EDUCATORS
SUMMIT

Teaching Modern Graphics: A Shader-first Approach

Dr. Sajid Farooq
Professor, Champlain College

GDC
GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19

Reminders

- Turn off any phones etc
- Pleas fill evaluations
- Probably no time for questions, so meet me for wrap-up the overlook

About me

- Professor of Game Programming at Champlain College
- PhD in Graphics from the University of Glasgow
- Team-Leader (XpoSim), Owner Ra'ed Entertainment
- Teaching for nearly 15 years, around the world
  - Malaysia, Pakistan, UK, USA

Why do we need this talk?

- Students face common *patterns* of problems in learning, regardless of geography
- Identified those problems in a Graphics context
- I present *principles* to solve the problems
- The principles are *not* specific to graphics

Caveats

- Graphics from a “gaming” perspective
- Non-realtime graphics (films etc) is very different
- Graphics from a “learning” perspective
- Shifts do not necessarily represent historical accuracy

Problem: Shifts in landscape

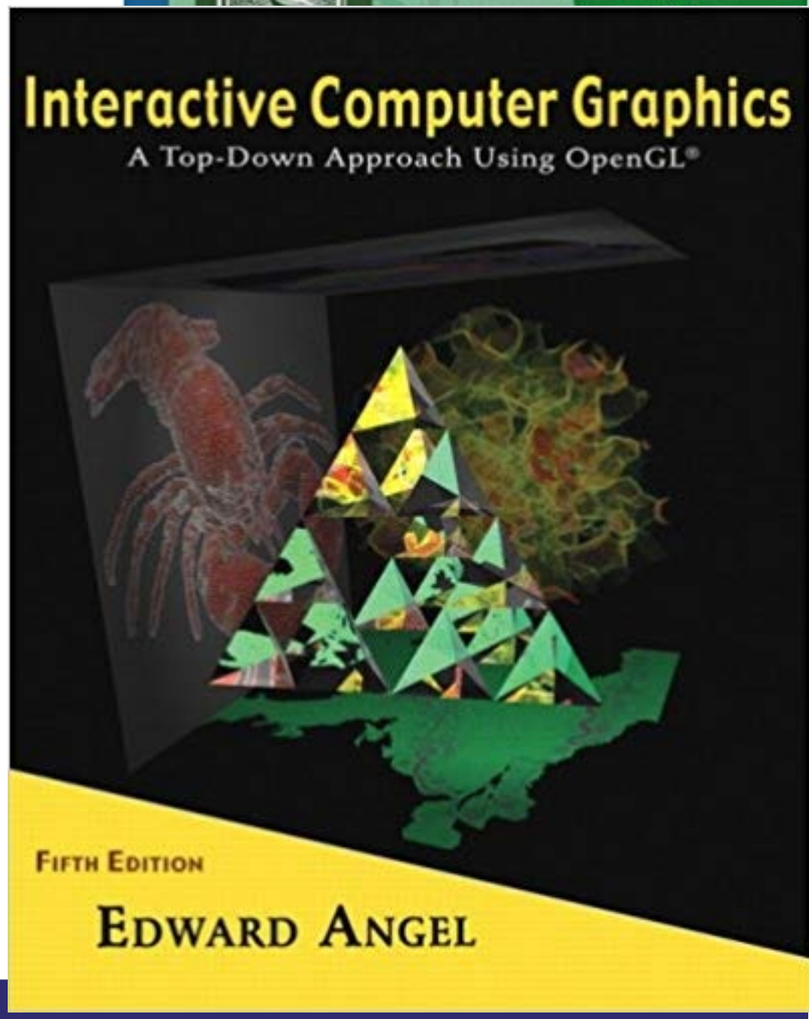
- Old days: Assembly. CPU optimization.
  - CPU architecture
  - Rasterization: (Line drawing -Bresenham’s algorithm)
  - Point->Lines->Shapes->Filling->Transformations

Shift 1: API Wars and the rise of 3D

- Real-time 3D goes mainstream
- Too many new topics to teach so lets use the library/API.
- Teach how the “library” is implemented: points->Lines…

Shift 2: The GPU

- API is not faithful anymore: Disconnect between how we teach and how API “actually” works
  - API could be running on CPU or perhaps GPU
  - New concepts: Display Lists, Draw Calls etc. Don’t make sense in the traditional CPU-only sense
- Solution: pretend everything is run on the CPU. API can emulate CPU.



Shift 3: The Programmable Pipeline

- Majority of graphics code is “shaders”, rest is glue.
- API no longer able to pretend:
  - No glMatrixMode, no glRotate, no CPU math
- Shaders are NOT CPU code, not emulated, entirely parallel.
- Shaders require thinking in parallel. Our points->Lines->Shapes thinking breaks down. E.g: In the FS, is line drawing in a for-loop?

