Generarea modelului terenului

Terenul in mod particular fata de alte obiecte va avea urmatoarele proprietati in plus (care se vor regasi si in fisierul de configurare):

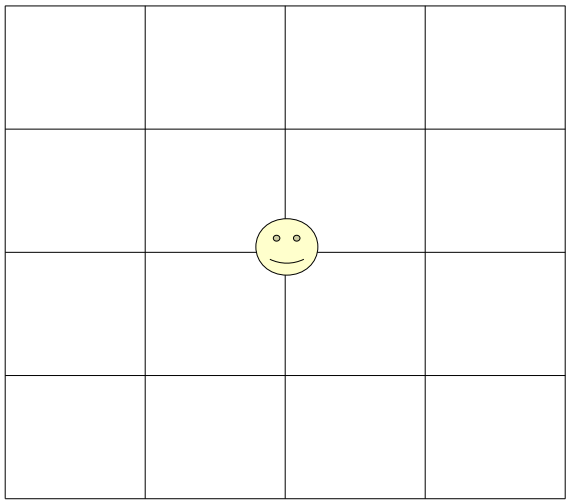
-numar celule pe orizontala

-numar celule pe verticala (de obicei egal cu numarul de celule pe orizontala, deci cele doua proprietati pot fi inlocuite de una singura: **nr_celule**)

-**dimensiunea unei celule (o vom nota dimCel)**

-un **offsetY** (explicat mai jos)

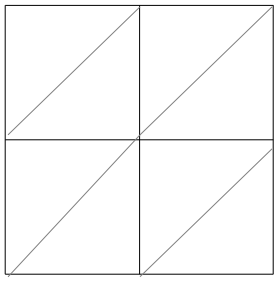
-La crearea terenului, acesta se va genera centrat fata de camera, asa cum se vede in imagine (in exemplu fiind un teren de 4*4 celule):



Terenul va fi paralel cu planul xOz si va avea un y prestabilit (**offsetY**).

In generarea vertecsilor terenului trebuie sa avem in vedere urmatoarele:

- va fi o grila dreptunghiulara de vertecsi
- in OpenGL ES nu avem quad-uri ca primitive, deci va trebui sa impartim patratelele din grid in triunghiuri, de exemplu:



-pozitiile initiale ale vertecsilor depind de pozitia camerei, deci putem face in doua moduri:

- Fie consideram centrul terenului ca fiind originea (coordonatele 0,0,0) si generam vertecsii fata de aceasta origine si translatam terenul cu centrul in pozitia camerei (doar pentru coordonatele x si z), iar y-ul sa fie egal cu **offsetY**.
- Fie (si solutia asta mi se pare mai usor de implementat si administrat) consideram de la inceput centrul terenului egal cu pozitia camerei (pentru coordonatele x si z) si y-ul egal cu **offsetY**, si generam pozitiile vertecsilor fata de acest centru asa cum am fi facut si mai sus.

Apoi generam si indicii corespunzatori, observand ca pentru fiecare quad avem cate 2 triunghiuri, deci 6 indici. O implementare simpla ar fi sa face doua for-uri care parcurg quad-urile (de ce 2 for-uri? Pentru ca putem considera ca avem o matrice de quaduri, deci parcurgem liniile si coloanele acestei matrici).

