Laborator 4

1 Obiective

Obiectivul acestui laborator este de a prezenta diferite metode de depanare a aplicațiilor OpenGL.

2 Notiuni teoretice

2.1 glGetError()

OpenGL va genera mesaje de eroare de fiecare dată când există o utilizare incorectă a funcțiilor sau a funcționalității. Puteți căuta mesaje de eroare folosind funcția glGetError (). Fiecare eroare are asociate un cod numeric și un nume simbolic. In caz de eroare, este setat un semnal (flag cu valoarea corespunzătoare a codului de eroare). Nu sunt generate alte erori până când se apelează glGetError. La fiecare apel, codul de eroare este returnat și semnalul (flag-ul) este resetat. Se pot genera următoarele coduri de eroare:

Error code	Error flag	Error description
0	GL_NO_ERROR	Nu a fost înregistrată nicio eroare. Valoarea acestei constante simbolice este garantată a fi 0.
1280	GL_INVALID_ENUM	Se specifică o valoare inacceptabilă pentru un argument enumerat. Comanda necorespunzătoare este ignorată și nu are alt efect secundar decât să seteze semnalul de eroare.
1281	GL_INVALID_VALUE	Un argument numeric este în afara intervalului acceptat. Comanda necorespunzătoare este ignorată și nu are alt efect secundar decât să seteze semnalul de eroare.
1282	GL_INVALID_OPERATION	Operația specificată nu este permisă în starea actuală. Comanda necorespunzătoare este ignorată și nu are alt efect secundar decât să seteze semnalul de eroare.
1286	GL_INVALID_FRAMEBUFFER_OPERATION	Obiectul framebuffer nu este complet. Comanda necorespunzătoare este ignorată și nu are alt efect secundar decât să seteze semnalul de eroare.
1285	GL_OUT_OF_MEMORY	Nu este suficientă memorie pentru executarea comenzii. Starea GL este nedefinită, cu excepția stării semnalelor de eroare, după ce această eroare este înregistrată.
1284	GL_STACK_UNDERFLOW	A fost făcută o încercare de a efectua o operațiune care ar determina o stivă internă să se prăbușească.

1283	GL_STACK_OVERFLOW	A fost făcută o încercare de a efectua o operație care ar determina o stivă internă să depășească limita admisa.

Pentru a verifica erorile, utilizați următoarea funcție:

```
GLenum glCheckError_(const char *file, int line)
{
    GLenum errorCode;
    while ((errorCode = glGetError()) != GL_NO_ERROR)
    {
        std::string error;
        switch (errorCode)
        {
            case GL_INVALID_ENUM:                  error = "INVALID_ENUM"; break;
            case GL_INVALID_VALUE:                 error = "INVALID_VALUE"; break;
            case GL_INVALID_OPERATION:             error = "INVALID_OPERATION"; break;
            case GL_STACK_OVERFLOW:                error = "STACK_OVERFLOW"; break;
            case GL_STACK_UNDERFLOW:               error = "STACK_UNDERFLOW"; break;
            case GL_OUT_OF_MEMORY:                 error = "OUT_OF_MEMORY"; break;
            case GL_INVALID_FRAMEBUFFER_OPERATION: error = "INVALID_FRAMEBUFFER_OPERATION"; break;
        }
        std::cout << error << " | " << file << " (" << line << ")" << std::endl;
    }
    return errorCode;
}
#define glCheckError() glCheckError_(__FILE__, __LINE__)
```

2.2 Depanare la nivel de shader

Din păcate la nivelul shaderului nu putem folosi funcții precum glGetError () și nu putem face o depanare pas cu pas prin programul nostru de shader. O soluție pentru depășirea acestei restricții este afișarea unor variabile pe canalul de iesire al fragment shader-ului. De exemplu, putem afișa vectorii normala pentru a verifica dacă au valori corecte.

Actualizați datele varfurilor pentru a încorpora vectorii normala:

```
GLfloat vertexData[] = {
    //vertex position, colour and normal vector
    0.0f, 0.7f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
    -0.5f, 0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f,
    -0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f,
    0.5f, -0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f,
    0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f, 1.0f
};

GLuint vertexIndices[] = {
    0, 1, 4,
    1, 2, 3,
    1, 3, 4
};

GLuint verticesVBO;
GLuint verticesEBO;
GLuint objectVAO;
```

Schimbați funcția initObjects pentru a trimite atributul normală către vertex shader.

