Dokumentace

Téma:

Nestandartní využití shadow mappingu při renderu scény

Autor:

David Nápravník

Datum: 11. 5. 2020

Podrobnosti tématu

Vytvoření 3D enginu, který dostane 3D Scénu a real-time ji vykreslí uživatelovi. Uživatel se pak může ve scéně volně pohybovat. Objekt bude **přijímat i vytvářet stíny**. Přesněji řečeno bude přijímat nebo propouštět světlo a to v kompletně nestatické scéně. (všechny výpočty se počítají s každým snímkem znovu)

Popis programu

Základem je knihovna **OpenGL**, která umožňuje komunikaci s grafickou kartou a knihovny glm pro práci s maticemi, glad jako loader pro OpenGL a GLFW pro správu okna.

Při startu si program natáhne všechny modely shadery a textury. Ty si následně uloží do datových struktur pro to určených. Textury ukládá na grafickou kartu. Soubory s modely sparsuje a vytvoří z nich data pro grafickou kartu (Vertex buffery) a Shadery zkompiluje a sestrojí z nich renderovací Program.

Poté sestaví scénu, vygeneruje kameru a světla, po dokončení úvodního načítání (které může trvat i pár vteřin) se spustí nekonečná renderovací smyčka.

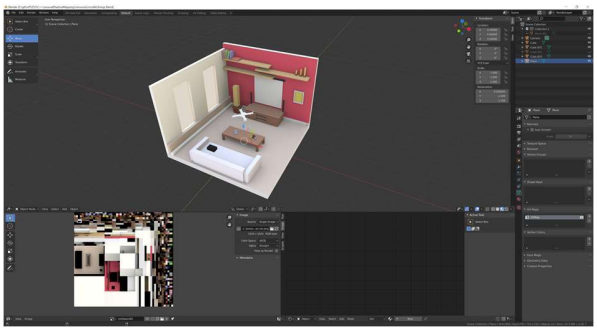
V renderovací smyčce se nejdříve zpracují změny od posledního vykreslení, jako jsou údery kláves nebo pohyby objektů podle předdefinované trajektorie.

Vzápětí přichází samotné vykreslení. To je rozděleno do tří částí. V první fázi se vypočítají data pro světla. V druhé fázi se vykreslí scéna podle dat z fáze předchozí. Na konci přichází fáze, která je pouze pro testovací a prezentační účely a v produkci by měla být skryta. Zde se vykreslí data z první fáze jak byly předány do fáze druhé (scéna z pohledu světla)

Řešené problémy

Načítání modelů

Modely byly vytvořeny v Blenderu a exportovány do souboru .obj (lidsky čitelný zápis modelu)



Program blender

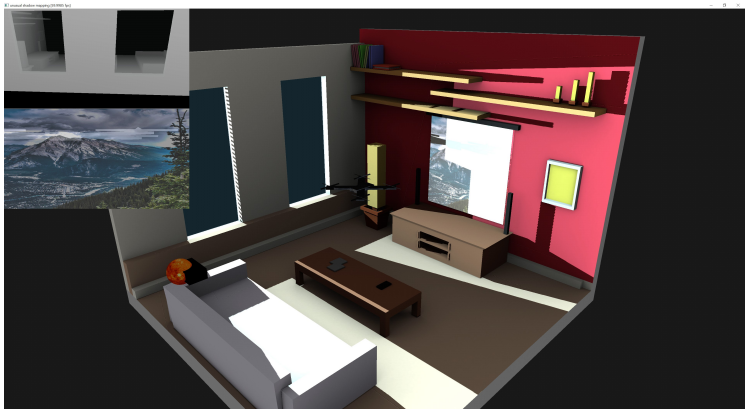
```
# Blender v2.80 (sub 74) OBJ File: 'cathedral.blend'
# www.blender.org
mtllib grass.mtl
o Plane
v -20.000000 0.000000 20.000000
v 20.000000 0.000000 20.000000
v -20.000000 0.000000 -20.000000
v 20.000000 0.000000 -20.000000
vt 4.499999 -3.500000
vt -3.499999 4.500000
vt -3.500000 -3.499999
vt 4.500000 4.499999
vn 0.0000 1.0000 0.0000
usemtl floor
s off
f 2/1/1 3/2/1 1/3/1
f 2/1/1 4/4/1 3/2/1
```

