Skybox

Pentru a da o nota de realism, jocul ar trebui sa aiba si un background. De exemplu, daca jucatorul se deplaseaza pe un teren deschis, sa poata sa vada zarea, cerul... Ca sa realizam acest efect vom folosi un cub (sau o sfera) care sa cuprinda scena (terenul, obiectele afisate). Cubul va fi destul de mare comparativ cu celelalte obiecte si va fi centrat fata de pozitia camerei. Centrarea se va realiza pe axele OX si OZ. Pentru axa OY vom da in fisierul de configurare o valoare-offset (fata de pozitia camerei).

Cand ne uitam in zare, si in acelasi timp ne deplasam, avem senzatia ca norii, sau alte obiecte foarte departate “se deplaseaza” odata cu noi. Aceasta iluzie exista din cauza distantei foarte mari la care se afla acele obiecte. Vom realiza acest efect deplasand skyboxul odata cu deplasarea camerei, pe axele OX,OY, OZ, astfel incat skyboxul sa isi pastreze aceeasi pozitie fata de camera pe toata durata jocului.

Pentru texturarea skyboxului vom folosi un *cube map*. Spre deosebire de cazul in care aplicam texturi 2D, pentru cube map vom defini o imagine pentru fiecare fateta. Adesea, cube map-urile se aplica din 6 fisiere-imagine separate. Astfel, functia glTexImage2D va fi apelata de 6 ori, iar pe primul parametru in loc sa avem GL_TEXTURE_2D vom avea o constanta care va indica fatea careia ii aplicam textura.

Constantele sunt:

```
GL_TEXTURE_CUBE_MAP_POSITIVE_X
GL_TEXTURE_CUBE_MAP_NEGATIVE_X
GL_TEXTURE_CUBE_MAP_POSITIVE_Y
GL_TEXTURE_CUBE_MAP_NEGATIVE_Y
GL_TEXTURE_CUBE_MAP_POSITIVE_Z
GL_TEXTURE_CUBE_MAP_NEGATIVE_Z
```

De exemplu, GL_TEXURE_CUBE_MAP_POSITIVE_Y e pentru fateta de sus (deoarece sensul pozitiv al axei OY e in sus), iar GL_TEXURE_CUBE_MAP_NEGATIVE_Y pentru fateta de jos. Formatul unei astfel de constante este de fapt: GL_TEXURE_CUBE_MAP_[sensul_axei]_[axa].

Din punct de vedere al valorilor, aceste constante sunt intregi pozitivi, consecutivi in ordine crescatoare (ordinea lor e cea precizata in lista de mai sus).

Si in shader se schimba un pic lucrurile. Nu vom mai avea sampler2D pentru textura, ci samplerCube si nu vom folosi functia textue2D() ci functia textureCube():

```
gl_FragColor=textureCube(u_cube_texture,  v_coord);
```

unde u_cube_texturea fost definit ca:

```
uniform samplerCube u_cube_texture;
```

De data asta *v_coord* nu mai e un vec2 ci un **vec3** pentru a indica coordonatele in cubul format

de textura. In cazul de fata v_coord va fi calculat in vertexshader din a_posL astfel:

```
v_coord=a_posL;
```

De ce acest lucru? Pentru ca, spre deosebire de coordonatele uv din textura 2D, in cazul texturii cub o sa avem drept "coordonate" un vector pornind din origine care intersecteaza cubul-textura. Culoarea texelului obtinut din acea intersectie este cea returnata de functia textureCube(). Astfel culorile asociate vertecsilor cubului ar fi chiar culorile din colturile cubului. Iar coordonatele pentru fragmentele generate in cadrul fatetelor cubului ar rezulta din interpolarea acelor vectori (de aceea folosim varyingul v_coord).

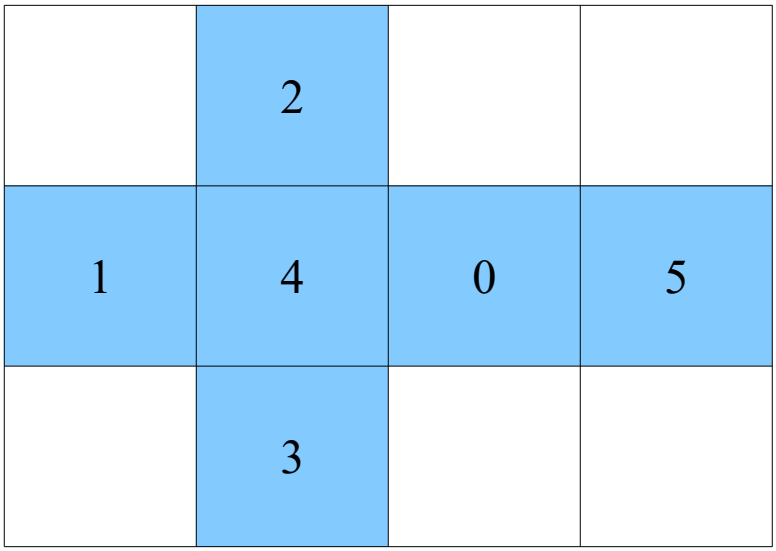
Pentru crearea skyboxului vom folosi:

- modelul SkyBox.nfg
- textura envMap.tga

Se observa faptul ca avem doar un fisier pentru textura *cube*. Prin urmare, fatetele vor trebui sa fie "decupate" din textura mare. Consideram W si H ca fiind latimea si inaltimea texturii intregi. Atunci, de exemplu, pentru fateta 2 vom avea width_fateta=W/4, height_fateta=H/3, iar pentru setul de texeli va fi obtinut astfel:

- pentru primul rand se sare w/4 texeli
- pentru al doilea rand se sare W+w/4 texeli
- etc.

