Laboratorul 01

Video Laborator 1: https://www.youtube.com/watch?v=YlzFkjJjh4Q.
Autor: Anca Morar

Introducere

Grafica pe calculator este un subiect amplu utilizat într-un număr din ce în ce mai mare de domenii. În acest laborator se vor prezenta conceptele ce stau la baza graficii cât și a utilizării procesorului grafic pentru acest scop. Domeniul graficii computerizate necesită cunoștințe variate: matematică, fizică, algoritmică, grafică digitală 2D & 3D, user experience design, etc.

Framework laborator

Întrucât scrierea unei aplicații simple OpenGL nu se poate realiza foarte ușor într-un timp scurt, dar și pentru a putea prezenta mai simplu conceptele de bază ale graficii computerizate moderne, în cadrul laboratoarelor se va lucra pe un framework ce oferă o serie de funcționalități gata implementate. Framework-ul utilizat oferă toate funcționalitățile de bază ale unui motor grafic minimal, precum:

- Fereastra de desenare având la bază un context **OpenGL 3.3+** (o să aflați ce înseamnă)
- Suport pentru încărcarea de modele 3D (cunoscute și ca **3D meshes**)
- Suport pentru încărcarea de imagini pentru texturarea modelelor 3D
- Suport pentru definirea și încărcarea de shadere OpenGL

De asemenea, pe langă funcționalitățile de bază, framework-ul implementează un model generic pentru scrierea de aplicații OpenGL. Astfel, sunt oferite următoarele aspecte:

- Control pentru fereastra de afișare
- Management pentru input de la tastatură și mouse
- Cameră de vizualizare cu input predefinit pentru a ușura deplasarea și vizualizarea scenei
- Model arhitectural al unei aplicații simple OpenGL, bazat pe toate aspectele prezentate

Funcționalitatea framework-ului este oferită prin intermediul mai multor biblioteci (libraries):

- **GLFW**
  - **Site oficial** http://www.glfw.org **Github** https://github.com/glfw/glfw
  - Oferă suportul de bază pentru API-ul OpenGL precum context, fereastră, input, etc
- **GLEW**
  - **Site oficial** http://glew.sourceforge.net **Github** https://github.com/nigels-com/glew
  - Asigură suportul pentru extensiile de OpenGL suportate de procesorul grafic
- **GLM**
  - **Site oficial** http://glm.g-truc.net **Github** https://github.com/g-truc/glm
  - Funcționalități matematice bazate pe specificațiile limbajului GLSL (shadere OpenGL)
  - Asigură interoperabilitate simplă cu API-ul OpenGL
- **ASSIMP**
  - **Site oficial** http://www.assimp.org **Github** https://github.com/assimp/assimp
  - Open Asset Import Library
  - Oferă suport pentru încărcarea de modele și scene 3D
  - Suportă majoritatea formatelor de stocare 3D utilizate în industrie
- **STB**
  - **Github** https://github.com/nothings/stb
  - Oferă suport pentru încărcare/decodare de imagini JPG, PNG, TGA, BMP, PSD, etc.
- **EGC-Components** – closed source
  - Oferă o serie de funcționalități ce vor fi utilizate începând din primul laborator dar care vor fi implementate și de către studenți pe parcursul laboratoarelor