Actualizați vertex shader:

```
#version 400

layout(location = 0) in vec3 vertexPosition;
layout(location = 1) in vec3 vertexColour;
layout(location = 2) in vec3 vertexNormal;

out vec3 colour;
out vec3 normal;

void main() {
    colour = vertexColour;
```

```
    normal = vertexNormal;
    gl_Position = vec4(vertexPosition, 1.0);
}
```

Actualizați fragment shader:

```
#version 400

in vec3 colour;
in vec3 normal;

out vec4 fragmentColour;

void main() {
    fragmentColour.rgb = normal;
    fragmentColour.a = 1.0;
}
```

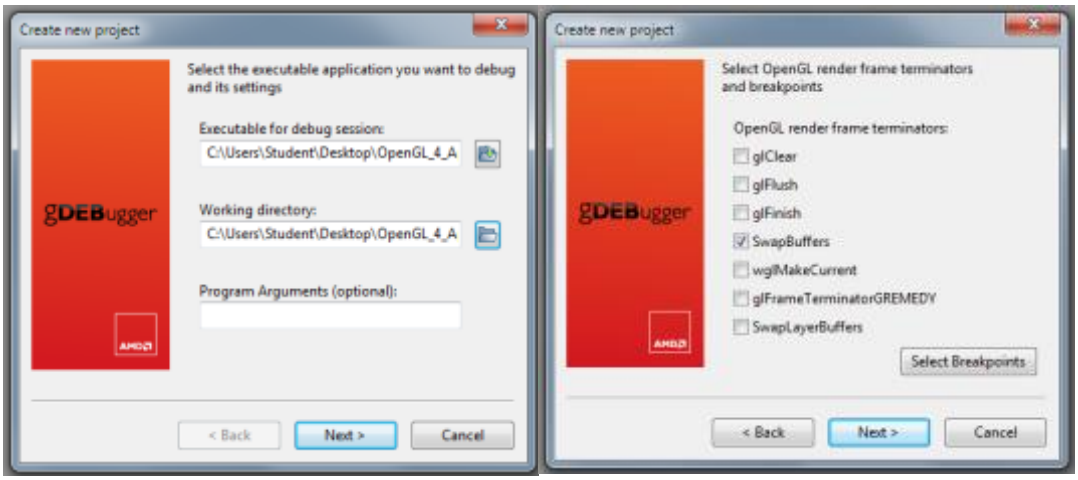
Întrebare: De ce obiectul este acum afișat în albastru?

2.3 Aplicație terță parte

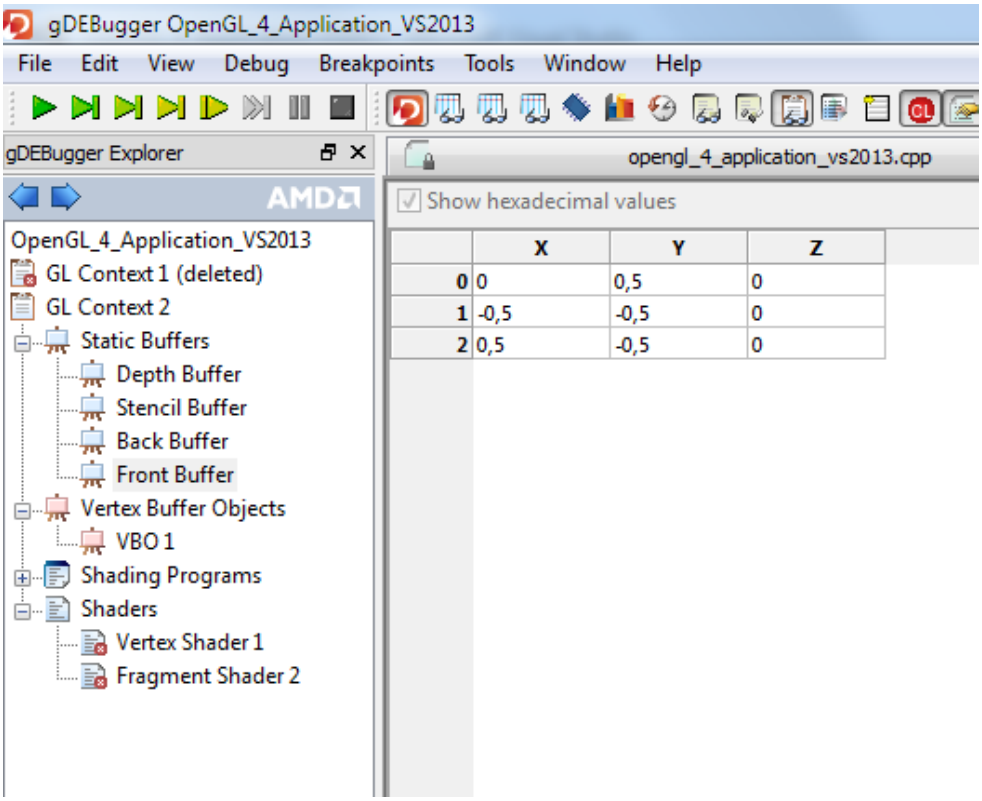
Aplicațiile de la terțe părți pot intercepta apeluri OpenGL diferite și pot fi folosite pentru a urmări utilizarea funcțiilor OpenGL, pentru a găsi blocaje, a inspecta memoria buffer și pentru a afișa texturile și atașamentele framebuffer. gDebugger este unul dintre instrumentele de depanare disponibile pentru aplicațiile OpenGL. Este o aplicație cross-platform care oferă o prezentare detaliată a stării aplicatiei OpenGL care rulează. Descărcați aplicația gDebugger de la:

