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 atribute,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 (triunghuiuri);
  - utilizarea `GL_CULL_FACE` pentru a “inlatura” fata/spatele poligonului

- **`02_03_poligoane3D_ex1_OLD.cpp` (“OpenGL vechi”)**

  - `GL_QUADS` ca mod de trasare a primitivelor
  - functii specifice OpenGL "vechi":

```
gluLookAt
glOrtho
glMatrixMode
// sunt generate transformari pentru vizualizare 3D
glPolygonMode
```

- **`02_04_poligoane3D.cpp`**

  - Acest cod sursa este un "echivalent" 2D al codului 02_03_poligoane_3d_old.cpp
  - Patratele sunt desenate folosind `GL_TRIANGLE_FAN`, varfurile au culori diferite
  - Este realizat cu OpenGL "nou"
  - Nu sunt indicati parametri pentru vizualizare / decupare, fiind selectate valorile implicite

- **`02_05_poligoane3D_ex2_OLD` (“OpenGL vechi”)**

  - indicarea varfurilor in vectori
  - 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 fereastra de "decupare" intr-o scena 2D putem folosi atat functia `glm::ortho`, cat si indicarea explicita a transformarilor
    - in exemplu este decupat dreptunghiul delimitat de xmin, xmax, ymin, ymax

- **`03_04_rotire.cpp`**

  - compunerea transformarilor, realizarea unei rotatii cu centrul diferit de origine
  - utilizarea GL_QUADS pentru desenarea unui dreptunghi

- **`03_05_transformari_keyboard.cpp`**

  Realizarea unei scene 2D in care obiectele se misca
  - unele primitive raman fixe, altele isi schimba pozitia
  - functii pentru tastatura: `processNormalKeys, processSpecialKeys`
  - pentru animatie: `glutIdleFunc`

Laborator 4

- **`04_02_indexare.cpp`**
  Indexarea varfurilor
  - folosirea indexarii varfurilor: elemente asociate (matrice, buffer)
  - desenarea se face folosind functia `glDrawElements()`

- **`04_03_texturare.cpp`**
  - Utilizarea texturilor.
  - Folosirea unor functii de amestecare in shader-ul de fragment.
  - Functii pentru reperul de vizualizare (glm::lookAt) si pentru proiectii.

1. **Folosirea `glDrawElements`**

   - In `CreateVBO`
     - Pozitia, culoarea, etc. (atribute ale varfurilor) sunt indicate in acelasi vector.
     - Indicii corespunzatori varfurilor sunt indicati intr-un vector.
     - Se creeaza un Vertex Array Object si Buffer-e pentru atributele varfurilor si pentru indici (`glGenVertexArrays` si `glGenBuffers`); se realizeaza "legarea" (`glBindBuffer`) si "copierea" (`glBindBuffer`) acestora : atentie la diferenta intre varfuri / atributele lor si indici!.
     - Se initializeaza lucrul cu atribute (`glEnableVertexAttribArray(i)`) si se precizeaza (`glVertexAttribPointer`) cum trebuie intrepretate datele din Vertex Buffer Objects. **Important:** atributul **i** se regaseste si in shader-ul de varfuri (`location=i`).
   - In functia de desenare:
     - Se apeleaza `glDrawElements` in loc de `glDrawArrays.`
   - Eliberare memorie si realocare resurse (`DestroyVBO`, `DestroyShaders`, etc.).
   - Comunicare cu shadere-le: se transmit shader-ului de varfuri informatiile referitoare la atributele varfurilor, folosind `location` (v. mai sus).

2. **Texturi**
   - 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.
   - Functia `LoadTexture` contine elementele necesare generarii, legarii, incarcarii texturii, precum si precizarea proprietatilor acesteia (`glTexParameteri`). Nu trebuie uitata eliberarea memoriei si realocarea.
   - Comunicare cu shader-ele:

- Shader-ul de varfuri: i se transmit coordonatele de texturare (v. atributele); 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. Se poate folosi functia `mix` pentru a “combina” culoarea sau diferite texturi.

