15-462: Intro to OpenGL
Adrian Biagioli
Graphics APIs

- A **graphics api** provides an abstraction over common rendering operations.
  - Important: **you don’t need a graphics API to do graphics!**
- Why use a Graphics API instead of writing it yourself?
  - GPUs are designed around them leading to massive speedups (this is what graphics drivers do!)
  - Standardization of Graphics APIs leads to better debugging and tooling (why DirectX is so popular!)
  - It takes way less time
Rasterization vs Pathtracing

**Rasterization**
Transform scene geometry via matrix operations to screen space, then use triangle fill algorithm.

*Optimized for performance*

**DrawSVG (A1)**

---

**Pathtracing**
Bounce simulated rays of light throughout your scene randomly for each pixel, and illuminate if it eventually intersects a light.

*Optimized for realism*

**Pathtracer (A3)**
Common Graphics APIs

- **OpenGL**: Runs on all platforms, but old, slow, and falling out of fashion. Mac support ending soon.
  - OpenGL ES: Subset of OpenGL for mobile GPUs
  - GL 1.1 != GL 2.1 != GL 3.3 != GL 4.6
    - **Massive** breaking API changes between GL versions — hard to find tutorials!
    - … and that’s not even counting extensions!
- **Vulkan**: Modern Graphics API, runs on every platform (macOS needs a Metal wrapper)
- **DirectX**: Windows / Xbox only, very popular in Game development due to engine support and tooling
  - DirectX 9: Used on Xbox 360 / WinXP, similar to GL 2.x
  - DirectX 10: Used on Xbox 360 / Vista, similar to GL 3.x
  - DirectX 11: Used on Xbox 360 / Xbox One / Win7, similar to GL 4.x
  - DirectX 12: Used on Xbox One, similar to Vulkan
- **Metal**: Apple’s low level graphics API for iOS / Mac
Choosing a Graphics API

• The graphics API you should use depends on:
  • Platform(s) you are publishing on (including OS!)
    • Example: On Windows DirectX performs better than OpenGL due to driver support
  • Specific API Features
    • Example: Vulkan offers lower level control of GPU memory than OGL, but may be harder to use
    • Whatever new hotness comes out (cough raytracing) tends to take a while to arrive to all graphics APIs
  • Your own preference / familiarity
    • Much more important when considering shader languages
drawsvg

- 5% of the assignment: regurgitate this recitation in `hardware_renderer.cpp`

- Everything we talk about today, up to blending

- 95% of the assignment: reimplement OpenGL calls in software!
Code!
OpenGL 2.x Render Pipeline

ES2.0 Programmable Pipeline

API → Primitive Processing → Vertices → Vertex Shader → Primitive Assembly → Rasterizer

Triangles/Lines/Points

Vertex Buffer Objects

Fragment Shader

Depth Stencil → Colour Buffer Blend → Dither → Frame Buffer
Wrap-up

• OpenGL and similar APIs are the bread and butter of practical computer graphics, so start learning them!
  
  • Recommend: 15-466 Computer Game Programming which uses OpenGL 3.3

  • Great tutorial for modern OpenGL: http://learnopengl.com

• Not mentioned today: **Shaders** are highly parallel code compiled for the GPU. Modern graphics libraries use shaders to implement many common GL 2.1 “fixed function” effects.
  
  • http://shadertoy.com (simple example here)

• There is a shading language for all modern graphics apis
Example Code:

https://github.com/Flafla2/GLTutorial