PGR - Shadows, procedural generation / Stíny, procedurální generování

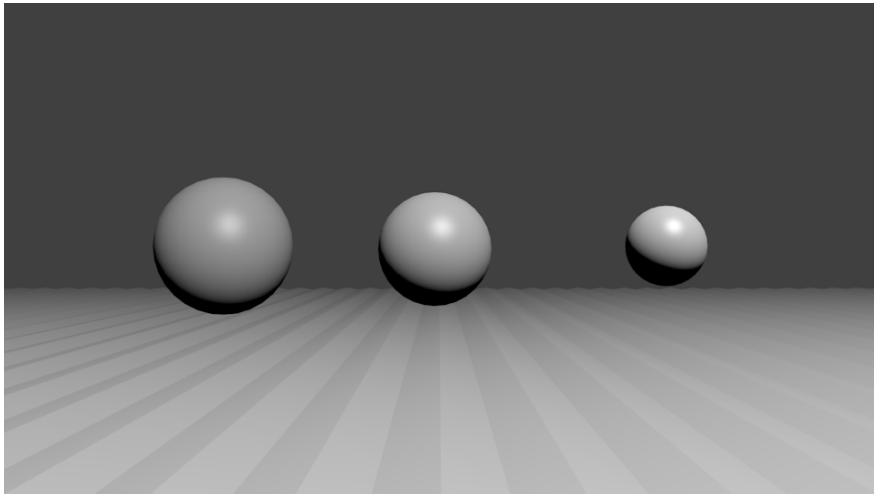
Tomáš Milet

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

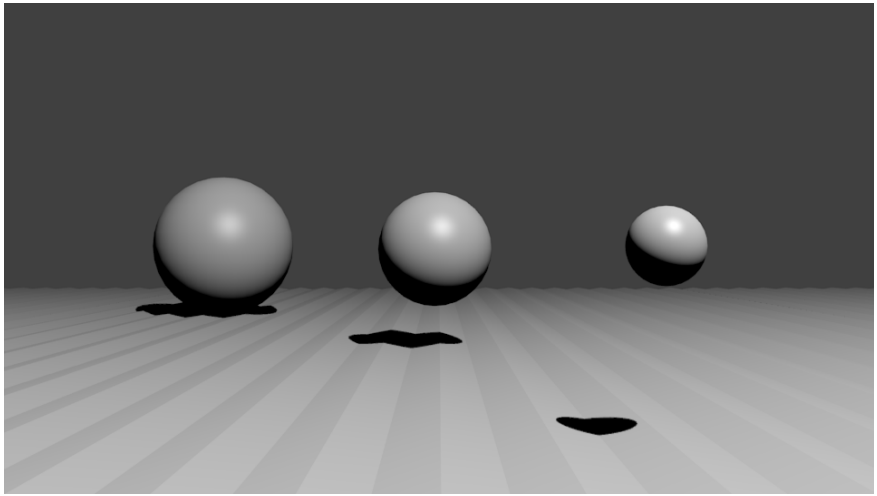
23. prosince 2022

Shadows / Stíny

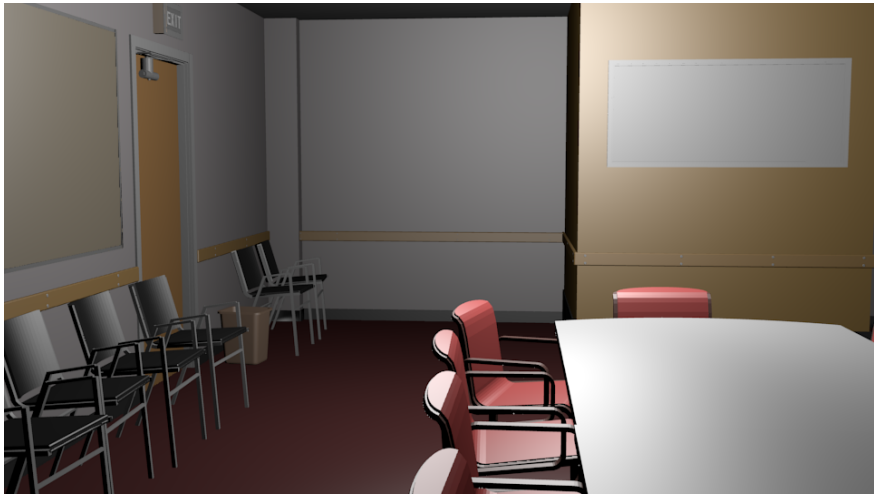
Why do we need shadows? / Proč potřebujeme stíny?



Why do we need shadows? / Proč potřebujeme stíny?

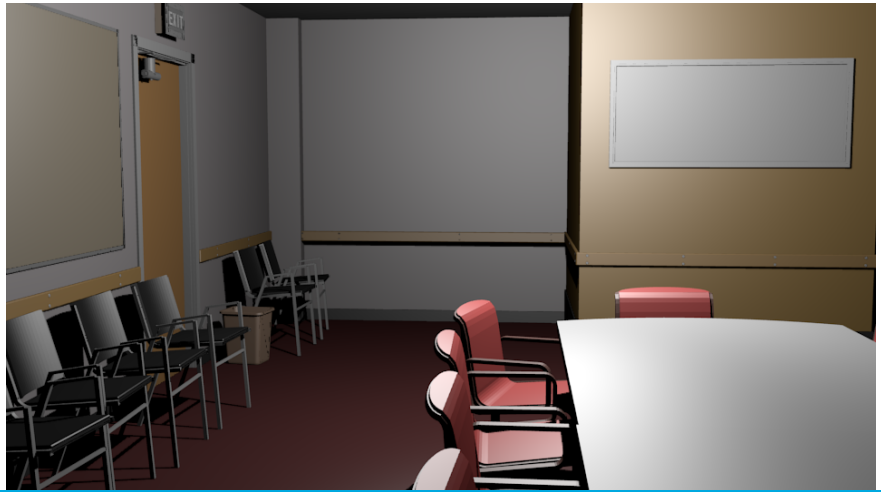


Why do we need shadows? / Proč potřebujeme stíny?



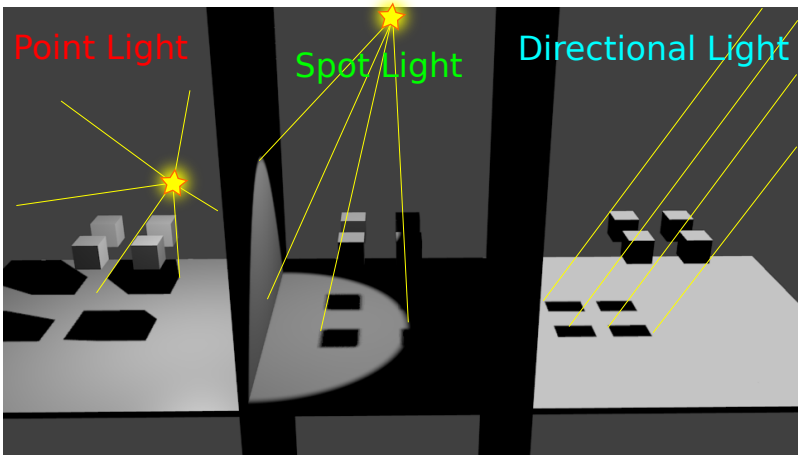
Why do we need shadows? / Proč potřebujeme stíny?

- Shadows help us to comprehend 3D structure of a scene.
- Mutual location of objects.
- Stíny pomáhají pochopit 3D scénu.
- Vzájemné polohy mezi objekty.