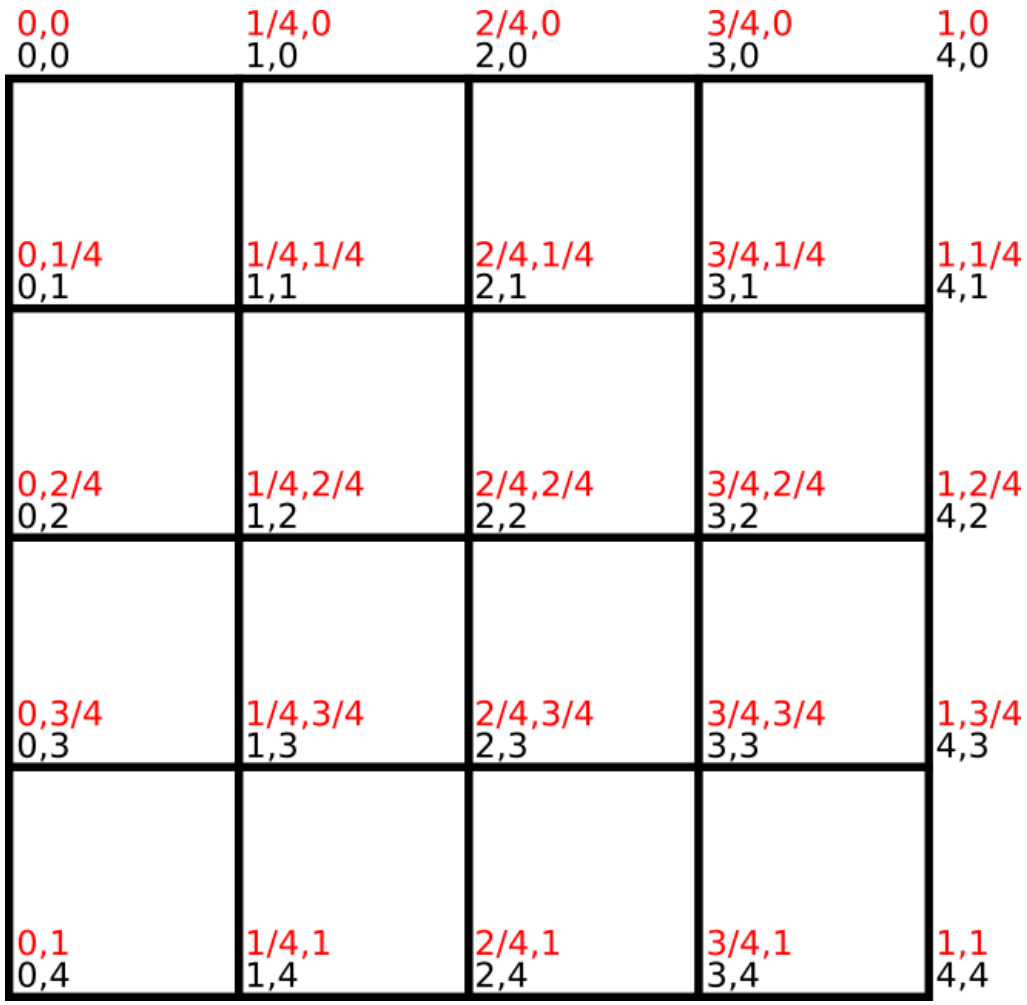
- pe teren se vor aplica 3 texturi de relief (Rock.tga, Dirt.tga, Grass.tga), fiecare din aceste trei texturi aplicandu-se in acelasi timp pe fiecare dintre celulele terenului => GL_REPEAT
- pe teren se va mai aplica o textura de amestec (Terrain_blend_map.tga). Aceasta va determina in fiecare fragment al terenului care textura se aplica dintre cele 3 mentionate mai sus. Pe textura de amestec se observa zonele rosii, verzi si albastre. Vom asocia, cu ajutorul fisierului de configurare cate o culoare fiecarei texturi. De exemplu pentru rosu avem pietris (rock), pentru albastru avem pamant (dirt) si pentru verde avem iarba (grass). Pentru a evita if-urile si pentru a putea trata usor si zonele de tranzitie intre culorile din textura de blending, vom folosi formula urmatoare:

```
c_final= c_blend.r*c_rock+c_blend.g*c_grass+c_blend.b*c_dirt;
c_final.a=1.0;
```

Unde, c_rock e culoarea fragmentului, preluat din textura rock, c_grass, culoarea din textura grass si c_dirt, culoarea din textura dirt.

Texturile rock, grass si dirt se aplica pe patratelele gridului (cu repeat), pe cand textura de blend se aplica pe tot gridul. Prin urmare o sa avem doua uv-uri: uv-ul pentru texturile mici, care e la fel pentru toate in cadrul unei patratele de grid, deoarece toate cele 3 texturi se aplica la fel pe un patratel; si uv-ul blend_map-ului, care se aplica pe intreg gridul.

De exemplu, pentru un grid 4X4 uv-urile vor arata asa (cu rosu uv-ul blend-map-ului si cu negru uv-urile texturilor mici):



La deplasarea camerei, trebuie sa avem senzatia de teren infinit. Astfel daca ne deplasam pe x sau pe z cu mai mult decat dimensiunea unei patratele trebuie “sa se genereze” un rand sau coloana noua in grid in sensul deplasarii (ca sa avem in continuare teren in fata pe care sa ne deplasam) si sa “se stearga” ultimul rand sau coloana in sensul opus deplasarii (ca sa nu avem un teren din ce in ce mai mare pe masura ce ne deplasam). Adaugarea si stergerea de randuri/coloane e de fapt fictiva pentru ca in final trebuie sa ramanem cu un grid cu acelasi numar de vertecsi si aceeasi dimensiune. Asa ca, practiv vom simula adaugarea si stergerea prin deplasarea terenului cu *d*=dimensiune_celula in sensul miscarii si shiftarea in sens opus a texturii de blend(de amestec) cu 1/nr_celule.

