## **Încărcarea modelelor**

### **Încărcarea datelor în buffer**

În taskul cu desenarea pătratului s-a observat repetarea datelor unor vertecși. Această redundanță încarcă inutil memoria plăcii grafice. Putem reutiliza vertecșii, specificându-i în buffer o singură dată. Reutilizarea se realizează cu ajutorul unui buffer de indici care specifică ordinea vertecșilor în cadrul primitivelor. Practic indicii sunt ca niște identificatori pentru vertecși.

Vom folosi un *index buffer*. Astfel vom avea doua tipuri de buffere:

* Cel pentru vertecși, de tip GL\_ARRAY\_BUFFER
* Cel pentru indici, de tip GL\_ELEMENT\_ARRAY\_BUFFER

Modelele pe care le vom încărca se găsesc în folderul NewResourcesPacket -> Models. Sunt fișierele cu extensia nfg. Această extensie nu este una standard, deci nu veți putea folosi o bibliotecă pentru parsarea fișierelor (va trebui să vă definiți voi propria funcție de citire).

În aceste două buffere vor fi puși vectorii obținuți în urma parsării fișierului nfg (care conține vectorul de vertecși și cel de indici).

### **Desenarea**

Pentru a desena cu ajutorul bufferului de indici (care specifică ordinea vertecșilor), vom folosi *glDrawElements* in loc de *glDrawArrays*.

glDrawElements( mod\_preluare\_vertecsi, numar\_indici, tip\_date\_indici, pointer\_locatie\_indici);

Explicație paramteri:

* mod\_preluare\_vertecsi - Specifică tipul de primitive (forma geometrică) ce va fi desenată. Poate fi:
  + GL\_TRIANGLES pentru triunghiuri individuale.
  + GL\_TRIANGLE\_STRIP pentru o bandă de triunghiuri legate.
  + GL\_TRIANGLE\_FAN pentru un evantai de triunghiuri.
  + Alte opțiuni includ GL\_LINES, GL\_LINE\_STRIP, etc.
* numar\_indici - Numărul total de indici din buffer pe care dorim să-i folosim pentru desenare. (practic câți vertecși vor fi extrași pentru desenare)
* tip\_date\_indici - Tipul de date al indicilor din buffer. De obicei, este GL\_UNSIGNED\_BYTE sau GL\_UNSIGNED\_SHORT, specificând că indicii sunt fie de tip GLubyte, fie de tip GLushort. **Noi vom folosi GL\_UNSIGNED\_SHORT**.
* pointer\_locatie\_indici - Un pointer către locația unde se află indicii în memoria GPU. În cazul în care avem un buffer de indici bind-uit folosind glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, iboID), pointer\_locatie\_indici reprezintă un offset (o distanță în bytes) de la începutul bufferului de indici , nu un pointer direct la bufferul de indici.

Deci in cazul nostru vom avea:

glDrawElements(GL\_TRIANGLES, nr\_indici, GL\_UNSIGNED\_SHORT, 0);

**Observație:** Am pus 0 pe ultimul parametru deoarece, avand datele stocate in buffer, ultimul parametru capata rol de offset si nu de locatie de unde sa ia datele, și noi dorim să ia datele de la începutul bufferului.

Cand facem apelul *glDrawElements*, ambele buffere (si cel de vertecși și cel de indici) trebuie să fie deschise cu ajutorul funcție glBindBuffer.

Vedeti si:

* <https://www.khronos.org/opengles/sdk/docs/man/xhtml/glDrawElements.xml>
* <http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-9-vbo-indexing/> (ce e scris acolo e valabil si pentru OpenGL ES)

### **Incarcarea texturilor**

**Texturile** sunt fișiere imagine (jpg, png, tga, etc.) care se încărca și aplică ("înfășoară) pe suprafața obiectului 3D. Ni le putem imagina ca pe niște abțibilduri pe care le putem redimensiona si din care putem decupa ce avem nevoie ca sa putem lipi imaginea rezultata pe o față a obiectului. Procesul de aplicare a unei texturi se numește mapare a texturii.

Texturile nu sunt folosite doar pentru aplicarea culorii pe obiecte. Există mai multe tipuri de texturi:

* **Texturi de culoare**: folosite pentru definirea culorii vizibile a obiectului.
* **Texturi de normale (Normal Maps)**: folosite pentru adăugarea de detalii subtile prin simularea denivelărilor sau a adânciturilor pe suprafață.
* **Texturi de reflexie/Speculare (Specular/Gloss Maps)**: folosite pentru crearea efectului de luciu sau reflexie.
* **Texturi de transparență (Alpha Maps)**: folosite pentru definirea părților transparente din textura aplicată.

