Programmability of Graphics Pipelines

i3D 2018
Aras Pranckevičius, Unity

- Internal build systems engineer
  - What does that have to do with graphics?
    - Nothing! ...however
Aras Pranckevičius, Unity

- Internal build systems engineer
  - What does that have to do with graphics?
    - Nothing! ...however

- At Unity since 2006
- Been doing graphics until 2017
- Unity might be the most widely used graphics engine in the world
Maybe a problem?

Increasing Fidelity
CPU Rendering (Elite, 1984)
CPU Rendering
(Magic Carpet, 1995)
CPU Rendering (Descent, 1995)
GPU DX7 level: T&L (Quake III, 1999)
GPU DX7 level: T&L
(Black & White, 2001)
GPU DX9 level: Shaders
(Far Cry, 2004)
GPU DX9 level: Shaders
(TES IV: Oblivion, 2006)
GPU DX10 level: More Shaders
(Mirror’s Edge, 2008)
GPU DX10 level: More Shaders (Bioshock Infinite, 2013)
GPU DX11 level: (Compute) Shaders (Rise of the Tomb Rider, 2016)
GPU DX11 level: (Compute) Shaders (Dreams, 2018)
Increasing Complexity
2000, Fixed Function: Simple

- Simple model: render states
- States are composable
2002, Shaders

• Lost composability aspect :(  
• Uber-shaders, shader variants, preprocessor, branching, ...
2009, Compute Shaders

- Now you program 1500-thread machines
- Good luck, have fun!

- Now you write half of the driver
- Good night, and good luck
2018, Raytracing (DXR, etc.)

- Maybe this one will actually make things easier
- ...eventually
Composability
“I want fog” in the days of yore

- glEnable(GL_FOG); // OpenGL 1.x
- dev->SetRenderState(D3DRS_FOGENABLE, TRUE); // D3D9 SM2.0
“I want fog” in shaders

- Have to modify **all shaders**, and add fog code in there
- 2x more variants, with & without fog code?
- A branch inside the shader?
- Specialization constant?
Modify all the shaders

- So we end up building abstractions in our shader code
  ```
  UNITY_APPLY_FOG(i.fogCoord, col);
  ```
  ...and the same for a whole bunch of other “states”

- Now our abstractions are project/engine-specific :(
- Shaders are not transferable across different tech stacks :(
Shaders are a big blob

- Large part of lost composability is the fact that a shader has to do “everything”
- All the code effectively inlined
- Previous attempts at fixing this (fragments, interfaces, subroutines) not successful
- Maybe with DXR & other raytracing APIs we’ll get “callable shaders”?
  - See also: “Hacking GCN via OpenGL” by Stachowiak https://h3r2tic.github.io/
Other complexities
Other axes of complexity

- Platforms
- Graphics APIs
- Hardware performance variety
- Hardware featureset variety
- Flexibility
All that stuff is complex!

- Research can ignore some of complexity
- “Production” often can not :(
Goal

Easy innovation in graphics techniques
Sharing of reproducible data

- Ability to validate research findings is critical for adoption
- Please!
  - Share your research code + data
  - We don’t really care if your code is “messy” or “not nice”
Lower amount of unrelated busywork

● Essential vs accidental complexity
● Modern APIs like Vulkan or D3D12 need a lot of plumbing
  ● Should not need 10 years of D3D experience to come up with a better BRDF
Game engines & frameworks a good fit!
Unity

- Popular, free version, tools, asset pipeline, platforms
- Fast iteration times
- Allows customizing rendering & shaders quite a lot
  - Even more so with Scriptable Render Pipelines (see later...)
NVIDIA Falcor & Slang

- [https://github.com/nvidiagameworks/falcor](https://github.com/nvidiagameworks/falcor)
  - D3D12 (including DXR) & Vulkan
  - Research & prototype oriented
- [https://github.com/shader-slang/slang](https://github.com/shader-slang/slang)
  - Extended HLSL
  - WIP, might not be production ready at the moment
Microsoft MiniEngine

- [https://github.com/Microsoft/DirectX-Graphics-Samples](https://github.com/Microsoft/DirectX-Graphics-Samples)
- “A DirectX 12 Engine Starter Kit”
bgfx

- [https://github.com/bkaradzic/bgfx](https://github.com/bkaradzic/bgfx)
  - Rendering library with many API backends/platforms
  - Bindings for many programming languages too!
Sokol

- [https://github.com/floooh/sokol](https://github.com/floooh/sokol)
- Minimalistic C (*not* C++) graphics API / app model wrapper
- D3D11, Metal, GLES3, GLES2
G3D Innovation Engine

- [https://casual-effects.com/g3d](https://casual-effects.com/g3d)
- Research oriented 3D engine
Shadertoy

- [https://www.shadertoy.com/](https://www.shadertoy.com/)
  - In-browser, shareable experiments
  - If your problem can be expressed in one/several shaders with WebGL limitations
Unity & Scriptable Render Pipeline (SRP)
Graphics Engine Pipeline