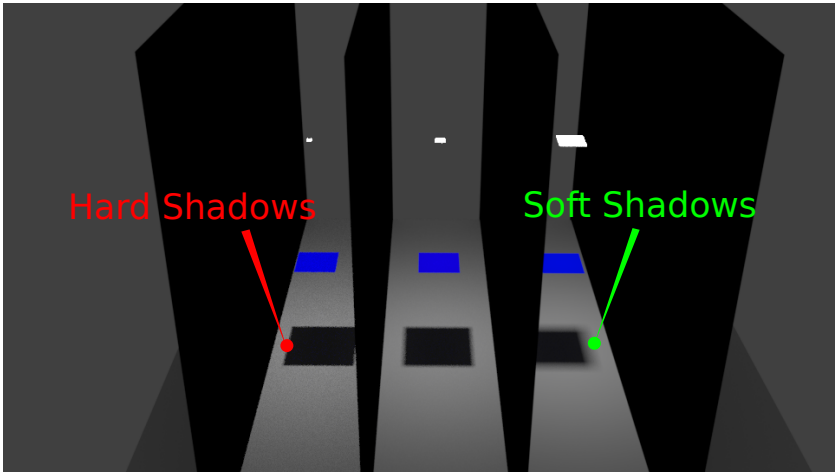
Types of light sources / Druhy světel

- Omnidirectional point light source.
- Spot light source.
- Directional light source.



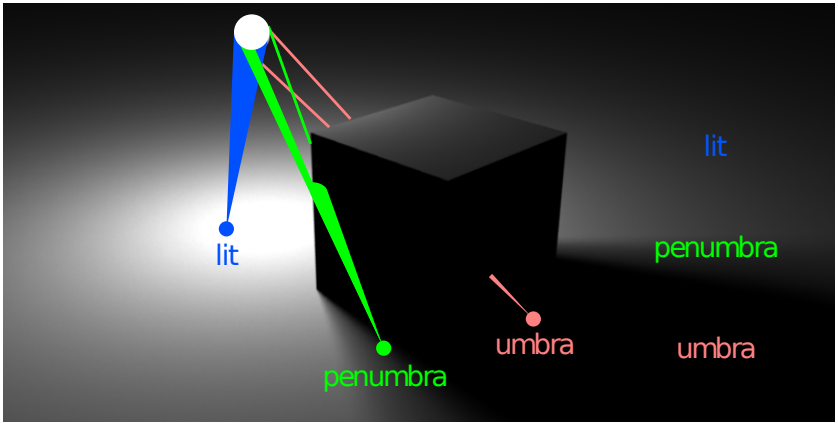
Types of shadows / Druhy stínů

- Hard shadows are created by infinitesimal light sources (common in computer graphics).
- Soft shadow are created from area light sources.
- Tvrdé stíny vznikají z nekonečně malých světelných zdrojů (časté v počítačové grafice).
- Měkké stíny vznikají z plošných zdrojů světla.



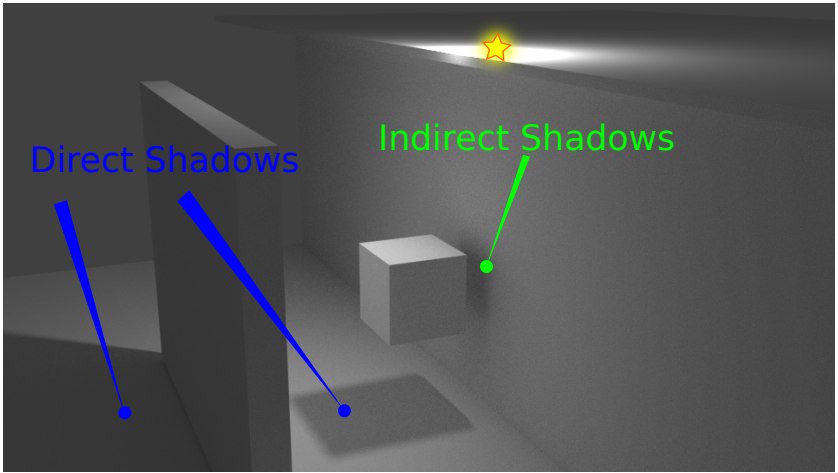
Types of shadows / Druhy stínů

- Fully lit parts of the scene see all points of the light source.
- Fully shadowed parts of the scene do not see any point of the light source.
- Half shadow (penumbra) is created in regions with partial visibility to the light source.



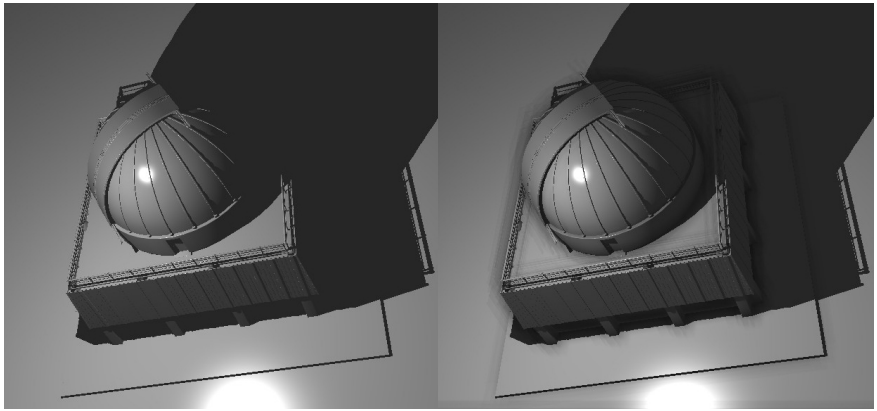
Types of shadows / Druhy stínů

- Shadows are created from direct illumination (can be hard, if the light source is small).
- Shadows are created from indirect illumination (soft shadows).
- Stíny vznikají z přímého osvětlení (můžou být tvrdé, pokud je světlo malé).
- Stíny vznikají z nepřímého osvětlení (měkké stíny).



Ambient occlusion

- Ambient occlusion - approximation of global illumination.
- Ambient occlusion - metoda pro aproximaci části globálního osvětlení a simulaci stínů nepřímého osvětlení.



Ambient occlusion

Ambient occlusion

Ambient occlusion

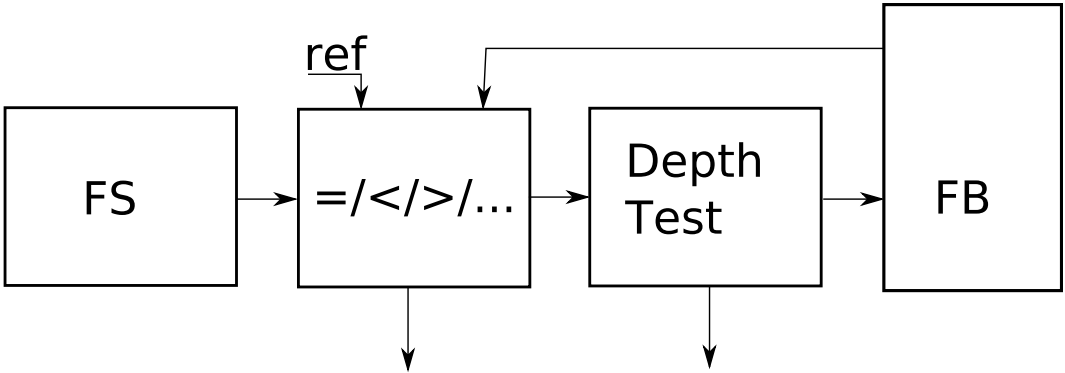
Shadow volumes

Stencil buffer

- Stencil buffer is a part of framebuffer.
- Intergers, 8bits
- Další část framebuferu
- Celočíselný, obvykle 8bpp

```
SDL_GL_StencilSize(8);//stencil buffer of default framebuffer
glClear(GL_STENCIL_BUFFER_BIT);//clearing of stencil buffer
glClearStencil(0);//cleraing value
glStencilMaskSeparate(GL_FRONT_AND_BACK,~0); //masking writes (bit mask)
```

