

PGR - Stíny, procedurální generování

Tomáš Milet

Brno University of Technology, Faculty of Information Technology

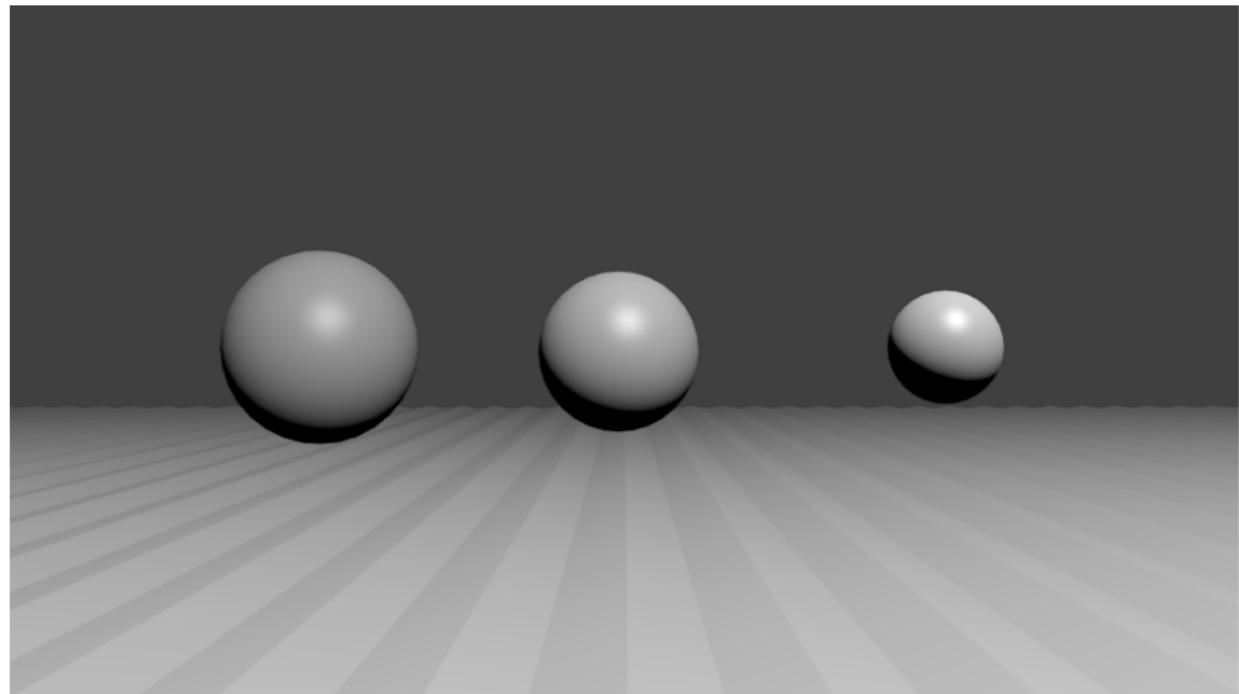
Božetěchova 1/2. 612 66 Brno - Královo Pole

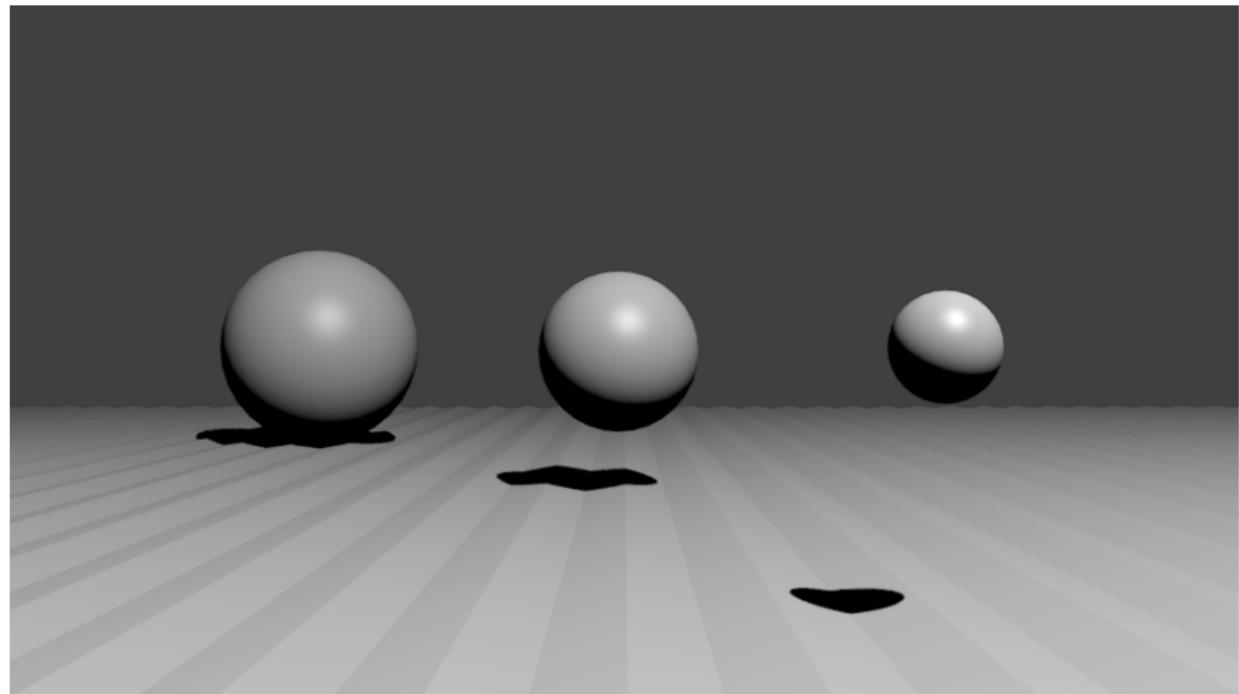
imilet@fit.vutbr.cz



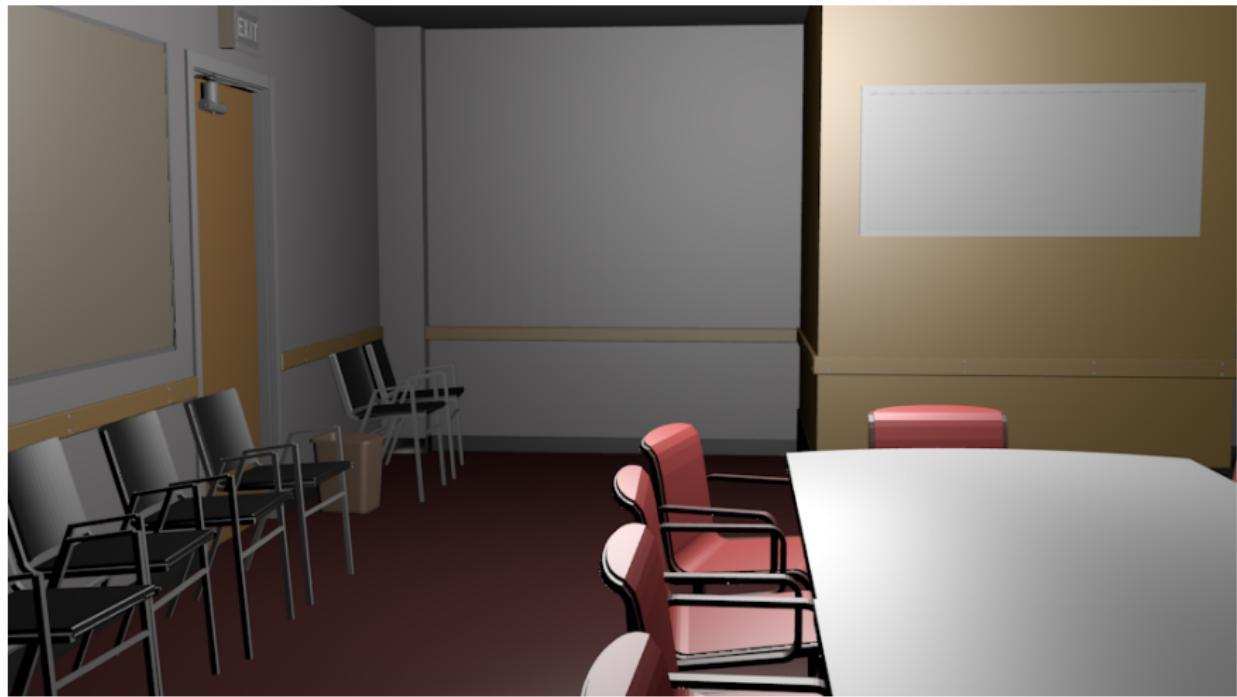
10. března 2020

Stíny



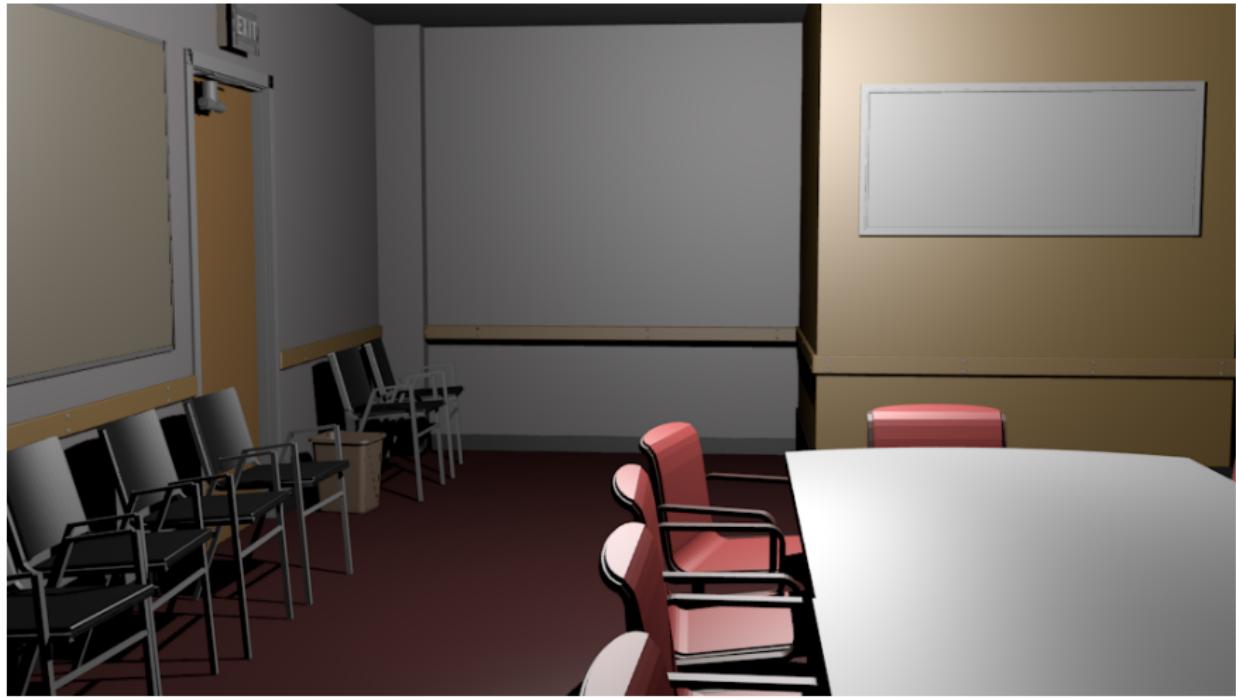


Proč potřebujeme stíny?

