Introductory Seminar

EDAF80: Computer Graphics

Rikard Olajos

LUND UNIVERSITY
LTH
FACULTY OF ENGINEERING

AGENDA

1. Lab info
2. OpenGL
3. C++ crash course

Lab info

LABS OVERVIEW

- 1 optional + 5 mandatory assignments
  - Week 2 – 7
  - "Lab 0" in week 2: optional attendance
  - Book sessions on course homepage
  - Labs take more than the 2 hours per week, so make sure to start in advance
- Work in pairs
  - Both must understand, and be able to present, the work done
  - If looking for a partner, post on forum `#seeking-lab-partner`
- E:Pluto
  - Located in E-huset basement
  - Windows 11, 64-bit, Core i7, 16 GB RAM
  - GeForce RTX 4070 Ti
  - Visual Studio 2019

OpenGL

OPENGL

- 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)

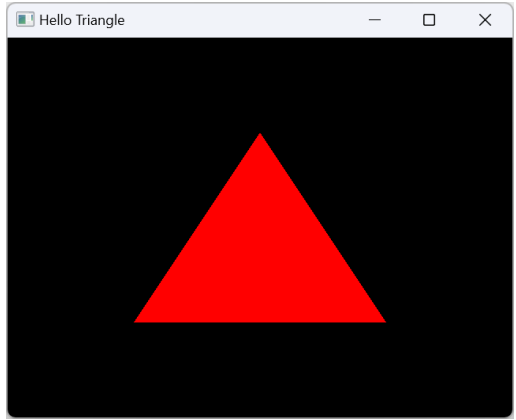
OPENGL

- 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

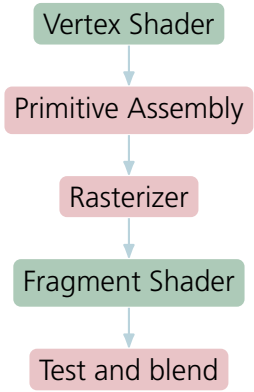
APPLICATION SETUP

- Create a window
  - Request from OS
  - Bind OpenGL context
- Create a `while` loop:
  - Grab inputs
  - Update scene objects
  - Render scene objects
  - Swap the buffers

HELLO TRIANGLE



GRAPHICS PIPELINE



- Shaders are programmable, other parts are not
- There are no default vertex and fragment shaders, you must provide them
- Vertex shader (VS) transforms vertices
- Primitive assembly (PA) connects vertices into primitives
- Rasterizer determines which pixels get cover by which primitive
- Fragment shader (FS) calculates colour of pixel
- Test and blend handles transparencies

VERTICES

- 3 vertices in (x, y, z)
  - Range is [-1, +1]

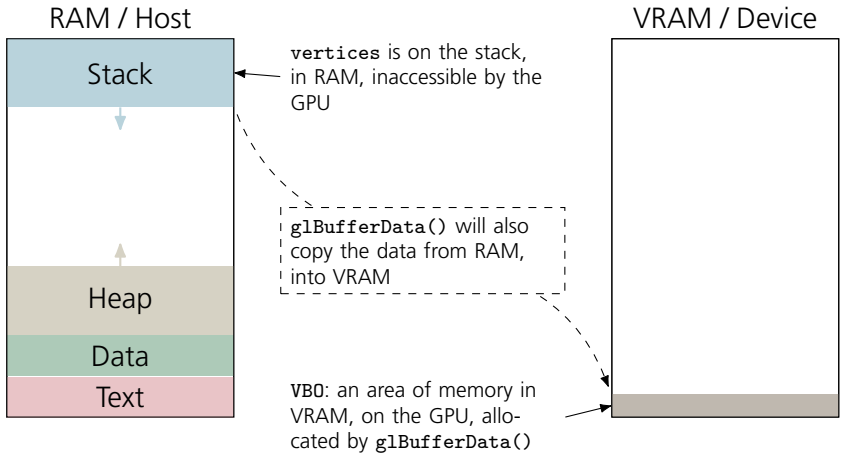
```
GLfloat vertices[] = {
    -0.5f, -0.5f, 0.0f,
     0.5f, -0.5f, 0.0f,
     0.0f,  0.5f, 0.0f
};
```

- 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);
```

OpenGL reference: `glBufferData()`

WHERE ARE YOUR VERTICES?



SIMPLE VERTEX SHADER

- 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);
}
```