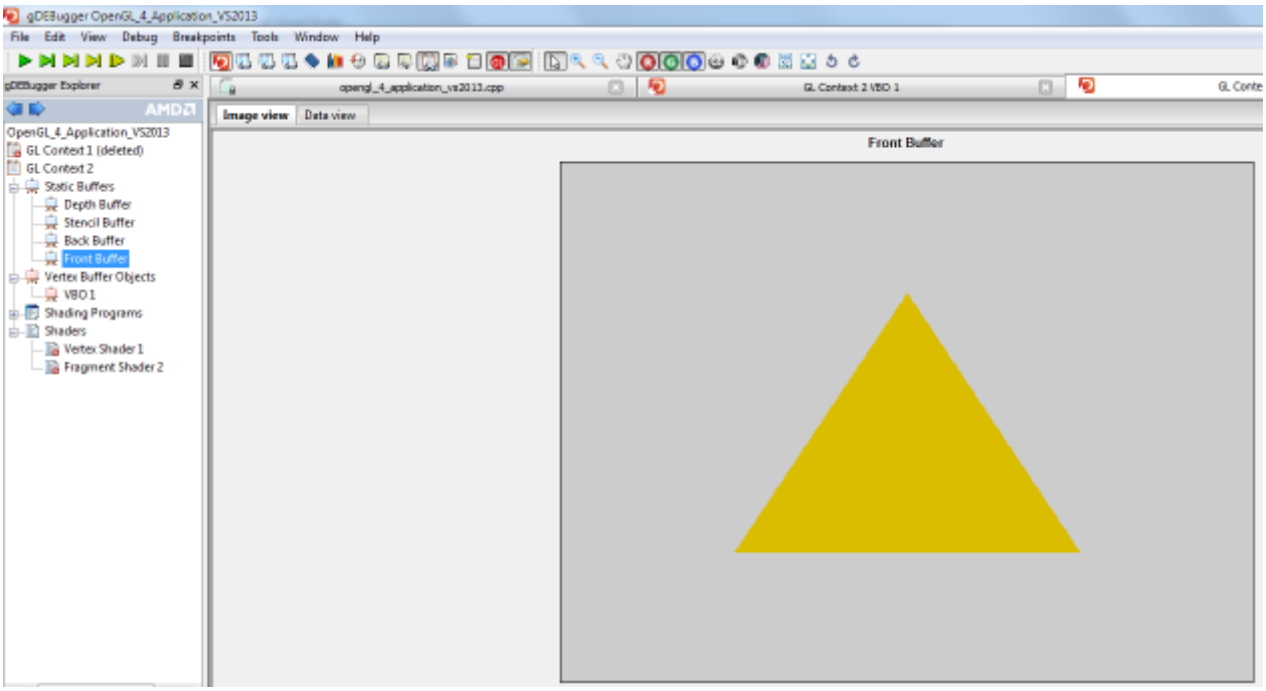
http://developer.amd.com/tools-and-sdks/archive/compute/amd-gdebugger/

Pentru a depana o aplicație OpenGL folosind gDebugger, trebuie să creați un nou proiect și să selectați locația programului și directorul de lucru al executabilului OpenGL.



Dacă întrerupeți aplicarea în orice moment, puteți examina starea curentă, memoria texturii și / sau utilizarea bufferului.





2.4 Sfaturi și trucuri

Ce trebuie să faceți dacă aplicația dvs. OpenGL afișează doar un ecran negru:

- Verificați erorile folosind glGetError ()
- Verificați dacă geometria se află în interiorul frustumului de vizualizare
- Schimbați culoarea de fundal diferită de negru
- Verificați dacă sunt configurate atributele vertexului
- Verificați dacă variabilele uniformele sunt setate

3 Further reading

Funcție (și legătură)	Descriere
glGenBuffers	generează nume de obiecte tampon buffer
glBindBuffer	leagă un obiect buffer la punctul de legare al buffer-ului specificat
glBufferData	creează și inițiază stocarea datelor unui obiect buffer
glGenVertexArrays	generează nume de vertex array object
glBindVertexArray	activează un vertex array object
glVertexAttribPointer	definește un șir of date generice associate unui vârf
glEnableVertexAttribArray	activează un șir generic de atribute
glCreateShader	creează un obiect shader
glShaderSource	înlocuiește codul sursă într-un obiect shader
glCompileShader	compilează un obiect shader
glCreateProgram	creează un obiect program
glAttachShader	atașează un obiect shader unui obiect program
glLinkProgram	operație de linking pentru un obiect program
glDeleteShader	șterge obiect shader

glUseProgram	instalează un obiect de program ca parte a stării curente de rasterizare
glDrawArrays	rasterizează primitive folosind șiruri de date
glGetUniformLocation	returnează locația unei variabile uniform
glUniform	specifică valoarea unei variabile uniform pentru obiectul program curent
glDrawElements	rasterizează primitive folosind șiruri de date
glGetError	Returnează valoarea semnalului de eroare

4 Temă

- Descărcați fișierele de pe site și actualizați proiectul anterior cu acestea. Programul ar trebui să se compileze, dar va afișa o fereastră gri. Încercați să depanați programul pentru a afișa următoarele: