PV112 Programování grafických aplikací

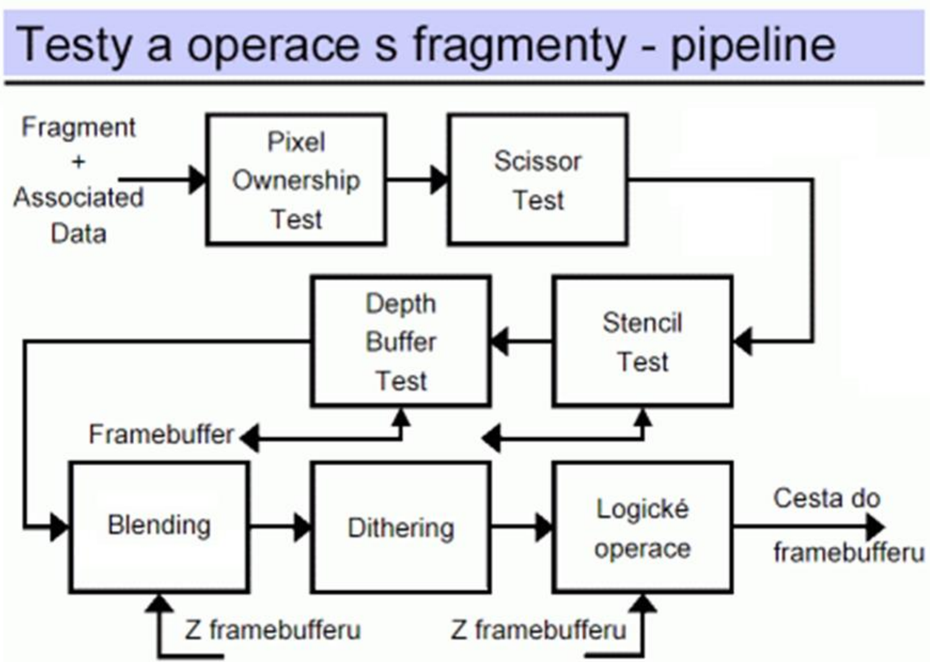
Jaro 2017

Výukový materiál

9. přednáška: Operace s fragmenty, efekty, culling

Operace s fragmenty

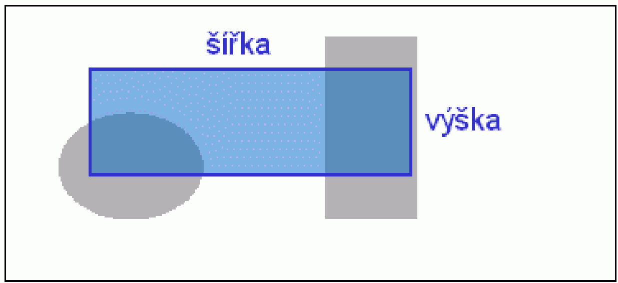
S fragmenty lze ještě před jejich zápisem do framebufferu provést řada operací. Na obrázku je zobrazeno pořadí operací, které se mohou provádět s vykreslovanými fragmenty. Šipky naznačují směr pohybu dat. Například v bloku Depth buffer test se přečte hloubka již zapsaného fragmentu a v případě úspěšného porovnání se do framebufferu zapíše hloubka nová – šipka je tedy obousměrná. Naopak, při provádění blendingu se hodnoty z framebufferu pouze čtou, neboť zápis se provede až po dalších operacích s vykreslovaným fragmentem – šipka je jednosměrná.



Při rasterizaci je možné podrobit vzniklé fragmenty před jejich zápisem do framebufferu celé řadě testů a operací, které mohou změnit obsah fragmentu nebo dokonce celý fragment odstraní z dalšího zpracování.

Scissor test

Scissor test neboli ořezání nůžkami patří mezi nejjednodušší operace, které jsou s fragmenty prováděny. Scissor test provádí ořezání fragmentů do obdélníku, který je specifikován svou pozicí, šířkou a výškou. Pokud se fragment nachází uvnitř tohoto obdélníku, putuje k dalším testům, pokud je mimo meze obdélníku, je odstraněn.



