EDAF80: Seminar 0

Lab info

OpenGL

Graphics pipeline

Vertices

n of

Buffers vs. textu

company so

More info

C++ crash

course

About

model

Stack

Heap and pointe

Classes

Arrays

Paramet

Operator querional

Output

Output

More info

Introductory Seminar

EDAF80: Computer Graphics

Rikard Olajos







AGENDA

3 C++ crash course

Lab info

2 OpenGL

LABS OVERVIEW

Lab info

- Application setup Graphics pipeline Vertices
- Buffers vs. texture
 Compiling shader
 Drawing
- C++ crasl

course

Simplified memory model

Heap and pointer

Classes

Parameter:

Operator overloadin Output

- 1 optional + 5 mandatory assignments
 - Week 2 − 7
 - "Lab 0" in week 2: optional attendance
 - Book sessions on course homepage
- Work in pairs
 - If looking for a partner, post on forum
- E:Uranus
 - Located in E-huset basement
 - Windows 10, 64-bit, Core i5, 8GB RAM
 - Visual Studio 2022
 - Geforce GTX 560

OPENGL

OpenGL

Graphics pipeline Vertices Fragments Buffers vs. texture

More info

C++ crash course

Simplified memo

model Stack

Heap and pointer

Arrays Paramete

Types
Operator overload

Operator overloading Output Application Programming Interface (API)

- Set of functions that create a 2D image of a 3D scene
- 3D scene is made of:
 - Primitives Triangles
 - Textures 2D images
 - and much more!
- Controls a graphics pipeline (graphics hardware)
 - Graphics Processing Unit (GPU)

OPENGL

OpenGL

Graphics pipeline
Vertices
Fragments
Buffers vs. textures
Compiling shaders

More info

C++ crast

Simplified memo

model

Heap and pointe

Classes

Paramete

Operator overload

Output

- We will focus on the core profile
 - no fixed function/immediate mode
- OpenGL is a state machine
 - Current state is the "OpenGL context"
 - There are many functions that change the current state
 - OpenGL uses objects that are a part of the state
 - Drawing uses the most recently bound buffers

APPLICATION SETUP

Application setup

- First, make a window, use GLFW library
- Second, create a while loop i.e. the render loop:
 - Grab inputs
 - Render the screen
 - Swap the buffers
- Third, do some rendering in the render loop...

GRAPHICS PIPELINE

OpenGL Application setup

Fragments
Buffers vs. textu

Compiling shad
Drawing
More info

C++ crash

About Simplified mem

model

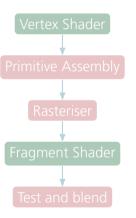
Heap and pointe

Arrays

Paramet

Operator overloadi

More info



- Shaders are programmable, other parts are not
- There are no default vertex and fragment shaders, you must provide them
- Primitive Assembly (PA) puts the vertices into the primitive that is currently specified

VERTICES

Lab Into

Application setup Graphics pipeline Vertices

Fragments Buffers vs. textur Compiling shade

C++ cras

About Simplified memor

Stack

Classes

Parameters

Operator overloadir Output • 3 vertices in (x, y, z)

• Range is [-1, +1]

```
GLfloat vertices[] = {
    -0.5f, -0.5f, 0.0f,
    0.5f, -0.5f, 0.0f,
    0.0f, 0.5f, 0.0f
};
```

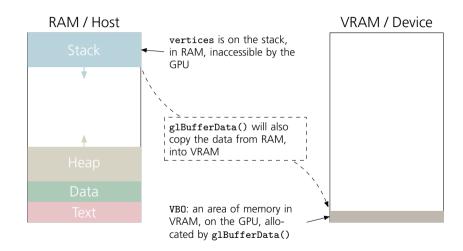
- Output from VS is in Normalized Device Coordinates (NDC)
 - Also [-1, +1]
 - Origin is in the middle of the screen
- Put vertices into Vertex Buffer Objects (VBO)

```
GLuint VBO;
glGenBuffers(1, &VBO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
```

FDAF80: Seminar 0

WHERE ARE YOUR VERTICES?

