Computer Graphics and GPGPU Programming

Donato D'Ambrosio

Department of Mathematics and Computer Science
Cubo 30B, University of Calabria, Rende 87036, Italy
mailto: donato.dambrosio@unical.it
homepage: http://www.mat.unical.it/~donato

Academic Year 2019/20

Course Introduction

Course Introduction

- Master degree course in Computer Science
  - *Artificial Intelligence and Games* profile
  - *Data Science* profile
- Main topics
  - Computer Graphics programming in OpenGL core profile
  - Global illumination (Ray Tracing)
  - GPGPU programming in OpenCL
- Prerequisites: basic level C/C++ programming
- Suggested resources/textbooks
  - Learn OpenGL web course (`http://learnopengl.com/`)
  - Course slides about fundamental algorithms of Computer Graphics
  - Introduction to Ray Tracing (`http://www.scratchapixel.com/`)
  - *Heterogeneous Computing with OpenCL 2.0*, D. Kaeli et al., Elsevier
  - Ray tracing with OpenCL (`https://www.gamedev.net/blogs/entry/2254170-realtime-raytracing-with-opencl-i/`)

Course Introduction

- Further readings
  - OpenGL Programming Guide Eighth Edition, Dave Shreiner et al., Addison-Wesley
  - OpenCL Programming Guide, A. Munshi, B.R. Gaster, T.G. Mattson, J.Fung, D. Ginsburg, Addison Wesley
  - 3D Computer Graphics, Alan Watt, Pearson, Addison-Wesley
  - OpenCL in Action, M. Scarpino, Manning
- Exam
  - Written examination (about 10 questions on Cmputer Graphics and GPGPU computing)
  - Student project (a 3D Copmuter Graphics project in OpenGL core profile or GPGPU one in OpenCL)
- Office ours: Thursday, from 16:30 to 17:30, Cubo 30B (in case of special needs, send me an email)

Table of contents

Introduction

Introduction

OpenGL core and compatibility profile

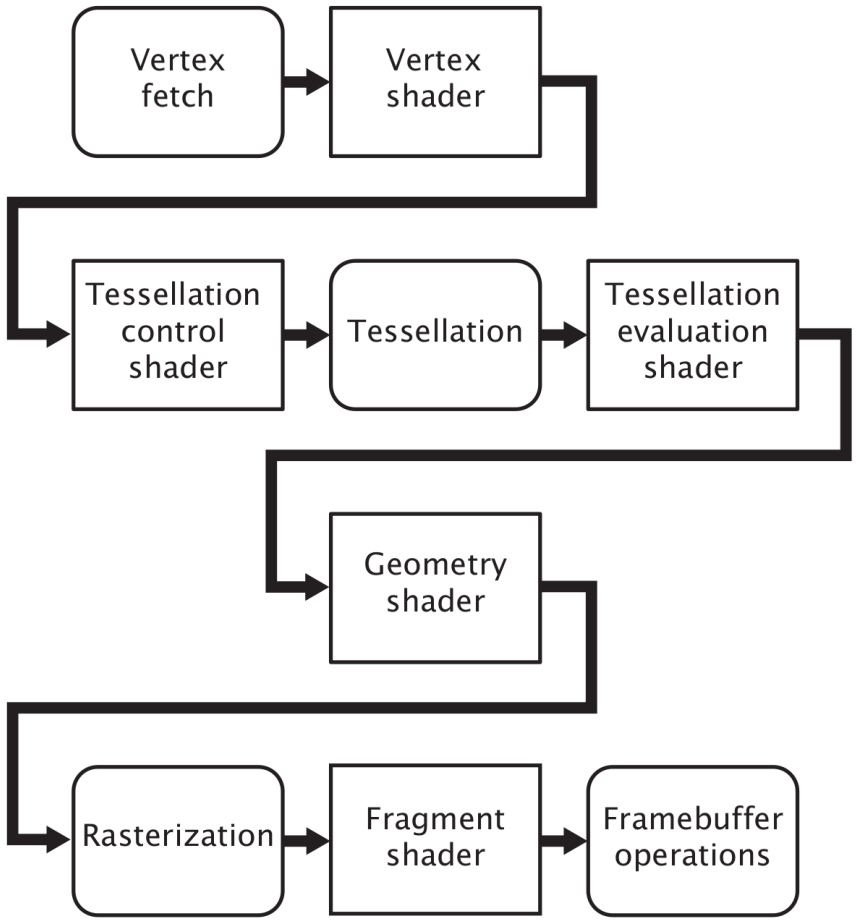
- OpenGL is a standard API defined by Khronos Group that applications can use to access and control the graphics subsystem (i.e. the Graphics Processing Unit, or GPU)
- Originally developed by Silicon Graphics (SGI), the first open (1.0) version was released in June of 1992
- In 2008, with the 3.3 specification, the Architecture Review Board (ARB) decided it would fork OpenGL into two profiles: core (strongly recommended) and compatibility