- Render State Management
- Resources
- Render Surfaces
- Device Abstraction
- Shader Management
- Job Management

Graphics API Layers
Graphics Engine Pipeline

- Engine Abstractions
- Graphics API Layers
- Feature Renderers (Mesh, Skinned, Particles, ...)
- Culling
- Light Management
- Render Jobs
- Command Buffer Generation
- Synchronization
Graphics Engine Pipeline

- Render Passes
- Engine Abstractions
- Graphics API Layers

- Rendering Passes
- Postprocessing
- View Management
- Camera Setup
Graphics Engine Pipeline

- Game Logic Based Control
- Render Passes
- Engine Abstractions
- Graphics API Layers

- Responding to in-game events
- Content based rendering choices
Graphics Engine Pipeline

- Game Logic Based Control
- Render Passes
- Engine Abstractions
- Graphics API Layers

“Interesting Bits”

“Plumbing”
Traditional development

Game Logic Based Control

Render Passes

Engine Abstractions

Graphics API Layers

C++

Shading language (HLSL, …)
Traditional development

- Slow iteration

- *(Unity specific)* rift between “engine dev” (C++) and “users” (C#)
High/Low level split from research community

- Python for high level, NumPy/TensorFlow/CUDA for low level
- R, MATLAB, Octave, Mathematica
High/Low split in graphics

- ATI demo engine Sushi (2003)
- Bitsquid/Stingray data driven renderer (2011)
- Destiny’s rendering architecture (2015)
- Frostbite Framegraph (2017)
Styles of games made with Unity
Styles of games made with Unity
Styles of games made with Unity
Styles of games made with Unity
Problem

Hard to serve all of them with one render pipeline
Traditional render pipeline in Unity

- Forward or Deferred
- A whole bunch of options & knobs
- Shaders mostly customizable
- Render pipeline itself less so
- Black box, complex, fragile
- Still enables all these different games, so that’s good :)}
Our wishes

- Lean
- User centric
- Optimal
- Explicit
Scriptable Render Pipelines! (SRP)

Game Logic Based Control

Render Passes

Engine Abstractions

Graphics API Layers

C#

C++ / Engine

Shading language (HLSL, ...)

unity
SRP Concept

- What to render: culling/filtering. World -> sets of objects
- Render: draw sets of objects with some flags/params
- Setup render passes around all that
- Setup per-frame/renderpass data

https://blogs.unity3d.com/2018/01/31/srp-overview/
SRP High/Low Level Split

- Perf-critical things (culling, drawing sets of objects, ...): C++
  - Might move to C#/Burst* at some point
- Control/logic, render pass setup: C#
- GPU code (shaders, compute): HLSL
  - Maybe subset of C# at some point?

* Unity Burst Compiler: LLVM-based compiler for a high performance subset of C#
  https://unity3d.com/unity/features/job-system-ECS
SRP Advantages

- Quick iteration of new algorithms
- All benefits of Unity engine/tooling
- Focus on algorithm, not busywork/plumbing
- Hot reload of C#/shader code
SRP, iterating on the render pipeline
SRP Disadvantages (*today*)

- If something needs native code tweaks/additions, it needs a new Unity release
- Not all the latest graphics features are exposed by Unity yet
  - Raytracing, conservative raster, bindless, ...
  - We’re trying to catch up though
- SRP with C#/HLSL code not easily transferable to other engines
Built-in SRP: Lightweight

- Simpler
- Runs on all platforms*
- Optimized for mobile / VR
- Single pass forward renderer

* At the very moment does not work on WebGL yet due to lack of threads/jobs
Built-in SRP: High-Definition

● More features!
  ● Materials: SSS, Anisotropic, Clearcoat, Iridescent, Rough Refraction, Layered
  ● Lighting: Area lights, better probes, better shadows, volumetrics, ...
  ● Lots of debug views

● Requires compute (DX11 HW)
● Tile/Clustered Forward/Deferred
Built-in SRPs

- Full live source code of both LWRP & HDRP
  - [https://github.com/Unity-Technologies/ScriptableRenderPipeline](https://github.com/Unity-Technologies/ScriptableRenderPipeline)
- Look at how things are done!
- Extend them!
- Build new research on top!
SRP in Research

- **Real-Time Polygonal-Light Shading with Linearly Transformed Cosines**
  Heitz, Dupuy, Hill, Neubelt; SIGGRAPH 2016

- **A Practical Extension to Microfacet Theory for the Modeling of Varying Iridescence**
  Belcour, Barla; SIGGRAPH 2017

- **Efficient Rendering of Layered Materials using an Atomic Decomposition with Statistical Operators**
  Belcour; SIGGRAPH 2018

- **Next up: you!**
SRP as Education Tool

- Simple API makes graphics pipeline more accessible
- Unity’s built-in LWRP/HDRP reference implementations
- Quick iteration & hot-reload
That’s it! Questions?