De exemplu, sa presupunem ca ne-am deplasat suficient de mult in dreapta (pe OX sensul pozitiv) astfel incat am depasit fata de centrul curent al terenului pe orizonatala distanta *d*. In acest moment deplasam tot gridul la dreapta, cu acel *d*, dar, ca sa nu observe utilizatorul ca s-a deplasat terenul, mutam in stanga textura de blend cu 1/nr_celule (scazand practic pentru toti vertecsii u cu 1/nr_celule). Daca punem GL_REPEAT si pentru textura de blend, nu vom aveam probleme cu u si v care ies din intervalul [0,1].

Generarea reliefului

Vom folosi blend_map-ul si pe post de height map. Astfel fiecare culoare (rosu, verde, albastru) va reprezenta o anumita inaltime (de exemplu 3,2,-1). Aceste inaltimi vor fi preluate din fisierul de configurare. De exemplu, partea corespunzatoare din xml poate arata asa:

```
<inaltimi>
	<r>15</r>
	<g>5</g>
	<b>-5</b>
</inaltimi>
```

Evident, aplicarea inaltimilor se va face in vertex shader. Se va prelua culoarea din textura pentru fiecare vertex si se va aplica o formula asemanatoare cu cea pentru aplicarea mai multor texturi.

Presupunand ca a_posL este atributul pentru pozitie, formula ar fi:

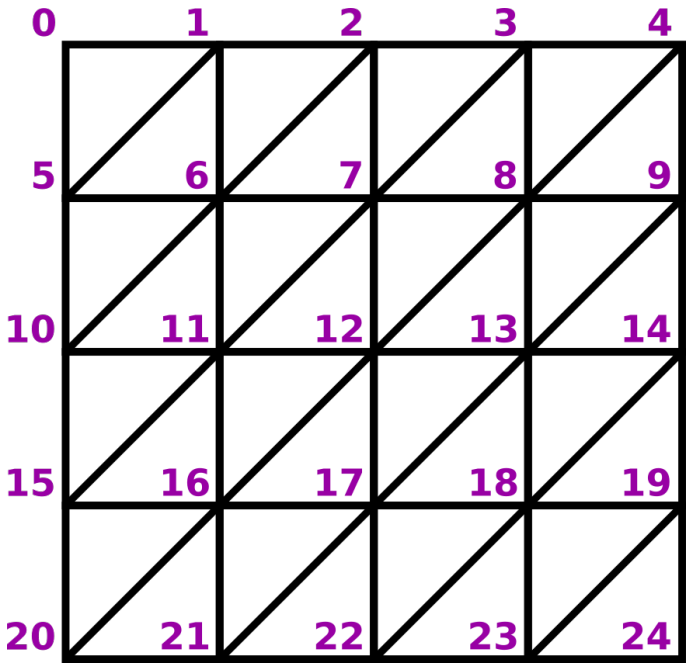
```
pos_nou = a_posL
pos_nou.y += c_blend.r*u_height.r+c_blend.g*u_height.g+c_blend.b*u_height.b
```

Unde height e un vec3 cu cele 3 inaltimi preluate din xml.

Pasi de urmat

Generarea modelului terenului

1) Consideram notat cu N numarul de celule pe o latura (presupunem N par si terenul patratic) si D dimensiunea laturii unei celule. Se va crea o metoda numita generateModel() a terenului, care va crea un mesh patratic, de de N*N celule. Pentru asta va fi nevoie sa se creeze un vector de vertecsi si un vector de indici. Vor fi (N+1)*(N+1) vertecsi.
2) Pentru vertecsi se genereaza coordonatele astfel incat centrul patratului sa fie in originea sistemului de coorodonate. Strict pentru observarea usoara a terenului, il vom genera intai pe verticala (in planul XoY) ca sa il putem gasi usor in scena. Consideram coordonatele x si y pornind de la -N/2 *D pana la N/2 * D si crescand din D in D. Setam un z astfel incat terenul sa se afle in fata camerei.
3) Generam indicii terenului, avand in vedere ca fiecare celula a terenului e impartita in 2 triunghiuri. Deci pentru fiecare celula, se vor genera cate 6 indici. Reamintim faptul ca indicii trebuie sa fie de tip GLushort. De exemplu pentru 4x4 celule vom avea indicii din imagine:



Astfel, pentru exemplul de mai sus, pentru primul triunghi, indicii vor fi 0, 1, 5, urmati de 1, 6, 5 etc. Ultimii vor fi 18, 19, 23, 19, 24, 23. Gasiti o formula pentru cei 6 indici corespunzatori fiecarei celule – aceasta trebuie sa depinda de numarul liniei si coloanei fiecarei celule. Parcurgand celulele din "matrice" generati indicii,adaugandu-i in vectorul cu elemente de tip GLushort.

4) Dupa ce ati creat vetorii de vertecsi si de indecsi, creati bufferele de tip GL_ARRAY_BUFFER si GL_ELEMENT_ARRAY_BUFFER in care veti copia vertecsii, respectiv indicii (asa cu ati facut in load-ul modelelor).
5) Copiati shaderele (vs si fs) de la obiectele obisnuite si redenumiti-le (de exemplu, le puteti da numele terrainShader). Deocamdata vrem sa vedem ca modelul este generat corect, astfel in fragment shader vom da tuturor fragmentelor o culoare constanta (de exxemplu, albastru) pentru a verifica faptul ca apare corect obiectul pe ecran.

Adaugarea texturii

1) Dupa ce am vazut patratul corespunzator terenului, e momentul sa ii adaugam si texturile. Le adaugam in resourceManager.xml (Grass.tga, Dirt.tga, Rock.tga si Terrain_blend_map.tga). Toate vor avea filtrele min si mag GL_LINEAR si wrap-urile de tip GL_REPEAT. Adaugam id-urile texturilor in sceneManager.xml pentru obiectul de tip teren.
2) Asa cum s-a vazut in partea de teorie avem doua uv-uri. Putem proceda in doua moduri (alegeti care vi se pare mai comod):
   a) Ori adugam un camp numit uv2 in structura Vertex, si transmitem acest nou atribut catre shader (fie in Draw-ul terenului fie in sendCommonData()). In acest caz ocupam mai multa memorie, insa putem refolosi campul uv2 si in alte cazuri cand dorim sa aplicam mai multe texturi cu coordonate diferite, pe acelasi obiect.
   b) Ori ne folosim de faptul ca daca uv.x=ind, atunci uv2.x=ind/N. Astfel transmitem N ca variabila de tip uniform si facem impartirea in shader. Totusi aceasta tehnica, desi mai eficenta pe acest caz, merge strictin aceasta situatie particulara, fiindca s-a nimerit sa avem o relatie intre cele doua coordonate de textura.
3) Pentru cei care nu si-au pregatit inca proiectul astfel incat sa se poata transmite mai multe texturi catre shader, urmam pasii descrisi in pdf-ul cu incarcarea modelelor:
   a) Pentru locatia texturii, in loc sa avem un singur *GLuint textureUniform*, vom avea un vector de locatii, pentru toate texturile pe care dorim sa le transmitem.
   Vom defini o constanta MAX_TEXTURES in fisierul Shader. Pentru acest proiect putem seta constanta la valoarea 5 (dar puteti schimba mai incolo). Astfel vom defini un vector de locatii pentru variabilele de tip uniform care vor tine texturile: *textureUniform[MAX_TEXTURES]*. Vom considera ca in shader, le vom denumi dupa urmatoarea conventie: u_texture_*i*, unde i este numarul texturii. De exemplu: u_texture_0, u_texture_1 etc.
   b) Pentru a lua locatiile acestor variabile, vom folosi functia glGetUniformLocation (codul trebuie adaugat in functia Load din clasa Shader). Putem fie sa hardcodam, avand in vedere ca avem putine texturi, fie sa facem un for si pentru numele variabilelor din shader sa facem o concatenare: "u_texture_" +std::to_string(i). Concatenarea dureaza mai mult (la executie) dar se scrie mai usor, scrierea de mana a fiecarui rand se executa mai repede, dar e mai greu de administrat daca vrem sa crestem numarul de texturi.
   c) In sendCommonData() (care transmite datele catre shader) vom avea un for care va itera prin texturile obiectului. Pentru fiecare textura va seta textura activa (cu glActiveTexture) la valoarea GL_TEXTURE*i* (putem sa ne folosim in for de faptul ca GL_TEXTURE*i*= GL_TEXTURE0+i). Apoi deschidem textura (cu glBindTexture)pentru a face legatura dintre ea si locatia i din VRAM. Apoi transmitem valoarea i catre locatia variabilei

u_texture_i cu ajutorul functiei glUniform1i.