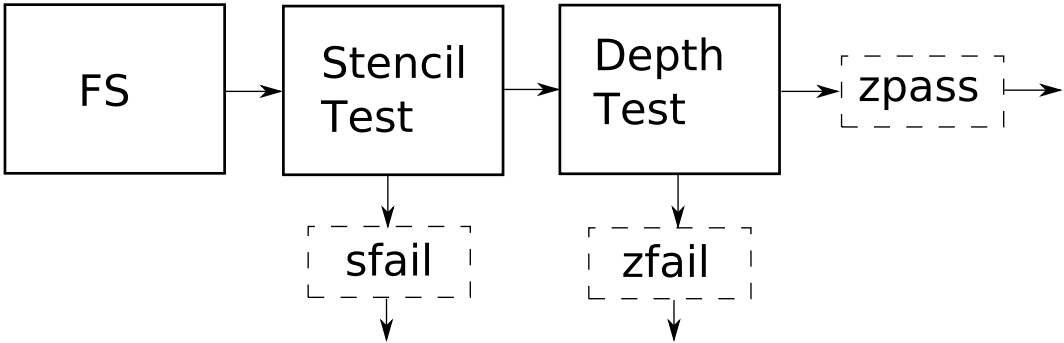
Stencil test




```
glEnable(GL_STENCIL_TEST);
glStencilFuncSeparate(face, func, ref, mask);
```

- func ∈ {GL_ALWAYS, GL_NEVER, GL_LESS, GL_GREATER, ...}
- if(S&mask == ref&mask)//GL_EQUAL

Changes to stencil buffer / Změny stencil bufferu




```
glStencilOpSeparate(face, sfail, zfail, zpass);
```

- GL_KEEP, GL_ZERO, GL_REPLACE, GL_INVERT
- GL_INCR, GL_DECR, GL_INCR_WRAP, GL_DECR_WRAP

$255 + 1 = 255 \iff$ GL_INCR
$255 + 1 = 0 \iff$ GL_INCR_WRAP

Shadow Volumes

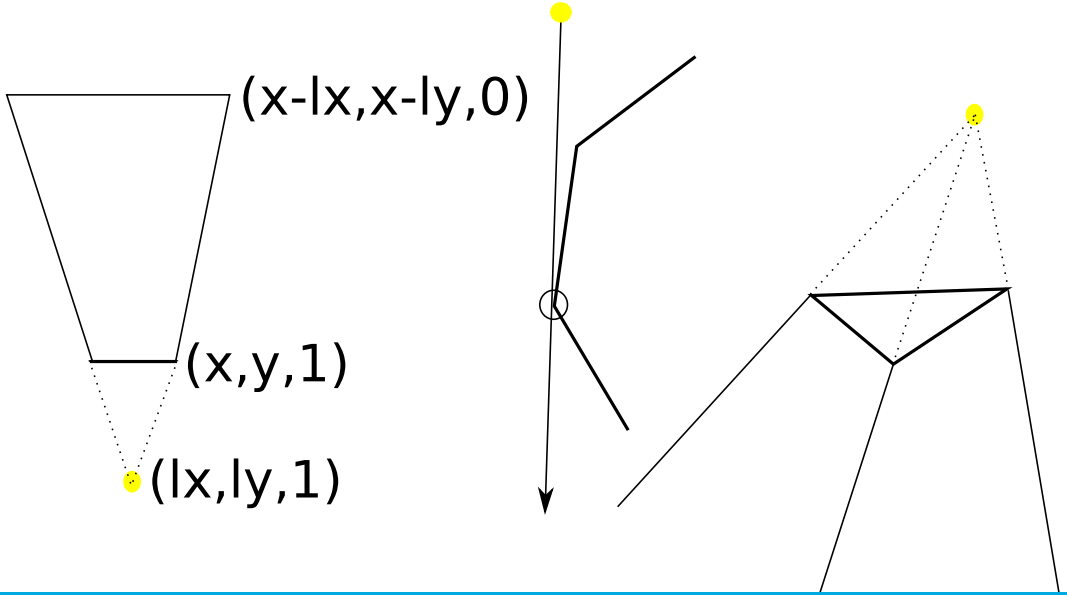
- Shadow volume is an algorithm for creation of precise hard shadows.
- It requires 3 render passes.
- 1. pass – rendering of the scene with ambient illumination.
- 2. pass – render of shadow volume geometry into stencil buffer.
- 3. pass – rendering of the scene with diffuse and specular illumination using stencil buffer.
- Two variants – zpass, zfail.

- Shadow volume je metoda pro tvorbu přesných tvrdých stínů.
- Vyžaduje 3 kreslící průchody.
- První průchod - vykreslení scény pouze s ambientním světlem.
- Druhý průchod - vykreslení stínové geometrie a vytvoření stencilové masky.
- Třetí průchod - vykreslení scény s difuzním a spekulárním světlem s využitím stencilové masky.
- Existují dvě verze: zpass a zfail.

Shadow Volumes algoritmus

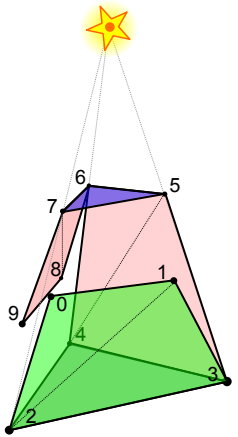
1. 1. pass – scene+ambient light → color+depth buffer
   1. Draw the scene with ambient illumination. / Vykresli scénu s ambietním osvětlením.
2. 2. pass – shadow volume geometry → stencil buffer
   1. Disable writes to color and depth. / Vypni kreslení barvy a modifikaci depth bufferu.
   2. Set stencil operation to increment on front-facing triangles and to decrement on back-facing triangles. / Nastav stencilovou operaci na incrementaci u přivrácených trojúhelníku a dekrementaci při odvrácených trojúhelníků.
   3. Enable writes to stencil buffer. / Povol zápis do stencil bufferu.
   4. Draw shadow volume geometry into stencil buffer. / Vykresli stínovou geometrii - vytvoří se stencilová maska.
   5. Disable writes into stencil buffer. / Vypni zápis do stencilového bufferu.
   6. Enable writes to color and depth buffer. / Povol kreslení barvy a modifikaci depth bufferu.
3. 3. pass – scene+diffuse+specular → color buffer
   1. Enable depth test (pass if stencil value is 0). / Zapni stencil test – při stencilové hodnotě 0 test projde.
   2. Enable additive bleding. / Povol aditivní blending.
   3. Draw the scene with diffuse and specular illumination. / Vykresli scénu s difuzním a spekuláním osvětlením.
   4. Disable stencil test. / Vypni stencil test.

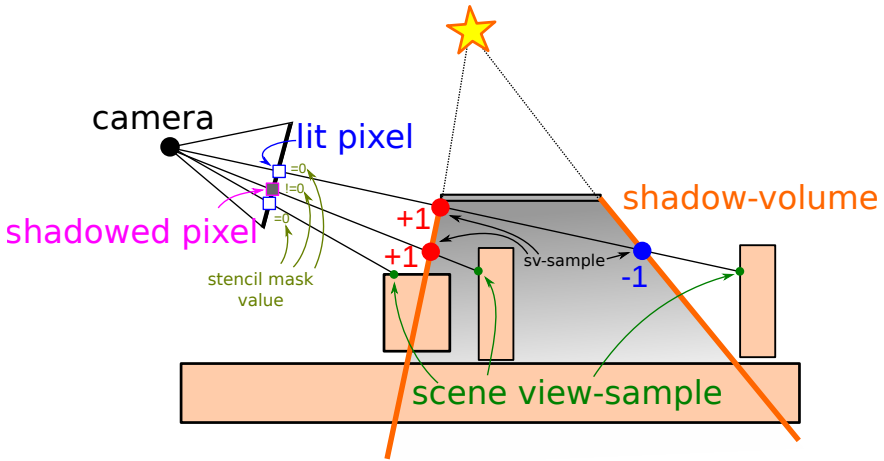
Jak na stínová tělesa?



