

## Despre codurile sursa de la laborator

### Generalitati

- **Despre LoadShaders.cpp**

- Permite afisarea erorilor de compilare.
- Poate fi instalata o extensie a MVS care sa indice eventuale erori de sintaxa in shadere

<https://marketplace.visualstudio.com/items?itemName=DanielScherzer.GLSL>

- Este recomandat ca functia care o utilizeaza - de exemplu createShaders ( ); sa fie apelata in functia de initializare.

- **Despre createVBO( )**

- Apelata in functia de initializare. Daca nu functioneaza, glBindBuffer( ) trebuie apelata inainte de functia de desenare.

- Structura:

- vectori cu varfuri, indici
- Generare nume ptr. buffer-objects: [glGenBuffers\( \)](#)
- Transfer date in buffer: [glBufferData\( \)](#)
- "Legare buffer" (eventual apelata inainte de functia de desenare):  
[glBindBuffer\( \)](#)
- Activarea lucrului cu attribute, indicarea locatiilor – vor fi utilizate in shader-ul de varfuri: [glEnableVertexAttribArray\( \)](#);  
[glVertexAttribPointer\( \)](#)

## Laborator 2

- **02\_01\_primitive.cpp + 02\_01\_shader.frag**
  - utilizarea unei singure culori pentru o primitiva in OpenGL "nou"
  - utilizarea variabilelor uniforme pentru "comunicarea" cu shader-ele
  - despre GLSL si shadere (detalii in [specificatiile GLSL](#)):
    - variabile si tipuri de variable (inclusiv vectori si matrice)
    - variabile: stocare (in / out /uniform)
    - calcule (operatii cu matrice) si decizii (if, switch,etc.)
  - folosirea GL\_POINT\_SMOOTH pentru reprezentarea punctelor
- **02\_02\_fata\_spate\_poligon.cpp**
  - fata si spatele poligoanelor (triunghiuri);
  - utilizarea GL\_CULL\_FACE pentru a "inlatura" fata/spatele poligonului
- **02\_03\_poligoane3D.cpp**
  - Patratele sunt desenate folosind GL\_TRIANGLE\_FAN, si GL\_QUADS, varfurile au culori diferite
  - Nu sunt indicati parametri pentru vizualizare / decupare, fiind selectate valorile implicite
  - utilizarea functiei de "mouse" glutMouseFunc

## Laborator 3

- **03\_01\_animatie\_OLD.cpp ("OpenGL vechi")**

- gluOrtho2D (indica dreptunghiul care este decupat) - DEPRECATED
- glTranslate, glRotate, glPushMatrix, glPopMatrix (ptr. transformari; DEPRECATED)
- glutSwapBuffers (v. GLUT\_DOUBLE); glutPostRedisplay; glutIdleFunc (animatie)

- **03\_02\_animatie.cpp ("OpenGL nou")**

- utilizeaza diverse transformari si compunerea acestora folosind [biblioteca glm](#). Aceasta biblioteca este deja disponibila in template.
- functii pentru utilizarea mouse-ului glutMouseFunc ( );

- **03\_03\_resize.cpp**

- pentru a stabili o fereastră de "decupare" într-o scenă 2D putem folosi atât funcția `glm::ortho`, cât și indicarea explicită a transformărilor
- în exemplu este decupat dreptunghiul delimitat de `xmin`, `xmax`, `ymin`, `ymax`

- **03\_04\_rotire.cpp**

- compunerea transformărilor, realizarea unei rotații cu centrul diferit de origine
- utilizarea `GL_QUADS` pentru desenarea unui dreptunghi

- **03\_05\_transformari\_keyboard.cpp**

Realizarea unei scene 2D în care obiectele se mișcă

- unele primitive rămân fixe, altele își schimbă poziția
- funcții pentru tastatură: `processNormalKeys`, `processSpecialKeys`
- pentru animație: `glutIdleFunc`

## Laborator 4

- **04\_02\_indexare.cpp**

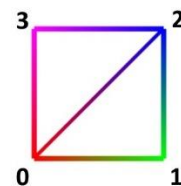
Indexarea varfurilor:

- folosirea indexarii varfurilor: elemente asociate (matrice, buffer) sunt indicate in `CreateVBO()`;
- in acest exemplu atat coordonatele varfurilor, cat si culorile sunt indicate in acelasi buffer, ca sub-buffere (tipul de buffer este `GL_ARRAY_BUFFER`) ;
- indicii sunt indicati intr-un vector, varfului  $m$  ii corespunde indicele  $m$ , ordinea indicilor precizeaza ordinea in care sunt considerate varfurile pentru a realiza desenarea (tipul de buffer este `GL_ELEMENT_ARRAY_BUFFER`);
- desenarea se face folosind functia [`glDrawElements\(\)`](#) in cazul folosirii indexarii exista si alte functii, de exemplu [`glDrawRangeElements\(\)`](#);
- in codul exemplu sunt patru varfuri, coordonatele si indexarea sunt indicate in figura:

```
// Coordonatele varfurilor;
static const GLfloat Vertices[] =
{
    -15.0f, -15.0f,  0.0f,  1.0f,
    15.0f,  -15.0f,  0.0f,  1.0f,
    15.0f,   15.0f,  0.0f,  1.0f,
    -15.0f,  15.0f,  0.0f,  1.0f
};

// Culorile ca atribut ale varfurilor
static const GLfloat Colors[] =
{
    1.0f, 0.0f, 0.0f, 1.0f,
    0.0f, 1.0f, 0.0f, 1.0f,
    0.0f, 0.0f, 1.0f, 1.0f,
    1.0f, 0.0f, 1.0f, 1.0f
};

// Indicii care determina ordinea de
static const GLuint Indices[] =
{
    0, 1, 2, 3, 0, 2
};
```