4) In vertex shader, asa cum avem deja creat un varying pentru coordonatele v_uv, facem unul si pentru v_uv2 (fie obtinut din atributul a_uv2, fie prin impartirea lui a_uv la numarul de celule)
5) In fragment shader preluam culoarea din textura de blend in c_blend (de la coordonatele v_uv2), si apoi culorile pentru rock, dirt si grass, de la coordonatele v_uv, punandu-le in variabilele c_rock, c_dirt, c_grass. Cu ajutorul formulei din teorie, folosim c_blend drept filtru. Si punem culoarea obtinuta in c_final. Setam componenta *alpha* la *1.0* astfel incat sa nu avem vreo zona transparenta in teren. Setam gl_FragColor sa aiba valoarea c_final.
6) Dupa ce observam aplicarea texturii, aducem terenul din pozitie verticala in pozitie orizontala, interschimband coordonatele y si z, pentru vertecsii terenului.
7) Atentie: initial nu folositi scalare pentru teren (aceasta trebuie sa fie 1.0 pentru toate coordonatele, altfel veti avea probleme la generarea terenului la infinit).

Aplicarea height map-ului

1) Adaugam in xml-ul terenului, inaltimile (asa cum s-a explicat in teorie), si le parsam. In functia Draw a terenului se vor trimite sub forma unui uniform (in shader vom considera variabila
   uniform vec3 u_height;
   Nu uitati sa declarati pentru ea o locatie in clasa Shader, si in Load-ul shaderelui sa preluati locatia.
2) In vertex shader, preluam culoarea vertexului din textura de blend (pe care o vom folosi ca height map). Vom folosi de data asta coordonatele din a_uv (din atribut, deoarece suntem in vertex shader, nu lucram inca cu fragmente). Notam cu c_blend culoarea preluata.
3) Tot in vertex shader, inainte de a inmulti cu matricea MVP, copiem a_posL (vec3-ul cu coordonatele vertexului) intr-un vec4 pos_nou, setand coordonata w la 1.0 (deoarece e punct). Apoi folosind formula din teorie, schimbam y-ul lui pos_nou astfel incat sa se schimbe in functie de culoarea din c_blend.
4) Inmultim MVP cu pos_nou

Generarea terenului la infinit

1. Pornim de la urmatoare presupunere. Numarul de celule e par, iar in local space-ul terenului (coordonatele initiale ale vertecsilor, inainte de inmultirea cu matricea model) centrul terenului are coordonatele 0,0. Va trebui calculata pozitia lui (din care rezulta matricea de translatie) astfel incat centrul terenului sa fie sub camera (x si z egale cu coordonatele corespunzatoare camerei, iar y-ul un pic mai mic astfel incat terenul sa fie mai jos de camera).
2. Presupunem ca deja am inceput sa ne deplasam prin scena. Ne intereseaza deplasarile pe x si pe z. In update-ul terenului vom face urmatoarea verificare.
   a) Notam cu dx distanta dintre coorodonatele x dintre centrul terenului si pozitia camerei. Daca dx este mai mare decat dimensiunea unei celule, inseamna ca e momentul sa “generam” o noua fasie de teren pe x (asa cum ati vazut in sectiunea de teorie, asta de fapt inseamna deplasarea terenului). Daca x-ul camerei e mai mare decat x-ul centrului terenului, inseamna ca s-a depasit in dreapta dimensiunea unei celule, deci trebuie sa transalatam terenul in dreapta pe o distanta egala cu dimCel. Daca x-ul camerei e mai mic decat x-ul centrului terenului, inseamna ca trebuie mutat terenul in stanga cu dimCel
   b) Facem acelasi lucru si pentru z .
3. Pentru a da senzatia ca suntem in acelasi loc pe teren “tragem” textura in sens opus. De exemplu, daca am mutat terenul in dreapta, textura se va muta in stanga. Veti modifica acum coordonatele uv pentru blend map, adunand/scazand (in functie de sens) o valoare egala cu 1/nr_celule pe x-ul respectiv y-ul uv-ului. (de exemplu, daca ati mutat terenul la dreapta, veti scadea din componenta x a tuturor uv-urilor 1/nr_celule; daca terenul se muta la stanga, operatia

era de adunare).