
  glTextureParameteri(Depth,GL_TEXTURE_BASE_LEVEL,Level-1);//starting mipmap level
  glTextureParameteri(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 framebffer
glTextureParameteri(Depth, GL_TEXTURE_BASE_LEVEL, 0);//starting mipmap level
glTextureParameteri(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?