

12/7/2015

FACULTATEA
DE
AUTOMATICA SI
CALCULATOARE

ELEMENTE DE GRAFICA PE
CALCULATOR



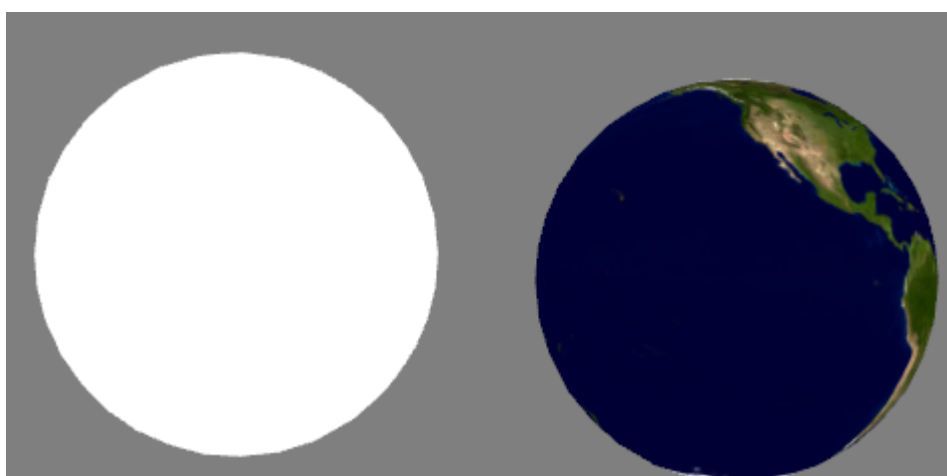
Laborator 9

Texturi

Introducere

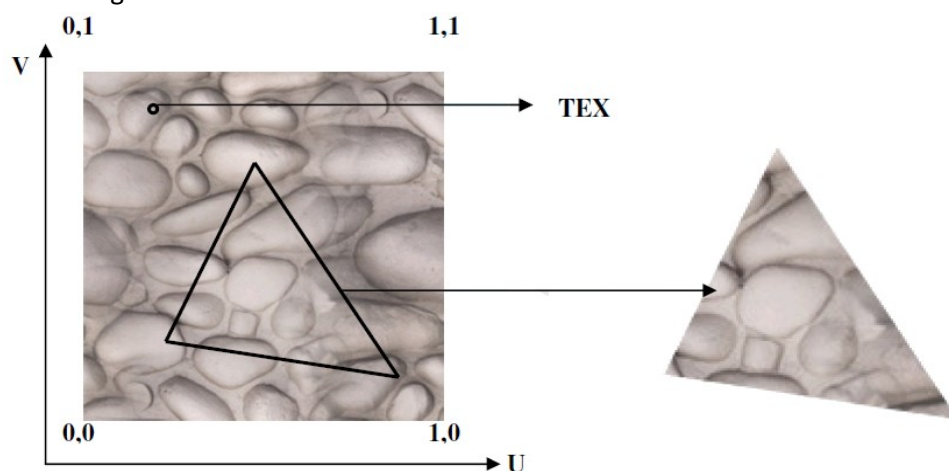
Maparea texturilor permite "lipirea" unei imagini pe un poligon. Maparea corecta a texturii asigura functionarea corecta in momentul in care poligonul transformat este afisat. Desi de obicei texturile sunt mapate pe poligoane, texturile pot fi aplicate pe toate tipurile de primitive: puncte, linii, poligoane, obiecte definite functional, etc.

Diferenta intre un obiect texturat si acelasi obiect netexturat este enorma din punct de vedere al acuratetei reprezentarii obiectului respectiv:



Maparea texturilor

Maparea texturilor se face placand de la conceptul de coordonate de textura. Cu niste coordonate definite de artistul care a creat obiectul in cauza putem lega o imagine de geometria desenata, exact ca in imaginea urmatoare.



Dupa cum se poate vedea, imaginea este parametrizata in spatiul $[0,1] \times [0,1]$ in care sunt definite coordonatele de mapare.

Deși există suficiente metode de generare funcțională a coordonatelor de textură, acestea nu vor fi prezentate în laborator, dar vom menționa faptul că nici una din aceste metode nu este aplicabilă la toate tipurile de topologie pentru care ne-am dori să generăm automat coordonate de textură.

Un element $dx \times dy$ din spațiul $[0,1] \times [0,1]$ se numește texel, numele venind de la texture element.

Construcție și utilizare

Pentru a construi o textură în OpenGL avem nevoie în primul rând de pixelii imaginii ce va fi folosită ca textură. Aceștia trebuie ori generați funcțional ori trebuie încărcați dintr-o imagine, iar acest task este independent de OpenGL. În codul de laborator folosim încărcare de imagini din format BMP datorită faptului că este un format comun și ușor de parsat. Există mai multe librării de încărcare de imagini, inclusiv încărcare automată a imaginilor ca și texturi de OpenGL precum Corona, DevIL, etc.

După ce avem pixelii imaginii încărcate putem genera o textură de OpenGL cu comanda:

```
unsigned int gl_texture_object;  
glGenTextures(1, &gl_texture_object);
```

Similar cu toate celelalte procese din OpenGL nu lucrăm direct cu textura ci trebuie să o mapăm la un punct de legare. Mai mult, la rândul lor punctele de legare pentru texturi sunt dependente de unitățile de texturare. O unitate de texturare este foarte similară ca și concept cu pipe-urile pe care trimitem atribute.

Setăm unitatea de texturare cu comanda (o singură unitate de texturare poate fi activă):

```
glActiveTexture(GL_TEXTURE0+nr_unitatii_de_texturare_dorite);
```

Iar pentru a lega obiectul de tip textură generat anterior la unitatea de textură activă folosim:

```
glBindTexture(GL_TEXTURE_2D, gl_texture_object);
```

Pentru a încărca date într-o textură folosim comanda:

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
```

Comanda încarcă o imagine definită prin un array de unsigned char-uri pe textură legată la punctul de legare GL_TEXTURE_2D al unității de texturare curent, nivelul 0 (o să luăm această constantă ca atare pentru moment), cu formatul intern GL_RGB cu lungimea width și cu înălțimea height, din formatul GL_RGB. Datele citite sunt de tip GL_UNSIGNED_BYTE (adică unsigned char) și sunt citite de la adresa data.

Pentru a folosi o textură în shader trebuie să urmărim acest proces:

- Trebuie activată unitatea de texturare pe care dorim să o folosim, cu comanda:
`glActiveTexture(GL_TEXTURE0+unitate_texturare);`
- Trebuie legată textura pe care dorim să o folosim la unitatea de texturare
`glBindTexture(GL_TEXTURE_2D, textura);`
- Trebuie găsită locația în shader folosind numele din shader al texturii
`unsigned int locatie = glGetUniformLocation(gl_program_shader, "textura");`
- Trebuie trimis pe această locație numărul unității de texturare pe care e legată textura
`glUniform1i(locatie, unitate_texturare);`

Urmatorul shader este un exemplu de shader ce poate folosi legarea precedenta, unde texcoord reprezinta coordonatele de texturare primite ca attribute in vertex shader si apoi pasate catre rasterizator pentru interpolare:

```
#version 330
layout(location = 0) out vec4 out_color;
uniform sampler2D textura1;
in vec2 texcoord;
void main(){