! Atentie la dimensunea unui texel in vectorul dat de loadTGA (poate fi de 3 octeti sau de 4 in functie de bpp).



Cu ajutorul schemei de mai sus puteti identifica mai usor de care constanta tine fiecare fateta, de exemplu:

0 - GL_TEXURE_CUBE_MAP_POSITIVE_X,
1 - GL_TEXURE_CUBE_MAP_NEGATIVE_X
...etc.

Pasi de urmat:

Clasa Skybox si modificari in xml-uri

1) Se creeaza o clasa numita SkyBox derivata din SceneObject.
2) In sceneManager.xml se defineste un obiect nou de tip skybox. In pozitie va fi setat initial doar y-ul. Se va modifica parsarea XML-ului astfel incat sa creeze un obicet de tip SkyBox.
3) Se creeaza fisierele vs si fs ale skybox-ului (si se adauga in resourceManager.xml si se trece id-ul shader-ului in sceneManager.xml). Puteti porni de la shaderele pentru obiectele obisnuite, si apoi sa le modificati. Initial in vs se calculeaza doar pozitia vertecsilor (inmultind pozitia din local space cu MVP), iar in fs e setata o culoare constanta – pentru testare). Se verifica faptul ca se afiseaza cubul. Apoi se scaleaza (in mod egal pe cele 3 axe) astfel incat sa cuprinda terenul si alte obiecte din scena (de exemplu scalare 1000). Daca in departare pare taiat, mariti dimensiunea proprietatii *far* a camerei.

Incarcarea texturii de tip *cube map*:

1) Se adauga in resourceManager textura pentru skybox (atentie, textura va fi de tip **cube** nu 2d). Pentru skybox, min/mag filter vor fi LINEAR si wrap-urile, CLAMP_TO_EDGE.
2) In load-ul clasei Texture (daca nu s-a adaugat deja) se va adauga un if care testeaza tipul texturii. Se defineste un Gluint pentru tip care va avea in functie de caz valorea GL_TEXTURE_2D, respectiv GL_TEXTURE_CUBE_MAP. Deschiderea texturii si setarea parametrilor texturii se vor modifica in sensul ca nu se mai trece direct tipul GL_TEXTURE_2D ci cel obtinut din XML.
3) Pentru textura cube va fi un cod special (pus in interiorul unui *if*, evident), dupa cum urmeaza:
4) Se creeaza cele 6 zone de textura cu glTexImage2d pentru cele 6 fatete: POSITIVE_X, NEGATIVE_X etc. Se va seta dimensiunea fiecarei subimagini la width-ul si heightul unei fatete. Avem doua variante de a umple bufferul texturii:
   a) fie se creeaza un char* subbuffer de dimensiunea unei fatete, in care se copiaza de fiecare data din textura mare doar fateta corespunzatoare. Apoi acest subbuffer este transmis functiei glTexImage2d (care il copiaza) si este rescris pentru fiecar fateta
   b) fie se seteaza cele 6 fatete (POSITIVE_X, NEGATIVE_X, ..etc) folosind glTexSubImage2D (ordinea o gasiti mai sus). Se seteaza pointerul din pixelArray. Se copiaza cate un rand din textura punand height-ul 1 si offsetul subimaginii chiar numarul randului la care s-a ajuns.
5) Din vertex shader se calculeaza interpolarea coordonatelor cubului din local space (se va seta un varying v_coord=a_posL) – acestea vor fi setate drept coordonate de textura pentru cube map
6) In fragment shader se calculeaza culoarea provenind din textura:
   culoare=textureCube(u_cube_texture, v_coord); //bineinteles, e posibil ca textura voastra sa aiba alt nume, de exemplu u_texture0 etc.

Modificari in SceneObject (sau in clasa Skybox daca decideti ca nu veti mai avea obiecte cu aceasta proprietate)

1) Se va implementa optiunea de following camera. Nu trebuie sa putem iesi din skybox (pe x, respectiv pe z). Ca sa realizam asta si, in plus, sa dam senzatia de departare (obiectele aflate la mare departare par sa se deplaseze odata cu noi cand ne miscam), vom deplasa skybox-ul dupa

camera. Pentru asta o sa implementam optiunea following camera din xml. Adaugati pentru skybox codul:

```
<followingcamera>
    <ox/>
    <oz/>
</followingcamera>
```

Prin asta cerem ca obiectul sa urmareasca camera pe ox si oz. Vom adauga in sceneObject o proprietate Vector3 (de exemplu numit followingCamera care va avea 1 pentru axele precizate in xml (in cazul de fata: x si z) si 0 pentru restul (in cazul de fata y).

2) In functia Update se va verifica daca obiectul are proprietatea followingCamera, fie verificand daca toate componetele vectorului sunt nule sau nu, fie printr-o variabila booleana (care indica daca exista proprietatea)
3) Se va modifica proprietatea position a obiectului astfel incat coordonatele following sa fie egale cu cele ale camerei. Aditional se poate specifica si un offset in pozitia initiala din xml:

```
<position>
    <x>12</x> <!-- e de fapt offsetul pe x daca obiectul are following Camera pe axa OX -->
    <y>5</y>
    <z>0</z>
</position>
```

Pentru skybox insa o sa punem 0 pe x si z. Coordonata y va fi setata astfel incat partea de culoare uniforma din josul skyboxului (e o culoare mov) sa ajunga la linia orizontului (fata de teren)

4) Din noul position reiese matricea de translatie (daca ati facut taskurile de pana acum aceasta parte e deja implementata).