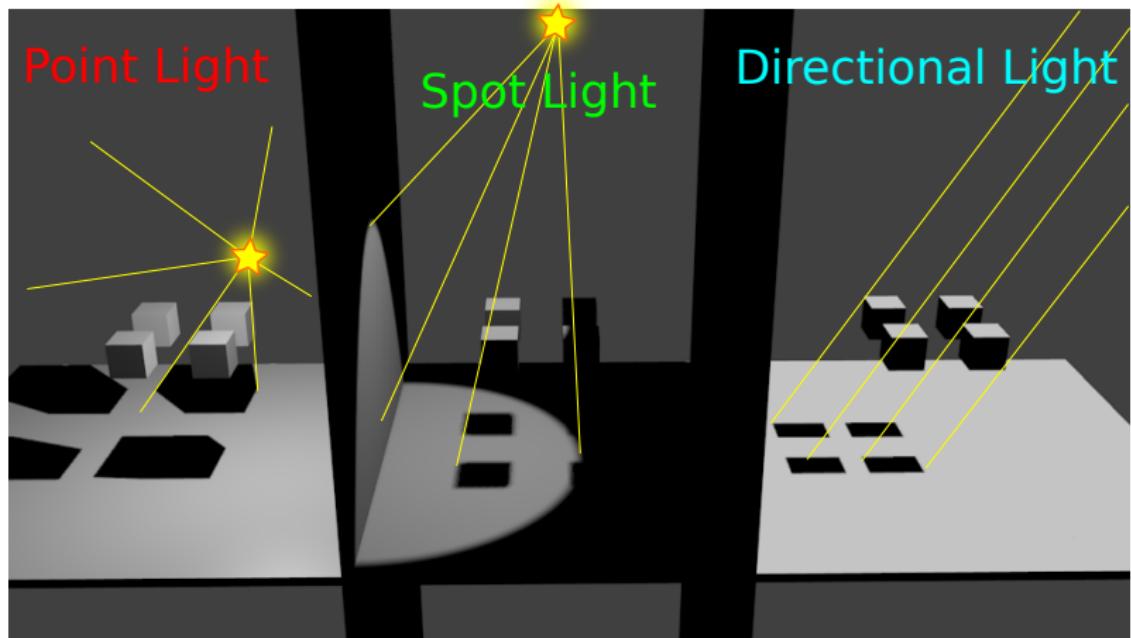


Proč potřebujeme stíny?

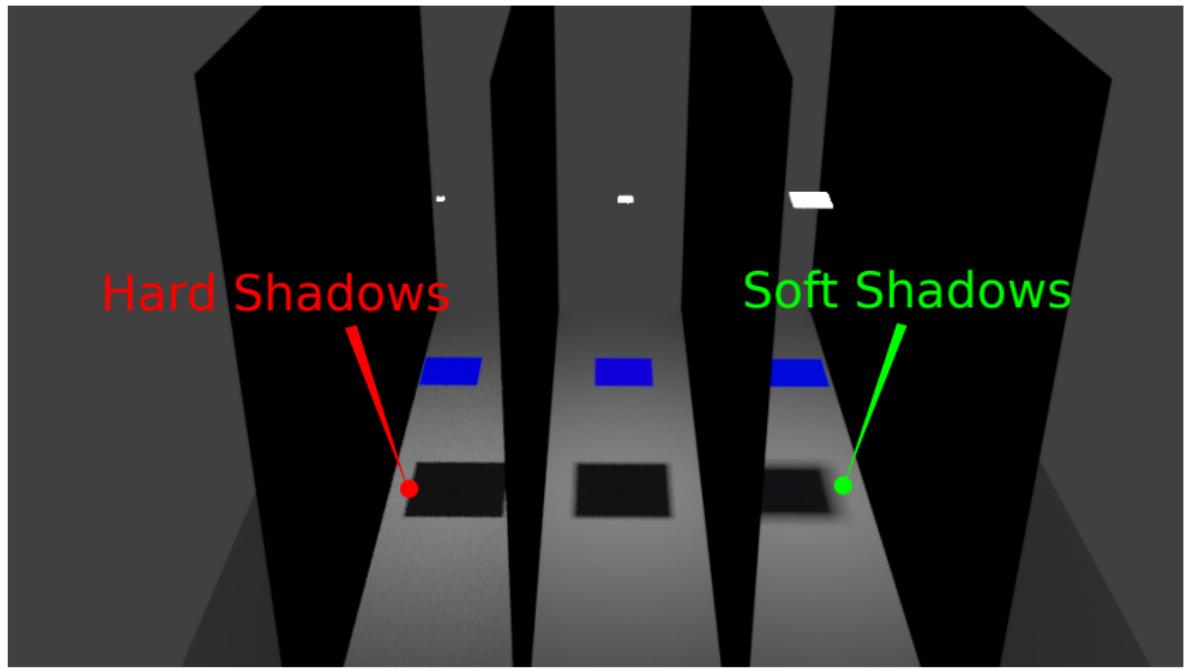
- Stíny pomáhají pochopit 3D scénu.
- Vzájemné polohy mezi objekty.



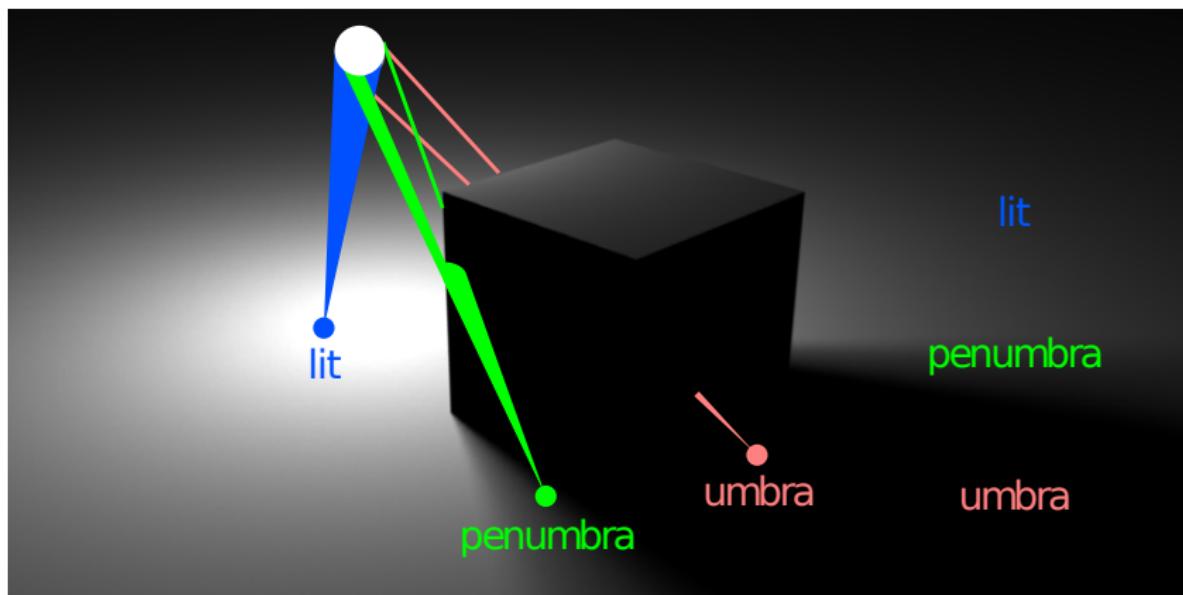
- Omnidirectional point light source.
- Spot light source.
- Directional light source.