Shadow volumes - geometry

```
#version 330
layout(triangles)in;
layout(triangle_strip,max_vertices=10)out;
uniform mat4 MVP,M;//matice
uniform vec4 LightPosition;//pozice svetla
void main(){
  vec4 LP=M*LightPosition;
  vec4 p[6];
  p[0]=gl_in[0].gl_Position;//body trojuhelniku
  p[1]=gl_in[1].gl_Position;
  p[2]=gl_in[2].gl_Position;
  p[3]=vec4(gl_in[0].gl_Position.xyz*LP.w-LP.xyz,0);//v nekonecnu
  p[4]=vec4(gl_in[1].gl_Position.xyz*LP.w-LP.xyz,0);
  p[5]=vec4(gl_in[2].gl_Position.xyz*LP.w-LP.xyz,0);
  vec3 N=normalize(cross((p[1]-p[0]).xyz,(p[2]-p[0]).xyz));
  float Distance=dot(N,LP.xyz)-dot(N,p[0].xyz);
  if(Distance<=0){//otocime volume vnitrkem ven
    vec4 c=p[0];p[0]=p[1];p[1]=c;
    c=p[3];p[3]=p[4];p[4]=c;
  }
  gl_Position=MVP*p[0];EmitVertex();
  gl_Position=MVP*p[1];EmitVertex();
  gl_Position=MVP*p[3];EmitVertex();
  gl_Position=MVP*p[4];EmitVertex();
  gl_Position=MVP*p[5];EmitVertex();
  gl_Position=MVP*p[1];EmitVertex();
  gl_Position=MVP*p[2];EmitVertex();
  gl_Position=MVP*p[0];EmitVertex();
  gl_Position=MVP*p[5];EmitVertex();
  gl_Position=MVP*p[3];EmitVertex();
  EndPrimitive();
}
```




camera
lit pixel
shadowed pixel
=0
!=0
=0
stencil mask value
+1
+1
sv-sample
-1
shadow-volume
scene view-sample

Shadow Volumes - 1. pass

```
//activate phong program
glUseProgram(phongProgram);

//enable ambient lighting
glProgramUniform4fv(phongProgram,
  lightAmbientColorUniform,light.amientColor);

//disable diffuse and specular lighting
glProgramUniform4f (phongProgram,
  lightDiffuseColorUniform,0,0,0,0);
glProgramUniform4f (phongProgram,
  lightSpecularColorUniform,0,0,0,0);

//draw the scene
glMultiDrawElementsIndirect(...);
```

Shadow Volumes - 2. pass

```
//activate shadow volume geometry program
glUseProgram(shadowVolumeProgram);

glColorMask(GL_FALSE,GL_FALSE,GL_FALSE,GL_FALSE);
glDepthMask(GL_FALSE);

glEnable(GL_STENCIL_TEST);
glStencilFunc(GL_ALWAYS,0,0);
glStencilOpSeparate(GL_FRONT,GL_KEEP,GL_KEEP,GL_INCR_WRAP);
glStencilOpSeparate(GL_BACK ,GL_KEEP,GL_KEEP,GL_DECR_WRAP);

//draw the shadow geometry
glMultiDrawElementsIndirect(...);

glStencilOp(GL_KEEP,GL_KEEP,GL_KEEP);
glDepthMask(GL_TRUE);
glColorMask(GL_TRUE,GL_TRUE,GL_TRUE,GL_TRUE);
```

Shadow Volumes - 3. pass - diffuse/specular

```
//activate phong program
glUseProgram(phongProgram);

glProgramUniform4f(phongProgram,lightAmbientColorUniform,0,0,0,0);
glProgramUniform4f(phongProgram,lightDiffuseColorUniform,light.diffuseColor);
glProgramUniform4f(phongProgram,lightSpecularColorUniform,light.specularColor);

//stencil tess will pass only when the value inside stencil buffer is 0
glStencilFunc(GL_EQUAL,0,0xff);

//we need to add the diffuse and specular illumination
glEnable(GL_BLEND);
glBlendFunc(GL_ONE,GL_ONE);

//draw the scene
glMultiDrawElementsIndirect(...);

glDisable(GL_STENCIL_TEST);
```

Multiple lights / Víc světel

N lights means 2^N combinations of shadowing :(What to do with it? *N* světel znamená 2^N kombinací zastíněností :(Co s tím?

N lights means 2^N combinations of shadowing :(What to do with it? N světel znamená 2^N kombinací zastíněností :(Co s tím?

$$
\begin{aligned}
L &= k_e + \sum_{i=1}^{n} k_a \cdot L_{a(i)} \quad + k_d \cdot L_{d(i)} \cdot (\vec{L}_i \cdot \vec{N}) + k_s \cdot L_{s(i)} \cdot (\vec{V} \cdot \vec{R}_i)^n \\
L_0 &= k_e + \sum_{i=1}^{n} k_a \cdot L_{a(i)} \\
L_i &= L_{i-1} \quad + k_d \cdot L_{d(i)} \cdot (\vec{L}_i \cdot \vec{N}) + k_s \cdot L_{s(i)} \cdot (\vec{V} \cdot \vec{R}_i)^n
\end{aligned}
$$

N lights means 2^N combinations of shadowing :(What to do with it? N světel znamená 2^N kombinací zastíněností :(Co s tím?

$$
\begin{aligned}
L &= k_e + \sum_{i=1}^{n} k_a \cdot L_{a(i)} \quad + k_d \cdot L_{d(i)} \cdot (\vec{L}_i \cdot \vec{N}) + k_s \cdot L_{s(i)} \cdot (\vec{V} \cdot \vec{R}_i)^n \\
L_0 &= k_e + \sum_{i=1}^{n} k_a \cdot L_{a(i)} \\
L_i &= L_{i-1} \quad + k_d \cdot L_{d(i)} \cdot (\vec{L}_i \cdot \vec{N}) + k_s \cdot L_{s(i)} \cdot (\vec{V} \cdot \vec{R}_i)^n
\end{aligned}
$$

- Draw the scene with ambient+emission / Vykreslit scénu s ambient+emission
- For every light source / Pro každé světlo
  - Prepare stencil buffer / Připravit stencil buffer
  - Blend together diffuse+specular / Přiblendovat diffuse+specular
  - (glDepthFunc(GL_LEQUAL), glBlendFunc(GL_ONE, GL_ONE))

Properties / Výhody a nevýhody

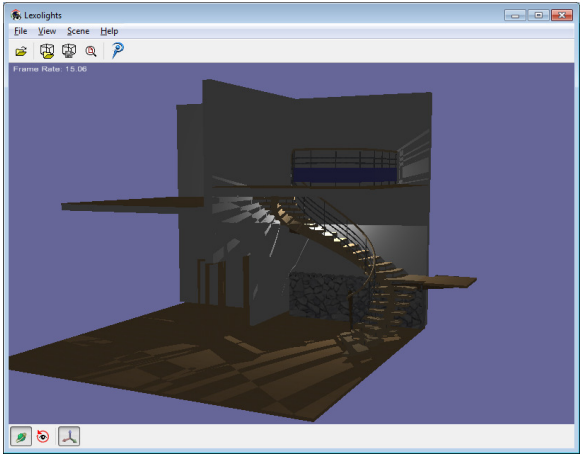
+

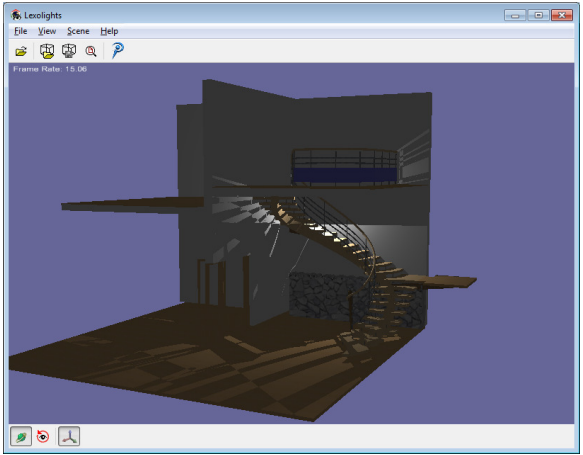
- Dynamic scenes / Dynamické scény
- Problematic scens / Problematické scény
- Per-sample precise / Sedí na pixel přesně

