GPU TESTING: PAST, PRESENT AND FUTURE (IN VULKAN)

Ádám István Szűcs

ELTE Faculty of Informatics
Department of Computer Algebra

July 12, 2019

EFOP-3.6.3-VEKOP-16-2017-00001

SZÉCHENYI 2020

HUNGARIAN GOVERNMENT

European Union
European Social Fund

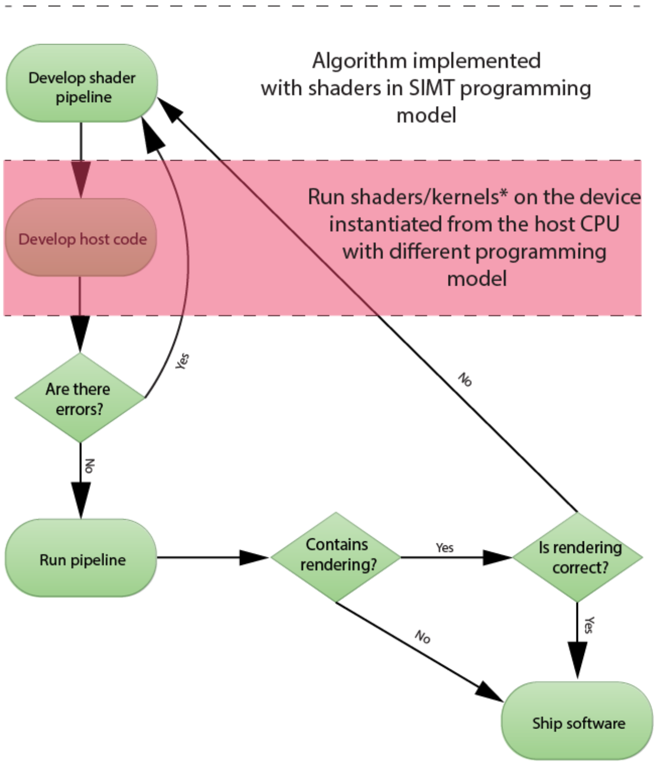
INVESTING IN YOUR FUTURE

GPU DEVELOPMENT PIPELINE

- Algorithm implemented in “shader” language

- Execution of shaders, data management implemented with API calls (mostly we have these API calls wrapped in C++ env. e.g.: UE4)

- There are no direct techniques to assure quality early in the development cycle that contain rendering and compute on GPUs



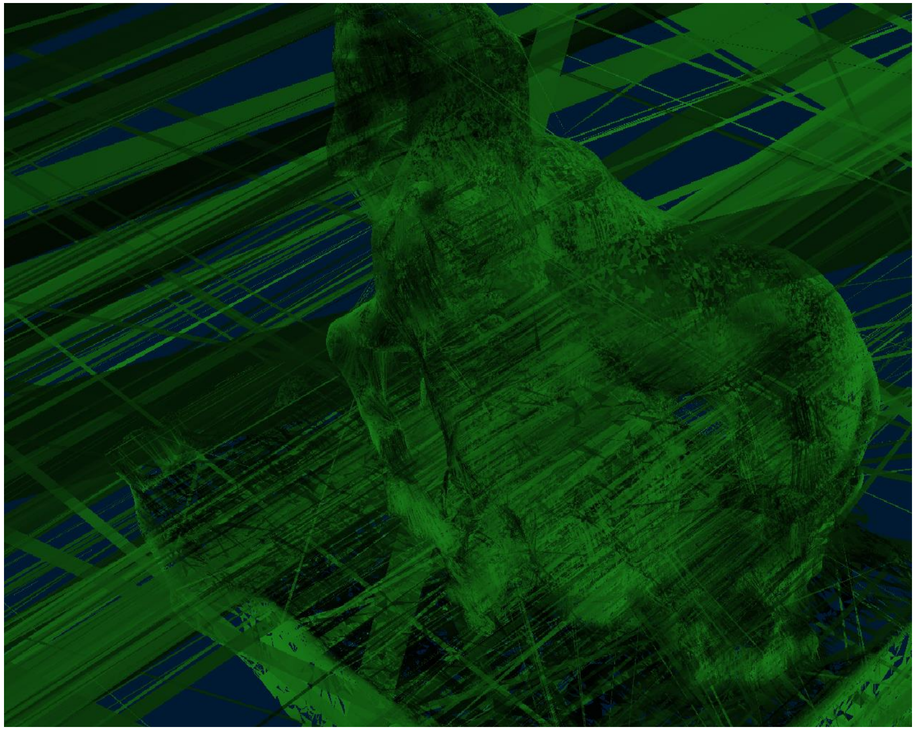
Develop shader pipeline
Algorithm implemented with shaders in SIMT programming model
Develop host code
Run shaders/kernels* on the device instantiated from the host CPU with different programming model
Yes
No
Are there errors?
No
Run pipeline
Contains rendering?
Yes
Is rendering correct?
No
Yes
Ship software

GRAPHICS AND COMPUTE

- For real-time rendering we have only 16-33ms per frame, imagine modern VR **[Elizabeth15]**
- CPU and GPU work together
- The GPU is powerful, but not all powerful
- GPUs act differently than CPUs and can be “mysterious”

COMMON GRAPHICS PROBLEMS

- Blank screen or things not drawing at all
- Corruption
- Flickering
- “Shader” bugs? **[Alastair17]**
- Slowness



PROBLEMS WITH GPU DEVELOPMENT

- GPU code execution goes through a driver
- Programming model needs to be changed when writing host code
- There are no techniques to assure quality for algorithms that contain rendering and compute on GPUs
- If something goes wrong only post-mortem analysis is possible with current tooling for X-{Platform,Device} development **RenderDoc, RGA, RGP**
- Compiler manifold: Microsoft DXC, GlSlang, XShaderCompiler **http://shader-playground.timjones.io/**

STANDARDIZED INTERMEDIATE REPRESENTATION

- Khronos Group Inc. has brought many technologies to life including the APIs such as OpenGL, OpenCL and Vulkan

- After creating many of these technologies for GPU development they have taken the effort in standardizing the intermediate languague for all these graphical processing unit APIs

- As a result SPIRV was created which is
  - Portable across vendors, architectures, platforms
  - Can be used as an intermediate step between high-level and assembly

KHRONOS™
GROUP

SPIR™

INITIAL DRAWBACKS

- However tool for SPIRV called SPIRV-X@Hans-Kristian has experimental C++ code generation it
    - Sometimes can create incorrect code
    - Doesn’t integrate well with type systems by default
    - Doesn’t help with API call generation
    - Doesn’t help with resource management

SOLUTION

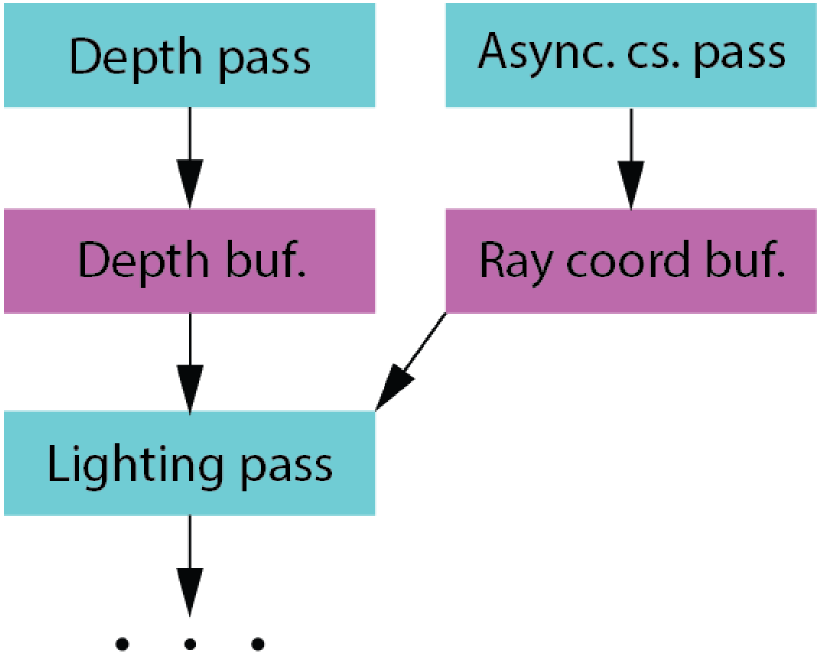
- Extend SPIRV-X to generate necessary types
- OOP the generated code -> inject into type system
- Maintain modularity
- Implement high-level resources for the RHI
- Result: Automatically testable equivalent C++ code