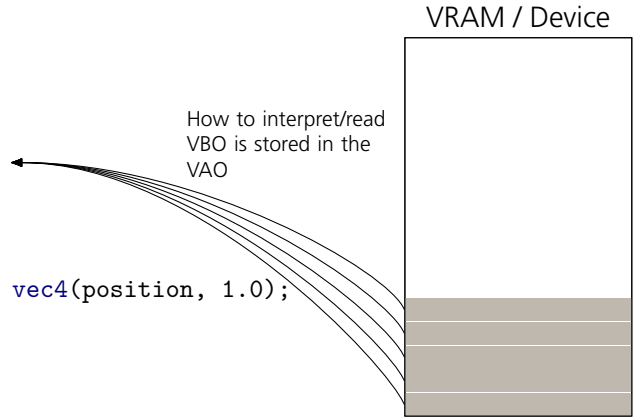
VERTEX ARRAY OBJECT

```
/* Create and bind a vertex array object */
GLuint VAO;
glGenVertexArrays(1, &VAO);
glBindVertexArray(VAO);

/* Detail how the vertex positions are laid out in the buffer */
GLuint index = 0;
glVertexAttribPointer(index, 3, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(index);

/* Bind the same index to the 'position' variable in the vertex shader */
glBindAttribLocation(shaderProgram, index, "position");
```

OpenGL reference: `glVertexAttribPointer()`

HOW TO ACCESS THE VERTICES

```
#version 410

in vec3 position;

void main()
{
    gl_Position = vec4(position, 1.0);
}
```



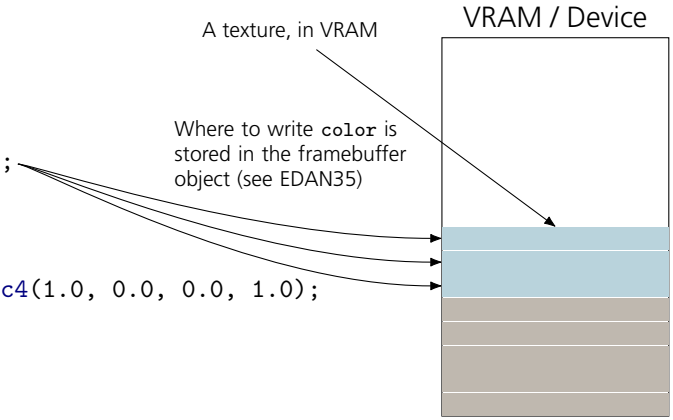
SIMPLE FRAGMENT SHADER

- 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
}
```

WHERE DOES FRAGMENT OUTPUT GO?

```
#version 410

out vec4 color;

void main()
{
    color = vec4(1.0, 0.0, 0.0, 1.0);
}
```



BUFFERS VERSUS 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

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

Done in two steps:

1. Compile each shader individually

```
GLuint vertexShader = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
glCompileShader(vertexShader);
```

…

   - Check for possible compile errors after `glCompileShader()`
   - Repeat for fragment shader

2. Link all shaders into a single shader program

```
GLuint shaderProgram = glCreateProgram();
glAttachShader(shaderProgram, vertexShader);
glAttachShader(shaderProgram, fragmentShader);
glLinkProgram(shaderProgram);
glUseProgram(shaderProgram);
```

…

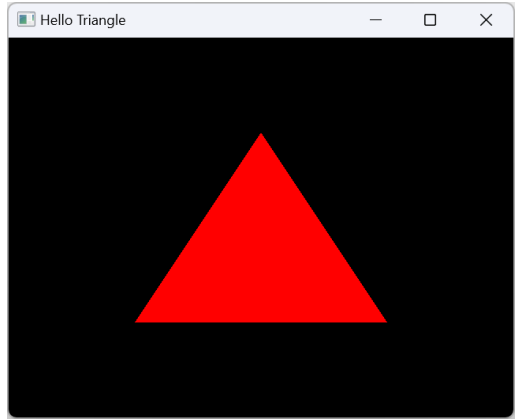
   - Check for possible linking errors after `glLinkProgram()`

DRAWING

- Tell OpenGL what to render
  - `glDrawArrays(GL_TRIANGLES, 0, 3);`
- All currently bound states are being used

OpenGL reference: `glDrawArrays()`

HELLO TRIANGLE (REVISITED)



Seminar Exercise 0-1: Hello Triangle!

- `https://cs.lth.se/edaf80/`
  - Assignments > Exercises

1. Move the triangle by changing the positions in the vertices list.
2. Change the appearance of the triangle by altering the colour in the fragment shader.

FOLLOW-UP QUESTIONS

1. How big is the viewport?
2. What happens when you change the z-coordinate?
3. How do you make the triangle yellow?
4. What happens with colour values above 1.0?

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

C++ crash course

HELLO C++

```cpp
#include <iostream>

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

Output

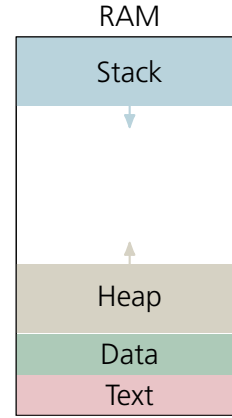
```
> Hello world!
```

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

SIMPLIFIED MEMORY MODEL

- 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



STACK INTEGER DECLARATION

```
int x;

std::cout << x;
```

Output

```
> 698683442
```

STACK INTEGER DECLARATION & INITIALIZATION

```
int x;
x = 35;

std::cout << x;
```