−

- Slow / Pomalé
- Fillrate dependent / Závislé na fillrate
- Needs to create shadow geometry / Konstrukce stínových těles
- Only hard shadows / Jen ostré stíny

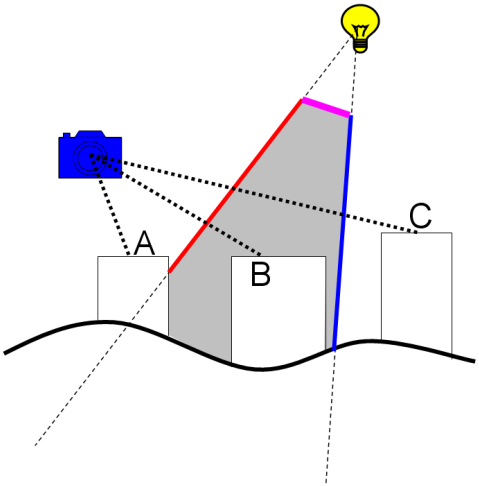
Problem / Problém...





If the camera enters shadow volume. Stačí vlézt dovnitř stínového tělesa.

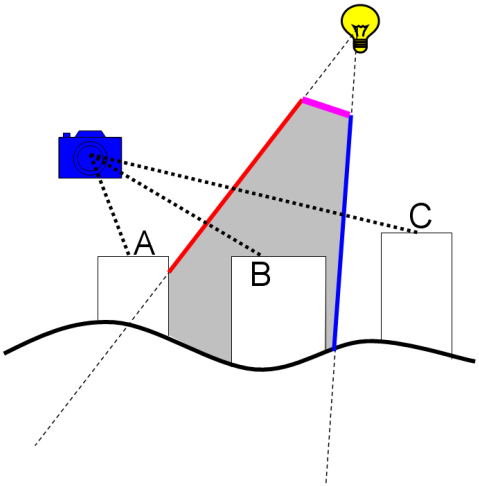
Pixel is shadowed, if the ray enters a shadow volume and does not get out.
Pixel je ve stínu když paprsek vletí dovnitř a nevyletí ven.



Z_PASS :
Computes visible shadow volume samples / Počítám viditelné vzorky stínových těles
+1, -1
Z_FAIL :
Computes invisible shadow volume samples / Počítám neviditelné vzorky stínových těles
-1, +1

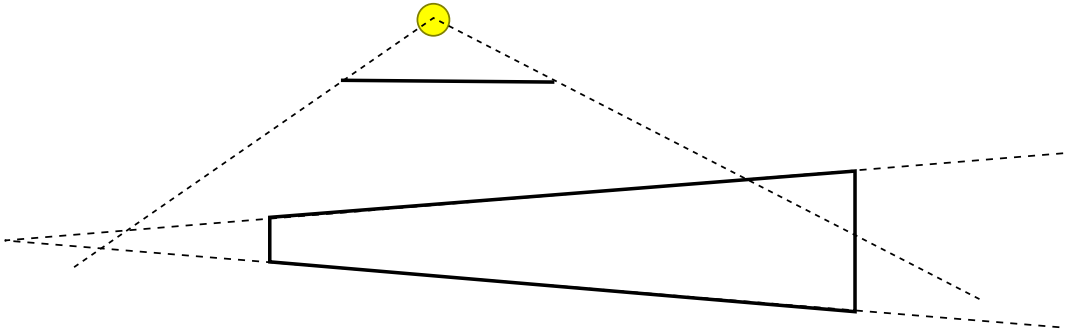
Z_PASS vs. Z_FAIL

Pixel is shadowed, if the ray enters a shadow volume and does not get out.
Pixel je ve stínu když paprsek vletí dovnitř a nevyletí ven.

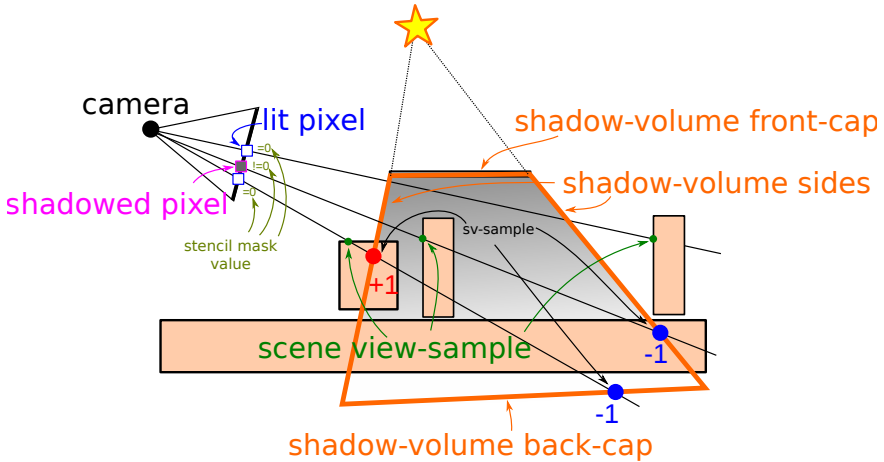


Z_PASS :
Computes visible shadow volume samples / Počítám viditelné vzorky stínových těles
+1, -1
Z_FAIL :
Computes invisible shadow volume samples / Počítám neviditelné vzorky stínových těles
-1, +1

Another problem / Další zádrhel



- Shadow volume has to be watertight.
- All sides has to be renderable.
- Problem with far-plane.
- Stínové těleso musí být uzavřené.
- A jeho stěny musí být "viditelné".
- Problém je far-plane.


camera
lit pixel
shadowed pixel
=0
=0
=0
stencil mask value
shadow-volume front-cap
shadow-volume sides
sv-sample
+1
scene view-sample
-1
-1
shadow-volume back-cap

Shadow Volumes - 2. pass is modified

```
//activate shadow volume geometry program
glUseProgram(shadowVolumeProgram);
glProgramUniformMatrix4fv(shadowVolumeProgram,mvpUniform,
  1,GL_FALSE,infiniteProjectionMatrix);

glColorMask(GL_FALSE,GL_FALSE,GL_FALSE,GL_FALSE);
glDepthMask(GL_FALSE);

glEnable(GL_STENCIL_TEST);
glStencilFunc(GL_ALWAYS,0,0);
glDepthFunc(GL_LESS);
glStencilOpSeparate(GL_FRONT,GL_KEEP,GL_INCR_WRAP,GL_KEEP);
glStencilOpSeparate(GL_BACK ,GL_KEEP,GL_DECR_WRAP,GL_KEEP);

//draw the shadow geometry with caps
glMultiDrawElementsIndirect(...);

glStencilOp(GL_KEEP,GL_KEEP,GL_KEEP);
glDepthMask(GL_TRUE);
glColorMask(GL_TRUE,GL_TRUE,GL_TRUE,GL_TRUE);
```