SOLUTION – FRAME GRAPHS

- During converting the shaders, we can gather information of the resources used by the GPU program

- Combining the resources we can generate and combine high level knowledge of a frame, temporary result during the algorithms pipeline **[Yuri17]**

- Resulting in
  - Simplified resource management
  - Simplified rendering/compute pipeline configuration
  - Simplified async compute and resource barriers
  - Self contained and modular pipelines



RT RAY TRACING IS HERE

- Running RT code on real-life scenes would drive us to the field of Disney/Pixar sized render farms
  - Moving to a software renderer does not help for small studios/firms
- Testing shall be moved up to the hardware
- Shall be as fast as possible

FUTURE

- Converting all the shaders can be a time-consuming job even in a distributed automated environment

- All the upcoming features need to be supported including extensions (>150 extensions already in Vulkan)

- Could be a viable solution to move all the implementation closer to the driver**[RADV]**
  - Only on Nix and AMD hardware
  - What about Nvidia?
  - Need GPU vendor collaboration

FUTURE

- Solution will be hard to implement in drivers, needs x-vendor collaborations, hard-to maintain and to keep control

- Try to do a round trip based on GPU Assisted Validation layers **[Lunarg19]**
  - Can be challenging to implement a Validation Layer
  - LunarG Validation Layer Factory to the rescue!
  - Mixed with VK_EXT_DEBUG_UTILS

- Save assertion results into buffer in shader
  - Sizing becomes a question (multiple terabytes of data is generated on the fly)
  - vkAssert(…) needs to be exposed by the API as well

- Need to “patch” the shaders under test
  - Can be challenging in large systems with material systems and permutation
  - SPIRV Optimizer

QUESTIONS

This work was supported in part by Advanced Micro Devices Radeon Technologies Group. Information on the

Radeon Technologies Group can be obtained from

https://radeon.com/. This research was also supported by

Zeno Vision Limited and ELTE EIT Digital and by the European Union, co-financed by the European Social Fund

(EFOP-3.6.3-VEKOP-16-2017-00001)

REFERENCES

[Elizabeth15] Tightening Up the Graphics: Tools and Techniques for Debugging and Optimization, BostonFIG Talk 2015

[Alastair17] Automated Testing of Graphics Shader Compilers, Proceedings of the ACM Programming Languages 2017

[SPIRV] Standard Portable Intermediate Representation, https://www.khronos.org/spir/

[Yuri17] FrameGraph: Extensible Rendering Architecture in Frostbite, Game Developers Conference 2017

[RADV] Radeon Vulkan driver "RADV“, https://github.com/airlied/mesa/tree/semi-interesting/src/amd/vulkan

[Lunarg19] Vulkan GPU-Assisted Validation https://www.lunarg.com/wp-content/uploads/2019/06/GPU-Assisted-Validation-Phase-2_final.pdf

THANK YOU FOR YOUR ATTENTION!

SZÉCHENYI 2020

HUNGARIAN GOVERNMENT

European Union
European Social Fund

INVESTING IN YOUR FUTURE