Ukázka modelu ve formátu .obj

Soubor je poté nutno projít a vyparsovat vrcholy, normálové vektory, souřadnice na texturování a plochy, jež popisují.
Data o modelu se poté nahrají do Vertex Bufferu. Pro každou plochu (trojúhelník) ve formátu [poziceX, poziceY, poziceZ, texturaX, texturaY, normálaX, normálaY, normálaZ].

Každý model má tedy svůj naplněný Vertex array object a ID textury. ID textury získává z grafické karty po jejím nahrání.

Výpočet stínu

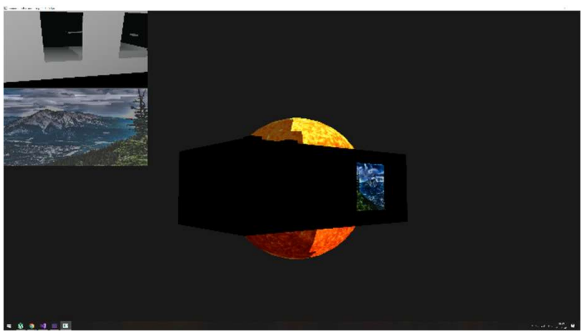


Při první fázi se vykreslí scéna z pohledu světla a tím získáme hloubku scény. (levý horní obdélník)
Dále při vykreslení průhledných objektů zjistíme i barvu světla. (levý spodní obdélník)

Při druhé fázi se vykresluje scéna a u každého pixelu se zjišťuje, zda je osvětlen, popřípadě jakou barvou.

Všechny výpočty stínu se řeší na grafické kartě pomocí vertex a fragment shaderu. První fáze používá shadery „shadow_mapping_color“ a „shadow_mapping_depth“. Kde se řeší pouze pozice vertexu a čistá barva textury. Druhá fáze používá shadery „shadow_mapping“. Ve vertex bufferu se kromě pozice daného vertexu spočítá i jeho relativní pozice k oběma světlům a kameře a dopočítají se normálové vektory. V fragment shaderu se potkají data o fragmentu daného modelu (pozice, normály, texturovací souřadnice), obě světla a jejich frame-buffery (znázorněné vlevo nahoře) a informace o kameře. Spočítá se tu průnik vektoru světla s vektorem kamery a porovná s dříve zjištěnou hodnotou uloženou v frame-bufferu. Pokud se schodují ± bias, daný fragment (pixel) je osvětlen. Barva světla se dopočítá ze sekundárního frame-bufferu u světla. Pokud se hodnoty ale neshodují, znamená to, že bod není osvícen, minimálně ne tímto světlem a tudíž je hodnota světla nastavena na RGB(0,0,0). Po zjištění světelných podmínek se sečte ambient světlo (neexistující světlo, jež má všude stejnou intenzitu, používá jako nahrazení výpočetně extrémně náročného light-bouncingu aka ray-tracingu) a světla z obou světel a výsledná barva se vyrenderuje do hlavního frame-bufferu.

Projekce



Promítačka s žárovkou (sluncem) a promítací destičkou

Stín na plátně za dronem a projekce na něm

V ukázkové scéně se na plátno promítá obraz z dataprojektoru. A to tak, že v dataprojektoru je umístěna textura obrázku, jenž má být promítán a jen pomocí shadow-mappingu se promítne na plátno.