$$\lim_{\text{far}\to+\infty}\begin{pmatrix} \frac{2\text{near}}{\text{right}-\text{left}} & 0 & \frac{\text{right}+\text{left}}{\text{right}-\text{left}} & 0 \\ 0 & \frac{2\text{near}}{\text{top}-\text{bottom}} & \frac{\text{top}+\text{bottom}}{\text{top}-\text{bottom}} & 0 \\ 0 & 0 & -\frac{\text{far}+\text{near}}{\text{far}-\text{near}} & -\frac{2\text{nearfar}}{\text{far}-\text{near}} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} \frac{2\text{near}}{\text{right}-\text{left}} & 0 & \frac{\text{right}+\text{left}}{\text{right}-\text{left}} & 0 \\ 0 & \frac{2\text{near}}{\text{top}-\text{bottom}} & \frac{\text{top}+\text{bottom}}{\text{top}-\text{bottom}} & 0 \\ 0 & 0 & -1 & -2\text{near} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$$\lim_{\text{far}\to+\infty} \begin{pmatrix} \frac{2\text{near}}{\text{right}-\text{left}} & 0 & \frac{\text{right}+\text{left}}{\text{right}-\text{left}} & 0 \\ 0 & \frac{2\text{near}}{\text{top}-\text{bottom}} & \frac{\text{top}+\text{bottom}}{\text{top}-\text{bottom}} & 0 \\ 0 & 0 & -\frac{\text{far}+\text{near}}{\text{far}-\text{near}} & -\frac{2\text{nearfar}}{\text{far}-\text{near}} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} \frac{2\text{near}}{\text{right}-\text{left}} & 0 & \frac{\text{right}+\text{left}}{\text{right}-\text{left}} & 0 \\ 0 & \frac{2\text{near}}{\text{top}-\text{bottom}} & \frac{\text{top}+\text{bottom}}{\text{top}-\text{bottom}} & 0 \\ 0 & 0 & -1 & -2\text{near} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

:) It is possible to render objects in infinity. / Vidíme do nekonečna.

? What about z-test? We are dividing infinite lenght into finite number of splits. / Co z-test? Dělíme nekonečnou délku na konečný počet dílků.

Z-buffer and inifinity

$$P(z) = \frac{z - 2n}{z} = 1 - \frac{2n}{z}$$

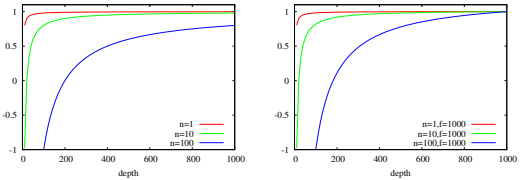
$-z$ is forward / Dopředu je $-z$

Z-buffer and inifinity

$$P(z) = \frac{z - 2n}{z} = 1 - \frac{2n}{z}$$

$-z$ is forward / Dopředu je $-z$

$$P(n) = -1 \qquad P(\infty) = 1$$
$$P(2n) = 0 \qquad P(4n) = 1/2$$



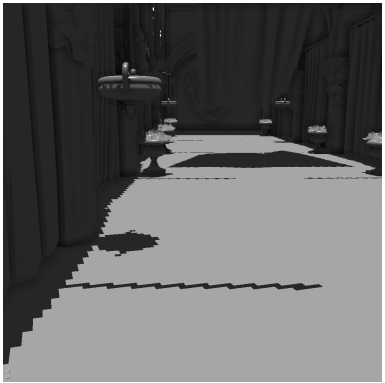
:) Most of the values of z-buffer are in "reasonable" distance.

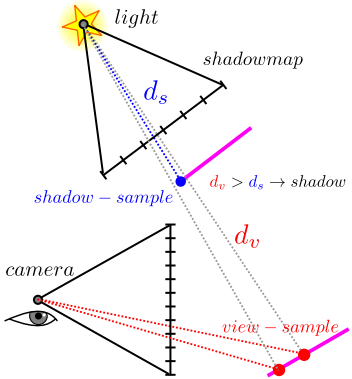
:) Většina hodnot z-bufferu je v "rozumné" vzdálenosti.

Shadow Mapping

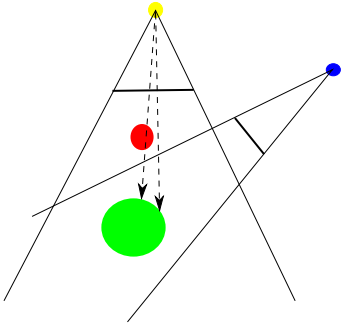
Shadow Mapping

- Two render passes.
- Two types of samples – (view-samples, shadow-samples)
- Every view-sample has corresponding shadow-sample.
- Dva průchody
- Dva druhy vzorků – (view-samples, shadow-samples)
- Každý view-sample má přiřazený – shadow-sample



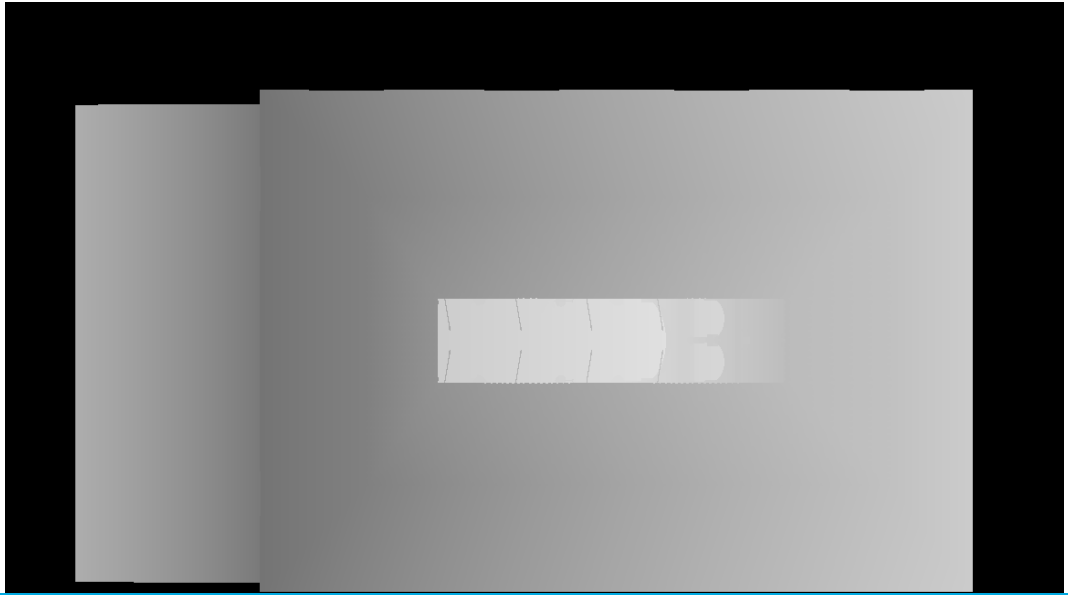


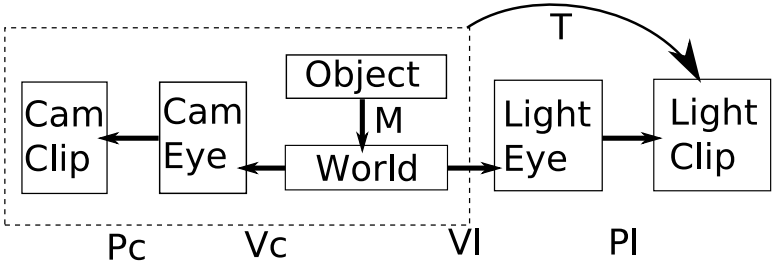
Shadow map texture / Stínové mapy



- We need to prepare z-buffer (shadow map) from the light source point of view.
- We need to compare the distance from view-sample to the light source with the depth stored in shadow map.
- Připravíme si z-buffer kamery ve světle.
- Porovnáváme hloubky vykreslovaných fragmentù.