Output

```
> 35
```

STACK INTEGER DECLARATION & INITIALIZATION

```
int x = 35;

std::cout << x;
```

Output

```
> 35
```

POINTER TO AN INTEGER

```
int* y;

std::cout << y;
```

Output

```
> 0000000B2394FB09
```

ALLOCATE HEAP MEMORY

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

std::cout << y;
```

Output

```
> 000001D7AD807FA0
```

POINTER DEREFERENCING

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

std::cout << *y;
```

Output

```
> 10
```

`*y` is dereferencing the pointer

HEAP DEALLOCATION

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

POINTER TO STACK INTEGER

```
int x = 35;
int* xp;
```

POINTER TO STACK INTEGER

```
int x = 35;
int* xp = x; // Wrong!
```

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

POINTER TO STACK INTEGER

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

`&x` takes the address of `x`

```
std::cout << *xp;
```

Output

```
> 35
```

POINTER TO STACK INTEGER

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

*xp = 80;
std::cout << x;
```

Output

```
> 80
```

POINTER TO STACK INTEGER

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

*xp = 80;
std::cout << x;
```

Output

```
> 80
```

This is what we use to initiate most handles in OpenGL

```
GLuint VBO;
glGenBuffers(1, &VBO);
```

C++ CLASS

```
class MyClass
{

};
```

CLASS SCOPE

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

CLASS ACCESS SPECIFIERS

```
class MyClass
{
private:
    // access within this class only (default)
protected:
    // access to this and inherited classes
public:
    // access to everyone
};
```

CLASS CONSTRUCTOR

```
class MyClass
{
    float mX;

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

CLASS CONSTRUCTOR + INITIALIZATION

```
class MyClass
{
    float mX;

    MyClass(float x) : mX(x)
    {

    }
};
```

CLASS CONSTRUCTOR & DESTRUCTOR

```
class MyClass
{
    float mX;

    MyClass(float x)
    {
        mX = x;
    }

    ~MyClass()
    {
        // mX is on stack, so automatically deallocated
    }
};
```

CLASS CONSTRUCTOR & DESTRUCTOR

```
class MyClass
{
    float* mpX;

    MyClass(float x)
    {
        mpX = new float(x);
    }

    ~MyClass()
    {
        delete mpX; // mpX is on heap, so deallocate manually
    }
};
```

CLASS MEMBER METHOD

```
class MyClass
{
    float mX;

public:
    float getX()
    {
        return mX;
    }

    void setX(float x)
    {
        mX = x;
    }
};
```

Stack

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

Heap

```
MyClass* pMyClass = new MyClass(5);
pMyClass->setX(2);
…
delete pMyClass;
```

CLASS DECLARATION + DEFINITION

MyClass.h

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

MyClass.cpp

```
#include "MyClass.h"

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

RAW ARRAYS: STACK & HEAP ALLOCATION

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;
```

STL ARRAYS: VECTOR & ARRAY

Includes

```
#include <array>
#include <vector>
```

Initialization

```
std::array<int, 3> arr;      // Static array with 3 integers
std::vector<float> vec;      // Dynamic array with floats (on the heap)
```

Element access & size

```
arr[0] = 1;                  // Set first element to 1
vec.push_back(1.0f);         // Add 1.0f to end of vector
std::cout << vec[0];         // Print first element of 'vec'
std::cout << vec.size();     // Print number of elements in 'vec'
```

PARAMETERS: VALUE, REFERENCE, POINTER

```
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      (pass by value)
    mc1.setX(10);  // edits original        (pass by reference)
    mc2->setX(10); // edits original        (pass by pointer)
}
```

TYPES

```
int                a = -1;          (32 bits)
unsigned int       b = 1u;          (32 bits)
long               c = -2l;         (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)
…                  …                …
```

C++ reference: Fundamental types
C++ reference: Fixed width integer types

- Have to use: `#include <cstdint>`

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

- Include header for `std::cout`

```
#include <iostream>
```

- or, for `printf()`

```
#include <cstdio>
```

OUTPUT

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

```
std::cout << "Rendering...\n";
```

- 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

- `https://en.cppreference.com/w/cpp`
- `https://cplusplus.com/`
- EDAF50 – C++ Programming

Seminar Exercise 0-2: Vertex ID

1. Move the triangle by offsetting the `gl_Position`.
   - `gl_Position.x += ...`
   - `gl_Position += vec4(...)`
2. Without changing the vertices list, move the top vertex of the triangle using `gl_VertexID`.

FOLLOW-UP QUESTIONS

1. Can you find any alternative ways of altering `gl_Position`?
2. In which order are the three vertices of the triangle ordered?