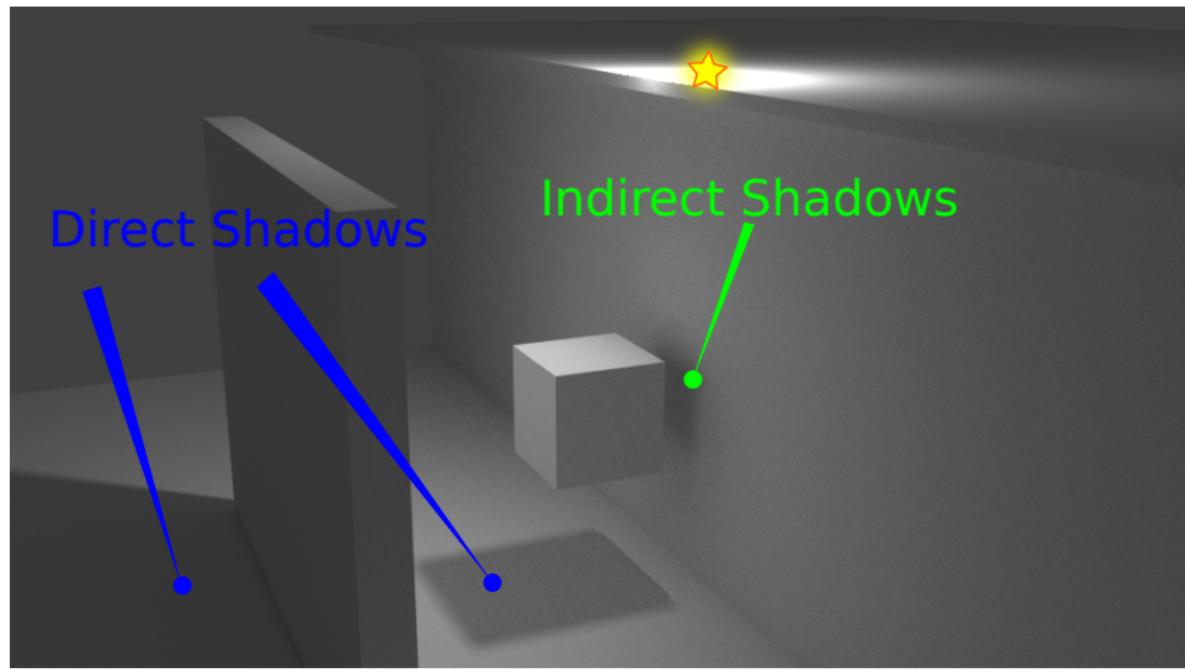
- Tvrde stíný vznikají z nekonečně malých světelných zdrojů (časté v počítačové grafice).
- Měkké stíny vznikají z plošných zdrojů světla.



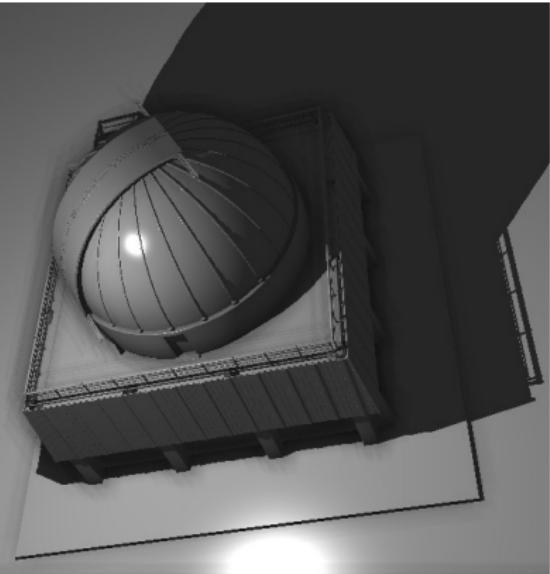
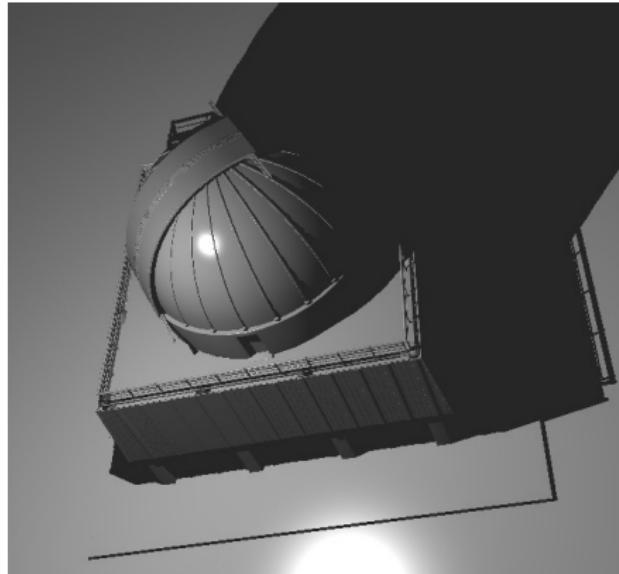
- Plně osvětlené regiony scény vidí na všechny plošky světla.
- Plně zastíněné regiony (umbra) nevidí na žádnou plošku světla.
- Polostín (penumbra) vzniká v regionech scény, které vidí jen na některé plošky švětla.



- Stíny vznikají z přímého osvětlení (můžou být tvrdé, pokud je světlo malé).
- Stíny vznikají z nepřímého osvětlení (měkké stíny).



- Ambient occlusion - metoda pro aproximaci části globálního osvětlení a simulaci stínů nepřímého osvětlení.





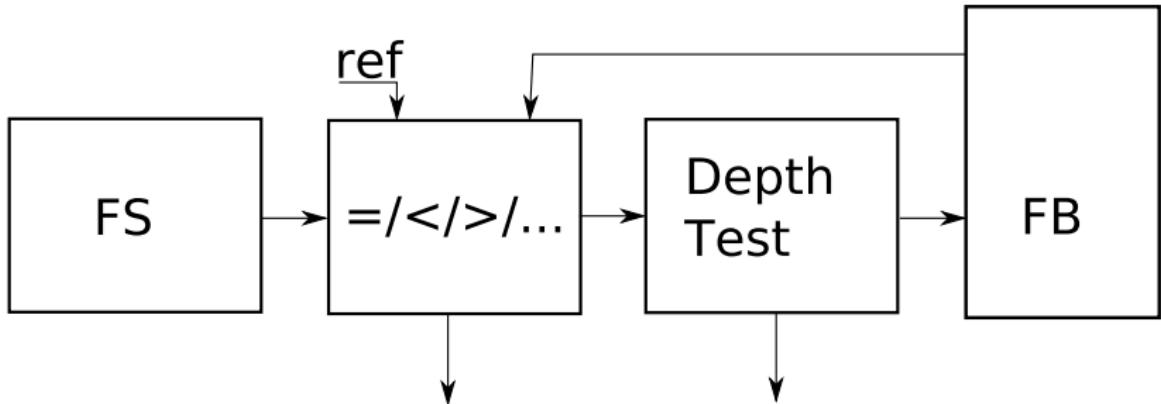




Shadow volumes

- Další část framebuferu
- Celočíselný, obvykle 8bpp

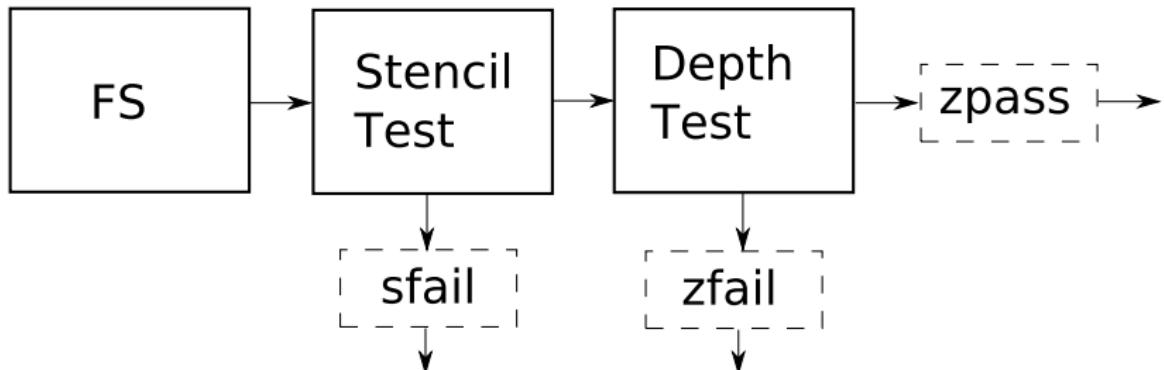
```
SDL_GL_SetStencilSize(8);  
glClear(GL_STENCIL_BUFFER_BIT);  
glClearStencil(0);  
glStencilMaskSeparate(GL_FRONT_AND_BACK, ~0); //!!!
```



```
glEnable(GL_STENCIL_TEST);
```

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

- $\text{func} \in \{\text{GL_ALWAYS}, \text{GL_NEVER}, \text{GL_LESS}, \text{GL_GREATER}, \dots\}$
- $\text{if}(\text{S} \& \text{mask} == \text{ref} \& \text{mask}) // \text{GL_EQUAL}$



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

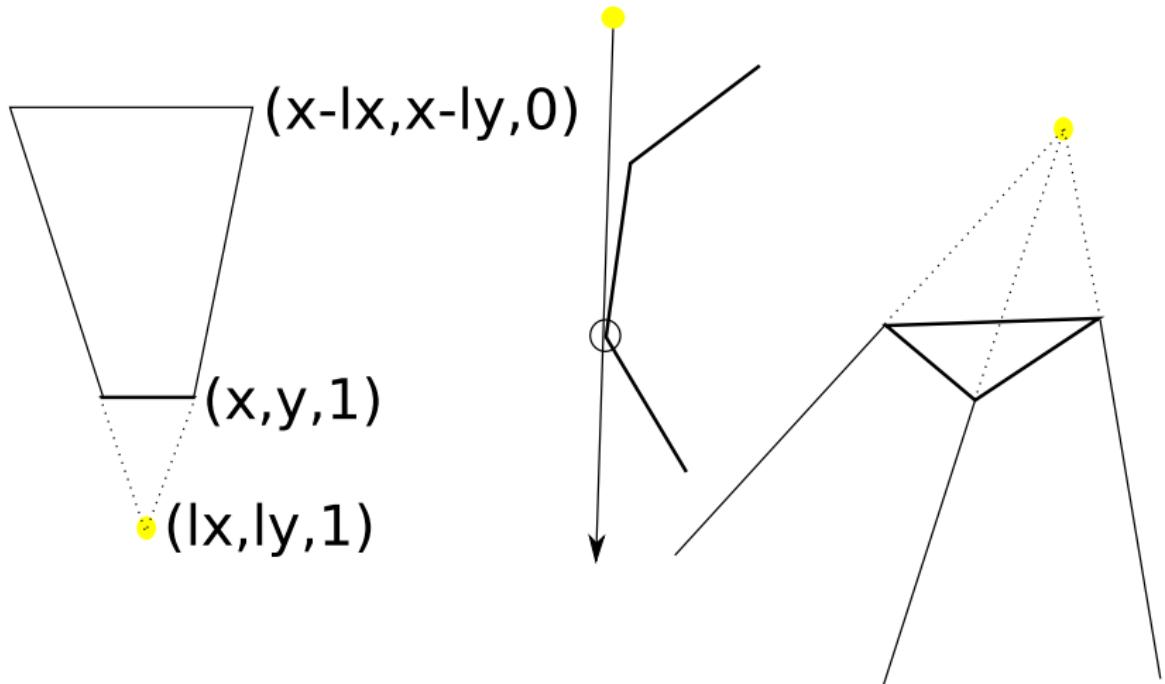
- GL_KEEP, GL_ZERO, GL_REPLACE, GL_INVERT
- GL_INCR, GL_DECR, GL_INCR_WRAP, GL_DECR_WRAP

$$255 + 1 = 255 \iff \text{GL_INCR}$$

$$255 + 1 = 0 \iff \text{GL_INCR_WRAP}$$

- 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.