Shadow map / Hloubková mapa



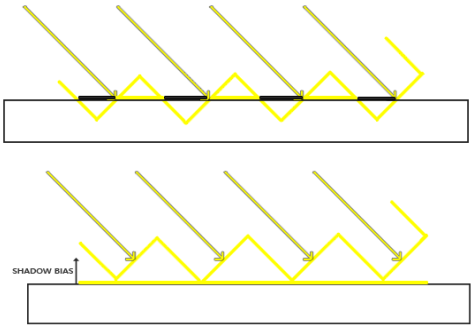


$$T = \begin{pmatrix} 0.5 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot P_l \cdot V_l \cdot M$$

```
//Vertex shader
in vec4 position;
out vec4 shadowPos;
uniform mat4 shadowMat;
void main() {
    //...
    shadowPos = shadowMat*position;
    //...
}
//Fragment shader
in vec4 shadowPos;
uniform sampler2D shadow;
void main() {
    vec3 sp = shadowPos/shadowPos.w;
    if(texture(shadow, sp.xy).x < sp.z)
        gl_FragColor = shadowed_color;
    else
        gl_FragColor = unshadowed_color;
}
```

Shadow Acne

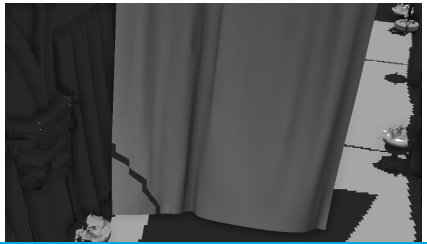
Bias

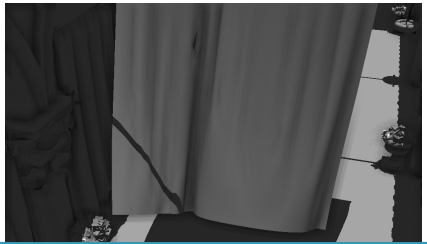


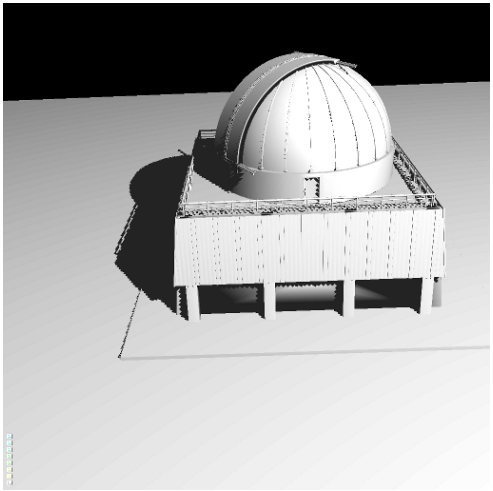

```
float depth = texture(shadowTexture, shadow_pos.xy).x;
float bias = max(0.05 * (1.0 - dot(N, L)), 0.005);
float shadow = shadow_pos.z - bias > depth ? 1.0 : 0.0;
```

Shadow-Mapping - Problems

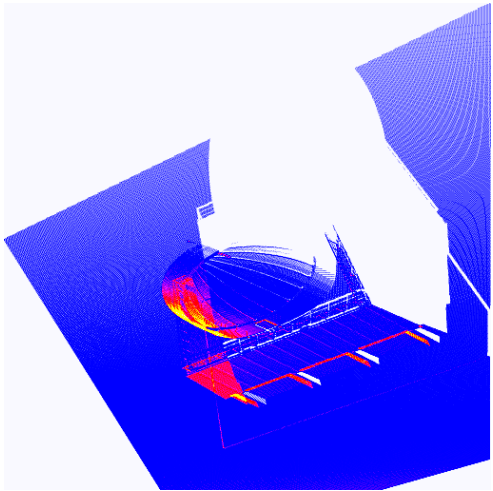
- Many shadow-samples are over utilized.
- Many shadow-samples are under utilized.
- $\rightarrow$ Jagged edges.
- There is a lot of method for mitigation of visual artifacts based on space partitioning or warping.
- Mnoho shadow-samples je nadužíváno.
- Mnoho shadow-samples není vůbec využito.
- $\rightarrow$ Zubaté hrany.
- Pro odstranění artefaktů je mnoho metod založeno na dělení prostoru nebo warpingu prostoru.







View-Samples



View-Samples projected to shadow map

Soft shadows / Měkké stíny

```
float shadow = 0.0;
vec2 texelSize = 1.0 / textureSize(shadowTexture, 0);
for(int x = -1; x <= 1; ++x)
{
    for(int y = -1; y <= 1; ++y)
    {
        float depth = texture(shadowTexture,
          shadow_pos.xy + vec2(x, y) * texelSize).r;
        shadow += shadow_pos.z - bias > depth ? 1.0 : 0.0;
    }
}
shadow /= 9.0;
```

Properties / Souhrn

\+

- Fast / Rychlé
- Simple / Jednoduché
- No new geometry / Žádná nová geometrie
- Supports "soft"shadows / Umí měkké stíny

\−

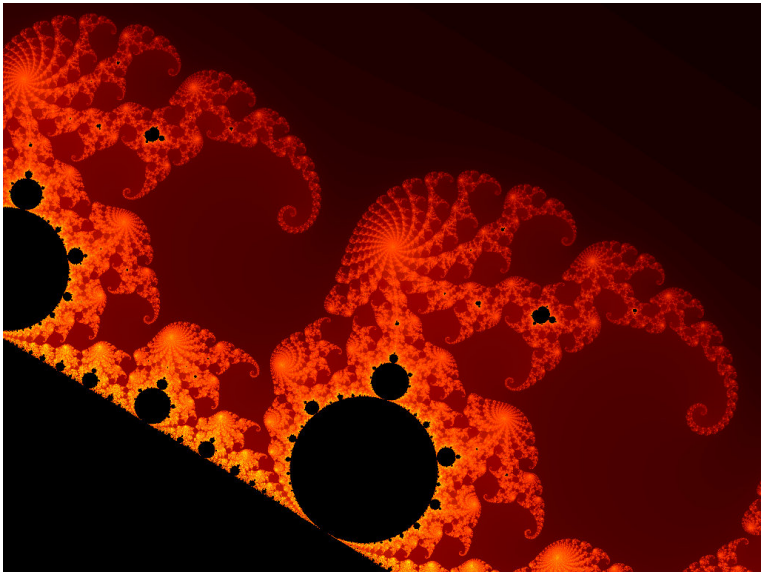
- Omnidirectional light sources / Všesměrová světla
- Shadow map resolution / Omezené rozlišení textur
- Alias

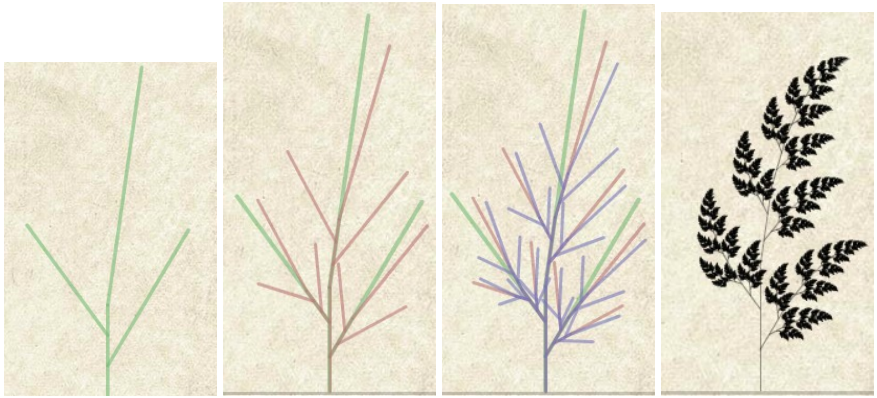
Procedural generation / Procedurální generování

Procedurální generování

- We do not have to store data of objects (vertices, textures, ...).
- We only stores templates and algorithms how to generate objects.
- Algorithm memory footprint is only few kB.
- Parameters can be stored in text files.
- Even template can be generated.
- Small memory footprint versus time to generate object.