Vertices



SIMPLE VERTEX SHADER

- Vertices

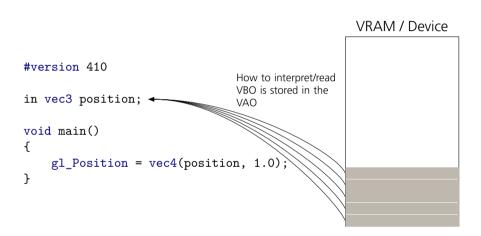
- Must set the predefined variable gl_Position
- Need to link vertex data to the vertex shader.
 - A Vertex Array Object (VAO) is also required

```
#version 410
in vec3 position;
void main()
   gl Position = vec4(position, 1.0);
```

FDAF80: Seminar 0

Vertices

HOW TO ACCESS THE VERTICES



SIMPLE FRAGMENT SHADER

Fragments

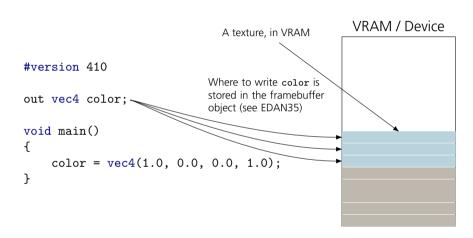
Requires one output variable of vec4, for the colour

```
#version 410
out vec4 color:
void main()
   color = vec4(1.0, 0.0, 0.0, 1.0); // set color to red
```

FDAF80: Seminar 0

WHERE DOES FRAGMENT OUTPUT GO?

Fragments





BUFFERS VERSUS TEXTURES

Ruffers us textures

Both reside in VRAM on the GPU

• Both represent a chunk of memory (both can be viewed as *n*-d arrays, with data in cells)

Buffers:

- Supports any data format (even custom)
- Only the cells can be read
- Stored linearly

Textures:

- Only specific data formats allowed
- You can read between cells. and get interpolated results
- Stored in tiles



OpenCl

Application setup Graphics pipeline

Fragments

Ruffers vs. textur

Compiling shaders Drawing

More info

C++ crash

course

Simplified memo

model Stack

Classes

Paramete

Operator overload

Operator overloadi Output

More info

COMPILING SHADERS

- Shaders run on the **GPU**, not CPU
- They are written in GLSL, which is C-based
- Like for CPUs, need to compile to machine-specific instructions
- Unlike CPUs, shader compilation is done at runtime by your GPU driver

COMPILING SHADERS

Lab info

Application setup Graphics pipeline Vertices

Buffers vs. textures Compiling shaders

Drawing

C++ crash

course

Simplified memory model Stack

Classes

Parameter:

Operator overloadii Output Done in two steps:

Compile each shader individually

```
glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
glCompileShader(vertexShader);
```

- Check for possible compile errors after glCompileShader()
- 2 Link all shaders into a single shader program

```
glAttachShader(shaderProgram, vertexShader);
glAttachShader(shaderProgram, fragmentShader);
glLinkProgram(shaderProgram);
glUseProgram(shaderProgram);
```

Check for possible linking errors after glLinkProgram()

DRAWING

Drawing

Tell OpenGL what to render

- glDrawArrays(GL_TRIANGLES, 0, 3);
- (what to draw, starting index, number of vertices)

MORE INFO

More info

https://learnopengl.com/

Joey de Vries

Most complete guide for modern OpenGL

https://open.gl/

Alexander Overvoorde

https://antongerdelan.net/opengl/

Anton Gerdelan

Lab info

OpenGL

Application setu Graphics pipelin

Fragmen

Buffers vs. texture Compiling shader

Drawing More info

C++ crash

course

Simplified memo

model

Heap and pointe

Classes

.

Paramet

Operator overloading

Operator overloadir

More info

```
#include <iostream>
```

```
int main()
{
    std::cout << "Hello world!\n";
    return 0;</pre>
```

Output

> Hello world!