1. kreslící průchod - ambientní světlo
 - 1 Vykresli scénu s ambientním osvětlením.
2. kreslící průchod - tvorba stencilové masky
 - 1 Vypni kreslení barvy a modifikaci depth bufferu.
 - 2 Nastav stencilovou operaci na incrementaci u přívrácených trojúhelníku a dekrementaci při odvrácených trojúhelníků.
 - 3 Povol zápis do stencil bufferu.
 - 4 Vykresli stínovou geometrii - vytvoří se stencilová maska.
 - 5 Vypni zápis do stencilového bufferu.
 - 6 Povol kreslení barvy a modifikaci depth bufferu.
3. kreslící průchod - difuzní, spekulární světlo
 - 1 Zapni stencil test - při stencilové hodnotě 0 test projde.
 - 2 Vykresli scénu s difuzním a spekuláním osvětlením.
 - 3 Vypni stencil test.

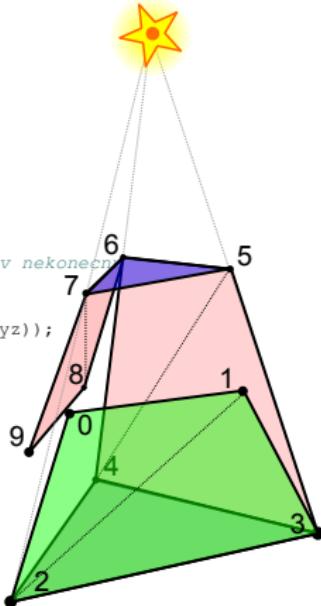


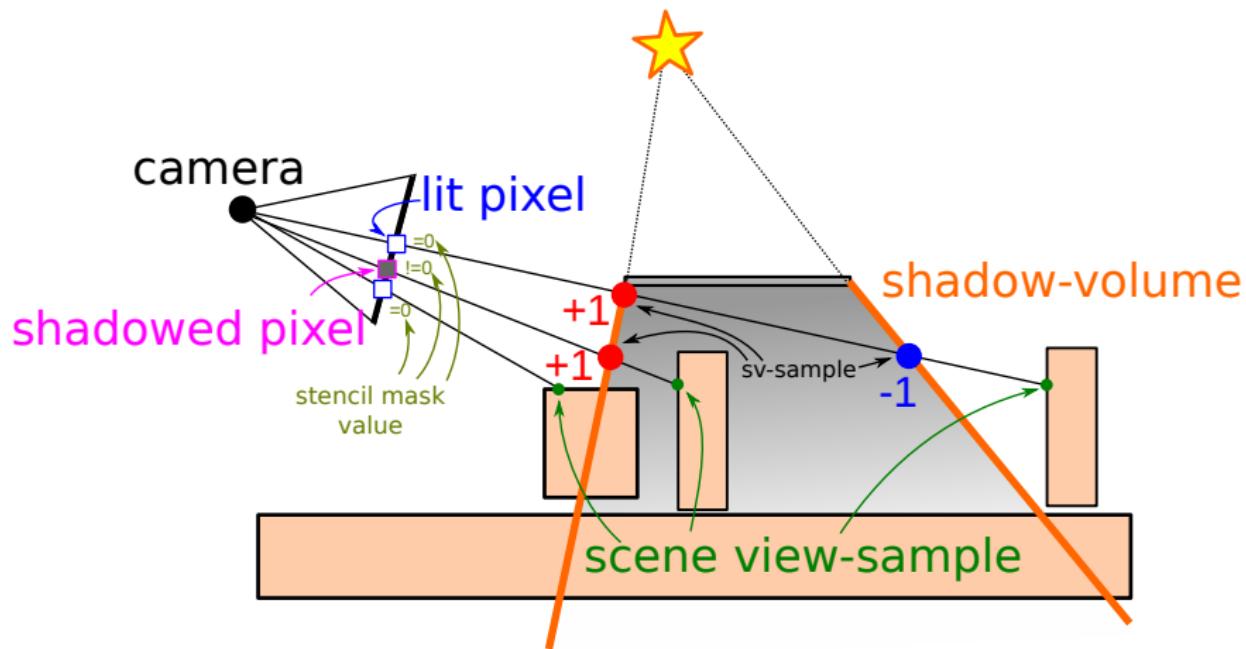
- Siluetu protáhnout do nekonečna
- Ideální bod $(x, y, z, 0)$

```

#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 nekonecne
    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 vnitrikem 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();
}

```





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

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

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

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

```
//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_WRA);
glStencilOpSeparate(GL_BACK, GL_KEEP, GL_KEEP, GL_DECR_WRA);

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

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

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

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

glStencilFunc(GL_EQUAL, 0, 0xff);

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

glDisable(GL_STENCIL_TEST);
```

N světel znamená 2^N kombinací zastíněností :(Co s tím?

N světel znamená 2^N kombinací zastíněností :(Co s tím?

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

N světel znamená 2^N kombinací zastíněností :(Co s tím?

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

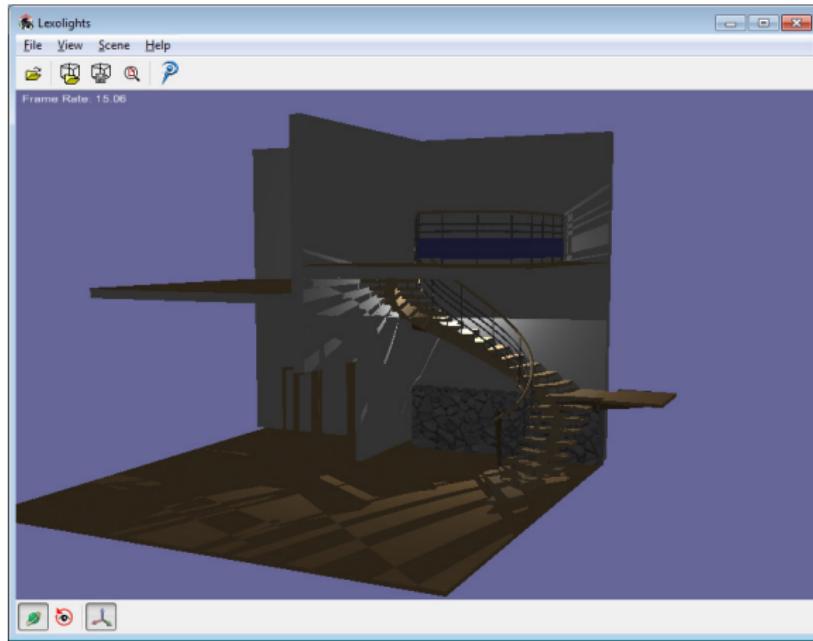
- Vykreslit scénu s ambient+emission
- Pro každé světlo
 - Připravit stencil buffer
 - Přiblendovat diffuse+specular
 - (glDepthFunc(GL_LESS), glBlendFunc(GL_ONE, GL_ONE))

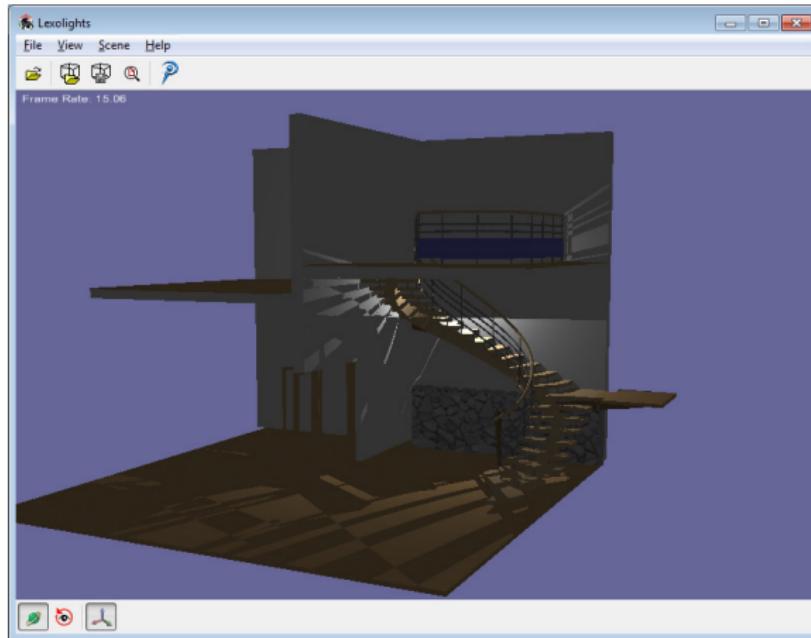
+

- Dynamické scény
- Problematické scény
- Sedí na pixel přesně

-

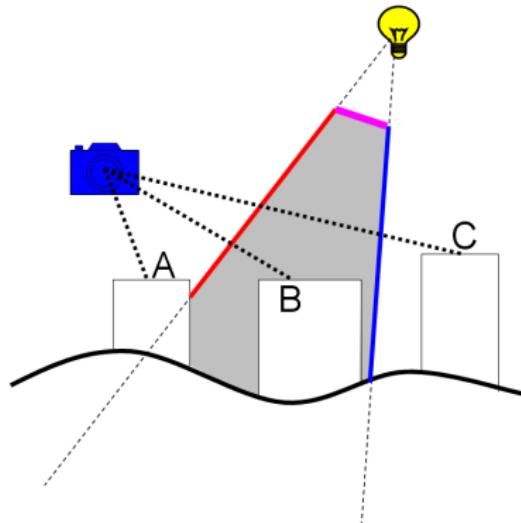
- Pomalé
- Závislé na fillrate
- Konstrukce stínových těles
- Jen ostré stíny





Stačí vlezit dovnitř stínového tělesa.

