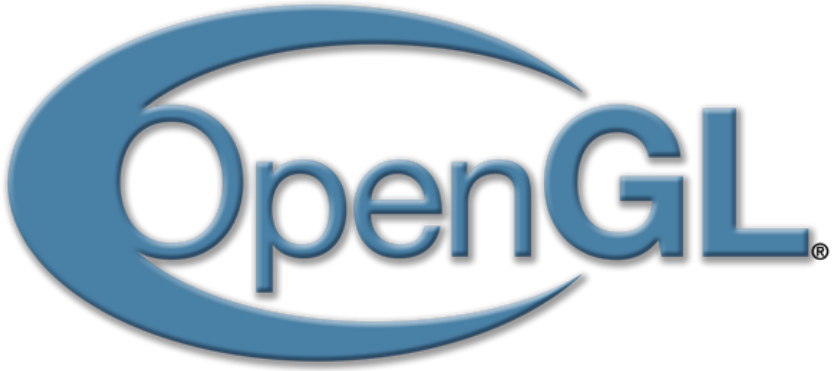
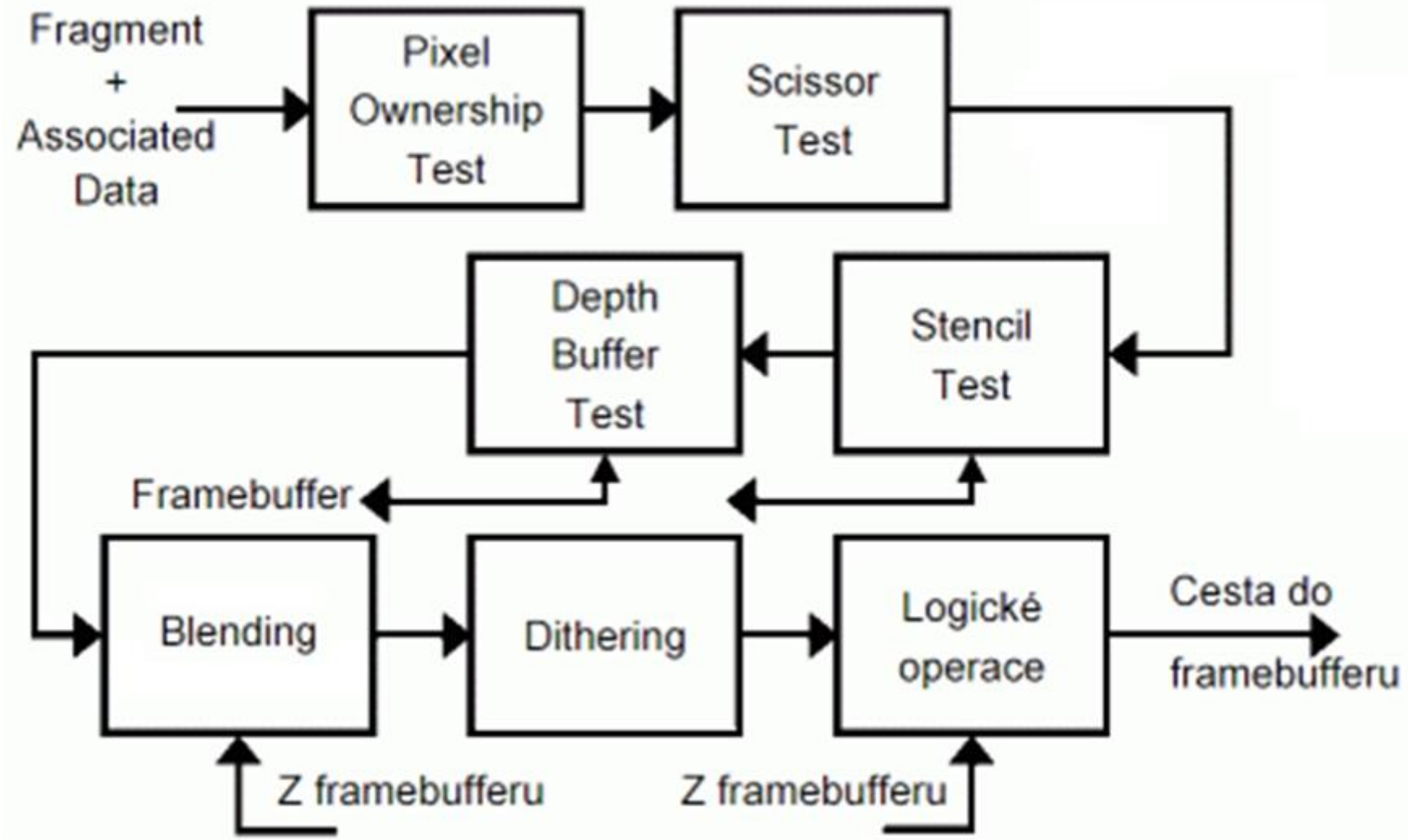