Jelikož nehledáme stín, ale promítáme světlo, je možné realisticky zobrazit promítané obrázky nejen na plátno, ale i na objekty před plátnem. Na plátno za objektem pak správně nedopadá, žádné světlo.

Naopak, pokud do promítajícího obrazu zasvítí denní světlo, obraz je realisticky přesvícen a téměř bez textury.

Pro možnost uchovávat nejen vzdálenost, ale i barvu, je každému světlu přiřazen dodatečný frame-buffer. Ačkoliv by se dali data o vzdálenosti a barvě vložit do jediné textury (vzdálenost by byla uložena jako alfa kanál),

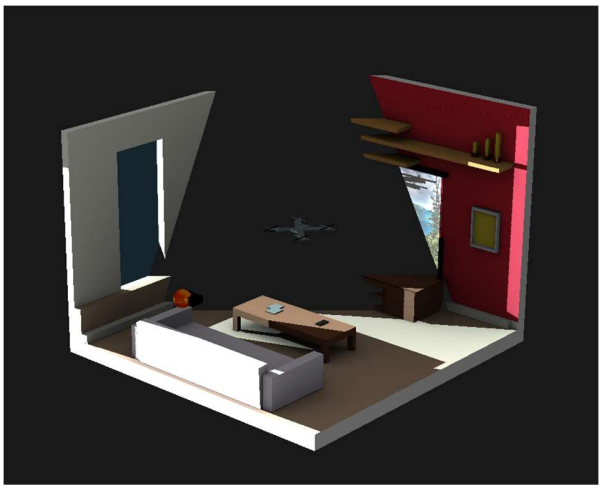
tak z pohledu rozšiřitelnosti je lepší tyto data rozdělit. V budoucnu je tudíž možné libovolně přidávat resp. ubírat přesnost jednotlivých složek, nebo klidně vynechat hloubkovou informaci pro objemové modely (volume object).

Mlha pro far_plane

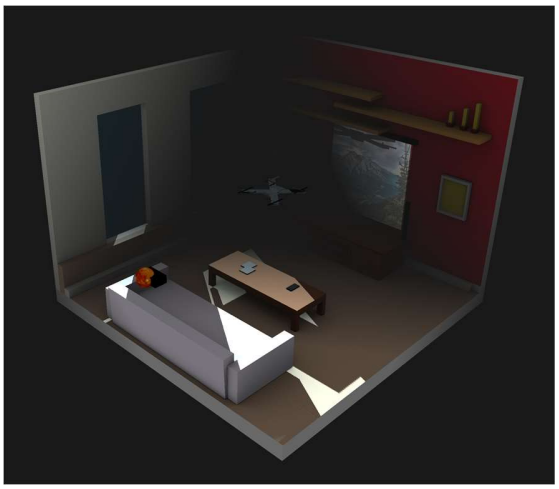
Jelikož kamera má určité rozmezí odkud kam může vykreslovat, existuje tkz. Far_plane, jež zamezuje renderu věcí za ní. Ta je však velmi výrazná a uživateli brzy dojde, že něco není v pořádku. Proto je ještě před touto hranicí obraz plynule zatemněn.

Výsledku dosáhneme na konci fragment shaderu, těsně předtím, než se vrátí výsledná barva se hodnoty v blízkosti far-plane smíchají s barvou pozadí

```
FragColor.rgb = mix(FragColor.rgb, vec3(.1,.1,.1), max(min(distance(viewPos, fs_in.FragPos)/2-4,1.0),0));
```