Pixel je ve stínu když paprsek vletí **dovnitř** a nevletí **ven**.



Z_PASS :

Počítám viditelné pixely

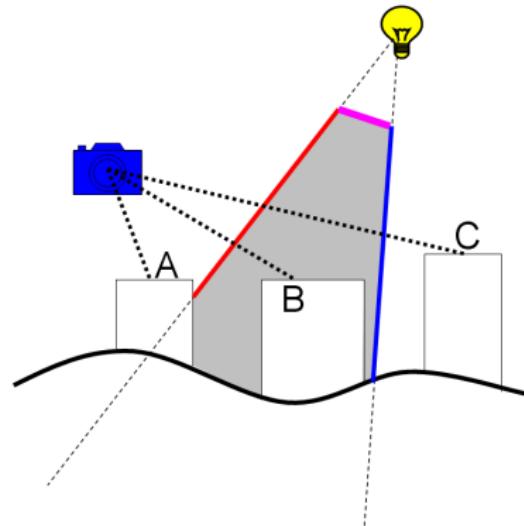
+1, -1

Z_FAIL :

Počítám neviditelné pixely **-1**,

+1

Pixel je ve stínu když paprsek vletí **dovnitř** a nevletí **ven**.



Z_PASS :

Počítám viditelné pixely

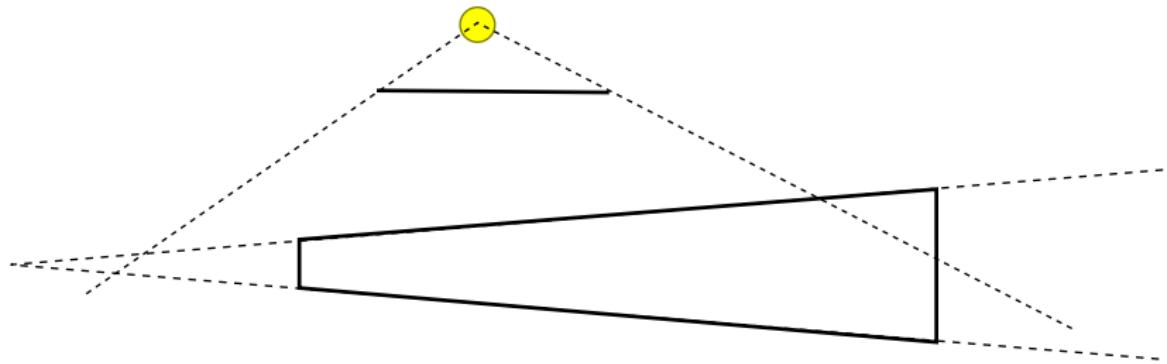
+1, -1

Z_FAIL :

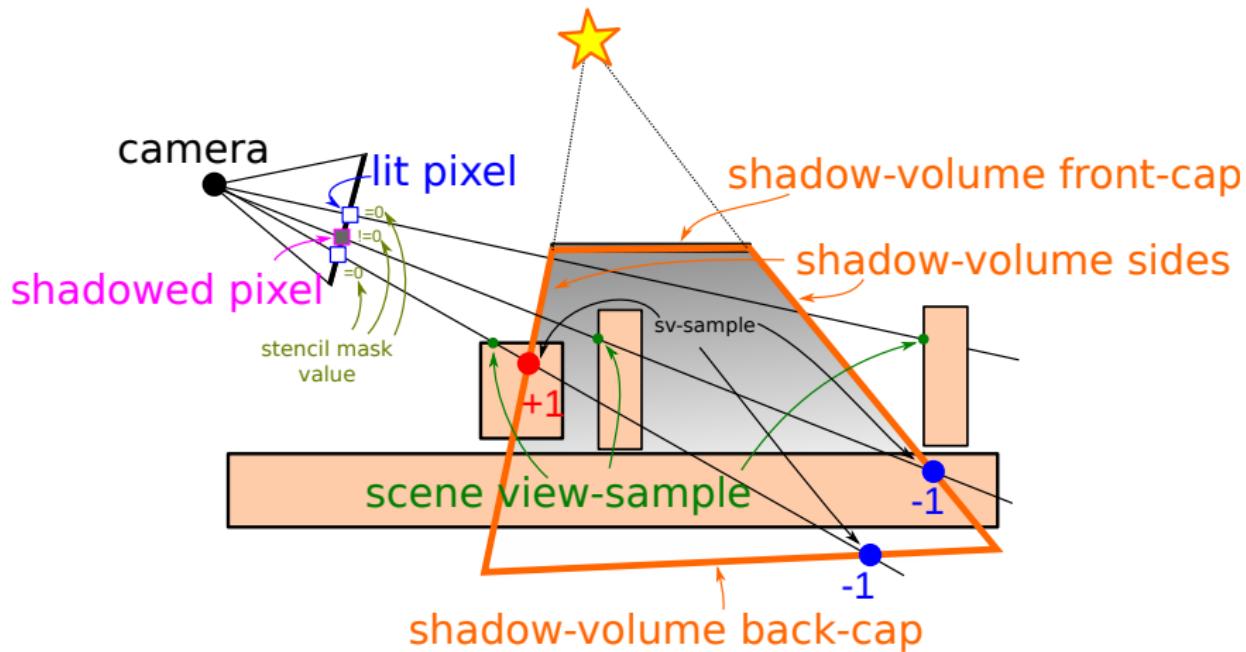
Počítám neviditelné pixely -1,

+1

```
glStencilOpSeparate(GL_FRONT, GL_KEEP, GL_DECR_WRAP,  
GL_KEEP);  
glStencilOpSeparate(GL_BACK, GL_KEEP, GL_INCR_WRAP,  
GL_KEEP);
```



- Stínové těleso musí být uzavřené.
- A jeho stěny musí být "viditelné".
- Problém je far-plane.



```
//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);
```

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

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

- :) Vidíme do nekonečna.
- ? Co z-test? Dělíme nekonečnou délku na konečný počet dílků.

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

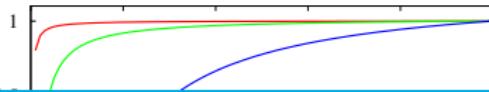
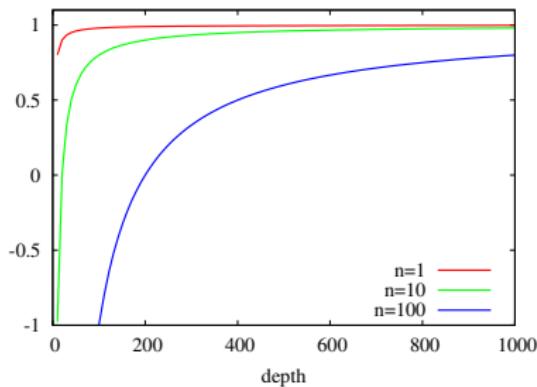
Dopředu je $-z$

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

Dopředu je $-z$

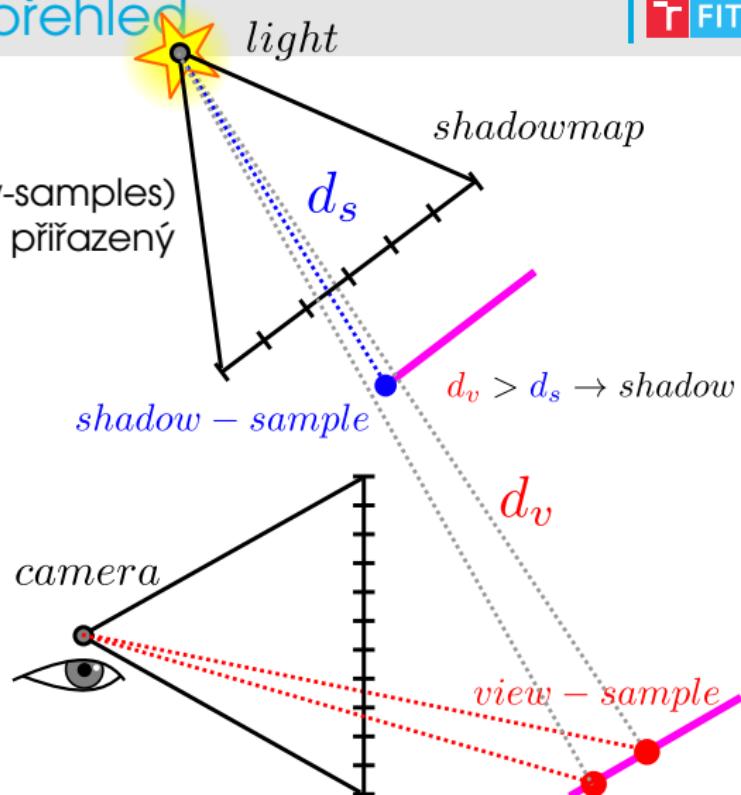
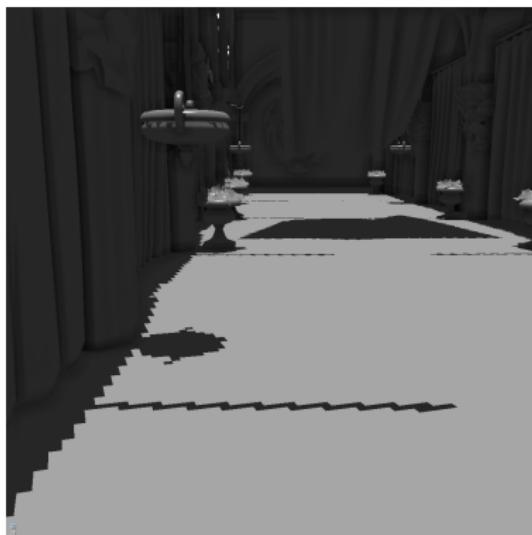
$$P(n) = -1$$
$$P(2n) = 0$$