- **04\_03a\_douaVAO.cpp, 04\_03b\_douaVBO.cpp**

- Folosirea a doua VAO (Vertex Array Objects), respectiv a doua VBO (Vertex Buffer Objects) pentru a desena doua obiecte diferite. Primul VAO (respectiv VBO) contine informatii (coordoanatele varfurilor, culori, indicii) despre primul obiect, cel de-al doilea VAO (respectiv VBO) informatiile referitoare la cel de-al doilea obiect.
- Inainte de desenarea propriu-zisa, trebuie „legat” in mod corespunzator VAO-ul, respectiv VBO-ul folosit. In cazul a doua VAO se foloseste `glBindVertexArray()`, iar in cazul a doua VBO se foloseste `glBindBuffer()`, in cazul coordonatelor/culorilor/etc (elemente care vor deveni *vertex attributes* in shader) este imediat apelata si `glVertexAttribPointer()`.

- **04\_04\_texturare.cpp**

- Utilizarea texturilor.
- Folosirea unor functii de amestecare in shader-ul de fragment.
- Functii pentru reperul de vizualizare (`glm::lookAt`) si pentru proiectii (`glm::ortho`).

## Texturare

- Folosirea unei biblioteci dedicate (de exemplu SOIL – Simple OpenGL Image Library) permite incarcarea rapida a unor texturi din fisiere avand formate standard, precum JPEG, PNG, etc.
  - Fisierul `SOIL.h` este utilizat ca fisier de tip header in proiect.
  - Template-ul pus la dispozitie are integrate aceasta biblioteca (fisierele lib aferente).
- In functia `CreateVBO()`, pentru fiecare varf sunt indicate coordonatele de texturare aferente (trebuie sa existe o coerenta intre coordonatele varfurilor si modul de alegere a coordonatelor de texturare). Aceste coordonate au asociata o locatie specifica, urmand sa devina *vertex attribute* (in shader-ul de varfuri). In codul sursa `04_04_texturare.cpp` coordonatele de texturare au locatia 2.
- Functia `LoadTexture()` contine elementele necesare generarii, legarii, incarcarii texturii, precum si precizarea proprietatilor acesteia ([`glTexParameteri`](#)). Nu trebuie uitata eliberarea memoriei si realocarea. Aceasta functie poate fi apelata la initializare.
- In functia de desenare/randare: textura trebuie activata/legata folosind functiile `glActiveTexture()` si `glBindTexture()`. Ulterior, trebuie transmisa shader-ului de fragment ca variabila de tip uniform (exemplu: `glUniform1i(glGetUniformLocation(ProgramId, "myTexture"), 0)`).
- Comunicare cu shader-ele:
  - Shader-ul de varfuri: i se transmit, pe langa coordonatele varfurilor si culori, coordonatele de texturare (v. attributele); ca output sunt si pozitia si culoarea si coordonatele de texturare.
  - Shader-ul de fragmente: are ca date de intrare atat informatiile transmise de shader-ul de varfuri, cat si textura – folosind o variabila uniforma (`uniform sampler2D`). Se poate folosi functia `mix` pentru a “combina” culoarea sau diferite texturi.

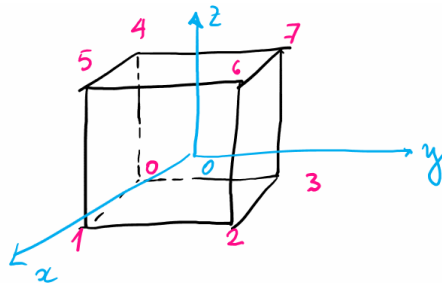
## Laborator 7

- 07\_01\_desenare\_cub.cpp

- Diverse tipuri de proiectii.
- Folosirea indexarii pentru a trasa separat fetele si muchiile unui obiect 3D (cub)
- Rolul testului de adancime

### Comentarii

- Coordonatele varfurilor si indicii. Varfurile 0, 1, 2, 3 sunt verzi, iar varfurile 4, 5, 6, 7 sunt rosii. Observatorul este la inceput in punctul (0, 0, 300) si se uita inspre punctul (0, 0, -100), deci vede fata rosie a cubului.



- Testul de adancime este esential pentru a reprezenta corect un obiect 3D.

