UNIVERSITATEA “ȘTEFAN CEL MARE”

FACULTATEA DE INGINERIE ELECTRICĂ ȘI ȘTIINȚA CALCULATOARELOR

SPECIALIZAREA: CALCULATOARE

REFERAT LA EGC

TEHNOLOGIA OpenGL

Student, Bogatu Patriciu, anul III

Grupa: 3132B

Anul univ. 2022-2023

OpenGL reprezintă un API creat pentru rendering-ul graficii bidimensiune și tridimensionale. Interfața de programare respectivă dispune de o mulțime de funcții care pot gestiona grafica în mai multe aplicații pe un număr mare de platforme.

OpenGL oferă o metodă ușoară pentru crearea jocurilor video crossplatform, de aceea acest API este în primul rând asociat cu industria jocurilor. AutoCAD și Blender sunt două exemple de aplicații CAD care folosesc ca bibliotecă grafică OpenGL.

Un **punct forte** al OpenGL este că e deschis spre extensii. Când niște firme care se ocupă cu grafică vin cu tehnici și optimizări noi pentru rendering, aceste inovări se vor regăsi în extensiile implementate pentru drivere.

OpenGL în sine reprezintă o mașină de stări, o colecție de variabile care definesc comportamentul curent pentru OpenGL. Contextul (curent) pentru OpenGL e determinat de starea definită de acele variabile. Lucrând cu OpenGL, se folosesc numeroase funcții destinate(state changing) modificării contextului și numeroase alte funcții(state using) care realizează niște operații ce se bazează pe starea curentă.

Bibliotecile OpenGL sunt toate scrise în limbajul de programare C, deși se pot realiza niște tranziții pentru a putea se putea folosi de acele librarii cu alte limbaje.

OpenGL a fost dezvoltat foarte abstract cu scopul ca obiectele pentru ca ele să poată fi referite în limbajele de nivel mai înalt. De asta a fost nevoie, pentru că C are niște mecanisme incompatibile cu limbaje high-level.

Un obiect, în OpenGL, reprezintă o colecție de opțiuni ce sunt o parte a stării OpenGL. Crearea unui obiect se face prin mai multe etape. În principiu, această instanțiere se poate interpreta ca o mapare a unui struct/clase mai mici(obiectul curent) la un struct mare(starea OpenGL). Se declara obiectul, apoi se mapeaza la context, urmează să se seteze parametrii obiectului mapat. La obiecte se pot referi după ID. Maparea la context se face după locația unui anumit TARGET.

După setarea parametrilor unui target prin intermediul datorită obiectului, se va realiza o demapare, aceasta realizându-se prin setarea obiectului respectiv ID-ul 0. În acest moment, setările se vor întoarce la default, iar dacă, ulterior, se va dori să se facă restore la acele setări, se va realiza o mapare repetată. Setările vechi sunt salvate în obiectul către care se poate referi după ID. Următoarele funcții sunt folosite în procesul dat: glGenObject – generează obiectul, glBindObject - mapează la context, glSetObjectOption - setează parametrii obiectului ce e deja mapat la target.

Un **plus** evident din tot acest procest, este că se pot crea mai multe obiecte în care se va salva niște setări, iar când se va dori preluarea unor setări, se va referi la obiectul necesar. De exemplu, un obiect poate conține informații despre un model 3D, se zicem un personaj, când se va dori desenarea acestuia, se va face maparea, iar apoi se vor desena alte obiecte 3D prin repetarea procesului de mai sus.

Pentru a începe rendering-ul, mai întâi e nevoie de o fereastră în care va fi contextul OpenGL propriu-zis. **GLFW** oferă o asemenea funcționalitate: o fereastră și un context OpenGL pentru rendering. Conținutul unui package GLFW poate fi compilat prin CMake. Pentru a genera build folder-ul dorit, trebuie de specificat folder-ul sursă care este package-ul GLFW și destinația(build folder) în care se va genera proiectul(de exemplu Visual Studio) împreună cu fișierele aferente. După crearea soluției, se va deschide IDE-ul și se va face link-ajul bibliotecilor glfw3 și opengl32.

O etapă importantă spre începerea rendering-ului este să se găsească locația funcțiilor dorite și să se păstreze în pointeri la funcții, pentru că locția acelor funcții nu se cunosc la run-time. Deja se poate menționa un **dezavantaj:** din cauza caracterului low-level a lui OpenGL, trebuie realizate foarte multe presetări înainte de efectiva creare și modificare a contextului. Developer-ul amator se va întâlni de la început cu mult cod complex care realizeză niște presetări, chiar dacă sunt biblioteci care ușurează procesul menționat mai sus.

Pentru a face query-uri la funcțiile dorite se folosește libăria **GLAD**.

Funcție de versiunea OpenGL, GLAD va defini și încârca toate funcțiile relevante.

După descărcarea bibliotecii GLAD și includerea acesteia în proiect, în fișier, deja se poate folosi GLAD împreună cu GLFW(care de asemenea trebuie inclus în fișier prin #include) pentru configurarea unui context OpenGL și crearea unei ferestre.

După setarea versiunilor minore și majore ai GLFW se poate deja crea obiectul asociat ferestrei. Este important mai întâi să se inițializeze GLAD-ul pentru ca să se poată folosi pointerii la funcțiile necesare.

Înainte de rendering, trebuie de specificat mărimea ferestrei de rendering, ca OpenGL să știe ce fel de valori și coordonate se vor afișa funcție de dimensiunea ferestrei. Dimensiunea la OpenGL viewport se setează funcție de fereastra GLFW.

La fiecare setare a ferestrei GLFW trebuie să se apeleze o funcție care deja va adapta viewport-ul la noile valori ale ferestrei GLFW.

Pentru ca fereastra să nu se închidă imediat după lansare, trebuie de creat o buclă infinită care să ruleze până utilizatorul explicit va închide fereastra, astfel aplicația va desena imagini și va procesa datele de intrare atâta timp cât utilizatorul își va dori. Deci, așa numitul render-loop este soluția. Dacă se va dori închiderea ferestrei, după aceasta, când se va părăsi render loop-ul, e important să se facă o ștergere a informațiilor alocate, aceasta realizându-se prin glfwTerminate.

În bucla de render se pun funcțiile pentru rendering, ele modifică conținutul ferestrei. Astfel, funcție de buffer, se poate modifica, de exemplu, culoarea de background a ferestrei. Ulterior, pentru a desena, de exemplu un triunghi, trebuie de creat vertex-uri(fiecare cu o poziție 3D) și de configurat niște shader-e.

În concluzie, pot afirma că OpenGL, fiind o tehnologie foarte low-level, ea totuși permite crearea unei grafice foarte detaliate, de la zero. OpenGL e portabil(de ex. spre WebGL) și se adaptă foarte rapid la tehnologiile noi(GPU-ri noi).