Obiectul 3D este cartografiat cu coordonate de textură, denumite coordonate u,v. Acestea sunt puncte care leagă locații specifice de pe textura 2D de vertecși de pe obiectul 3D. Coordonatele u,v sunt parametrice (valori între 0 și 1), unde (0, 0) este colțul din stânga jos al texturii, iar (1, 1) este colțul din dreapta sus.

Pentru a specifica ce portiune de textura e asociata unei fete de pe suprafata modelului, se asociaza fiecarui vertex câte un set de **coordonate u,v** din intervalul [0,1] (pot fi si din afara acestui interval insa vor fi tratate in mod special, asa cum e explicat mai jos). Coordonatele u,v pentru fragmente sunt calculate prin interpolare. Intervalul [0,1] este mapat pe latimea si, respectiv, inaltimea imaginii-textura. Astfel, punctul de culoare specificat prin coordonatele u,v subunitare va fi egal cu **texelul** din imagine de la coordonatele (u\*lungime,v\*inaltime).

Pentru a genera un id de textura:

*glGenTextures(nr\_texturi, locatie\_in\_care\_sa\_fie\_stocat\_id-ul/id-urile)*

Exemplu:

*glGenTextures(1, &id\_textura);*

Ca si in cazul bufferelor, textura trebuie "deschisa" pentru a putea face setarile necesare asupra ei.

Tipul de textura cu care vom lucra deocamdata este GL\_TEXTURE\_2D

Deci instructiunea va fi:

*glBindTexture(GL\_TEXTURE\_2D,id\_textura);*

Se vor seta apoi parametrii texturii:

1. Cum se comporta textura in cazul in care trebuie marita/micsorata pentru a se potrivi pe suprafata modelului.

Proprietatile sunt:

* GL\_TEXTURE\_MIN\_FILTER
* GL\_TEXTURE\_MAG\_FILTER

Valorile posibile sunt:

* GL\_LINEAR
* GL\_NEAREST
* Pentru MIN mai sunt si niste proprietati care tin de mipmapping

Exemplu:

*glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);*

1. Felul in care se comporta textura pentru coordonate u,v in afara intervalului [0,1].

Proprietatile sunt:

* GL\_TEXTURE\_WRAP\_S (asociat coordonatei u)
* GL\_TEXTURE\_WRAP\_T (asociat coordonatei v)

Valorile posibile sunt:

* GL\_CLAMP\_TO\_EDGE
* GL\_REPEAT
* GL\_MIRRORED\_REPEAT

Exemplu:

*glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);*

Se creeaza imaginea (ca un fel de buffer) de tip textura cu functia:

*glTexImage2D(tip\_textura, nivel\_mipmap, format\_intern\_textura, latime, inaltime, border=0, format\_pixel\_data, mod\_memorare\_pixel\_data, pointer\_vector\_pixeli)*

Parametrii sunt obtinuti din proprietatile fisierul textura calculate cu functia LoadTGA (din TGA.h, proiectul Utilities).

Exemplu:

*glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, width, height , 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, pixelArray);*

Pentru desenare se va transmite catre shader un intreg de tip uniform, care va reprezenta indicele texturii. Cand avem o singura textura, acesta este 0, altfel ar trebui sa corespunda indicelui dat in functia *glActiveTexture*. Unui model ii pot fi atribuite mai multe texturi, caz in care vor trebui pe rand legate de indici (distincti, evident) de textura activa, pentru a putea fi folosite in shader.

Exemple:

*glActiveTexture(GL\_TEXTURE0);* //nu e necesar acest apel daca avem o singura textura, deoarece implicit textura activa e GL\_TEXTURE0.

*glActiveTexture(GL\_TEXTURE3);*//in acest caz va trebui transmis 3 catre shader (vezi mai jos).

Dupa un apel glActiveTexture(GL\_TEXTURE***i***), se realizeaza un glBindTexture(tip\_textura, idt), pentru a lega textura cu id-ul *idt* de indicele de textura GL\_TEXTURE***i***.

Apoi se transmite indicele de textura activa catre shader cu un glUniform1i:

*glUniform1i(textureUniform,i);*

In cazul in care avem o singura textura, instructiunea va fi:

*glUniform1i(textureUniform,0);*

**Lucrul cu texturi in shader**

In **vertex shader** trebuie doar sa ne asiguram ca se va realiza interpolarea coordonatelor uv:

*v\_uv=a\_uv;* // unde a\_uv este un vec2, atributul corespunzator coordonatelor uv asociate vertexului, iar v\_uv este un varying, tot vec2.

In **fragment shader** se va folosi variabila de tip uniform care identifica textura, insa cu un tip special de date: sampler2D:

*uniform sampler2D u\_texture;*

Culoarea fragmentului va fi calculata cu ajutorul functiei *texture2D()*:

*gl\_FragColor = texture2D(u\_texture, v\_uv);*

## Pași detaliați de rezolvare

### Încărcarea unui model (deocamdată fără textură)

1. Se deschide un fișier cu extensia nfg din NewResourcesPacket/Models și se observă structura lui. În zonele care explică sintaxa unui rând, ce este între paranteze drepte e variabil, iar restul textului e constant.
   1. Pe primul rând este indicat numărul de vertecși (îl vom nota cu NV). Sintaxa este:

NrVertices: [NV]

* 1. Urmează NV rânduri indexate, fiecare rând corespunzător unui vertex. Sintaxa este:

[indice vertex]. pos:[trei numere raționale cu coordonatele vertexului]; norm:[trei numere raționale cu coordonatele normalei]; binorm:[trei numere raționale cu coordonatele binormalei]; tgt:[trei numere raționale cu coordonatele tangentei]; uv:[două numere raționale cu coordonatele de textură];

* 1. După cele NV rănduri de vertecși încep indicii. Numărul lor (notat cu NI) e dat de rândul:

NrIndices: [NI]

* 1. Urmează NI rânduri indexate. Pe fiecare rând sunt câte trei indici corespunzători vertecșilor de mai sus. Sunt dați în grupuri de câte trei, pentru a evidenția triunghiurile modelului. Sintaxa unui rând este:

[numar rând]. [primul indice], [al doilea indice], [al treilea indice]

1. Se va completa structura Vertex din Vertex.h cu proprietățile de tip Vector3 *norm*, *binorm*, *tgt* și proprietatea Vector2 *uv*.
2. În funcția **Init()** din **NewTrainingFramework.cpp** se vor crea doi [vectori STL](https://cplusplus.com/reference/vector/vector/) unul care va conține vertecșii (deci elementele vor fi de tip **Vertex**) și unul care va conține indicii (cu elemente de tip **unsigned short**).
3. Veți crea o funcție **readNfg(caleNfg, vectorVertecsi, vectorIndecsi)** care primește calea către nfg, o referință către vectorul de vertecși și o referință către cel de indecși. Funcția va citi fișierul nfg si va popula cei doi vectori cu datele din fișier. Pentru a construi vectorul de vertecși, e posibil să aveți nevoie să definiți un constructor de copiere în cadrul clasei Vertex, deoarece e apelat de metoda push\_back(). Funcția va fi apelată în **Init()** din **NewTrainingFramework.cpp**. Puteți alege pentru început modelul cu crocodilul: Croco.nfg.
4. Se va crea bufferul de vertecși pentru model:
   1. veți crea un id de buffer nou (cu glGenBuffers) salvat în variabila modelVboId (de tip GLuint)
   2. folosind funcția glBindBuffer, veți deschide (și asocia) un buffer de tip GL\_ARRAY\_BUFFER cu id-ul modelVboId
   3. cu glBufferData veți seta datele bufferului preluându-le din vectorul de vertecși citit din fișier. Atenție la cum dați dimensiunea bufferului; nu va mai merge cu sizeof(vector) deoarece acesta e un obiect din clasa vector. Trebuie să înmulțiți numărul de vertecși cu dimensiunea unui vertex. De asemenea, veți folosi metoda data() pentru a oferi adresa de unde încep datele.
   4. închideți bufferul cu glBindBuffer(GL\_ARRAY\_BUFFER, 0)
5. Se va crea bufferul de indici pentru model:
   1. veți crea un id de buffer nou (cu glGenBuffers) salvat în variabila modelIboId (de tip GLuint)
   2. folosind funcția glBindBuffer, veți deschide (și asocia) un buffer de tip GL\_ELEMENT\_ARRAY\_BUFFER cu id-ul modelIboId
   3. cu glBufferData veți seta datele bufferului preluându-le din vectorul de indici citit din fișier. Ca și pentru vertecși, pentru a seta dimensiunea bufferului, trebuie să înmulțiți numărul de indici cu dimensiunea unui indice (unsigned short). De asemenea, veți folosi metoda data() pentru a oferi adresa de unde încep datele.
   4. închideți bufferul cu glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, 0)
6. Crearea shader-ului pentru model:
   1. creați un vertex shader si un fragment shader speciale pentru model. Puteți porni de la shaderele pentru triunghi, doar că nu mai interpolați culoarea (**setați în fragment shader direct culoarea verde**). Veți intra cu Windows Explorer în folderul Resources/Shaders unde creați cele două fișiere (le puteți numi modelShaderVS.vs și modelShaderFS.fs). Apoi în Visual Studio, în filtrul Shaders le adăugați cu Add -> Existing Item
   2. Atenție, în vertex shader trebuie să înmulțiti cu MVP pozițiile vertecșilor
   3. creați o variabilă de tip Shaders numită modelShader. Apelați în funcția Init() din NewTrainingFramework.cpp, metoda Init() a obiectului modelShader cu căile către cele două fișiere shader.
7. În funcția Draw() veți comenta desnarea liniei și a triunghiului/pătratului (adică de la apelul glUseProgram() până la apelul glDrawArrays()). Veți adăuga codul pentru desenarea modelului..
   1. veți folosi shader-ul pentru model (glUseProgram)
   2. veți deschide bufferul de vertecși al modelului, cu glBindBuffer(), folosind modelVboId dar și bufferul de indici al modelului, cu glBindBuffer(), folosind modelIboId
   3. veți transmite datele pentru model. Veți verifica pentru fiecare variabilă din shader dacă locația e diferită de -1 (în acest caz ar trebui sa fie doar poziția vertecșilor) și veți transmite datele cu glVertexAttribPointer()
   4. veți desena linia cu glDrawElements() folosind modul de desenare GL\_TRIANGLES, și trimițând un numărul de indici citit din fișier, tipul de date GL\_UNSIGNED\_SHORT și pointerul pentru locația indicilor să fie 0.
8. **Debugging**. Dacă nu vedeți modelul:
   1. afișați vectorii de vertecși și de indici (fie în watch cu un breakpoint, fie în consolă cu std::cout) și verificați măcar primele 3 și ultimele 3 elemente că au fost citite corect din fișier.
   2. verificați că setați o culoare constantă în fragmentShader (în gl\_FragColor)
   3. verificați în output că nu s-au afișat erori de shader
   4. verificați că planul **far** al camerei este suficient de departe cât să cuprindă coordonatele modelului. Setați planul far la o valoare mai mare, măcar 1000.

### Aplicarea texturii

1. În funcția **Init()** din **NewTrainingFramework.cpp** veți defini variabilele width, height, bpp de tip int și o variabilă pixelArray de tip char\*. Variabilele width și height vor fi folosite pentru a memora dimensiunea unei texturi. Variabila bpp este o prescurtare de la "bits per pixel" și poate fi 24 (pentru canalele R,G,B) sau 32 (dacă în plus culoarea are și canalul ALPHA). Veți apela metoda LoadTGA (vedeți TGA.h) și valoarea returnată de funcție va fi pusă în *pixelArray*. Parametrii width, height, bpp, vor fi trimiși prin adresă pentru a fi calculați de LoadTGA.
2. Se va crea bufferul de textură pentru imaginea citită (tot în funcția Init()):
   1. veți crea un id nou de textură (cu glGenTextures) salvat în variabila *idTexture* (de tip GLuint)
   2. folosind funcția glBindTexture(), veți deschide (și asocia) un buffer de tip GL\_TEXTURE\_2D cu id-ul *idTexture*
   3. cu *glTexImage2D* veți seta datele texturii deschise de tip GL\_TEXTURE\_2D cu valorile:
      1. 0 pentru nivelul de mipmap
      2. GL\_RGB (pentru bpp de valoare 24) sau GL\_RGBA(pentru bpp de valoare 32), atăt pentru parametrul corespunzător formatului intern al texturii cât și pentru formatul datelor pixelilor (cum vor fi stocați în buffer).
      3. Argumentele pentru lățime și înălțime vor fi date prin variabilele width și height.
      4. Argumentul border va fi setat la 0.
      5. Penultimul argument care indică modul de reprezentare în memorie a culorilor, va fi setat la GL\_UNSIGNED\_BYTE (câte un octet pentru fiecare canal de culoare).
      6. Iar ultimul argument al funcției (care reprezintă o adresă către conținutul efectiv al imaginii) va avea valoarea variabilei *pixelArray*
3. Setați parametrii texturii folosind funcția *glTexParameteri()*:
   1. pentru GL\_TEXTURE\_WRAP\_S și GL\_TEXTURE\_WRAP\_T setați valoarea GL\_CLAMP\_TO\_EDGE
   2. pentru GL\_TEXTURE\_MIN\_FILTER și GL\_TEXTURE\_MAG\_FILTER setați valoarea GL\_LINEAR
4. Închideți bufferul de textură cu glBindTexture(GL\_TEXTURE\_2D, 0)
5. Pentru a desena textura trebuie să trimitem către shader coordonatele uv (vec2) și textura. Coordonatele uv au fost citite din fișierul modelului doar pentru vertecși, deci trebuie interpolate pentru a fi calculate pentru fiecare fragment în parte. În **modelShaderVS.vs** veți defini un atribut de tip vec2, numit a\_uv și un varying tot de tip vec2, numit v\_uv. În funcția main veți atribui lui v\_uv pe a\_uv, pentru a interpola coordonatele.
6. În fragment shader, adică **modelShaderFS.fs** veți defini un varying-ul v\_uv de tip vec2 și un uniform de tip sampler2D numit u\_texture (care va conține textura). Definiți o variabila vec4 numită *color*. Apelați funcția *texture2D* care primește ca parametri textura și coordonatele și returnează culoarea găsită în textură la acele coordonate, setând variabila *color* la această valoare returnată. În gl\_FragColor veți seta valoarea din *color*.