Na specifikaci oblasti pro ořezání se používá funkce:
void glScissor(GLint x, GLint y, GLsizei width, GLsizei height);

Zapnutí (povolení) scissor testu:
void glEnable(GL_SCISSOR_TEST);

Vypnutí scissor testu:
void glDisable(GL_SCISSOR_TEST);

Dotaz na povolení či zakázání scissor testu:
GLboolean glIsEnabled(GL_SCISSOR_TEST);

Dotaz na souřadnice nastaveného ořezávacího obdélníku:
void glGetIntegerv(GL_SCISSOR_BOX, ¶ms);

Stencil test

Stencilový test je umožněn pouze za předpokladu přítomnosti stencilového bufferu. Stencil test (test na šablonu) se provádí pro každý vykreslovaný fragment tak, že se porovnává hodnota ve stencil bufferu se zadanou referenční hodnotou. V závislosti na výsledku testu může být hodnota ve stencil bufferu změněna, což představuje široké možnosti použití, například při tvorbě těles pomocí CSG nebo při zobrazování stínů.

Nastavení stencil testu se provádí pomocí funkce:
void glStencilFunc(GLenum func, GLint ref, GLuint mask);

Parametr func představuje relační funkci, která se bude provádět, parametr ref referenční hodnotu a parametr mask bitovou masku, která se aplikuje jak na referenční hodnotu, tak

i na hodnotu uloženou ve stencil bufferu. Možnosti relační funkce jsou stejné jako u alfa testu:
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)

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()

Implicitně: **GL_KEEP, GL_KEEP, GL_KEEP**
fail – Jestliže fragment nesplnil stencilový test, jinak:
zfail – Jestliže fragment nesplnil hloubkový test
zpass – Jestliže fragment splnil hloubkový test

Povolení stencil testu:
void glEnable(GL_STENCIL_TEST);

Zakázání stencil testu:
void glDisable(GL_STENCIL_TEST);

Dotaz na povolení či zakázání stencil testu:
GLboolean glIsEnabled(GL_STENCIL_TEST);

Získání parametrů stencil testu:
glGetIntegerv(GL_STENCIL_FUNC, ¶ms;); // relacni funkce
glGetIntegerv(GL_STENCIL_REF, ¶ms;); // referencni hodnota

glGetIntegerv(GL_STENCIL_VALUE_MASK, ¶ms;); // bitova maska
glGetIntegerv(GL_STENCIL_FAIL, ¶ms;); // funkce zadana glStencilOp
glGetIntegerv(GL_STENCIL_PASS_DEPTH_FAIL, ¶ms;);
glGetIntegerv(GL_STENCIL_PASS_DEPTH_PASS, ¶ms;);

Depth test

Test na hodnotu uloženou v paměti hloubky si ještě jednou vysvětlíme dále. Zde si pouze popíšeme nastavení relační funkce, která se provádí mezi hloubkou právě vykreslovaného fragmentu a hodnotou uloženou v paměti hloubky. Relační funkci nastavujeme pomocí příkazu:

void glDepthFunc(GLenum func);
kde parametr *func* může nabývat následujících 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 $Z_{fragmentu} <> Z_{bufferu}$

Povolení testu na hloubku:
void glEnable(GL_DEPTH_TEST);

Zakázání testu na hloubku:
void glDisable(GL_DEPTH_TEST);

Dotaz na povolení či zakázání testu na hloubku:
GLboolean glIsEnabled(GL_DEPTH_TEST);

Blending

Operaci blendingu už jsme poznali, zde si pouze stručně řekneme, že se jedná a výpočet barevné kombinace právě zpracovávaného fragmentu s fragmentem uloženým ve framebufferu. Výpočet výsledné barvy závisí na nastavené blendovací (míchací) funkci:

void glBlendFunc(GLenum sfactor, GLenum dfactor);