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

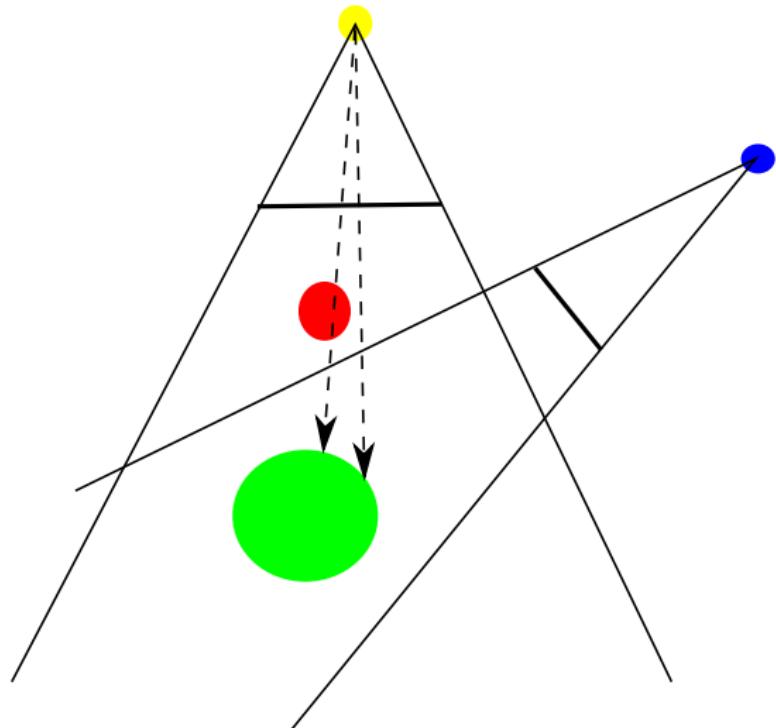


Shadow Mapping

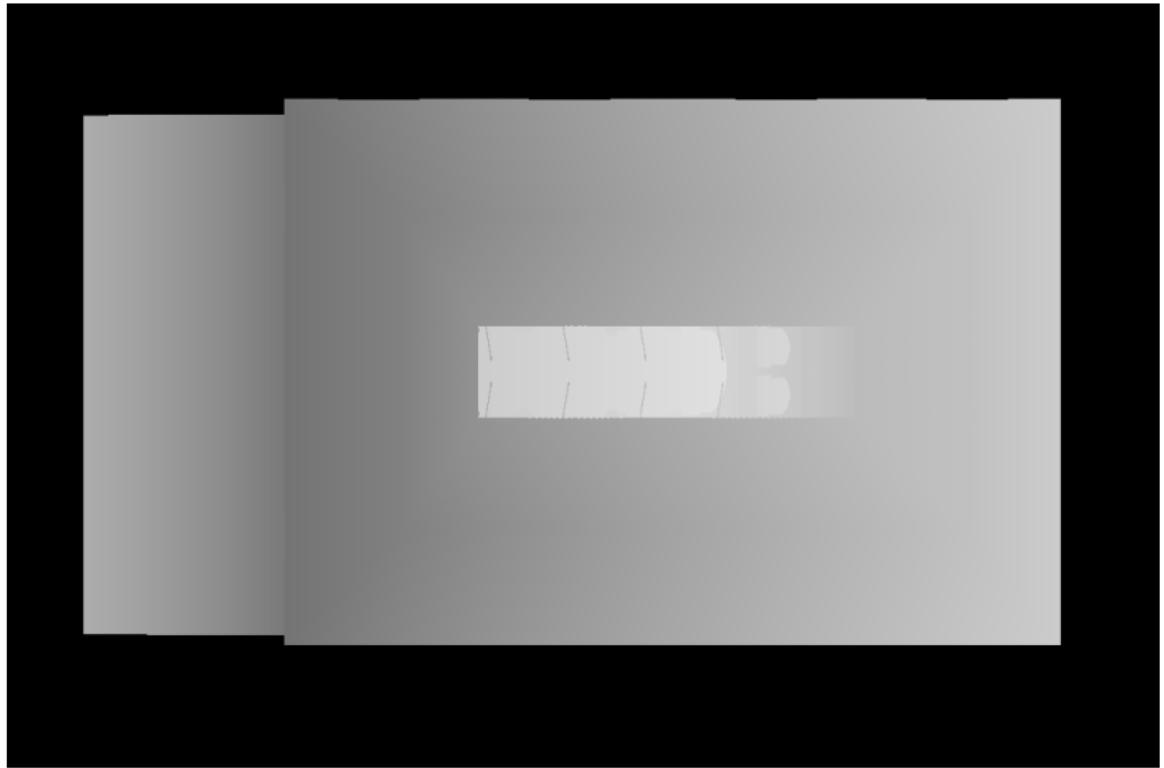
- Dva průchody
- Dva druhy vzorků (view-samples, shadow-samples)
- Každý view-sample má přiřazený shadow-sample

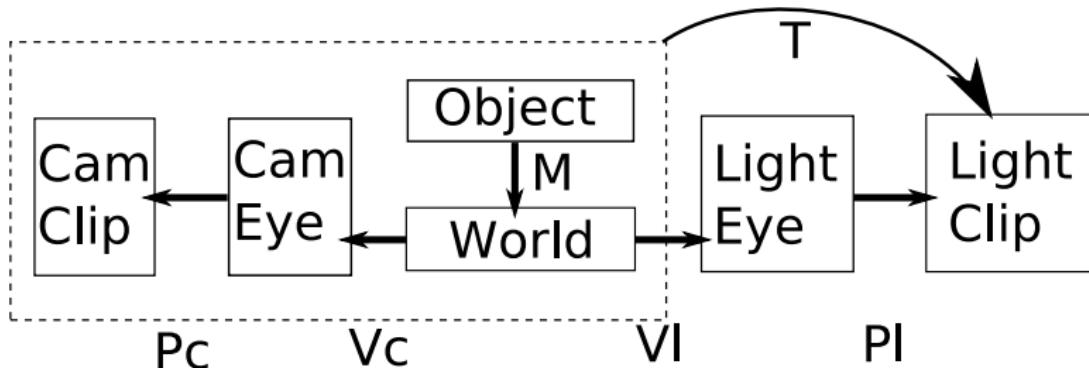






- Připravíme si z-buffer kamery ve světle.
- Porovnáváme hloubky vykreslovaných fragmentů.



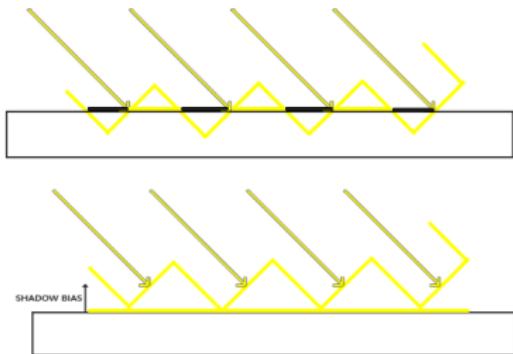


$$T = \begin{pmatrix} 0.5 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot P_l \cdot V_l \cdot M$$

```
//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;
}
```

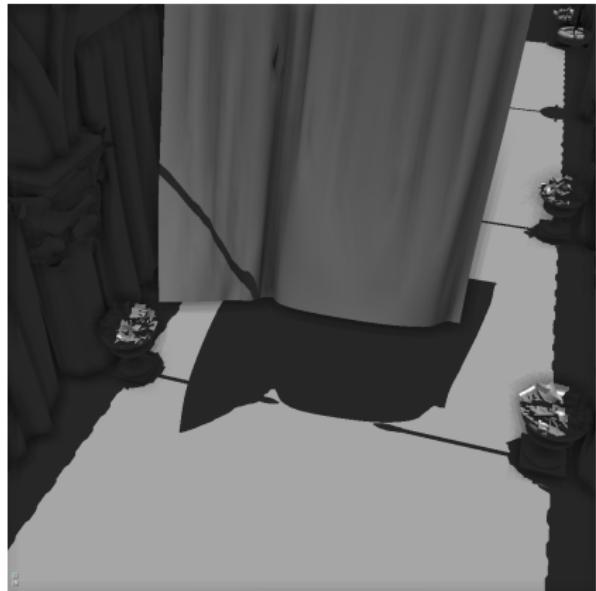
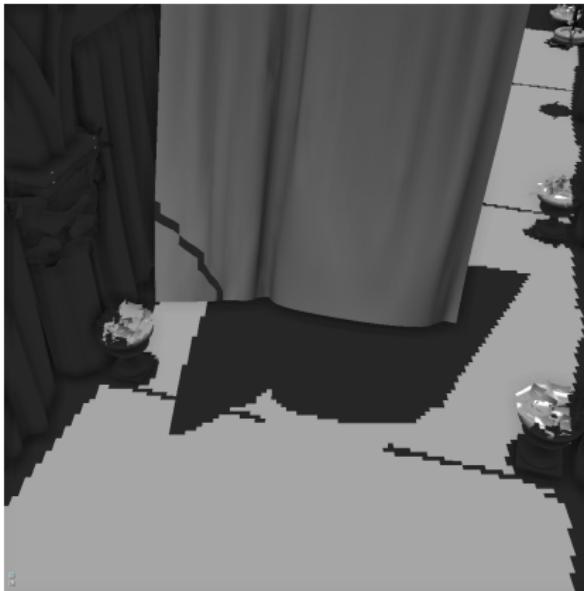


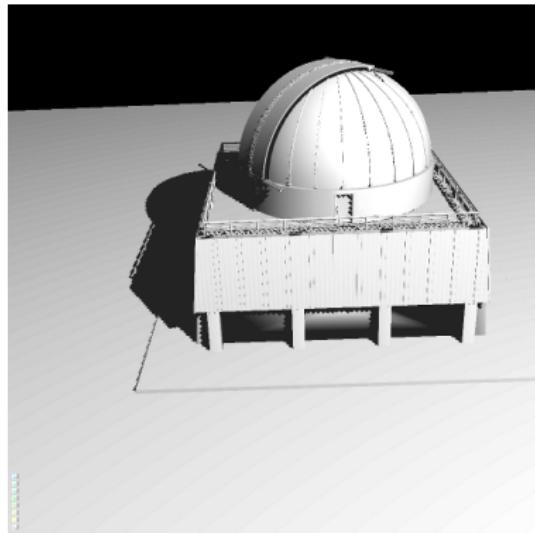


```
float depth = texture(shadowTexture, shadow_pos.xy).x;
float bias = max(0.05 * (1.0 - dot(N, L)), 0.005);
float shadow = shadow_pos.z - bias > depth ? 1.0 : 0.0;
```