OpenGL core and compatibility profile

- The compatibility profile maintains backwards compatibility with all revisions of OpenGL back to version 1.0
- On some platforms, newer features are only available if you are using the core profile of OpenGL
- Application written using the core profile of OpenGL will run faster

OpenGL Shaders and the Graphics Pipeline

- The graphics system is broken into a number stages, each represented either by a programmable shader (square boxes) or by a fixed-function (rounded boxes)
- The minimal useful pipeline configuration consists only of a vertex shader (or just a compute shader), but if you wish to see any pixels on the screen, you will also need a fragment shader

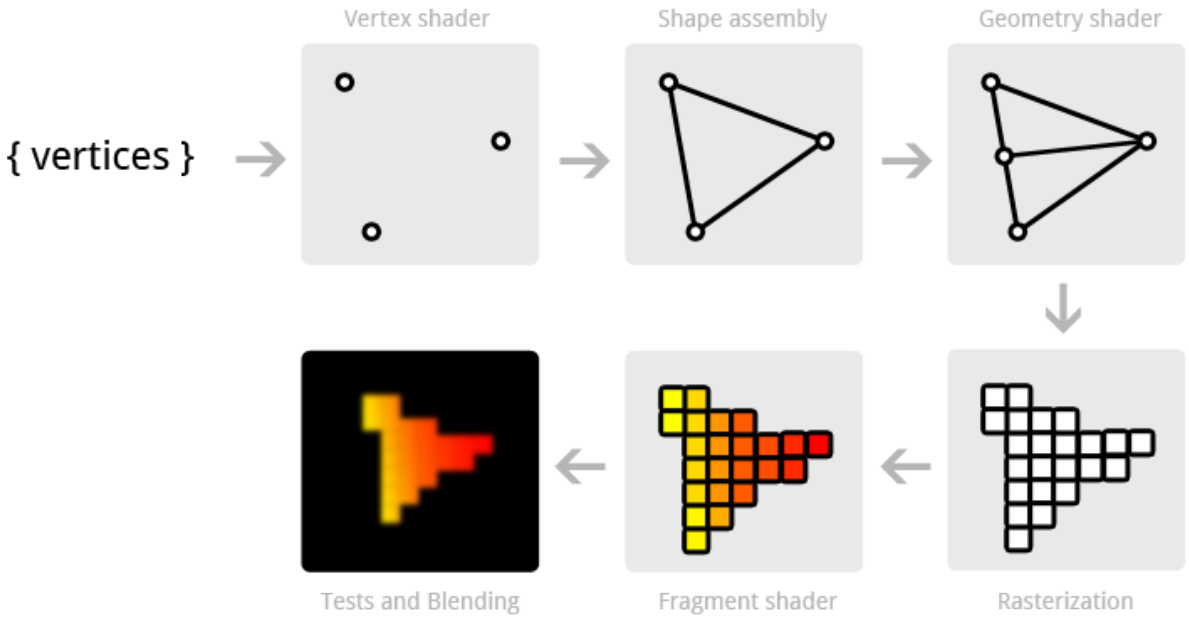


Graphics Pipeline's Front and Back End

- The graphics pipeline is broken down into two major parts
- The first part, often known as the front end, is constituted by the vertex, tessellation and geometry shaders and processes vertices and primitives, eventually forming them into the points, lines, and triangles that will be handed off to the rasterizer. This is known as primitive assembly
- After the rasterizer, the geometry has been converted from what is essentially a vector representation into a large number of independent pixels
- These pixels are handed off to the back end, which includes depth and stencil testing, fragment shading, blending, and updating the output image