ABOUT

About

- Based on C
- Create by Bjarne Stroustrup in 80's
- Object-oriented (classes and structs)
- Constructors & destructors
- Inheritance & virtual functions
- Operator overloading (+, -, *, /, etc.)
- Templates
- C++11 began a 3-year cycle of updates

EDAF80: Seminar 0

SIMPLIFIED MEMORY MODEL

Lab into

Application setup Graphics pipeline Vertices

Buffers vs. textur Compiling shade

C++ crack

course

Simplified memory

Stack

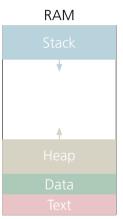
Classes

Arrays

Types

Operator overloadir Output

- Stack
 - Stores local variables
 - Managed by the compiler
- Heap
 - Dynamic memory
 - Managed by the programmer
- Data
 - Stores global variables
 - Initialized and uninitialized
- Text
 - Stores code being executed



EDAF80: Seminar 0

STACK INTEGER DECLARATION

Lab IIIIO

Opendi

Graphics pipeline

Fragment

Proffession Assets

bullers vs. textu

Drawing

More info

C++ crash

course

Simplified me

model Stack

Hean and n

Classes

Arrays

Types

Operator overloadi

Output

More info

int x;

std::cout << x;

Output

> 698683442



STACK INTEGER DECLARATION & INITIALIZATION

Lab into

Application set

Graphics pipelir

Eraamonto

Proffession Assets

Butters vs. textur

Drawing

More info

More into

C++ cras

Jourse

Simplified mei

Stack

Hean and noin

Heap and point

Arrauc

Parame

Operator overloadi

Outrast

More info

int x; x = 35;

std::cout << x;

Output

> 35

EDAF80: Seminar 0

POINTER TO AN INTEGER

Lab IIIIo

Application se

Graphics pipelin

Fragment

Buffers vs. textur

Compiling

More info

C++ crash

course

Simplifie

model Stack

Heap and pointers

.

Paramo

Operator overload

Operator overload

More info

int* y;

std::cout << y;</pre>

Output

> 0000000B2394FB09



ALLOCATE HEAP MEMORY

O-- --- C1

Application setup
Graphics pipeline

Fragments

Buffers vs. texture

Drawing

More info

C++ crash

About

Simplified memoral model

Heap and pointers

ricap and pointer

Arrays

Parame

Operator overloadi

Output

More info

```
int* y;
y = new int(10);
std::cout << y;</pre>
```

Output

> 000001D7AD807FA0

EDAF80: Seminar 0

POINTER DEREFERENCING

Lab into

Application setu Graphics pipelin

Fragments

Buffers vs. textur

Drawing

More info

C++ crash

course

Simplified memoral

Stack

Heap and pointers

A manua

Parame

Operator overload

Operator overloadi

More info

penGL

```
int* y;
y = new int(10);
```

std::cout << *y;

*y is dereferencing the pointer

Output

> 10

POINTER TO STACK INTEGER

int x = 35; int* xp;

Heap and pointers

```
EDAF80:
Seminar 0
```

POINTER TO STACK INTEGER

Heap and pointers

x is an int, not an int* (pointer to an int)

POINTER TO STACK INTEGER

Heap and pointers

int x = 35; int* xp = &x;

&x takes the address of x

std::cout << *xp;</pre>

Output

> 35

HEAP DEALLOCATION

Heap and pointers

int* y = new int(10);delete y;

Lab info

OpenGI

Application setu Graphics pipelin

Fragment

Buffers vs. texture

Drawing

More info

C++ crash course

About

Simplified memory model

loan and pointors

Classes

Parame

Operator querles

Output

More info

class MyClass

};

CLASS SCOPE

```
class MyClass
   // class scope
};
```

CLASS ACCESS SPECIFIERS

Classes

class MyClass

private:

// access within this class only (default)

protected:

// access to this and inherited classes

public:

// access to everyone

};

CLASS CONSTRUCTOR

Lab info

```
Classes
```

```
class MyClass
{
    float mX;

    MyClass(float x)
    {
        mx = x;
    }
};
```

EDAF80: Seminar 0

CLASS CONSTRUCTOR + INITIALIZATION

```
class MyClass
    float mX;
    MyClass(float x) : mX(x)
};
```

CLASS CONSTRUCTOR & DESTRUCTOR

Lab info

```
class MyClass
    float mX;
   MyClass(float x)
        mX = x;
    ~MyClass()
        // mX is on stack, so automatically deallocated
};
```