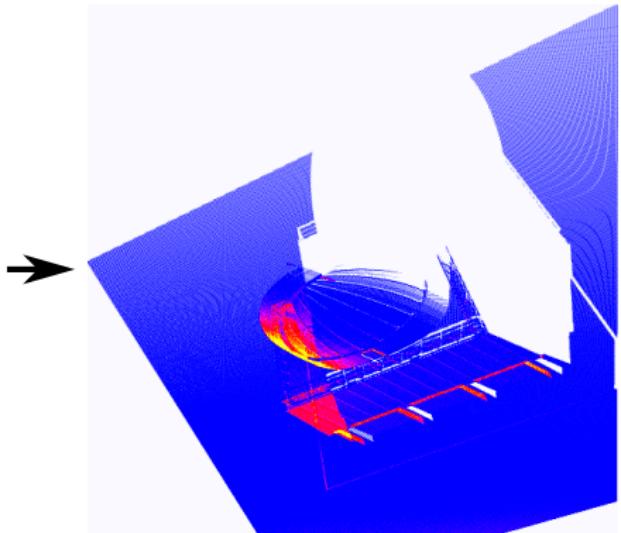


- Mnoho shadow-samples je nadužíváno.
- Mnoho shadow-samples není vůbec využito.
- → Zubaté hrany.
- Pro odstranění artefaktů je mnoho metod založeno na dělení prostoru nebo warpingu prostoru.





View-Samples



View-Samples projected to shadow map



```
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;
```

+

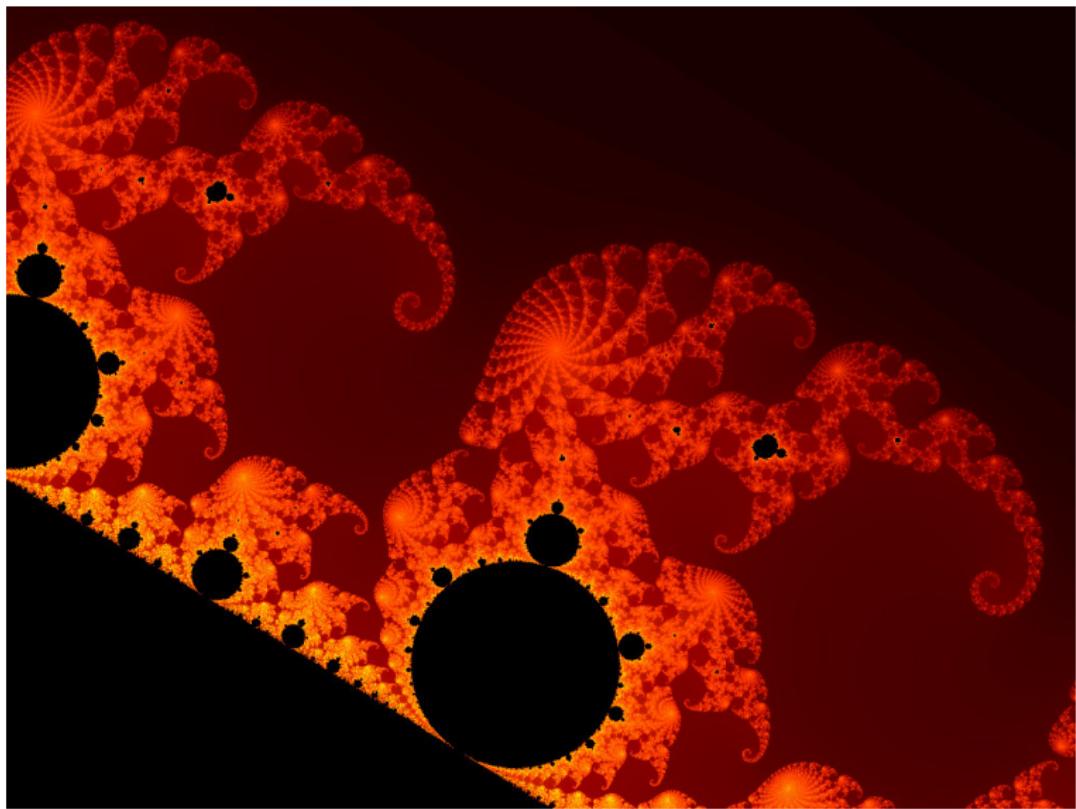
- Rychlé
- Jednoduché
- Žádná nová geometrie
- Umí měkké stíny

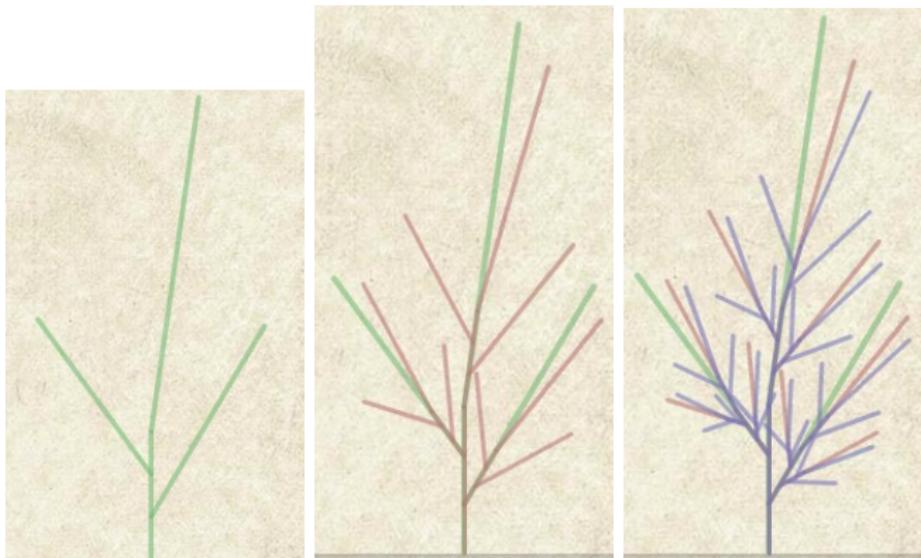
-

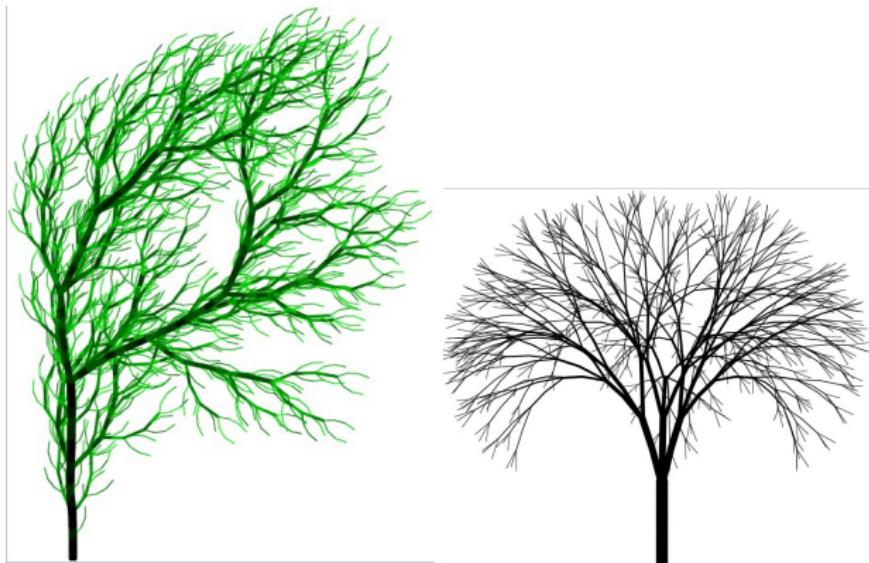
- Všesměrová světla
- Omezené rozlišení textur
- Alias

Procedurální generování

- 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.





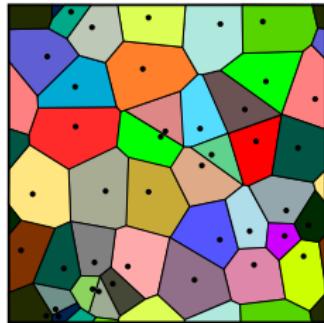
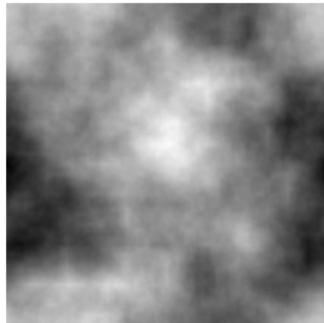
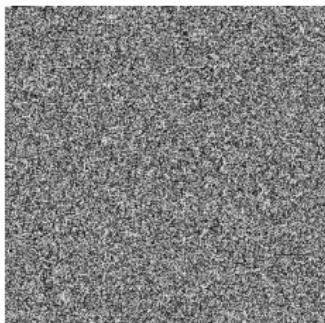