Primitives, Pipelines, and Pixels

- The fundamental unit of rendering in OpenGL is known as the primitive (such as points, lines, triangles and polygons)
- Each primitive is basically defined by its verteices, each one defining information about a point into the 3D world such as position, color, besides other (that we will see later), and then processed by the OpenGL pipeline to produce the final image



Rasterizers (GPUs)

- Modern GPUs consist of thousands of small programmable processors called shader cores which run mini-programs called shaders

Rasterizers (GPUs)

- Here you can find Nvidia GTX 1080 tech specs: `http://international.download.nvidia.com/geforce-com/international/pdfs/GeForce_GTX_1080_Whitepaper_FINAL.pdf`
- For instance, vertices are processed (for instance, each of them is translated from its current position to anoter one into the 3D space) in parallel, each one by a different shader core
- There is no interaction between them, so that they can be processed concurrently without the need of inter communication among shader cores
- However, since OpenGL acts as an abstraction layer, applications do not need to know details about the graphics processor: who made it, how many cores it is made by, how it works, or how well it performs

Practice

- Now, it's time to draw your first triangle

- For this purpose, we will consider the GLFW and GLAD (or even GLEW) APIs and CMake to build our projects
- Later, we will introduce other libs, e.g. GLM (for math purposes)

Table of contents

OpenGL Core Profile

OpenGL Core Profile

LearnOpenGL Table of Contents

Basics

- Introduction to OpenGL `https://learnopengl.com/Getting-started/OpenGL`
- Creating a window `https://learnopengl.com/Getting-started/Creating-a-window`
- Hello Window `https://learnopengl.com/Getting-started/Hello-Window`
- Hello Triangle (shaders, vertex attributes, VBO, EBO, etc.) `https://learnopengl.com/Getting-started/Hello-Triangle`

GLSL (shaders and data)

- Shaders (types, uniforms, more attributes, etc.) `https://learnopengl.com/Getting-started/Shaders`
- Textures `https://learnopengl.com/Getting-started/Textures`

LearnOpenGL Table of Contents

Tranformations

- Transformations (some math) `https://learnopengl.com/Getting-started/Transformations`
- Coordinate Systems (projections, hidden surface removal, etc.) `https://learnopengl.com/Getting-started/Coordinate-Systems`
- Camera abstraction `https://learnopengl.com/Getting-started/Camera`

LearnOpenGL Table of Contents

Lighting

- Colors `https://learnopengl.com/Lighting/Colors`
- Phong lighting model
  `https://learnopengl.com/Lighting/Basic-Lighting`
- Materials
  `https://learnopengl.com/Lighting/Materials`
- Lighting maps
  `https://learnopengl.com/Lighting/Lighting-maps`
- Light casters
  `https://learnopengl.com/Lighting/Light-casters`
- Multiple lights
  `https://learnopengl.com/Lighting/Multiple-lights`
- Blinn-Phong model
  https://learnopengl.com/Advanced-Lighting/Advanced-Lighting

LearnOpenGL Table of Contents

Model loading

- Assimp
  `https://learnopengl.com/Model-Loading/Assimp`
- Mesh class
  `https://learnopengl.com/Model-Loading/Mesh`
- Model class
  `https://learnopengl.com/Model-Loading/Model`

Optional

- Instancing `https://learnopengl.com/Advanced-OpenGL/Instancing`
- Shadow mapping `https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping`

Algoritms Table of Contents (slides)

Clipping

- Cohen-Sutherland

Rasterization

- DDA
- Bresenham's line algorithm
- Scanline polygon algorithm

Hidden Surface Removal

- Culling
- Z-buffer

Lightinh

- Phong
- Ray-tracing

OpenCL Table of Contents (slides)

Hands On OpenCL `https://handsonopencl.github.io/`

- Introduction to Heterogeneous Parallel Computing
- An overview of OpenCL
- Important OpenCL concepts
- Overview of OpenCL APIs
- Introducing OpenCL kernel programming
- Understanding the OpenCL memory hierarchy
- Synchronization in OpenCL