Laborator 7

- `06_02_desenare_cub.cpp`

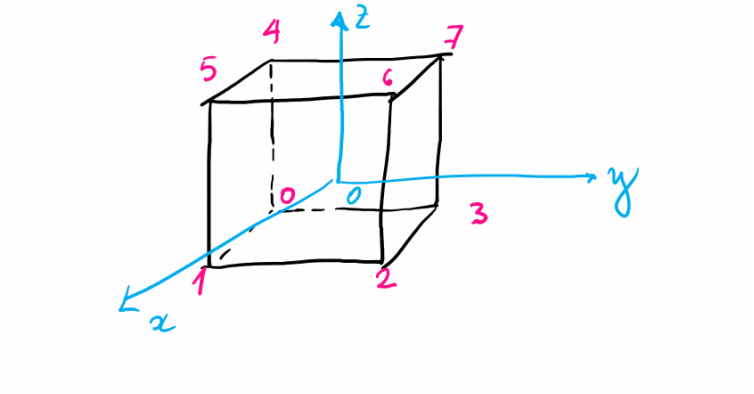
  - Diverse tipuri de proiectii.

  - 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_01_survolare_cub.cpp`

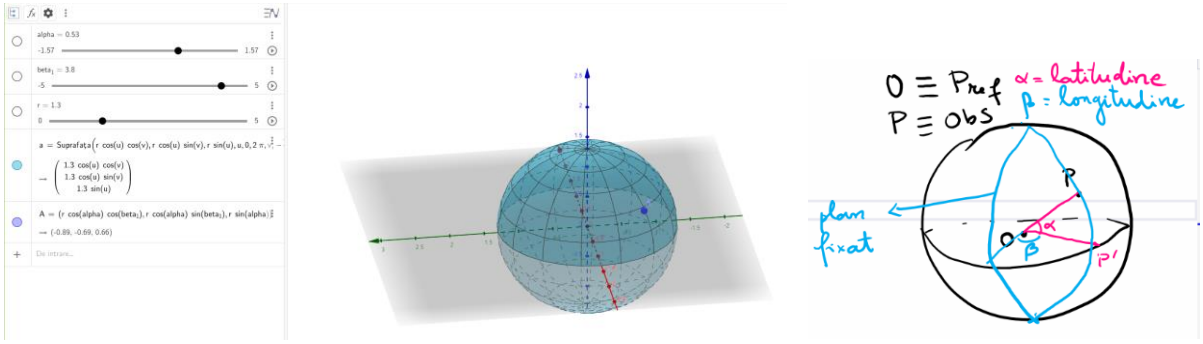
  - Survolarea unui obiect folosind coordonate sferice.

  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).



- **`07_02_instanced_rendering.cpp`**

  - Sunt desenate mai multe instante ale aceluiasi obiect, fiecare dintre ele avand culorile proprii ale varfurilor si o pozitie proprie.

  - 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)*sin(t), r(t)*cos(t)); in plus, fiecare instanta este rotita (in spatiu).

```
glm::translate(glm::mat4(1.0f),
glm::vec3(80* n * sin(10.f * n * 180 / PI), 80 * n * cos(10.f * n * 180 / PI), 0.0)) *
glm::rotate(glm::mat4(1.0f), n * PI / 8, glm::vec3(n, 2 * n * n, n / 3));
```

    - 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.

Laborator 8

Detaliile matematice / teoretice se gasesc in fisierul L8_suprafete.pdf

- **`08_02_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.

- **`08_04_obiecte.cpp`**
  - Sunt desenate doua obiecte folosind doua VAO diferite

Laborator 10

- **`10_01_iluminare.cpp`**
  Aplicarea modelului de iluminare in cazul unui cub
  - Normalele sunt calculate la nivelul fetelor
  - Din programul principal sunt transferate in shader-ul de varfuri toate informatiile geometrice (coordonate, normale, pozitia observatorului, pozitia sursei de lumina)
  - Din shader-ul de varfuri in shaderul de fragment sunt transferate (dupa aplicarea transformarilor de vizualizare si proiectie!)

```
out vec3 FragPos; // pozitia fragmentului
out vec3 Normal; // normala
out vec3 inLightPos; // pozitia sursei de lumina
out vec3 inViewPos; // pozitia observatorului
```