- 07\_02a\_instanced\_rendering.cpp

- Sunt desenate mai multe instante ale aceluiasi obiect, fiecare dintre ele avand culorile proprii ale varfurilor si o pozitie proprie. **Culorile si pozitiile instantelor sunt calculate in programul principal.**

- In createVBO() apar o serie de elemente specific randarii instantiate:

- Coordonatele varfurilor sunt indicate separat de culorile/matricele de pozitie ale instantelor. Pentru fiecare instanta este precizata pozitia, pe o curba de forma  $(r(t) \cdot \cos(t), r(t) \cdot \sin(t))$ ; in plus, fiecare instanta este rotita (in spatiu).

```
MatModel[instID] =
    glm::translate(glm::mat4(1.0f), glm::vec3(90 * instID * cos(instID), //
                                                90 * instID * sin(instID), //
                                                0.0))
    * glm::rotate(glm::mat4(1.0f), (instID + 1) * PI / 10, glm::vec3(1, 0, 0))
    * glm::rotate(glm::mat4(1.0f), instID * PI / 6, glm::vec3(0, 1, 0))
    * glm::rotate(glm::mat4(1.0f), 3 * instID * PI / 2, glm::vec3(0, 0, 1));
```

- O functie specifica randarii instantiate este [glVertexAttribDivisor\(\)](#). Aceasta indica rata cu care are loc distribuirea atributelor per instanta (de testat cazul cand al doilea parametru este 0, 2, 5, INSTANCE\_COUNT).
- In cazul atributului "pozitie a instantei" trebuie tinut cont ca este indicata o matrice 4x4, sunt alocate 4 atribute, corespunzator celor 4 coloane.

## • 07\_02b\_instanced\_rendering.cpp

- Sunt desenate mai multe instante ale aceluiasi obiect, fiecare dintre ele avand culorile proprii ale varfurilor si o pozitie proprie. **Culorile si pozitiile instantelor sunt calculate in shader-ul de varfuri.**

- Pentru a accesa fiecare instanta individual este folosita variabila gl\_instanceID

- In shader nu poate fi accesata biblioteca GLM si trebuie scrise functii pentru generarea matricelor 4x4 pentru translatii / rotatii in jurul axelor de coordonate.

- Instantele sunt distribuite pe aceeasi curba ca in codul 07\_02a

```
// Generarea matricei de translatie
matrTransl=translate
(90 * instID * cos(instID), 90 * instID * sin(instID), 0.0);

// Matricea de rotatie este produsul a trei rotatii
matrRot = rotateX((instID + 1) * PI / 10)
          * rotateY(instID * PI / 6)
          * rotateZ(3 * instID * PI / 2);

// Pozitia finala a varfului corespunzator instantei procesate
gl_Position = projectionMatrix * viewMatrix * matrTransl * matrRot * in_Position;
```

## • 07\_03\_survolare\_cub.cpp

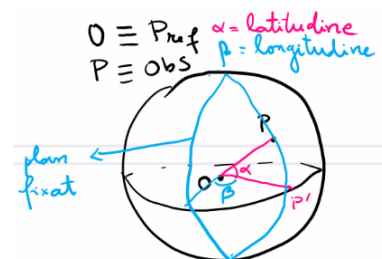
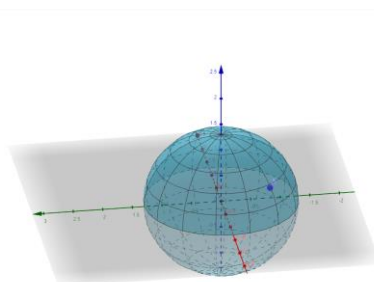
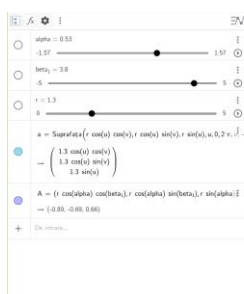
- Survolarea unui obiect folosind coordonate sferice.

### Comentarii

- Observatorul se misca pe o sfera de raza float dist (variabila) in jurul punctului de referinta (centrul cubului).

- Verticala din planul de vizualizare este (0, 0, 1).

- Este implementata reprezentarea unui punct variabil pe sfera (si implicit a sferei) folosind unghiurile alpha si beta (latitudine, respectiv longitudine).



## Laborator 8

- **08\_01\_stive\_matrice.cpp**

- Codul ilustreaza miscarea relativa a obiectelor 3D – este creat un mini-sistem solar folosind doua cuburi.
- Transformarile de modelare si cea de vizualizare sunt inglobate intr-o singura matrice;
- Folosirea stivelor de matrice;
- Utilizarea timpului scurs de la initializare;
- In **08\_01\_Shader.frag**: stabilirea culorii obiectului in functie de pozitia fragmentului

Pentru codurile urmatoare, detaliile matematice / teoretice se gasesc in fisierul [L8 suprafete.pdf](#)

- **08\_02a\_cerc.cpp, 08\_02b\_cerc\_cu\_disc.cpp**

- Trasarea unui cerc si a unui disc folosind reprezentarea parametrica.
- Folosirea functiei de callback [glutReshapeFunc](#) (relevanta pentru pastrarea proportiilor).

- **08\_03\_sfera.cpp**

- Desenarea unei sfere si survolarea acesteia.