Structura framework-ului

- **/libs**
  - Bibliotecile utilizate în cadrul framework-ului
- **/Visual Studio**
  - Proiect Visual Studio 2017 preconfigurat
- **/Resources**
  - Resurse necesare rulării proiecutului
  - **/Textures**
    - Diverse imagini ce pot fi încărcate și utilizate ca texturi
  - **/Shaders**
    - Exemple de programe shader - Vertex Shader și Fragment Shader
  - **/Models**
    - Modele 3D ce pot fi încărcate în cadrul framework-ului
- **/Source**
  - Surse C++
  - **/include**
    - O serie de headere predefinite pentru facilitarea accesului la biblioteci
    - gl.h
      - Adaugă suportul pentru API-ul OpenGL
    - glm.h
      - Adaugă majoritatea headerelor glm ce vor fi utilizate
      - Printare ușoara pentru `glm::vec2, glm::vec3, glm::vec4` prin intermediul operatorului C++ supraîncărcat: `operator«`
    - math.h
      - Simple definiții preprocesor pentru MIN, MAX, conversie radiani ⇔ grade
    - utils.h
      - Simple definiții preprocesor pentru lucrul cu memoria și pe biți
  - **/Components**
    - Diverse implementări ce facilitează lucrul în cadrul laboratoarelor
    - SimpleScene.cpp
      - Model de bază al unei scene 3D utilizată ca bază a tuturor laboratoarelor
    - CameraInput.cpp
      - Implementare a unui model simplu de control FPS al camerei de vizualizare oferite de biblioteca EGC-Components
  - **/Core**
    - API-ul de bază al framwork-ului EGC
    - **/GPU**
      - GPUBuffers.cpp
        - Asigură suportul pentru definirea de buffere de date și încărcarea de date (modele 3D) pe GPU
      - Mesh.cpp
        - Loader de modele 3D atât din fișier cât și din memorie
      - Shader.cpp
        - Loader de programe Shader pentru procesorul grafic
      - Texture2D.cpp
        - Loader de texturi 2D pe GPU
    - **/Managers**
      - ResourcePath.h
        - Locații predefinite pentru utilizarea la încărcarea resurselor
      - TextureManager.cpp
        - Asigură încărcare și management pentru texturile Texture2D
        - Încarcă o serie de texturi simple predefinite
    - **/Window**
      - WindowCallbacks.cpp
        - Asigură implementarea funcțiilor de callback necesare de GLFW pentru un context OpenGL oarecare
        - Evenimentele GLFW sunt redirecționare către fereastra definită de Engine
      - WindowObject.cpp
        - Oferă implementarea de fereastră de lucru, suport predefinite definire pentru callbacks, dar și un model de buffering pentru evenimente de input tastatură și mouse
      - InputController.cpp
        - Prin moștenire oferă suport pentru implementarea callback-urilor de input/tastatură. Odată instanțiat, obiectul se va atașa automat pe fereastra de lucru (pe care o obține de la Engine) și va primi automat evenimentele de input pe care le va executa conform implementării
        - În cadrul unui program pot exista oricâte astfel de obiecte. Toate vor fi apelate în ordinea atașării lor, dar și a producerii evenimentelor
      - Engine.cpp
        - Asigură inițializarea contextului OpenGL și a ferestrei de lucru
      - World.cpp
        - Asigură implementarea modelului de funcționare al unei aplicații OpenGL pe baza API-ului oferit de Framework
  - **/Laboratoare**
    - Implementările pentru fiecare laborator EGC
    - Fiecare laborator va pleca de la baza oferită de SimpleScene

Funcționarea unei aplicații grafice (OpenGL)

Orice aplicație trebuie să asigure funcționalitatea pe o anumită perioadă de timp. În funcție de cerințe această perioadă poate fi :

- **deterministă** - programul va executa un anumit task iar apoi se va închide (majoritatea programelor create in cadrul facultății până în acest moment respectă acest model)
- **continuă** - rulează unul sau mai multe task-uri în mod continuu (până in momentul în care utilizatorul sau un eveniment extern închide aplicația).

Aplicațiile grafice cu suport de vizualizare în timp real (de exemplu jocurile, sau de exemplu ce vom face noi la EGC) se regăsesc în cel de-al doilea model și au la bază funcționarea pe baza unei bucle (loop) de procesare.
În cadrul framework-ului EGC acest loop constă într-o serie de pași (vezi `World::LoopUpdate()`):

1. Se interoghează evenimentele ferestrei OpenGL (input, resize, etc.)
   - Evenimentele sunt salvate pentru a fi procesate mai tarziu
2. Se estimează timpul de execuție pentru iterația actuală (timpul de execuție al iterației precedente)
3. Se procesează evenimentele salvate anterior
4. Se procesează frame-ul actual (este indicat să se ia în considerare timpul de execuție în cadrul modificărilor pentru a oferi actualizare independentă de timp)
5. *Opțional*: În cazul double sau triple buffering se interschimbă bufferele de imagine
6. Se trece la următorul frame (se revine la primul pas)