Shift 3: The Programmable pipeline

- We still start with the CPU! We teach:
  - Setting up OpenGL etc (GLEW/SDL/GLFW)
  - Math libraries
  - Setting up data (all the buffers)
  - Setting the correct state
  - Sending data to the GPU, and then weeks before first shader…
  - Real-world: Most of these are done once and wrapped in boilerplate.

Shift 4: Zero overhead/Low overhead Libs

- Even more front-loaded
- Now even the commands and state have to be pre-recorded, pre-validated, and set up
- A month (or more) before first hello-triangle!

How do we solve this?

- Remember: Most game programmers will interact with shaders far more than glue/pipeline code.
- So: Are there tools that allow us to “jump” to shaders first, skipping boiler-plate until we need it?

The “backwards” approach:

1. Start with the modern and major technique (parallelism in this case) as a basis. In graphics: Shader-first approach.
2. Start with the output first, and move towards how we got to it (i.e, backwards). In graphics: Fragment shader first, then vertex, then CPU, and so on.
3. **(a)** Find tools relevant to the domain to "simplify" obtaining the final results at first (see point 3.b), then slowly let go of each tool (the crutches/training wheels).

(b) Focus on "interactivity first, explain later" approach: Use tools that focus on interactivity, practice, and immediate visual results (ShaderToy in Graphics).

Graphics

- We start with the fragment shader (1&2) (using shader toy as a crutch), and pretend the world is 2D.
- Students get familiar with fundamental graphics concepts (like drawing shapes, rasterization, image processing, and blending).
- Then, we reveal that the "canvas" that they have been drawing on is in-fact a texture, mapped on a polygon in 3D space. This provides an intuitive introduction to vertex shaders (using KickJS Shader Editor as a crutch).
- We proceed to teaching remaining topics (transformations, projections) thereby completing in initial pass of the modern programmable pipeline.
- Now that students have already written their own shaders, we get rid of the crutches and teach students how to perform the "grunt-work" of setting up up the environment in C++ so they don't need ShaderToy or other tools to run their shaders and pass data to it from the CPU.
- I sometimes add an additional step/crutch of using Unity to do this before heading off completely to C++.

Results

- At Champlain College, we switched to the shader-first approach two years ago.
- Results are dramatically different:
  - Students fail less
  - learn more
  - are far more confident about graphics
  - They are able to produce far more complex graphical projects by the end of their first year.

Results: Fragment Shader topics covered

- First Shader: Hello world
- Colors and Gradients
- Conditionals (quadrants)
- Textures
- Convolution (Blurring, Sharpening etc)
- Shapes
- Blending/Compositing

Results: Vertex Shader topics covered

- Vertex Shader: Hello world
- Transformations
- Homogenous coordinates
- Projection Matrix (from scratch)
- Viewing/Camera transformations
- Passing data from the CPU (Unity/C++)

Take-aways

- Make the modern technique the "basis" rather than treating it like a "latch-on".

- Start with the end result, and move backward. This way, students always see what they are expected to reproduce. In this particular case, it's also simpler.

- Allow students to "play", i.e, learn by "interaction". This is the fastest way to get them up to speed on modern techniques.

Other Domains

- Luckily, domain-specific tools exist now to allow us to do that for almost any field:
  - Desmos for Math
  - ShaderToy for Fragment Shaders and/or tech-art,
  - Tech.io and Repl.it for Programming,
  - Unity or Unreal for game design.

- Use them. This is perfectly complimentary to the notion that all learners fall into the Visual, Auditory, or Kinesthetic kind. A lecture is auditory, while the output and interaction of an interactive tool are the Visual and Kinesthetic component.

Graphics

- Domain: Graphics
- The modern technique: Shaders (parallel programming)
- The interactive tools: ShaderToy, KickJS Shader Editor, Unity
- Backward approach: Fragment Shader, then Vertex Shader, then Unity, then C++

Art

- Domain: Art
- The modern technique: Procedural Techniques
- The interactive tools: Unreal Engine
- Backward approach: Start with Material Editor, then Construction Scripts, then custom node Blueprints, then c++

Math

- Domain: Math
- The modern technique: Clifford Algebra (also known as Geometric Algebra)
- The interactive tools: GAViewer, Desmos
- Backward approach: Perform basic arithmetic in 4 dimensions directly in GAviewer then move to doing it manually.

Resources

- Learning Shaders:
  - https://thebookofshaders.com/
- Shader Tools:
  - https://www.shadertoy.com/
  - http://www.kickjs.org/example/shader_editor/shader_editor.html
- Programming Tools:
  - https://tech.io/
  - https://repl.it/
- Math Tools:
  - http://tobyschachman.com/Shadershop/
  - https://www.desmos.com/calculator
  - http://www.geometricalgebra.net/gaviewer_download.html

Contact

- sfarooq@champlain.edu

- Wrap-up at the overlook