GSOC 2017 Project Proposal - Software Renderer for Vulkan

Jacob Lifshay
programmerjake@gmail.com

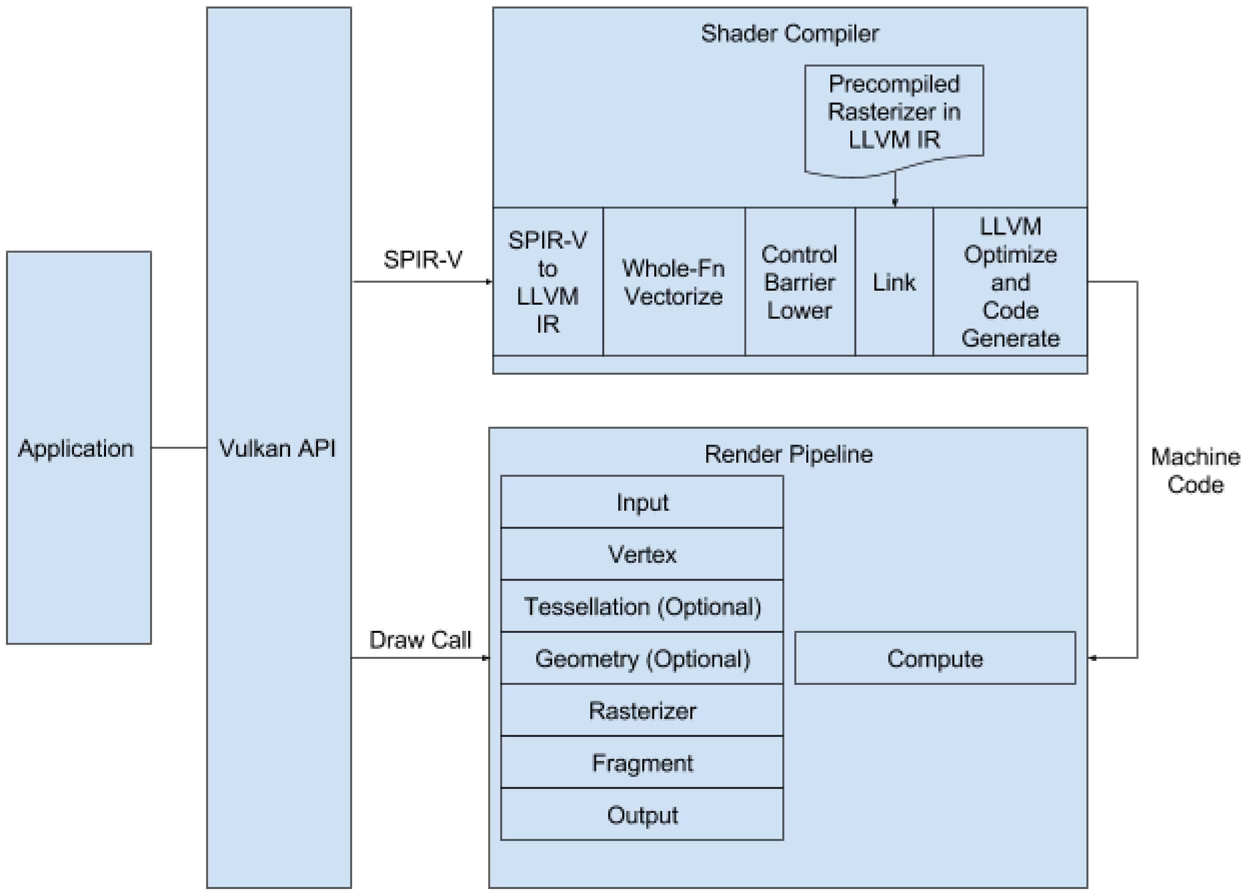
Software Renderer for Vulkan

Write a software renderer for Vulkan that implements graphics and compute shaders by translating directly from SPIR-V to LLVM IR, running a custom whole-function vectorization pass, running a custom control-barrier lowering pass, then using LLVM's MC framework to generate code.
This will benefit the community by providing an implementation of Vulkan that can be used as a software fallback or on virtual machines that don't have access to a video card.

Timeline:

1. Both of (simultaneously) (2-4 weeks):
    1.1. Write SPIR-V to LLVM IR translation code (required)
    1.2. Write custom control-barrier lowering pass (required)
2. Write rasterization code using a hierarchical tiled architecture, translating (using clang) from C++11 to LLVM IR; Integrate generated rasterization code with translated shaders to create the basic rendering pipeline (required) (1-2 weeks).
3. Write and/or reuse basic Vulkan framework code to support calling the rendering pipeline with support for only F32 depth, and RGBA 8888 image formats (required) (1-3 weeks)
4. (at this point, except for the image formats and multisampling, the implementation should support most of the basic requirements of the Vulkan spec).
5. Submit initial code to Mesa project (optional, will submit all code later) (1 day)
6. Write or reuse WSI code for X11 on Linux and for Win32. (optional) (2-3 days)
7. Write documentation for code (required) (2-3 days)
8. Write whole-function vectorization pass (optional) (3-8 weeks)
9. In any order (depending on how much time is left)
    9.1. Add support for more image formats (optional)
    9.2. Implement multisampling (optional)
    9.3. Implement vectorized math functions (eg. sin, cos, log) (optional)
    9.4. Implement other optional portions of Vulkan spec (optional)
    9.5. Write more documentation (optional)
10. Submit code to Mesa project (required) (1 day)

Estimated time for required deliverables: 4.5 weeks to 9.5 weeks



The software will consist of a Vulkan API implementation that implements the bare minimum required to be Vulkan compliant, except that it may not implement all of the required image formats, and probably won't support multisampling. The shader compiler will translate from SPIR-V directly to LLVM IR. I will probably write my own SPIR-V translation code as the currently existing code, released by Khronos, seems to be designed to only work with OpenCL. The vertex and fragment shaders will be linked with the LLVM IR form of the rasterizer, then optimized as a whole. The optimal image layouts will be a 2D array of chunks where each chunk is a 2D array whose size is dependent on the SIMD vector width. The initially supported image formats will be RGBA8888 for color images, and float32 for depth. The rasterizer will use a hierarchical tile rasterization scheme based off of the edge equations in 2D projective space (x, y, w). The algorithm I was going to use to implement control barriers is what you'd get from treating the shader as if it's a javascript generator function by replacing the control barriers with yield statements, then inlining the generator function into a while loop that loops until all of the generator functions return.

Test implementation of rasterizer: https://github.com/programmerjake/tiled-renderer
Previous 3D renderer project (implements scanline renderer with texture mapping and alpha testing): https://github.com/programmerjake/lib3d link to some generated output lib3d-output.ogv

Related Work:
Inspiration for rasterization algorithm:
https://software.intel.com/en-us/articles/rasterization-on-larrabee hierarchical tiled rasterization algorithm description
Only part of the final software

Whole-Function Vectorization:
http://compilers.cs.uni-saarland.de/publications/theses/karrenberg_msc.pdf
Only part of the final software

LLVMPipe: software rasterizer for OpenGL 3.3
Doesn’t implement Vulkan API and doesn’t parallelize vertex shaders. Doesn’t implement control barriers efficiently.

Khronos SPIR-V LLVM translator:
https://github.com/KhronosGroup/SPIRV-LLVM
Doesn’t implement support for SPIR-V for graphics.

Note: This document is a modified version of my project proposal for GSOC 2017 -- I changed the title and removed some personal details.