PV112 – Programování grafických aplikací

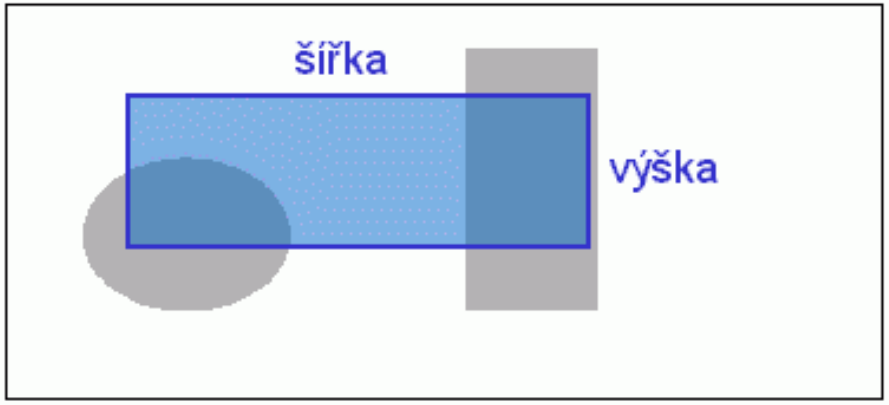
9. přednáška – Operace s fragmenty, efekty, culling

Testy a operace s fragmenty - pipeline



Operace s fragmenty při vykreslování

- **Scissor test**
  - Nejjednodušší operace, provádí ořezání fragmentů do obdélníka, který je specifikován pozicí, výškou a šířkou. Fragment mimo meze obdélníka je odstraněn.



Scissor test

- Specifikace oblasti pro ořezání:

```
void glScissor( GLint x, GLint y, GLsizei width, GLsizei height );
```

- Zapnutí scissor testu:

```
void glEnable( GL_SCISSOR_TEST );
```

- Vypnutí:

```
void glDisable( GL_SCISSOR_TEST );
```

- Dotaz na povolení či zakázání:

```
GLboolean glIsEnabled( GL_SCISSOR_TEST );
```

- Dotaz na souřadnice ořezávacího obdélníka:

```
void glGetIntegerv( GL_SCISSOR_BOX, &params );
```

Stencil test

- Pro každý vykreslovaný fragment se porovnává hodnota ve stencil bufferu se zadanou referenční hodnotou
- V závislosti na výsledku testu může být hodnota ve stencil bufferu změněna
- Nastavení stencil testu:

```
void glStencilFunc( GLenum func, GLint ref, GLuint mask );
```

Stencil test

- Parametry funkce glStencilFunc() jsou:
  - *func* – relační funkce
  - *ref* – referenční hodnota
  - *mask* – bitová maska, která se aplikuje na referenční hodnotu
- *func* může nabývat hodnot:

  GL_NEVER – fragment bude vždy odstraněn
  GL_ALWAYS – fragment vždy projde do dalšího zpracování
  GL_LESS – přijmout, když (*ref* & *mask*) < (*stencil* & *mask*)
  GL_LEQUAL – přijmout, když (*ref* & *mask*) <= (*stencil* & *mask*)
  GL_EQUAL – přijmout, když (*ref* & *mask*) = (*stencil* & *mask*)
  GL_GREATER – přijmout, když (*ref* & *mask*) > (*stencil* & *mask*)
  GL_GEQUAL – přijmout, když (*ref* & *mask*) >= (*stencil* & *mask*)
  GL_NOTEQUAL – přijmout, když (*ref* & *mask*) <> (*stencil* & *mask*)

Stencil test

- Pomocí funkce ***glStencilOp(fail, zfail, zpass);*** se specifikuje, co se má stát s hodnotou ve stencil bufferu v případě, že:
  1. test na šablonu fragment odmítne z dalšího zpracování
  2. test na hloubku fragmentu bude neúspěšný
  3. test na hloubku fragmentu bude úspěšný
- Možné hodnoty všech tří parametrů této funkce jsou:
  - *GL_KEEP* – ponechat původní hodnotu
  - *GL_ZERO* – vynulovat hodnotu ve stencil bufferu
  - *GL_INCR* – inkrementace uložené hodnoty
  - *GL_DECR* – dekrementace uložené hodnoty
  - *GL_INVERT* – vynásobení konstantou –1 (inverze bitů)
  - *GL_REPLACE* – nastaví hodnotu ve stencil bufferu na *ref* specifikované funkcí glStencilFunc()

Stencil test

- Povolení:

```
void glEnable(GL_STENCIL_TEST);
```

- Zakázání:

```
void glDisable(GL_STENCIL_TEST);
```

- Dotaz na povolení či zakázání:

```
GLboolean glIsEnabled(GL_STENCIL_TEST);
```

- Získání parametrů stencil testu:

```
glGetIntegerv(mode, &params);
```

Stencil test

- Parametr *mode* může nabývat hodnot:
  - GL_STENCIL_FUNC
  - GL_STENCIL_REF
  - GL_STENCIL_VALUE_MASK
  - GL_STENCIL_FAIL
  - GL_STENCIL_PASS_DEPTH_FAIL
  - GL_STENCIL_PASS_DEPTH_PASS

Depth test

- Relační funkci nastavujeme:

```
void glDepthFunc(GLenum func);
```

  - Parametr *func* může nabývat hodnot:

    GL_NEVER – fragment bude vždy odstraněn

    GL_ALWAYS – fragment vždy projde do dalšího zpracování

    GL_LESS – přijmout, když $Z_{fragmentu} < Z_{bufferu}$ – standardní nastavení

    GL_LEQUAL – přijmout, když $Z_{fragmentu} <= Z_{bufferu}$

    GL_EQUAL – přijmout, když $Z_{fragmentu} = Z_{bufferu}$

    GL_GREATER – přijmout, když $Z_{fragmentu} > Z_{bufferu}$

    GL_GEQUAL – přijmout, když $Z_{fragmentu} >= Z_{bufferu}$

    GL_NOTEQUAL – přijmout, když $Z_{fragmentu} <> Z_{bufferu}$

Depth test

- Povolení:

```
void glEnable(GL_DEPTH_TEST);
```

- Zakázání:

```
void glDisable(GL_DEPTH_TEST);
```

- Dotaz na povolení či zakázání:

```
GLboolean glIsEnabled(GL_DEPTH_TEST);
```

Blending

- Výpočet výsledné barvy závisí na nastavené míchací funkci:

```
void glBlendFunc(GLenum sfactor, GLenum dfactor);
```