Cel mai simplu model de aplicație OpenGL va trata evenimentele de input (mouse, tastatură) la momentul producerii lor. Acest model nu este indicat deoarece are numeroase dezavantaje:

- nu oferă posibilitatea de a trata combinații de taste (Exemplu: utilizatorul apasa W și A pentru a deplasa caracterul in diagonală)
- nu oferă informații ce țin de starea continuă a unui eveniment
  - **Exemplu**: Un personaj dintr-un joc trebuie să se deplaseze în față atât timp cât utilizatorul ține apasată tasta **W**.
    - Pentru a trata corespunzător o astfel de logică este necesar să menținem starea tastei **W** iar atunci când se face deplasarea personajului, aceasta să fie direct proporțională cu timpul trecut de la ultimul frame procesat
    - Același lucru se aplică și în cazul butoanelor de la mouse

De asemenea, un model bazat pe buffering al evenimentelor de input oferă posibilitatea de a interoga starea input-ului în orice moment al unui frame, deci ofera și o flexibilitate generală mai mare pentru a implementa noi comportamente/logici. Clasa `WindowObject` asigură suportul pentru buffering, dar și pentru procesarea ulterioară a evenimentelor prin intermediul obiectelor de tipul `InputController`.

Recomandăm să citiți documentația GLFW despre tratarea evenimentelor de input pentru a înțelege mai bine conceptele prezentate: http://www.glfw.org/docs/latest/input_guide.html

Multi-buffering

În general, aplicațiile grafice folosesc mai multe buffere de imagini separate pentru a evita apariția artefactelor grafice prin modificarea directă a imaginii randate pe ecran. Astfel, imaginea afișată la momentul `T` a fost procesată la momentul `T-1`, sau `T-2` (în funcție de dimensiunea bufferului).
Informații adiționale despre această tehnică multi-buffering pot fi obțiunute de pe wiki:

- https://en.wikipedia.org/wiki/Multiple_buffering
- https://en.wikipedia.org/wiki/Multiple_buffering#Double_buffering_in_computer_graphics
- https://en.wikipedia.org/wiki/Multiple_buffering#Triple_buffering

Modelul de funcționare al aplicației de laborator

În cadrul unui laborator modelul aplicației grafice prezentat mai sus este implementat de către clasa `World`.
Pasul 2 este tratat de către instanțele InputController în timp ce pasul 4 este asigurat de funcțiile `FrameStart()`, `Update(float deltaTime)`, și `FrameEnd()` moștenite de la clasa `World`. Clasa `World` extinde deja `InputController` pentru a ușura munca în cadrul laboratorului.
Toate laboratoarele EGC vor fi implementate pe baza `SimpleScene` ce oferă următoarele facilități:

- scena 3D cu randarea unui sistem cartezian de referință în coordonate OpenGL
  - plan orizontal XOZ
  - evidențierea spațiului pozitiv (OX, OY, OZ)
- camera predefinită pentru explorarea scenei
- shadere predefinite pentru lucrul în primele laboratoare
- management pentru stocarea shaderelor și modelelor nou create, pe baza unui nume unic

Etapele rulării aplicației

1. Se definesc proprietățile pentru fereastra de lucru (`Main.cpp`)
2. Se inițializează Engine-ul astfel - `Engine::Init()`
   a. Se inițializează API-ul OpenGL (`glfwInit()`)
   b. Se creează fereastra de lucru cu un context OpenGL 3.3+
      I. Se atașează evenimentele de fereastră prin intermediul WindowsCallbacks.cpp
   c. Se inițializează managerul de texturi
3. Se creează și inițializează o nouă scenă 3D de lucru având la bază modelul de update prezentat anterior (Main.cpp)
4. Se pornește rularea scenei încărcate (`LoopUpdate()`)

Standardul OpenGL

OpenGL este un standard (API) pe care îl putem folosi pentru a crea aplicații grafice real-time. Este aproape identic cu Direct3D, ambele având o influență reciprocă de-a lungul anilor.