Bez mlhy na přelomu

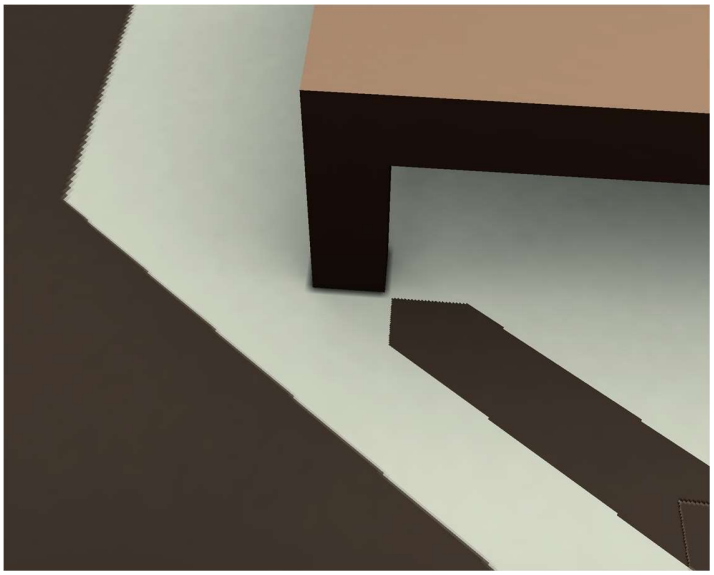


S mlhou na přelomu

Fresnel efekt stínu

S rostoucí vzdáleností se stín rozptyluje. Blízko objektu je ostrý a s rostoucí vzdáleností se změkčují jeho hrany.

Toho je docíleno porovnáváním okolních pixelů, zda jsou zakryty stínem a v případě rozdílu stín-světlo se pomocí vzdálenosti dopočte rozptyl.



V ukázce výše je dobře vidět, že levý stín (způsobený hranou od okna) je rozmazaný, ale pravý stín (od nohy stolu) je ostrý (tak jak nám to rozlišení dovolí).

To že stín nezačíná v místě kde se noha stolu dotýká země je způsobeno nedostatečnou přesností floatu, ta je v této ukázce pouze 8-bitová, avšak dá se za mírného snížení výkonu zvýšit až k 32-bitové přesnosti.

Výpočet stínu se provádí uvnitř fragment shaderu „shadow_mapping.fs“, kde jak bylo dříve zmíněno se porovnává vzdálenost fragmenu od kamery a fragment od světla. Pro hladký stín se však kontrolují I okolní fragment a na ty se aplikuje obrazový filtr[1]. Filt je podobný Sobelově operaci, která detekuje hrany v obraze. Podle vzdálenosti stínu od jeho zdroje se volí, který filtr se použije. Aktuálně jsou 3 filtry:

pro blízké: $\begin{bmatrix}0 & 0 & 0\\ 0 & 9 & 0\\ 0 & 0 & 0\end{bmatrix}$, pro středně daleké: $\begin{bmatrix}.5 & .5 & .5\\ .5 & 5 & .5\\ .5 & .5 & .5\end{bmatrix}$ a pro daleké: $\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$

všechny se sumou 9, aby se optimalizoval kód běžící na grafické kartě a snížilo se množství větvení kódu (if-else)

Spuštění

V projektu je soubor *unusualShadowMapping.sln*, který lze otevřít ve visual studiu (osobně používám verzi 2019), nebo ve složce *bin* jsou soubory unusualShadowMapping.exe a konfiguracne scén. Pokud jednu z těchto scén zadáte jako parametr programu (stačí drag-and-drop configu na .exe soubor), scéna se načte.

[1] obrazový filtr – algoritmus který dostane množinu 3x3 defunující filtr a množinu dat 3x3, výsledkem je jediné číslo.

Příklad: $\begin{bmatrix}0 & 1 & 0\\ 1 & 5 & 1\\ 0 & 1 & 0\end{bmatrix} * \begin{bmatrix}255 & 255 & 125\\ 255 & 125 & 0\\ 125 & 0 & 0\end{bmatrix} = \begin{bmatrix}0*255 & 1*255 & 0*125\\ 1*255 & 5*125 & 1*0\\ 0*125 & 1*0 & 0*0\end{bmatrix}$ => 1135 / 9 = 126 tudíž výsledná barva bude 126