- Povolení:

```
void glEnable(GL_BLEND);
```

- Zakázání:

```
void glDisable(GL_BLEND);
```

- Dotaz na povolení či zakázání blendingu:

```
GLboolean glIsEnabled(GL_BLEND);
```

Dithering

- Prováděn za účelem zdánlivého přidání počtu barev do zobrazované scény
- Povolení:

```
void glEnable(GL_DITHER);
```

- Zakázání:

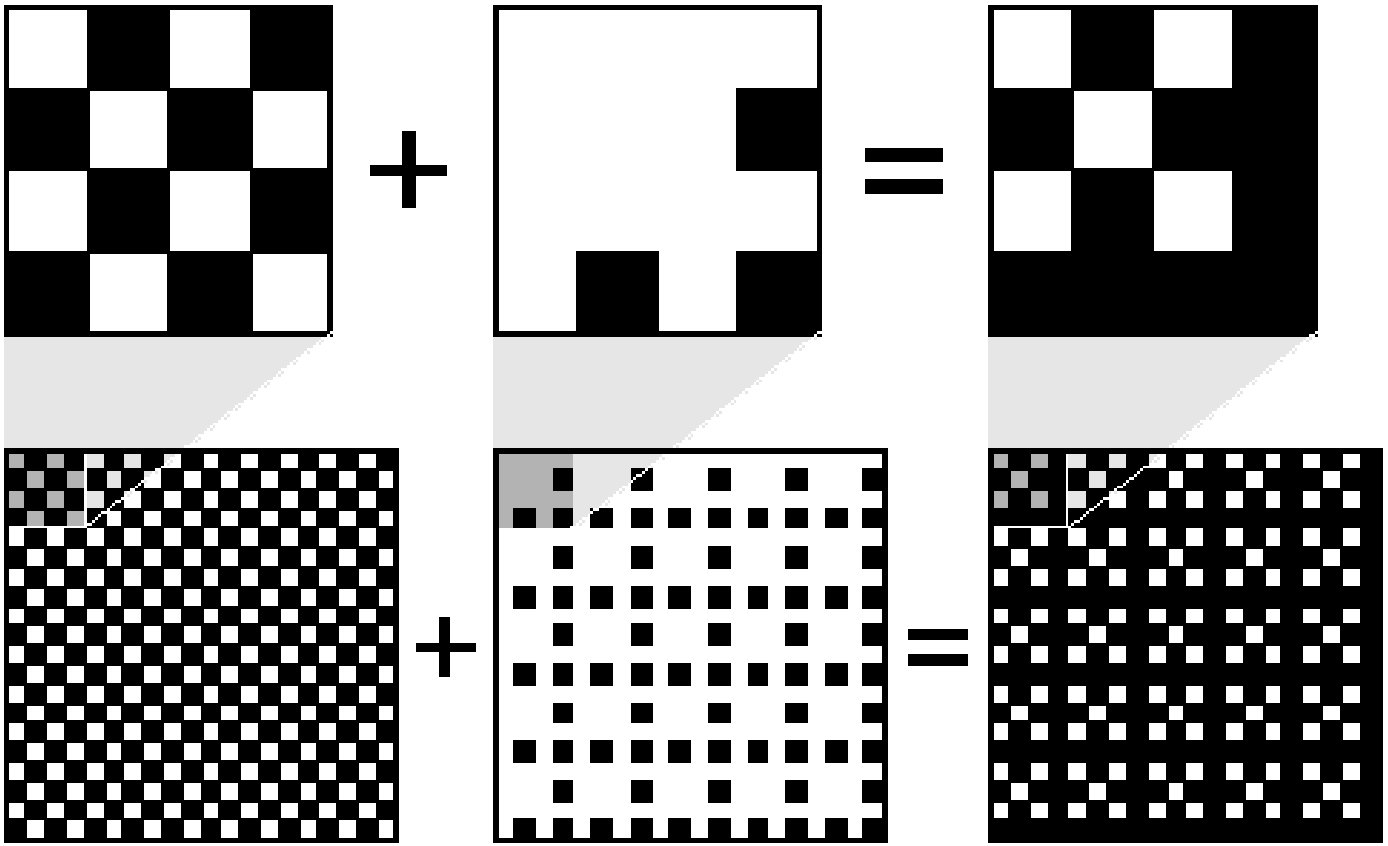
```
void glDisable(GL_DITHER);
```

- Dotaz na stav:

```
GLboolean glIsEnabled(GL_DITHER);
```

Dithering

- Příklad vytvoření šedé barvy



Maskování zápisu do bufferů – logické operace

- Při zápisu do jednotlivých bufferů se mohou provádět bitové nebo logické operace mezi zapisovanými daty a předem zadanou hodnotou
- Logické operace jsou aplikovány na hodnoty příchozích fragmentů (source) a hodnoty fragmentů ve fragmentovém bufferu (destination).