- Mai multe informații despre istoricul OpenGL se pot găsi la adresa: https://en.wikipedia.org/wiki/OpenGL
- Explicații complete prinvind API-ul OpenGL cât și utilizarea acestuia se pot găsi pe pagina oficială a standardului: https://www.opengl.org/sdk/docs/man/

Atunci când nu sunteți siguri ce face o anumită comandă sau ce reprezintă parametrii funcțiilor este recomandat să consultați documentația: https://www.opengl.org/sdk/docs/man/

Versiunea curentă a acestui standard este 4.6. Pentru cursul de EGC vom folosi standardul 3.0/3.3, care este în același timp și versiunea actuală pentru varianta pentru mobile a OpenGL, numită OpenGL ES https://en.wikipedia.org/wiki/OpenGL_ES.

Începând cu 2016 a fost lansat și API-ul Vulkan ce oferă access avansat low-level la capabalitățile grafice moderne ale procesoarelor grafice. Standardul Vulkan este orientat dezvoltării aplicațiilor de înaltă performanță iar complexitatea acestuia depășește cu mult aspectele de bază ce vor fi prezentate în cadrul cusului/laboratorului.

Utilizarea API

Pe parcursul laboratoarelor (dar și a cursului) se va trece prin toate etapele importante ce stau la baza redării grafice. Astfel vor fi învățate concepte precum:

- încărcare și randare de obiecte 3D simple
- funcționarea pipeline-ului grafic
- vizualizare, proiecție, control camera
- utilizare shadere (vertex și fragment shader)
- iluminare
- texturare

Cerințe generale de laborator

- Citiți cu foarte mare atenție Framwork-ul de laborator întrucât îl veți utiliza pe tot parcursul laboratorului de EGC inclusiv și la temele de casă
- Citiți comentariile din cod – ar trebui să răspundă la majoritatea întrebărilor pe care le aveți
- Citiți documentația de la __InputController.h__ întrucât veți utiliza constant funcțiile din cadrul acestei clase (prin suprascriere) pentru definirea de interacțiuni și comportament personalizat
- Dacă nu ințelegeți modelul de funcționare al aplicației rugați asistentul să explice încă o dată cum funcționează toată aplicația

C++

Framework-ul este scris în limbajul C++, ce va fi utilizat pe tot parcursul laboratoarelor. Conceptele utilizate în cadrul laboratorului și care trebuie știute sunt:

- concepte de bază de OOP - obiecte, moștenire, metode virtuale, etc
- utilizarea bibliotecilor standard: în special std::vector, std::list și std::unorderd_map

Pentru cei mai puțin familiarizați cu limbajul C++ recomandăm să parcurgeți tutoriale: Learn C++

Visual Studio 2019

- În cadrul laboratorului vom utiliza Visual Studio 2019 Community Edition
- Installer-ul de Visual Studio vine cu posibilitatea de a instala modular doar ceea ce este necesar. Pentru acest laborator trebuie instalat doar modulul default **Desktop development with C++**, care se regăsește în **Workloads**
- Framework-ul conține deja un proiect preconfigurat pentru Visual Studio Framework_EGC.sln (folderul /Visual Studio)
- Deschideți soluția în Visual Studio

Cei care nu au mai utilizat IDE-ul **Visual Studio** pentru scrierea de aplicații C++ sunt rugați să citească toturialul Getting Started with C++ in Visual Studio