7. Deoarece avem un attribute și uniform nou, trebuie să le preluăm locațiile și să le trimitem datele. Vom defini în Shaders.h variabilele textureUniform și uvAttribute. În Shaders.cpp vom prelua locațiile pentru uniform-ul u\_texture și attribute-ul a\_uv cu funcțiile glgetUniformLocation și glGetAttribLocation
8. În funcția **Draw()** din **NewTrainingFramework.cpp** verificăm dacă locația texturii (textureUniform) e diferită de -1, caz în care cu funcția glUniform1i trimitem valoarea 0. Trimitem această valoare deoarece avem o singură textură și aceasta e încărcată implicit în GL\_TEXTURE\_0. Trimitem cu glVertexAttribPointer valorile pentru coordonatele uv (atenție la parametrul offset; peste câți octeți trebuie să săriți ca să ajunga în memorie la proprietatea uv?)
9. Rulăm și vedem dacă apare textura. Dacă nu, verificați întâi următoarele:
   1. calea pentru textură este corectă? verificați folosind un breakpoint dacă aveți valorile corecte în width și height
   2. verificați interpolarea corectă a valorilor uv. în fragment shader comentați linia curentă pentru gl\_FragColor și scrieți gl\_FragColor =vec4(v\_uv, 0.0, 1.0). Rezultatul ar trebui să fie un gradient aplicat pe model (deoarece coordonatele uv sunt între 0 și 1 ca și componentele culorii, și își schimbă valorile gradual de la un fragment la altul vecin).
   3. după apelul glDrawElements, scrieți int err=glGetError() și verificați valoarea variabilei err. Dacă e diferită de 0 înseamnă că s-a dat o valoare invalidă de parametru într-una din funcțiile OpenGL, deci verificați din nou toate funcțiile.
10. După ce se afișează modelul, veți observa că pare că puteți vedea prin el și că nu arată natural. Motivul este că testul de adâncime e implicit oprit și desenează fragmentele în ordinea procesării, suprascriind pixelii indiferent care se află în față pe axa OZ. Pentru a avea o afișare naturală, trebuie în funcția Init() să apelați glEnable(GL\_DEPTH\_TEST) care va activa și bufferul de adâncime. În funcția Draw() va trebui, așa cum ștergeți conținutul bufferulul de culoare să ștergeți și conținutul bufferului de adâncime: glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT). Observați operatorul "sau" binar, care activează astfel ambele flag-uri în masca de biți.
11. Veți observa că solzii crocodilului apar pe o bandă neagră. Acest lucru se datorează punctelor de culoare din textură cu alpha 0 care în continuare sunt afișate, deoarece implict nu se ține cont de canalul alpha. Pentru eficiență, vom da discard fragmentelor cu alpha foarte mic. Prin urmare, în fragment shader, în funcția main() veți verifica (înainte de setați gl\_FragColor) dacă proprietatea *a* a lui *color* e mai mică decât 0.1, caz în care folosiți instrucțiunea discard:

if (color.a<0.1) discard;

această instrucțiune face ca fragmentul să nu mai fie desenat.

# Linkuri utile

1. **Tutoriale vectori STL**
   1. <https://www.mygreatlearning.com/blog/vectors-in-c/>
   2. <https://www.programiz.com/cpp-programming/vectors>
2. **Explicații constructor de copiere**: <https://www.geeksforgeeks.org/copy-constructor-in-cpp/>