Maskování zápisu do bufferů – logické operace

```
void glLogicOp(GLenum opcode);
```

GL_CLEAR	0	GL_COPY	s
GL_NOOP	d	GL_SET	1
GL_AND	s&d	GL_OR	s|d
GL_AND_REVERSE	s&not(d)	GL_OR_REVERSE	s|not(d)
GL_COPY_INVERTED	not(s)	GL_INVERT	not(d)
GL_AND_INVERTED	not(s)&d	GL_OR_INVERTED	not(s)|d
GL_XOR	s XOR d	GL_EQUIV	not(s XOR d)
GL_NOR	not(s|d)	GL_NAND	not(s&d)

```
glEnable(GL_COLOR_LOGIC_OP);
```

- Pro každý typ bufferů z framebufferu je určena jedna z následujících funkcí:

Maskování zápisu do bufferů

- Při zápisu do barevných bufferů je každá barevná složka buď zapsána nebo nezapsána

- Nastavení příznaku zapsání pomocí

```
void glColorMask(GLboolean red,
GLboolean green, GLboolean blue,
GLboolean alpha);
```

Maskování zápisu do bufferů

- Pro paměť hloubky lze nastavit příznak, zda se má či nemá fragment zapsat, pomocí funkce:

```
void glDepthMask(GLboolean flag);
```

- Pro paměť šablony lze operací *and* nulovat jednotlivé bity:

```
void glStencilMask(GLuint mask);
```

```
void glStencilMaskSeparate(GLenum face,
GLuint mask);
```

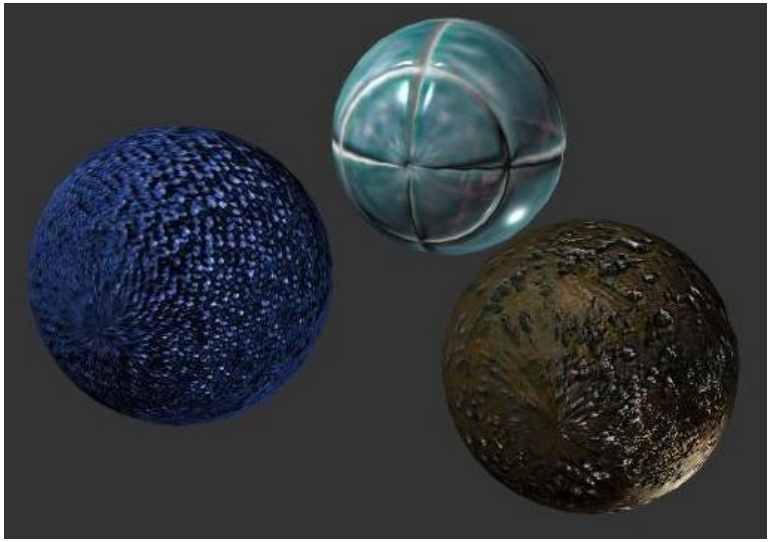
Maskování zápisu do bufferů

- Inicializace:

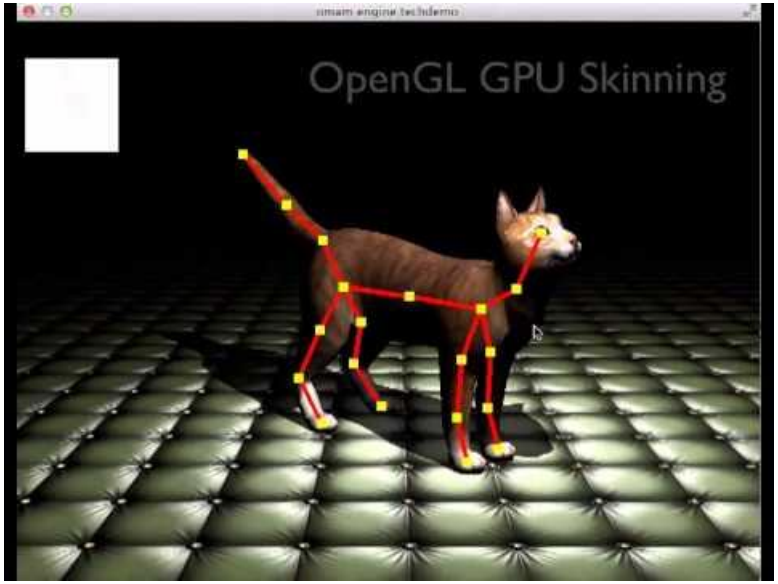
```
glColorMask(GLtrue, GLtrue, GLtrue, GLtrue);
glDepthMask(GLtrue);
glStencilMask((GLuint)~0);
```