    out_color = texture(textura1, texcoord).xyz;
}
```

Exista anumite cazuri in care o textura nu este suficienta, acest efect fiind numite multitexturare. De exemplu putem folosi acest efect in momentul in care reprezentam o un obiect topologic complex , sa spunem ca niste frunze, cu o topologie mult inferioara ca si complexitate, ca de exemplu un quad. Daca utilizam o asemenea reducere de complexitate trebuie sa avem o metoda prin care se putem elimina la nivel de fragment, fragmentele ne-necesare (sunt evidente in imaginea urmatoare).



Puntru aceasta putem sa folosim o textura de opacitate(alpha) care ne spune care sunt fragmentele reale ale obiectului. Combinatia de textura de opacitate si textura de culoare este suficienta pentru definirea acestui bambus:



ALPHA/OPACITATE

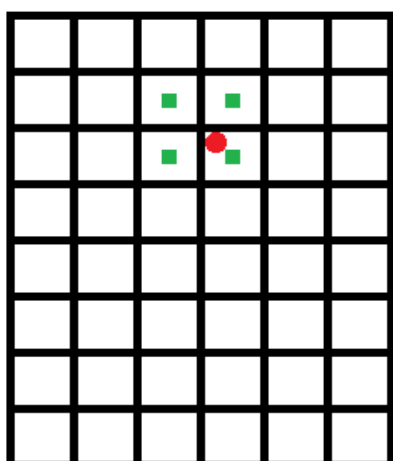
CULOARE

Filtrare

Pentru a utiliza o textura nu este suficient sa citim doar imaginea si sa o incarcam ca un obiect de tip textura ci sa o si filtram. Ce inseamna filtrare? Teoria sistemelor ne spune ca este o metoda de esantionare si reconstrucie pe un semnal. Semnalul este realitatea (reala sau virtuala!) prinsa in poza iar pixelii imaginii rezultate reprezinta esantioanele pe care le stocam din aceasta realitate.

Reconstrucia reprezinta procesul prin care utilizand acesti pixeli putem obtine valori pentru oricare din pozitiile in textura (adica nu neaparat exact la coordonatele din mijlocul pixelului, acolo unde a fost esantionata realitatea in spatiul post proiectie).

Cea mai simpla metoda de filtrare e de a alege cel mai apropiat texel (sample de realitate) relativ la coordonata de texturare unde ne dorim sa reconstruim realitatea. Cand folosim filtrarea nearest alegem tot timpul cel mai apropiat punct de esantionare si ii folosim valoarea ca rezultat pentru citirea de la coordonatele de texturare initiale. In imaginea de mai jos am folosi valoarea texelului dreapta jos dintre texeli cu centrele colorate cu verde.



coordonat de
texturare
centru texel

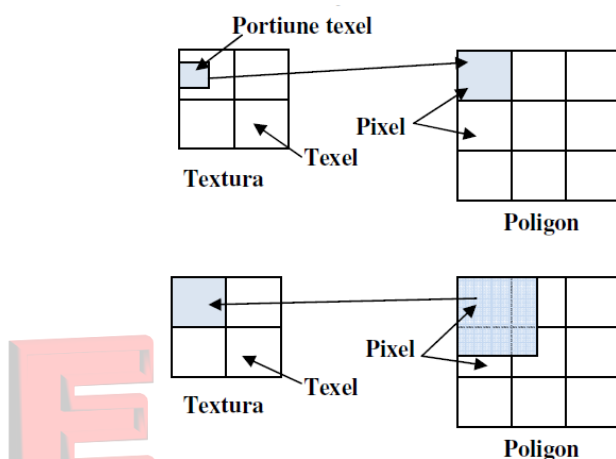
Putem face filtrare nearest cu comenzile:

```
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST );
```

In cazul in care vom folosi filtrare biliniara, vom folosi cei 4 cei mai apropiati texeli si vom interpola liniar intre perechile sus si jos si apoi liniar intre rezultatele obtinute. Astfel vom avea tot timpul continuitate in spatiul culorii. Putem face filtrare bilineara cu comezile:

```
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

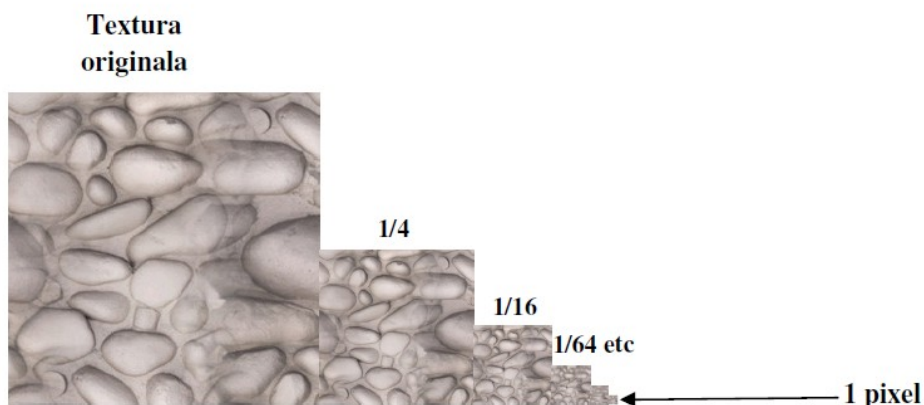
Totusi, nu tot timpul texelii din imagine se vor mapa perfect pe pixelii ecranului. Aproape tot timpul vom avea un raport diferit de 1. Putem observa acest fenomen in imaginea urmatoare:



Motivatia pentru un algoritm mai bun este urmatoarea: daca suntem in situatia in care raportul texeli/pixeli este subunitar, adica am mai multi pixeli pentru un singur texel sunt in pozitia in care fara algoritmi foarte avansati de reconstructie predictiva nu am solutie. Dar daca sunt in situatia in care am raportul texeli/pixeli supraunitar atunci inseamna ca am mai multi texeli pe acelasi pixel si se pune intrebarea : cum calculam culoare pixelului? Prin medie? Prin interpolare? Mai mult, trebui sa consideram si performanta: de exemplu putem citi cate 500 de texeli per pixel daca geometria pe care mapam textura e mica!

De aceea introducem un algoritm care rezolva toate aceste probleme de filtrare: filtrul trilinear. Atunci cand incarc textura dintr-o imagine folosim functia:

```
glGenerateMipmap(GL_TEXTURE_2D); se creeaza urmatoare structura de memorie (numita mipmap):
```



Atunci cand folosesc filtrare trilineara banda grafica determina pe baza raportului texeli/pixeli (1 e ideal!) care sunt cele 2 cele mai apropiate nivele din mipmap. Se face filtrare biliniara pentru citirea din amandoua nivelele alese iar apoi se face filtrare liniara intre rezultatele obtinute. Putem face filtrare biliniara cu setarea:

```
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);  
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
```

Chiar si aceasta metoda are probleme atunci cand privim geometria suport (pe care e mapata textura) la unghiuri extreme, dupa cum se poate observa in imaginea urmatoare.



Daca in apropiere filtrarea de textura are o calitate similara in distanta mipmapurile duc la un fenomen de blur datorita interpolarii. Ce se intampla? Problema este ca la distanta intra foarte multi texeli pe acelasi pixel DAR modul in care intra e unul particular atunci cand lucram la un unghi. In cazul acesta foarte multi texeli de pe o SINGURA directie intra in pixel in timp ce putini de pe cealalta directie conteaza. Acest fenomen se numeste ANISOTROPIC din cauza faptului ca nu este ISOTROPIC, adica nu toate axele conteaza la fel de mult in rezultatul final.

Pentru a rezolva si aceasta problema, deci pentru a obtine rezultatul din imaginea din dreapta de mai sus folosim filtrarea anisotropica, prin urmatoarele comenzi:

```
float maxAnisotropy;  
glGetFloatv(GL_MAX_TEXTURE_MAX_ANISOTROPY_EXT, &maxAnisotropy);  
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAX_ANISOTROPY_EXT, maxAnisotropy);
```

Mai mult, atunci cand adresam textura la niste coordonate de texturare precum 1.1 sau -0.5 avem mai multe optiuni. Putem considera ca exista o margine dupa care toate valorile sunt egale cu aceasta margine, iar acest comportament de limitare il numim „clamping”. Practic, in acest caz $1.1 = 1$ si $-0.5 = 0$; Il putem seta cu comenzile:

```
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP );  
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP );
```

In acelasi timp putem considera ca dupa ce trec de marginile texturii aceasta se repeta, deci avem $1.1 = 0.1$ si $-0.5 = 0.5$. Acest comportament de repetare il putem seta cu comenzile:

```
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT );  
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT );
```

Observatie: exista multe alte subiecte care nu au fost atinse ce tin tot de texture: alinierea in memorie, conversiile implicite din diferite formate, compresia, filtrarea functional, etc, deci daca intampinati o problema consultati si familiarizati-va cu documentatia!

