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).

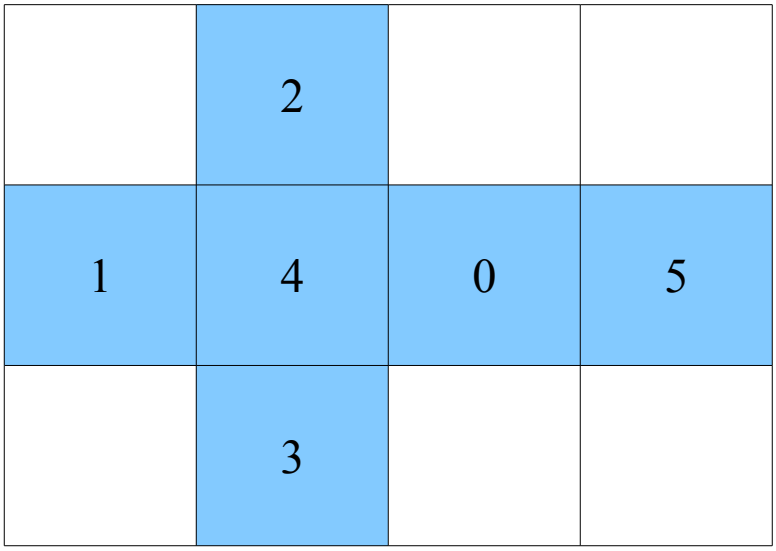
Pentru crearea skyboxului vom folosi:

- 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 w=W/4, h=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.