Další efekty

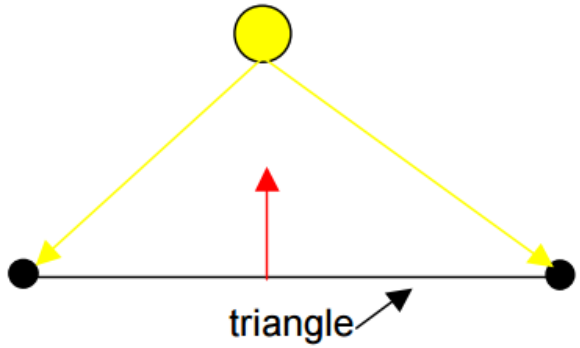
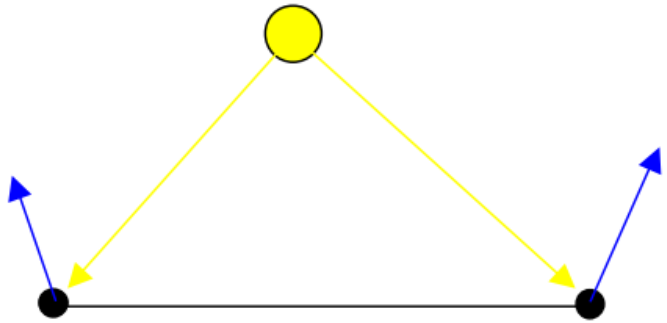
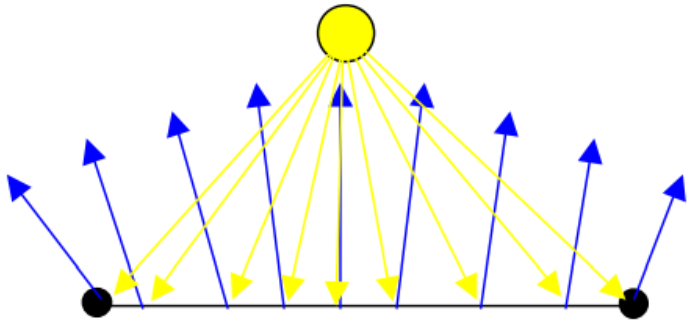
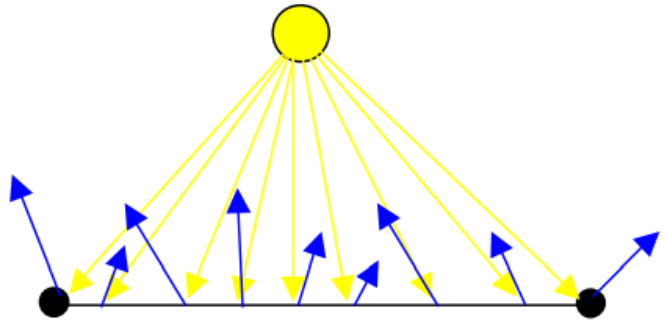
- Bump mapping



- Skinning



Bump mapping

Flat shading	Goraud shading
	
Only the first normal of the triangle is used to compute lighting in the entire triangle.	The light intensity is computed at each vertex and interpolated across the surface.
Phong shading	**Bump mapping**
	
Normals are interpolated across the surface, and the light is computed at each fragment.	Normals are stored in a bumpmap texture, and used instead of Phong normals.

Bump mapping

- Nutné znát normálovou mapu = textura, která se aplikuje na každý fragment obsahující základní texturu
- Normálová mapa může být odvozena z výškové mapy základní textury