GLM

În grafică, matematica este folosită peste tot, de la simple matrici pentru rotații până la integrale infinit dimensionale pentru algoritmii folosiți în industria filmului, de aceea ne dorim să avem un suport de matematică robust, bine documentat și nu în ultimul rând cât mai apropiat de formatul OpenGL. În loc să scriem noi o bibliotecă de matematică vom folosi biblioteca GLM. GLM ne oferă rotații, translații, vectori de dimensiune 2/3/4, matrici și multe alte funcționalități avansate (de ex. modele de zgomot). Vom folosi doar cele mai simple funcționalități în laboratoarele de la această materie.

```
glm::mat4 identity = glm::mat4 (1, 0, 0, 0,
                                0, 1, 0, 0,
                                0, 0, 1, 0,
                                0, 0, 0, 1);
glm::mat4 identity2 = glm::mat4(1); // short form for writing identity matrices
glm::vec3 culoare = glm::vec3(1, 0.5, 1);
glm::vec2 directie = glm::vec3(-1, 1);
glm::vec3 pozitie = glm::vec3(100, 10, -20);
pozitie.x = 2; // you can select components like so: .x .y .z .t .r .g. b. a
```

Laboratorul 1

Framework

Framework-ul de laborator se găsește pe Github
Puteți să descărcați direct arhiva accesând acest link

Informații laborator

Sursele ce stau la baza fiecărui laborator se află în directorul: /Source/Laboratoare/Laborator**N**/, **N** reprezentând numărul laboratorului.

În cadrul laboratorului 1 puteți încărca modele 3D în cadrul scenei și cere afișarea scenei utilizând funcția

```
RenderMesh(Mesh * mesh, glm::vec3 position, glm::vec3 scale)
```

Culorile pixelilor prin care se reprezintă scena sunt salvate într-un buffer, numit Framebuffer. Contextul definit oferă automat un astfel de buffer și este configurat să ruleze cu double-buffering

API-ul OpenGL utilizat în cadrul laboratorului:

```
// defineste un spatiu de desenare in spatiul ferestrei de afisare a aplicatiei
// x, y reprezinta coordonatele coltului stanga jos
// width, height reprezinta dimensiunea spatiului de desenare.
void glViewport(GLint x,  GLint y,  GLsizei width,  GLsizei height);

// seteaza culoarea cu care va fi colorat tot ecranul la operatia de clear
void glClearColor(float r, float g, float b, float a);

 // implementeaza operatia de clear
void glClear(GL_COLOR_BUFFER_BIT);
```

Culorile în OpenGL sunt specificate ca float în intervalul 0 - 1

Bufferul de culoare utilizat (atât în cadrul laboratorului dar și în mod uzual datorită limitărilor impuse de afișarea pe monitoare) este în format RGBA8. Fiecare componentă (red, green, blue, alpha) este memorată pe 8 biți, deci are o valoare in intervalul 0 – 255. Astfel:

- roșu (255, 0, 0) este reprezentată ca (1, 0, 0) pe procesorul grafic
- galben este (1, 1, 0) și tot așa

Control aplicație

Programul rulat oferă posibilitatea vizualizării scenei create prin intermediul unei camere predefinite.

Taste de control pentru cameră

- **W, A, S, D, Q, E** - deplasare față, stânga, spate, dreapta, jos, sus
- **MOUSE RIGHT + MOUSE MOVE** - rotație cameră

Cerințe laborator

1. Descărcați framework-ul, compilați și rulați proiectul
   - Trebuie să deschideți proiectul Framework_EGC.sln (folderul /Visual Studio) în Visual Studio 2019
2. Încărcați un alt model 3D și randați-l în scenă la o poziție diferită față de cele 2 cuburi
   - /Resources/Models conține o serie de modele 3D ce pot fi încărcate
   - În `Laborator1::Init()` găsiți modul în care puteți să declarați (și încărcați) un nou obiect de tip `Mesh`
3. La apăsarea unei taste să se schimbe culoarea de ștergere a ecranului
4. La apăsarea unei taste să se schimbe obiectul afisat (render) la o poziție (să cicleze prin 3 obiecte, de ex cube, teapot, sphere)
5. Să se miște prin spațiu un obiect oarecare la apăsarea tastelor W, A, S, D, E, Q (pozitiv și negativ pe toate cele 3 axe)

Citiți cu atenție documentația evenimentelor de input din fișierul `InputController.h` întrucât le veți utiliza în cadrul fiecărui laborator