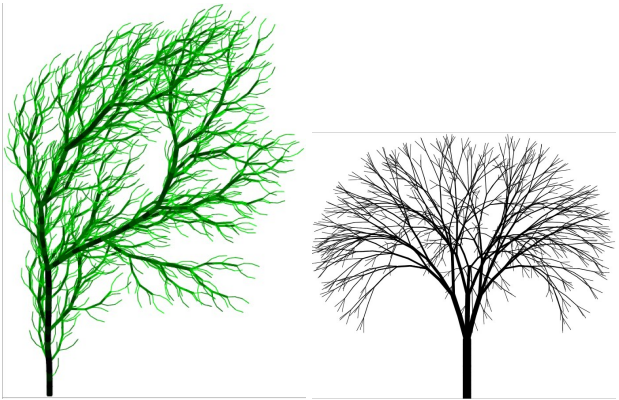
- Neukládáme data grafických objektů.
- Neukládáme vrcholy, pixely, barvy, ...
- Ukládáme šablony a způsob vygenerování grafického objektu.
- Uložení algoritmu pro vygenerování zabírá pár kB.
- Uložení v textovém souboru - parametry.
- Můžeme generovat i šablony.
- Málo místa pro uložení vs doba generování.





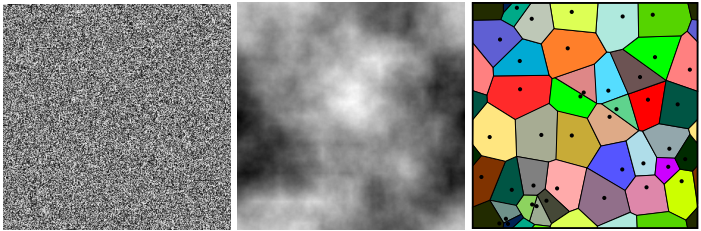
http:
//www.dangries.com/Flash/FractalMakerExp/FractalMaker_exp.html

L-systems / L-systém

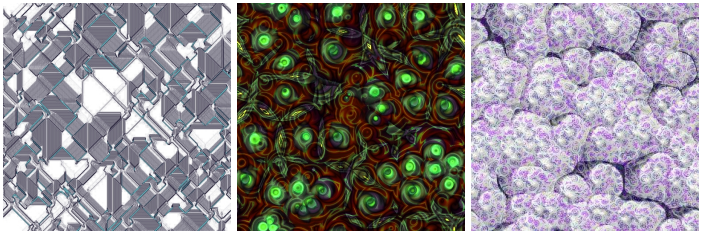


http://malsys.cz/

- Perlin/Simple noise
- Voronoi diagram / Voroného diagramy



Noise based textures / Textury založené na šumech



- Texture is computed in shader on demand.
- Small memory footprint.
- Infinite resolution.
- Suitable for demos.
- Noise.
- Výpočet hodnoty v shaderu.
- Málo paměti.
- Neomezené rozlišení.
- Vhodné pro dema.
- Šum

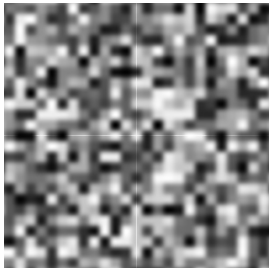
We need noise functions / Chceme šumové funkce

- float noise(float p)
- float noise(vec2 p)
- ...

- Pseudorandom / Pseudonáhodné
- Input is in range: / Výstup v: $[0, 1]$ / $[-1, 1]$
- Deterministic / Deterministické
- Limited frequency spectrum / Omezené frekvenční spektrum

Value noise / Hodnotový šum

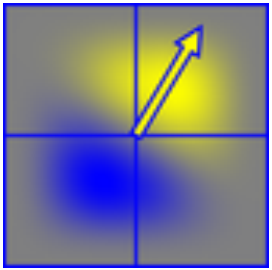
- Randomly generated textures / Náhodně vygenerovaná textura.
- The repetition pattern is visible / Rychle se opakuje.
- Interpolation and tiling can be performed by OpenGL. / Interpolaci a opakování zařídí OpenGL.

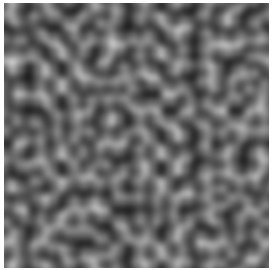


```
float noise(vec2 p)
{
  return 2*texture(noiseTexture, p).x - 1;
}
```

Gradient noise / Gradientní šum

- Texture stores gradients. / V textuře jsou **gradienty**.
- We need to compute directional vector from texels to samples. / Spočítáme si směrové vektory od **texelů** ke **vzorku**.
- Scalar multiplication od directions and gradient computes the resoluting noise. / Skalární součin směrů s gradienty dává výsledný šum.



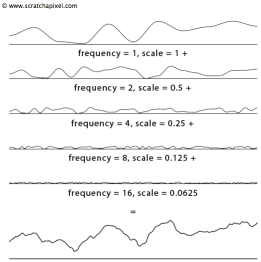


Perlin K.; Implementing Improved Perlin Noise, GPU Gems

Green S.; Implementing Improved Perlin Noise, GPU Gems 2

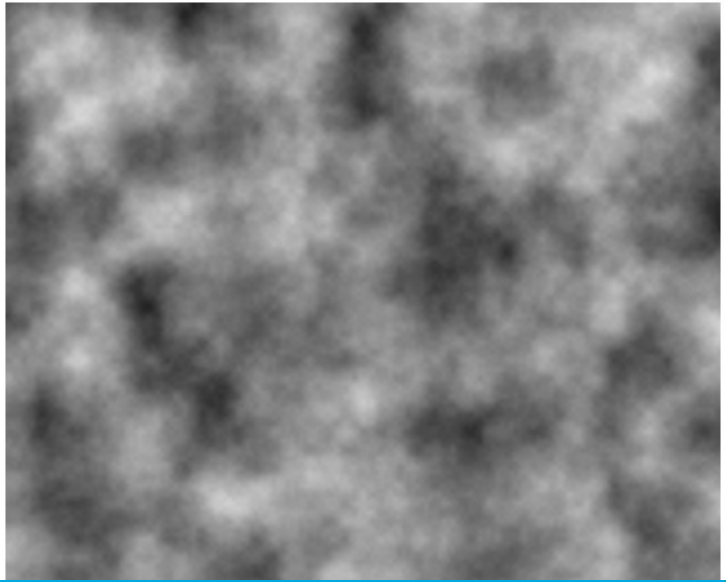
Fractal noise / Fraktální šum

- Add layers with different frequencies and amblitudes. / Sčítáme vrstvy o různé frekvenci a amplitudě
- Octaves / Oktávy



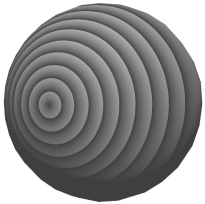

```
float fnoise(vec2 p)
{
  return noise(p)
       + 0.5*noise(2*p)
       + 0.25*noise(4*p);
}
```

Result / Výsledek



Simple example / Jednoduchý priklad

Annual ring / Letokruhy



```
float d = length(model_pos.xy);
float t = mod(d*10, 1);
```

Coloring of annual rings / Letokruhy obarvíme

```
vec3 brown = vec3(140, 90, 27)/255;
vec3 yellow = vec3(188, 140, 42)/255;

float t = smoothstep(0.2, 0.4, mod(length(model_pos.xy)*10, 1));
vec3 color = (1-t)*brown + t*yellow;
```

Add turbulency with noise / Přidáme šum

```
vec2 noisy_pos = model_pos.xy
    + 0.1*fnoise(model_pos.xy);

float t = smoothstep(0.2, 0.4, mod(length(noisy_pos)*10, 1));
vec3 color = (1-t)*brown + t*yellow;
```

Thank you for your attention! Questions?