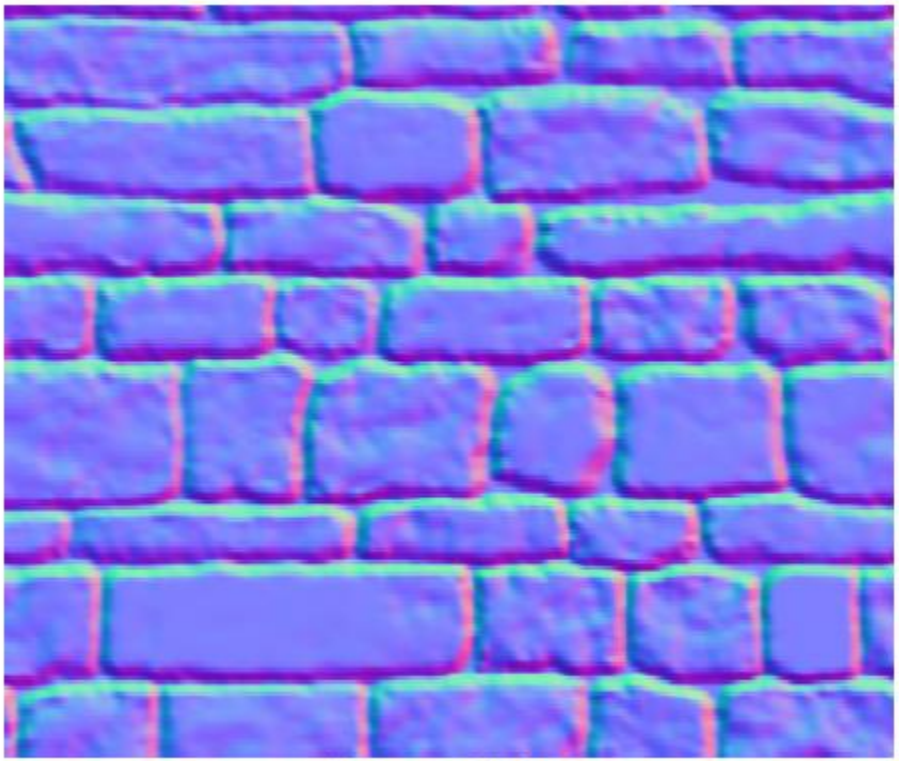
Bump mapping

Base texture

Height map

Bump mapping

- Převod na normálovou mapu



Normal Map

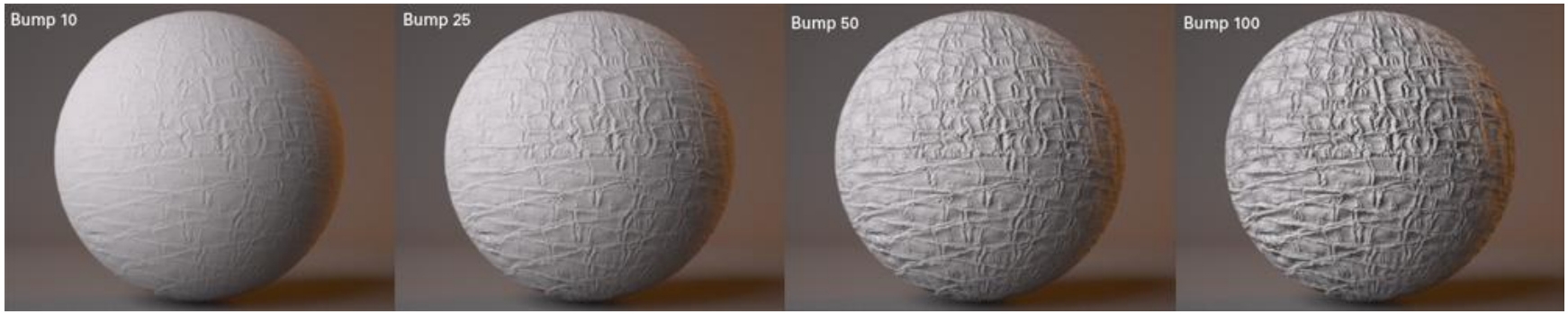
Bump mapping

- Normály jsou škálovány a posunuty do rozsahu [0, 255]
- Čtení z normálové mapy

```
vec3 normal = texture(myNormalMap,
texcoord.st).xyz * 2.0 – 1.0;
```

- Problém – rozdílný souřadný systém oproti vektoru světla, nutné unifikovat
  - detaily lze dohledat v literatuře

Bump mapping



Skinning

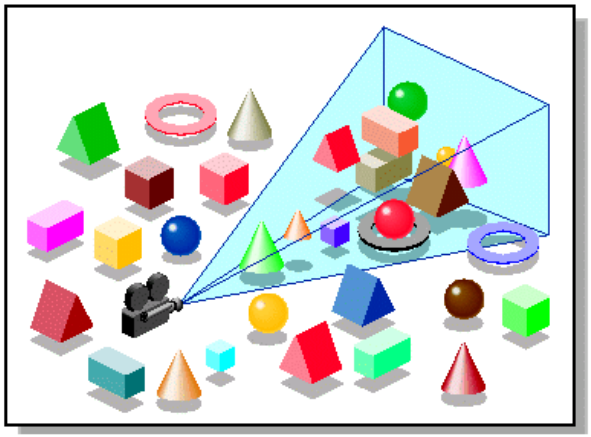
- Čeho chceme dosáhnout?

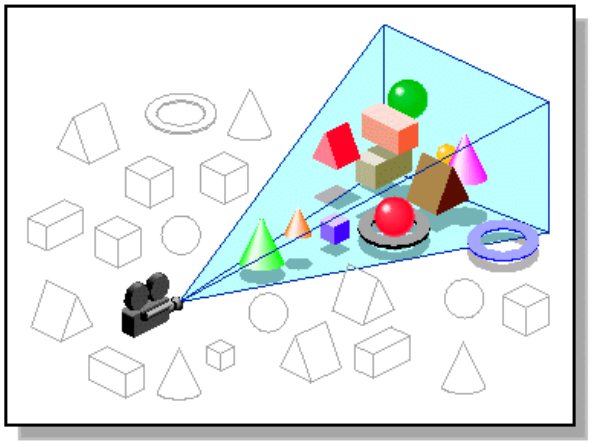
  https://www.youtube.com/watch?v=9xX7OotZ3P0

- Jak?
  - Transformací pozic vrcholů a normál meshe na základě matice reprezentující animaci kostí, ke kterým jsou vrcholy přiřazeny pomocí vah