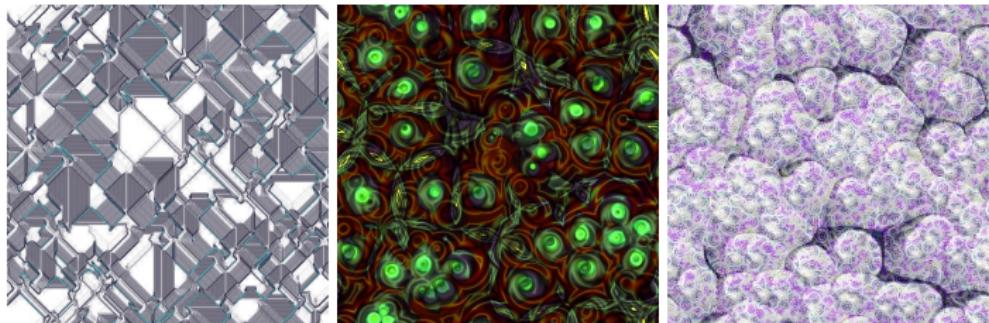
<http://malsys.cz/>

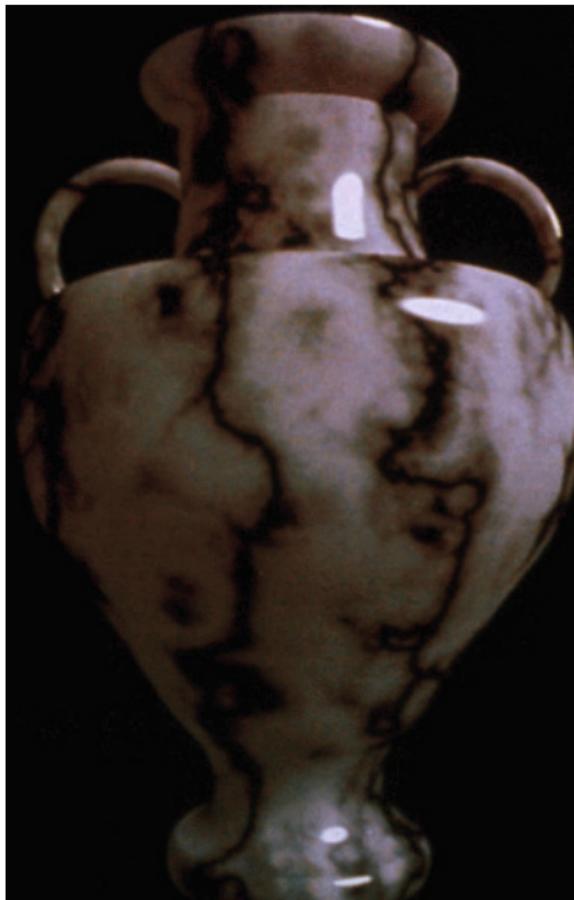


<http://procworld.blogspot.com/>

- Perlin/Simple noise
- Voroného diagramy







- Výpočet hodnoty v shaderu.
- Málo paměti.
- Neomezené rozlišení.
- Vhodné pro dema.
- Šum

Chceme šumové funkce

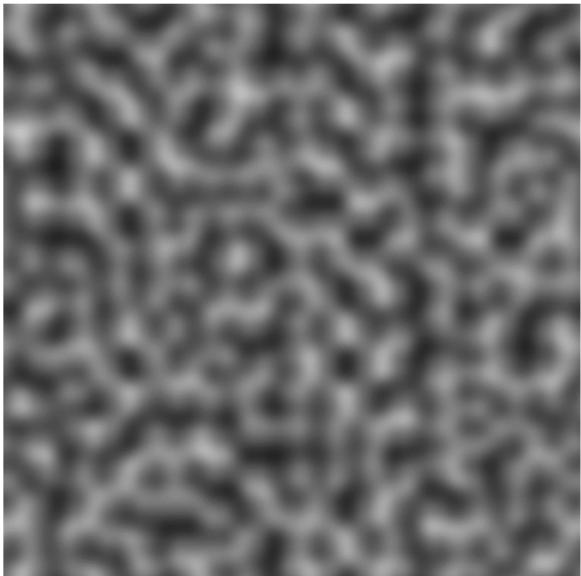
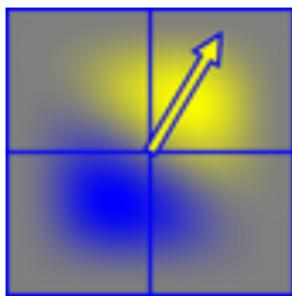
- `float noise(float p)`
 - `float noise(vec2 p)`
 - ...
-
- Pseudonáhodné
 - Výstup v $[0, 1]$ nebo $[-1, 1]$
 - Deterministické
 - Omezené frekvenční spektrum

- Náhodně vygenerovaná textura.
- Rychle se opakuje.
- Interpolaci a opakování zařídí OpenGL.



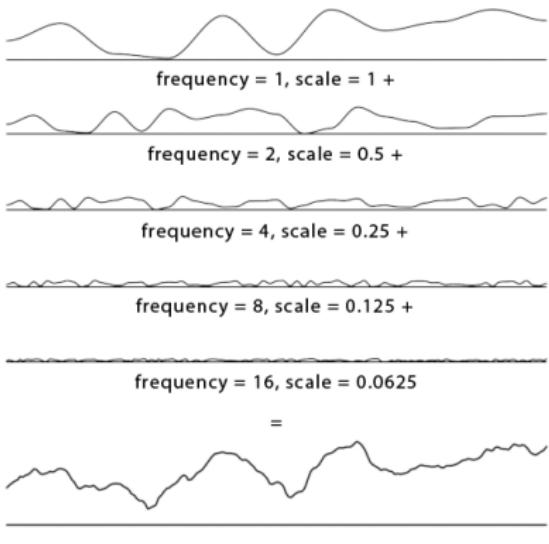
```
float noise(vec2 p)
{
    return 2*texture(noiseTexture, p).x - 1;
}
```

- V textuře jsou **gradienty**.
- Spočítáme si směrové vektory od **texelů** ke **vzorku**.
- Skalární součin směrů s gradienty dává výsledný šum.



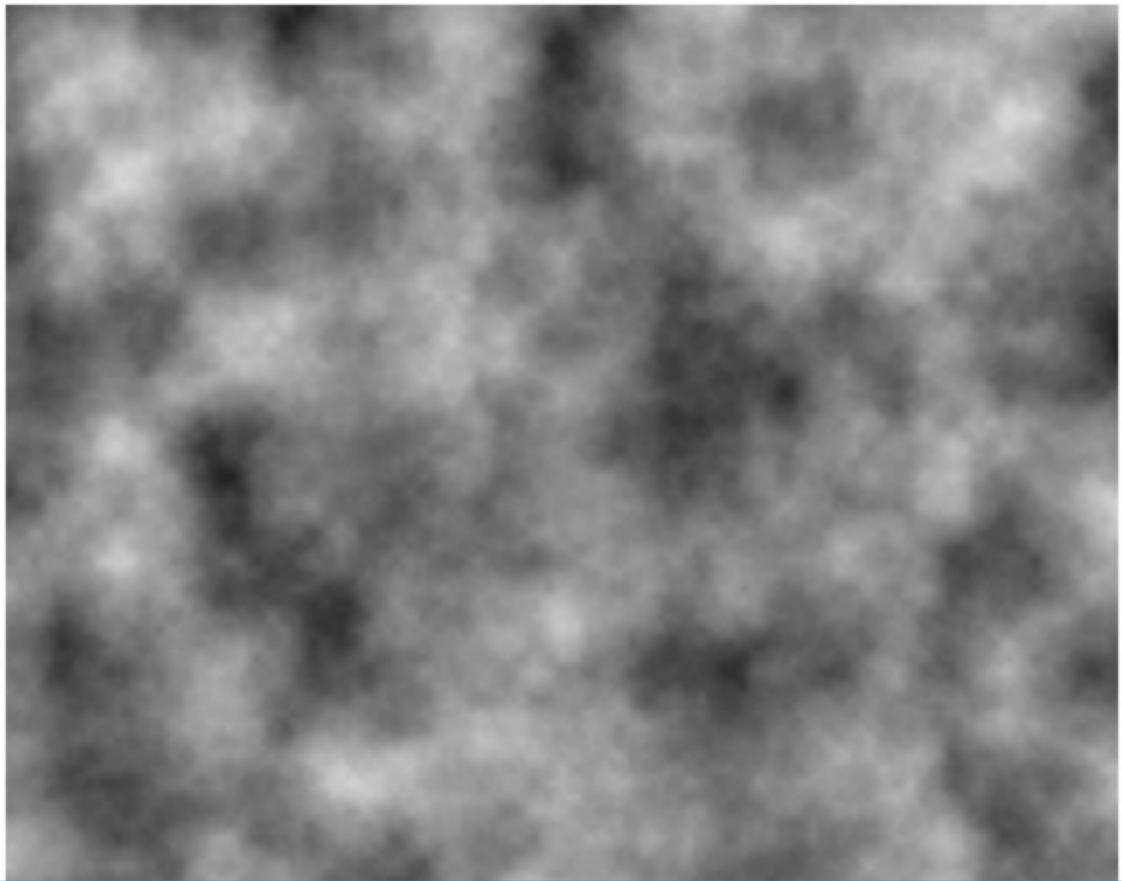
Perlin K.; Implementing Improved Perlin Noise, GPU Gems
Green S.; Implementing Improved Perlin Noise, GPU Gems 2

© www.scratchapixel.com

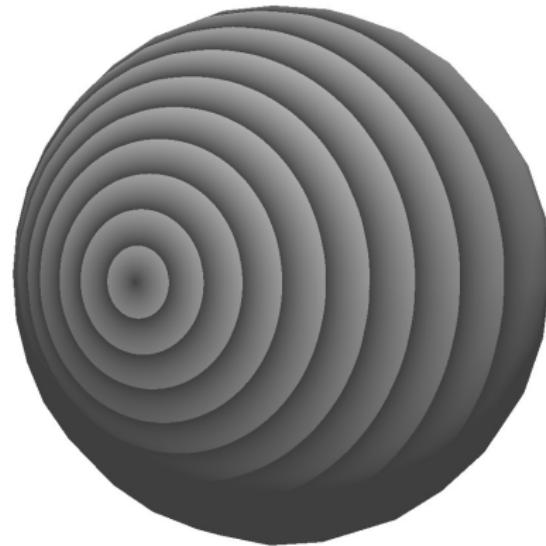


- Sčítáme vrstvy o různé frekvenci a amplitudě
- Oktávy

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







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



```
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) *
```



```
vec3 color = (1-t)*brown + t*yellow;
```



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

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

Thank you for your attention! Questions?