  - Modelul de iluminare este implementat in shader-ul de fragment

- **`10_02_model_iluminare.cpp`**
  Aplicarea modelului de iluminare in cazul unui cub
  - In acest cod sursa toate varfurile au aceeasi culoare.
  - Sunt patru posibilitati, intrucat sunt testate variante pentru:
    (i) implementarea modelului de iluminare (a. in shader-ul de fragment, b. in shader-ul de varfuri). In acest scop sunt scrise shader-e diferite – **`10_02f*`**, respectiv **`10_02v*`**)
    (ii) modul de alegere a normalelor (a. la nivelul fetelor sau b. la nivelul varfurilor, prin mediere). Implementarea pentru alegerea normalelor este legata doar de programul principal.
  - Din programul principal sunt transferate in shader-ul de varfuri toate informatiile geometrice (coordonate, normale, pozitia observatorului, pozitia sursei de lumina).
  - Din shader-ul de varfuri in sunt transferate shaderul de fragment informatii diferite, in functie de shader-ul in care este implementat modelul de iluminare (de exemplu, daca modelul de iluminare este implementat in shader-ul de varfuri, in shader-ul de fragment este transferata culoarea varfului, aceasta poate fi apoi randata ca atare).
  - Folosirea meniurilor `glutMenu`.

- **`10_03_iluminare_sfera.cpp`**
  Aplicarea modelului de iluminare in cazul unei sfere.
  - Fiecare varf are asociata o culoare (eventual poate fi aceeasi). Coordonata z a fiecarui varf este "perturbata" (se adauga "zgomot"). Fiecare varf are asociata o normala.
  - Sunt doua posibilitati, intrucat sunt testate variante pentru:
    (i) implementarea modelului de iluminare (a. in shader-ul de fragment, b. in shader-ul de varfuri). In acest scop sunt scrise shader-e diferite – **`10_03f*`**, respectiv **`10_03v*`**)

- Din programul principal sunt transferate in shader-ul de varfuri toate informatiile geometrice (coordonate, normale, pozitia observatorului, pozitia sursei de lumina).
- Din shader-ul de varfuri in sunt transferate shaderul de fragment informatii diferite, in functie de shader-ul in care este implementat modelul de iluminare (de exemplu, daca modelul de iluminare este implementat in shader-ul de varfuri, in shader-ul de fragment este transferata culoarea varfului, aceasta poate fi apoi randata ca atare).
- In shader-ul de varfuri **`10_03v*`** exista posibilitatea de a selecta sa fie randata doar culoarea varfurilor, fara aplicarea iluminarii.
- Folosirea meniurilor `glutMenu`.

Laborator 11

- **`11_01_umbra.cpp`**
  Iluminare: generarea umbrelor folosind transformari.
  - Pentru a genera umbra unei surse este utilizata o matrice 4x4 (umbra unui obiect = transformarea acelui obiect printr-o proiectie, cf. curs)
  - In shader-ul de varfuri este inclusa si matricea umbrei.
  - In shader-ul de fragment umbra este colorata separat.
  - Sursa de lumina este punctuala (varianta de sursa directionala este comentata).

- **`11_02_amestecare_2D.cpp`**
  Amestecare in context 2D
  - Folosirea celei de-a 4-a componente din codul RGBA.
  - Utilizarea functiilor specifice pentru amestecare (`glEnable(GL_BLEND); glBlendFunc(GL_SRC_ALPHA, GL_SRC_ALPHA);`)

- **`11_03_amestecare_3D.cpp`**
  Amestecare in context 3D
  - Folosirea celei de-a 4-a componente din codul RGBA.
  - Utilizarea functiilor specifice pentru amestecare (`glEnable(GL_BLEND); glBlendFunc(GL_SRC_ALPHA, GL_SRC_ALPHA);`)
  - Combinarea (i) ordinii de desenare a obiectelor, (ii) testului de adancime, (iii) efectelor de amestecare.