CLASS CONSTRUCTOR & DESTRUCTOR

Lab info

```
class MyClass
    float* mXp;
    MyClass(float x)
        mXp = new float(x);
    ~MyClass()
        delete mXp; // mX is on heap, so deallocate manually
};
```

CLASS MEMBER METHOD

Classes

};

float mX; void setX(float x)

mX = x;

class MyClass

CLASS MEMBER ACCESS

Lab info

Graphics pipeline Vertices Fragments Buffers vs. textures Compiling shaders Drawing

C++ crash

```
About
Simplified memo
```

Heap and pointers

Classes

Arrays

Types

Operator overloadir Output Stack

```
MyClass myclass = MyClass(5);
myclass.setX(2);
```

Heap

```
MyClass* myclassp = new MyClass(5);
myclass->setX(2);
...
delete myclassp;
```

```
FDAF80:
Seminar 0
```

CLASS DECLARATION + DEFINITION

Classes

```
MyClass.h
```

```
class MyClass {
    float mX;
    void setX(float x);
};
```

MyClass.cpp

```
#include "MyClass.h"
void MyClass::setX(float x) {
   mX = x;
```

ARRAY: STACK & HEAP ALLOCATION

Lab info

Application setup Graphics pipeline Vertices Fragments Buffers vs. texture

C++ crasl

course

Simplified memor model

Heap and pointer

Arrays

Types Operator overloadin Output

Stack

```
float numbers[3];
numbers[0] = 1.0f;
...
```

Stack: direct initialization

```
float numbers[3] = { 1.0f, 2.0f, 3.0f };
```

Heap

```
float* numbers = new float[3];
number[0] = 1.0f;
...
delete[] numbers;
```

```
FDAF80:
Seminar 0
```

PARAMETERS: VALUE, REFERENCE, POINTER

Parameters

```
MyClass mc0 = MyClass(1);
MyClass mc1 = MyClass(1);
MyClass* mc2 = new MyClass(1);
foo(mc0, mc1, mc2);
int foo(MyClass mc0, MyClass& mc1, MyClass* mc2)
   mc0.setX(10); // edits local copy only
```

mc1.setX(10); // edits original

mc2->setX(10); // edits original

TYPES

Lab info

Graphics pipeline Vertices

Fragments Buffers vs. texture

Compiling shade Drawing

C++ crack

course

Simplified men

model

Heap and point

Classes

Paran

Types Operator ove

Operator overloadin Output

int	a = -1;	(32 bits)
unsigned int	b = 1u;	(32 bits)
long	c = -21;	(64 bits)
unsigned long	d = 2lu;	(64 bits)
float	e = 1.0f;	(32 bits)
double	f = 3.14;	(64 bits)
bool	g = true;	(8 bits)
char	h = 'x';	(8 bits)
char	i[] = "abcd";	(5 * 8 bits)
		•••

```
FDAF80:
Seminar 0
```

OPERATOR OVERLOADING

Operator overloading

May customize +, -, *, /, and many others

• Very useful for linear algebra, e.g.:

```
glm::mat3 A, B;
glm::vec3 u;
glm::mat3 M = A * B;
glm::vec3 v = M * u;
```

OUTPUT

Lab info

Application setup Graphics pipeline Vertices Fragments Buffers vs. texture

C++ crack

course

Simplified memory model Stack

Heap and pointers Classes

Parameters Types

Operator overloadii
Output

More info

• Print "Rendering..." to standard output, followed by a new line:

```
std::cout << "Rendering...\n";</pre>
```

• Or, with the same result:

```
printf("Rendering...\n");
```

• Inclusion of variables (many formatting options available):

```
std::cout << "an integer: " << 1 << ", a float: " << 3.14f << '\n';
```

• Or, with the same result:

```
printf("an integer: %d, a float: %f\n", 1, 3.14f);
```

MORE INFO

Lab IIIIo

Application setup Graphics pipeline

Fragments Buffers vs. texture

Drawing

C++ cras

course

Simplified memory model

Stack

Classes

Arrays

Paramet

Operator overloadi

More info

• EDAF50 – C++ Programming

• https://cplusplus.com/

• https://en.cppreference.com/w/cpp