Povolení blendingu:
void glEnable(GL_BLEND);

Zakázání blendingu:
void glDisable(GL_BLEND);

Dotaz na povolení či zakázání blendingu:
GLboolean glIsEnabled(GL_BLEND);

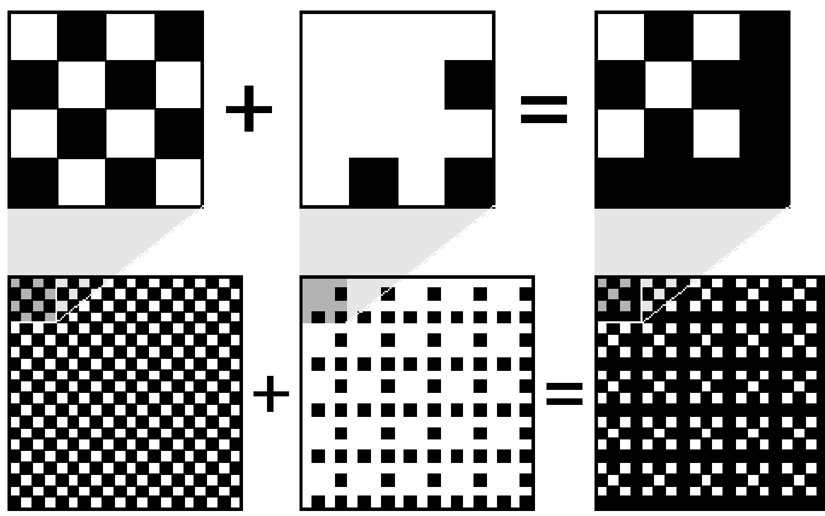
Dithering

Dithering může být prováděn za účelem zdánlivého přidání počtu barev do zobrazované scény. Jedná se o dobře známý postup, který se běžně používá při tisku na všech tiskárnách – z pixelů několika základních barev se jejich vzájemným promícháním získají další barevné odstíny.
Povolení ditheringu:
glEnable(GL_DITHER);

Zákaz ditheringu:
glDisable(GL_DITHER);

Dotaz na stav ditheringu:
GLboolean glIsEnabled(GL_DITHER);

Jako příklad si uvedeme vytvoření různých odstínů šedé pomocí ditheringu. Uvedené vzorky představují situaci pro 50% 19% a 69% šedé. Jednotlivé pixely jsou ve všech případech černé nebo bílé, liší se pouze jejich poměrem. Šedá barva se projeví až z pohledu z větší dálky.



Maskování zápisu do bufferů – logické operace

Při zápisu dat do jednotlivých bufferů se mohou provádět bitové nebo logické operace logického součinu mezi zapisovanými daty a předem zadanou hodnotou. Logické operace jsou aplikovány na hodnoty příchozích fragmentů (source) a hodnoty fragmentů ve fragmentovém bufferu (destination). Pro každý typ bufferů z framebufferu je určena jedna z následujících funkcí.

```
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¬(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:

Každá barevná složka zapisovaného fragmentu je podle nastaveného příznaku buď zapsána, nebo nezapsána. Nastavování příznaků se provádí pomocí funkce:

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

Pro paměť hloubky lze také nastavit příznak, zda se má, či nemá fragment zapsat. Použitá funkce má hlavičku:

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

Posledním bufferem je paměť šablony, kde lze při zápisu bitovou operací *and* nulovat jednotlivé bity, podobně jako u barvového bufferu nastaveného do paletového režimu. Funkce pro nastavení masky má hlavičku:

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

Rozlišujeme dva různé typy stencilových masek – jedna ovlivňuje polygony tvořící zadní stěny, druhá přední polygony a nepolygonální primitiva. Funkce *glStencilMask*() nastavuje hodnoty pro obě masky stejně. Pokud chceme pro tyto primitiva použít masky odděleně, můžeme od verze OpenGL 2.0 využít funkci *glStencilMaskSeparate()*.

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

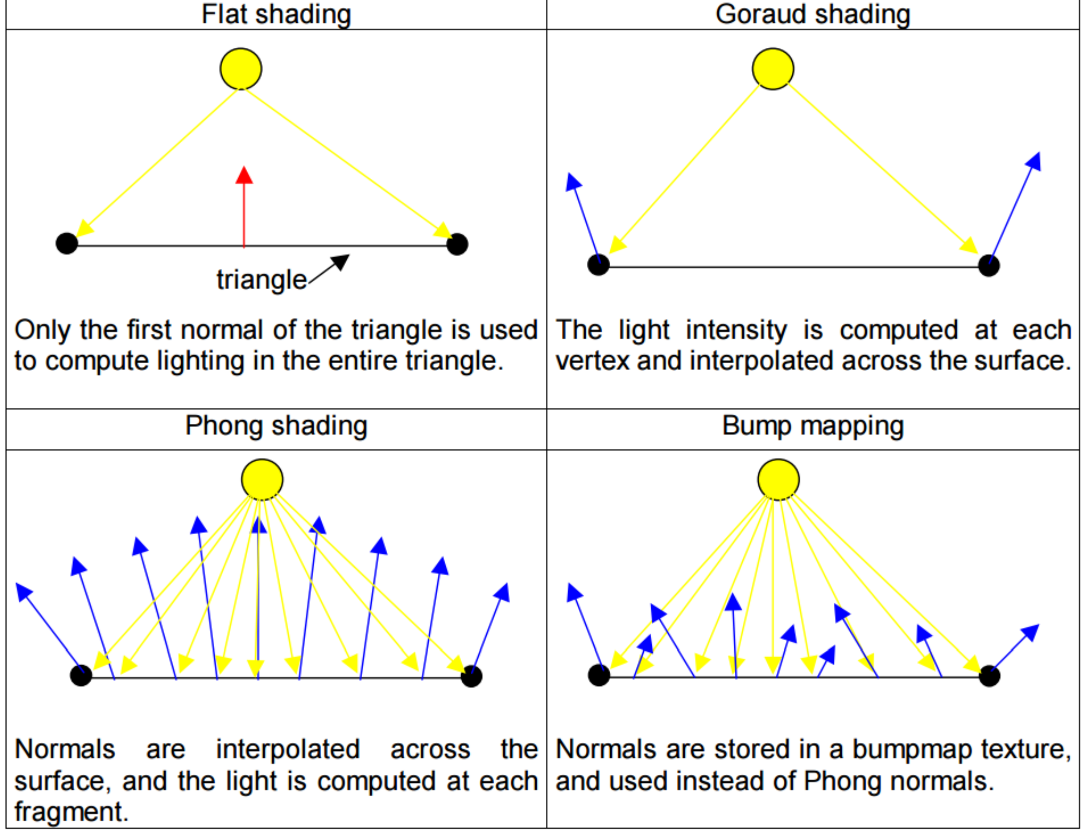
Pokud jako parametr face nastavíme GL_FRONT_AND_BACK, výsledek bude stejný jako za použití *glStencilMask()*.
Inicializační hodnoty při spuštění aplikace jsou nastaveny tak, jako by se volala sekvence funkcí:

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

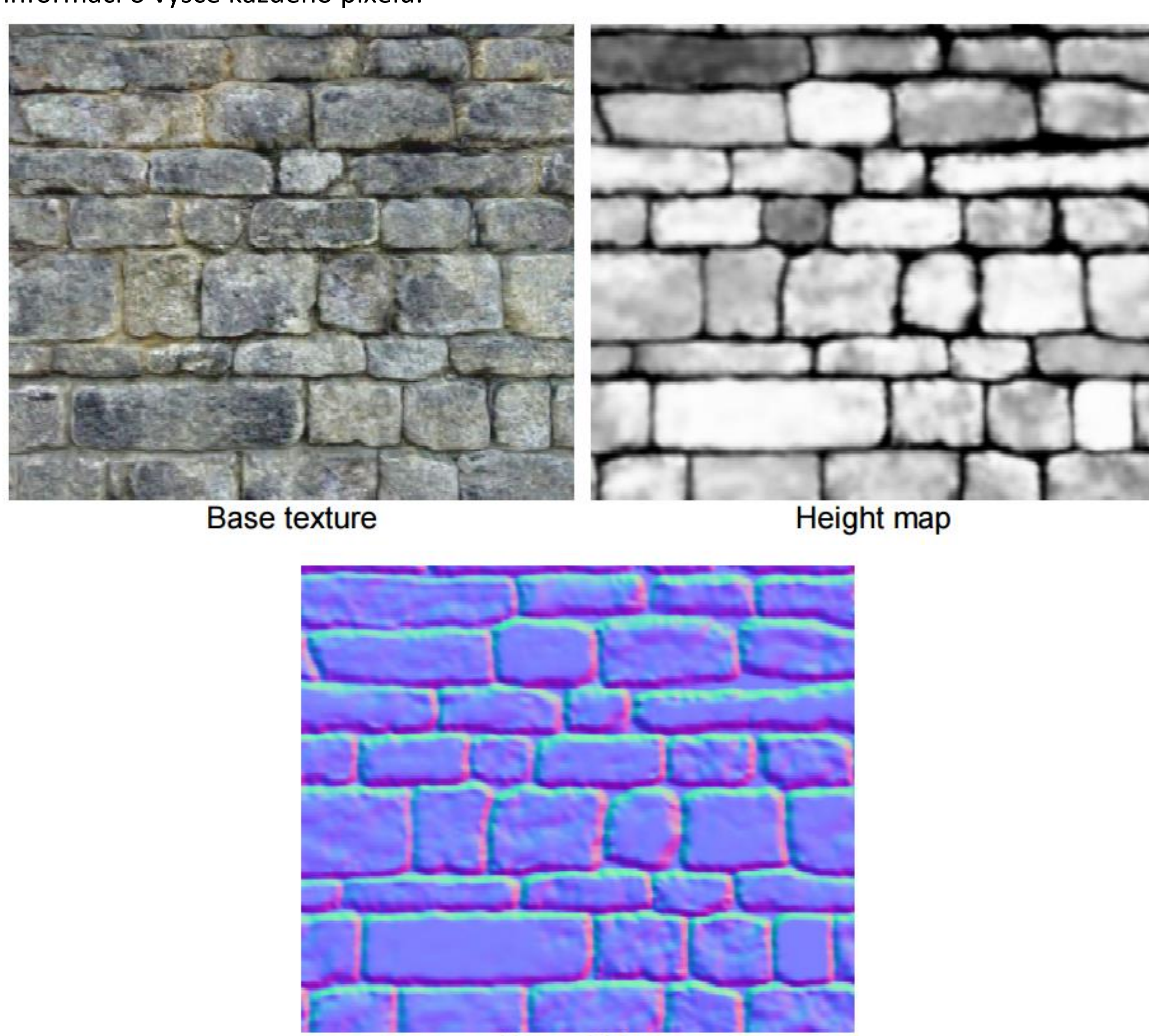
Další efekty

Mezi další efekty, o kterých se stručně zmíníme, patří bump mapping a skinning.

Bump mapping je jeden z nejznámějších a nejpoužívanějších světelných efektů. Abychom si vysvětlili, jak funguje, rekapitulujeme si principy různých typů stínování, které jsme již probírali. Nejjednodušší flat shading používá pouze jednu normálu pro výpočet osvětlení celého trojúhelníku. Při Gouraudově stínování je intenzita světla spočtena pro každý vrchol trojúhelníka a hodnota normály je interpolována. U Phongova stínování jsou normály interpolovány na povrchu trojúhelníka a světlo je spočteno pro každý fragment zvlášť. U bump mappingu jsou normály určeny podle textury, tedy texturové souřadnice ovlivňují směr a velikost normálových vektorů jednotlivých fragmentů.



Abychom měli dostatek informací pro spočtení bump mappingu, musíme mít definovánu tzv. *normálovou mapu*. Normálová mapa je v podstatě textura, která je aplikována na každý fragment se základní texturou. Pro získání normálové mapy můžeme použít například výškovou mapu základní textury. Výšková mapa je v podstatě šedotónní textura uchovávající informaci o výšce každého pixelu.



Normály z normálové mapy jsou poté škálovány a posunuty, aby byly v rozsahu [0, 255]. Při čtení z normálové mapy musíme zavolat pouze

vec3 normal = texture2D(myNormalMap, texcoord.st).xyz * 2.0 – 1.0;

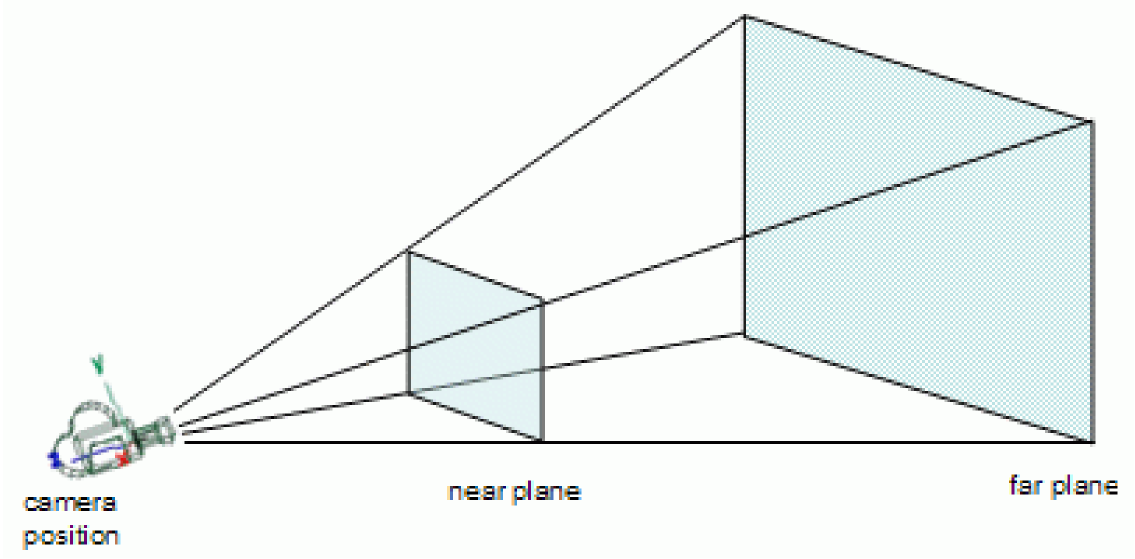
Poté již máme normály v požadovaném rozsahu [-1,+1] a připraveny k použití. Tyto normály však nejsou ve stejném souřadném systému jako vektor světla ve vertex shaderu, je třeba sjednotit. Tato operace je netriviální, detaily lze dohledat v literatuře.

U **skinningu** chceme dosáhnout realistické deformace textury, která je namapována na pohyblivý object (s kostrou). Dá se toho dosáhnout transformací pozic vrcholů a normál meshe na základě matice reprezentující animaci kostí, ke kterým jsou vrcholy přiřazeny pomocí vah. Detaily lze dohledat v literatuře.

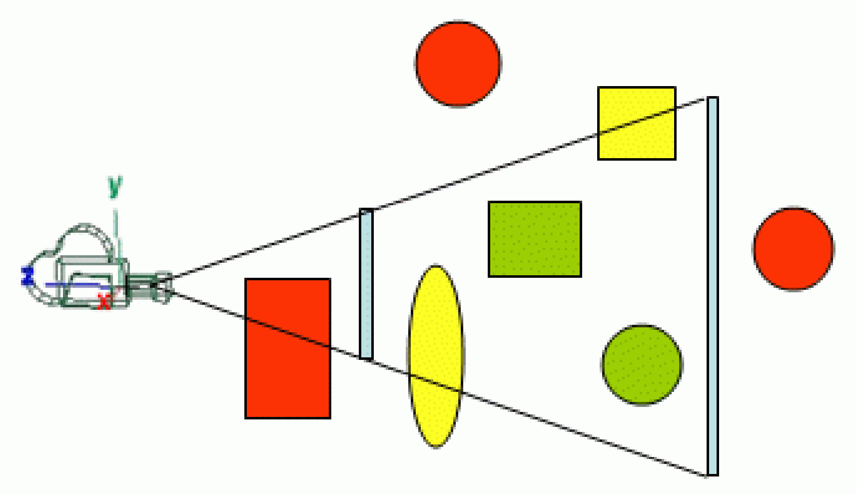
Culling

Culling, neboli odstřel objektů, je technika, která se používá pro odstraňování těhc objektů nebo jejich částí, které nebudou vykresleny. Mezi běžné techniky používané pro odstřel patří:

View frustum culling (ořezávání podle pohledového objemu)
Ořezávání podle pohledového objemu využívá informaci o pozici kamery a pohledový komolý jehlan. Objekty protínající tento jehlan jsou vykresleny. Základna jehlanu reprezentuje zadní ořezávací rovinu, vrchol jehlanu pozici kamery. Komolý jehlan je pak vytvořen přidáním přední ořezávací roviny.



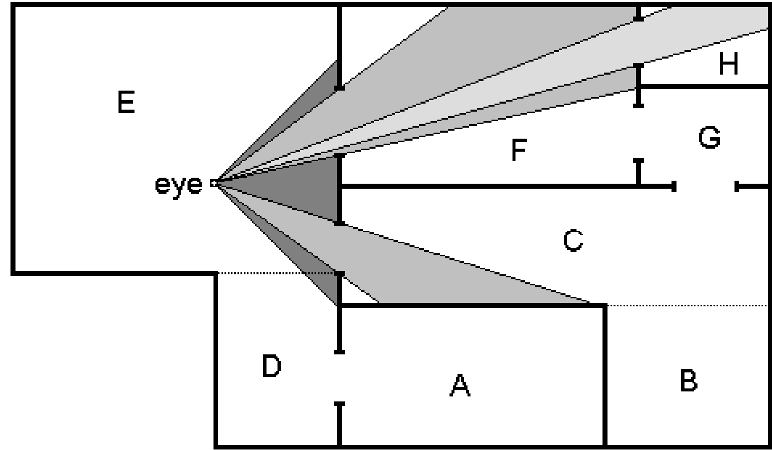
Objekty, které nespadají do objemového jehlanu a ani jej neprotínají nemá smysl renderovat – na obrazovce se neobjeví. Na obrázku budou vyrenderovány zelené a žluté objekty. Naopak červené objekty leží mimo jehlan, nebudou tedy vykresleny.



Cílem view frustum cullingu je tedy identifikovat objekty, které jsou uvnitř pohledového jehlanu nebo jej protínají a poté zbývající objekty ořezat. V důsledku dochází ke zlepšení výkonu grafické karty, která renderuje méně objektů.

Portal culling

Portal culling rozdělí scénu na sadu buněk, které obsahují díry (dveře nebo okna) označované jako portály. Poté je každá buňka analyzována, aby bylo možné určit, které další buňky jsou viditelné z dané buňky skrz její portály. Výsledkem je potenciálně viditelná sada (potentially visible sets - PVS), která pro každou buňku definuje sadu buněk, které je třeba vykreslit v případě, že je pozorovatel umístěn do této buňky.



Detail culling

Detail culling, neboli geometrický level of detail, určuje, jak blízko je daný objekt k pozorovateli a na základě této informace přidává nebo odebírá detaily z daného objektu.

Úroveň detailu je ovlivněna tím, jestli se objekt přibližuje k pozorovateli nebo naopak oddaluje.