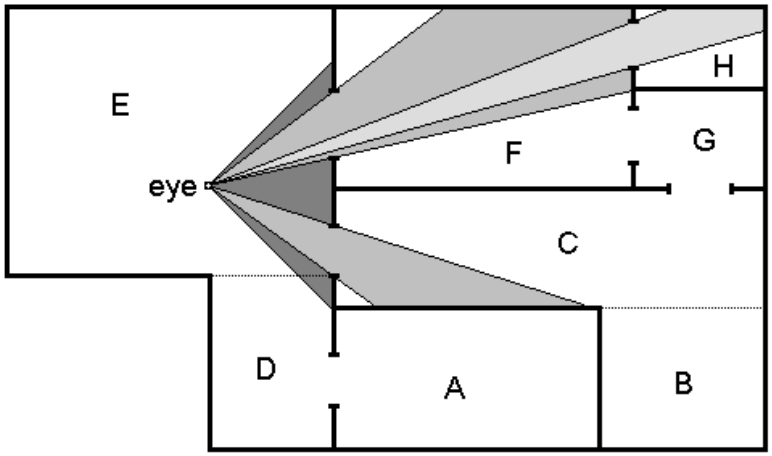
Culling

- View frustum culling



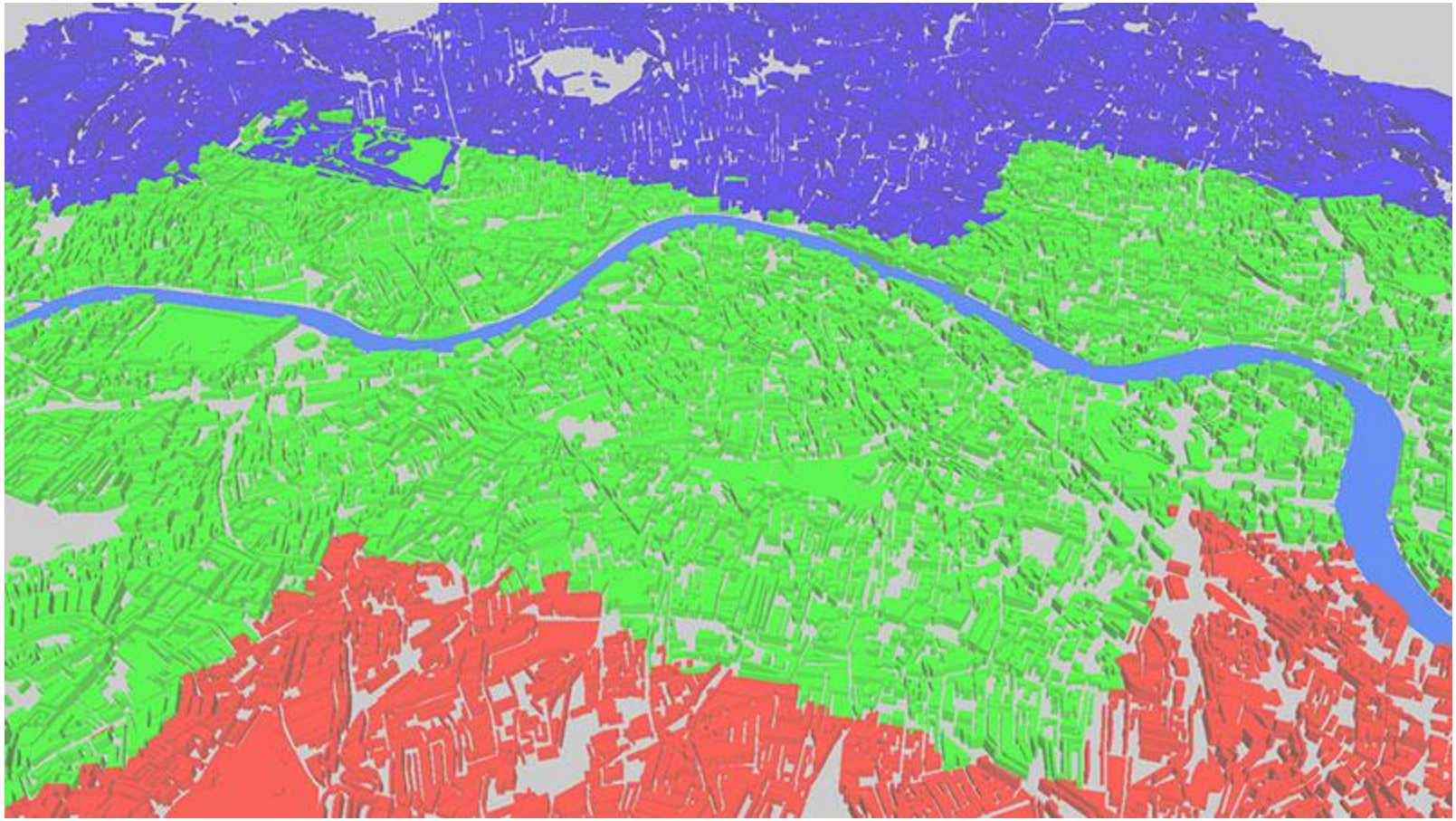


- Portal culling



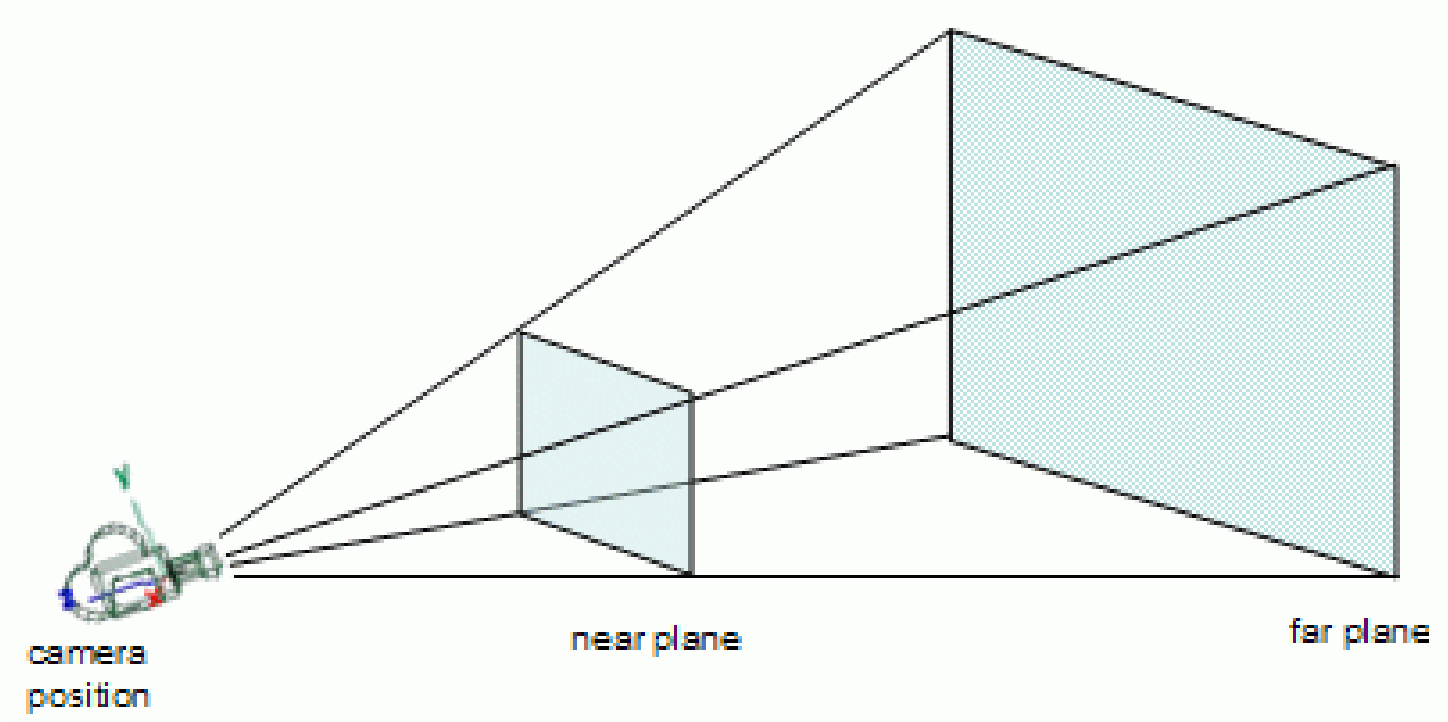
Culling

- Detail culling (LOD)



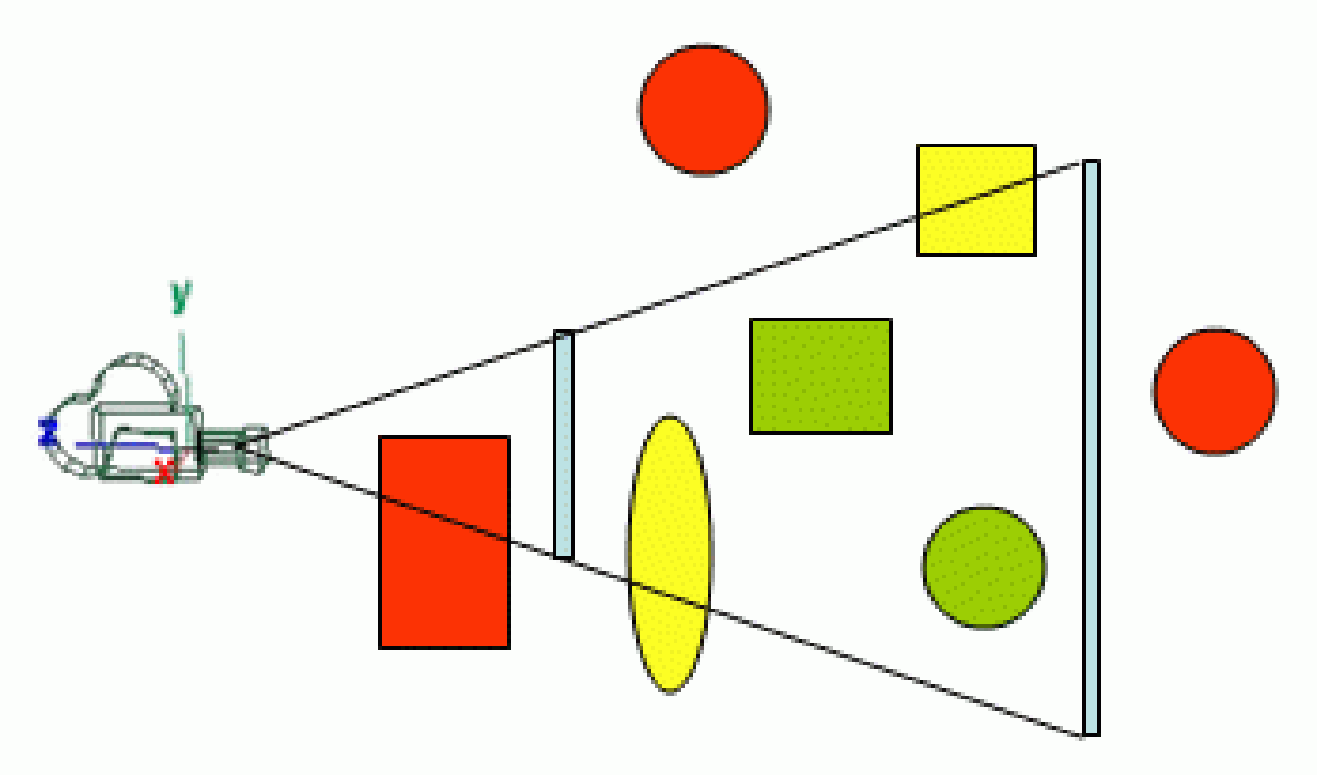
View frustum culling

- Ořezávání podle pohledového objemu



View frustum culling

- Objekty mimo pohledový objem nemá smysl vykreslovat



Portal culling

- Rozdělení scény na buňky obsahující portály
- Pro každou buňku je spočtena potenciálně viditelná sada buněk
  - Ty je třeba vykreslit v případě, že je pozorovatel do příslušné buňky umístěn

Detail culling

- Level of detail technika
- Na základě vzdálenosti od pozorovatele přidává nebo odebírá detaily