The Vulkan Sample Code Included with These Notes

Mike Bailey
mjb@cs.oregonstate.edu

This work is licensed under a
Creative Commons
Attribution-NonCommercial-NoDerivatives 4.0
International License

The Vulkan Sample Code Included with These Notes

Sample Program Output

Sample Program Keyboard Inputs

1. 'l', 'L': Toggle lighting off and on.
2. 'm', 'M': Toggle display mode (textures vs. colors, for now)
3. 'p', 'P': Pause the animation
4. 'q', 'Q': quit the program
5. Esc: quit the program
6. 'r', 'R': Toggle rotation-animation and using the mouse
7. 't', 'T': Toggle using a vertex buffer only vs. an index buffer
   (in the index buffer version)
8. '1', …,'9','a',…'g': Set the number of instances (in the instancing version)

Caveats on the Sample Code, I

1. I've written everything out in appalling longhand.
2. Everything is in one .cpp file (except the geometry data).  It really should be broken up, but this way you can find everything easily.
3. At times, I could have hidden complexity, but I didn’t.  At all stages, I have tried to err on the side of showing you everything, so that nothing happens in a way that’s kept a secret from you.
4. I’ve setup Vulkan structs every time they are used, even though, in many cases (most?), they could have been setup once and then re-used each time.
5. At times, I’ve setup things that didn’t need to be setup just to show you what could go there.

Caveats on the Sample Code, II

6. There are great uses for C++ classes and methods here to hide some complexity, but I’ve not done that.
7. I’ve typedef’d a couple things to make the Vulkan phraseology more consistent.
8. Even though it is not good software style, I have put persistent information in global variables, rather than a separate data structure
9. At times, I have copied lines from vulkan_core.h into the code as comments to show you what certain options could be.
10. I’ve divided functionality up into the pieces that make sense to me.  Many other divisions are possible.  Feel free to invent your own.

Main Program

```cpp
int main( int argc, char * argv[] )
{
    Width  = 1024;
    Height = 1025;
    errno_t err = fopen_s( &FpDebug, DEBUGFILE, "w" );
    if( err != 0 )
    {
        fprintf( stderr, "Cannot open debug print file '%s'

FpDebug: Width = %d ; Height = %d

Reset( );
InitGraphics( );

// loop until the user closes the window:
while( glfwWindowShouldClose( MainWindow ) == 0 )
{
    glfwPollEvents( );
    Time = glfwGetTime();          // elapsed time, in double-precision seconds
    UpdateScene( );
    RenderScene( );
}
    fprintf(FpDebug, "Closing the GLFW window

vkQueueWaitIdle( Queue );
vkDeviceWaitIdle( LogicalDevice );
DestroyAllVulkan( );
glfwDestroyWindow( MainWindow );
glfwTerminate( );
return 0;
}
```
```cpp
void InitGraphics()
{
    HERE_I_AM( "InitGraphics" );
    VkResult result = VK_SUCCESS;
    Init01Instance();
    InitGLFW();
    Init02CreateDebugCallbacks();
    Init03PhysicalDeviceAndGetQueueFamilyProperties();
    Init04LogicalDeviceAndQueue();
    Init05UniformBuffer( sizeof(Matrices), &MyMatrixUniformBuffer );
    Fill05DataBuffer( MyMatrixUniformBuffer, (void *) &Matrices );
    Init05UniformBuffer( sizeof(Light), &MyLightUniformBuffer );
    Fill05DataBuffer( MyLightUniformBuffer, (void *) &Light );
    Init05MyVertexDataBuffer( sizeof(VertexData), &MyVertexDataBuffer );
    Fill05DataBuffer( MyVertexDataBuffer, (void *) VertexData );
    Init06CommandPool();
    Init06CommandBuffers();
    Init07TextureSampler( &MyPuppyTexture.texSampler );
    Init07TextureBufferAndFillFromBmpFile( "puppy.bmp", &MyPuppyTexture );
    Init08Swapchain();
    Init09DepthStencilImage();
    Init10RenderPasses();
    Init11Framebuffers();
    Init12SpirvShader( "sample-vert.spv", &ShaderModuleVertex );
    Init12SpirvShader( "sample-frag.spv", &ShaderModuleFragment );
    Init13DescriptorSetPool();
    Init13DescriptorSetLayouts();
    Init13DescriptorSets();
    Init14GraphicsVertexFragmentPipeline( ShaderModuleVertex, ShaderModuleFragment,
                                   VK_PRIMITIVE_TOPOLOGY_TRIANGLE_LIST, &GraphicsPipeline );
}
```

```
#include "SampleVertexData.cpp"
```
Vulkan Software Philosophy

Vulkan has lots of typedefs that define C/C++ structs and enums. Vulkan takes a non-C++ object-oriented approach in that those typedef'd structs pass all the necessary information into a function. For example, where we might normally say using C++ class methods:

```cpp
result = LogicalDevice->vkGetDeviceQueue ( queueFamilyIndex, queueIndex, OUT &Queue );
```

Vulkan has chosen to do it like this:

```cpp
result = vkGetDeviceQueue ( LogicalDevice, queueFamilyIndex, queueIndex, OUT &Queue );
```

Vulkan Conventions

**My Conventions**

- VkXxx is a typedef, probably a struct
- vkYyy( ) is a function call
- VK_ZZZ is a constant
- "Init" in a function call name means that something is being setup that only needs to be setup once
- The number after "Init" gives you the ordering
- In the source code, after main( ) comes InitGraphics( ), then all of the InitxxYYY( ) functions in numerical order. After that comes the helper functions
- "Find" in a function call name means that something is being looked for
- "Fill" in a function call name means that some data is being supplied to Vulkan
- "IN" and "OUT" ahead of function call arguments are just there to let you know how an argument is going to be used by the function. Otherwise, IN and OUT have no significance. They are actually #define'd to nothing.

Querying the Number of Something and Allocating Enough Structures to Hold Them All

```cpp
uint32_t count;
result = vkEnumeratePhysicalDevices( Instance, OUT &count, OUT (VkPhysicalDevice *)nullptr );
VkPhysicalDevice * physicalDevices = new VkPhysicalDevice[count];
result = vkEnumeratePhysicalDevices( Instance, OUT &count, OUT physicalDevices[0] );
```

This way of querying information is a recurring OpenGL and Vulkan pattern (get used to it):

```cpp
result = vkEnumeratePhysicalDevices( Instance, OUT &count, nullptr );
result = vkEnumeratePhysicalDevices( Instance, OUT &count, OUT physicalDevices[0] );
```

Reporting Error Results, I

```cpp
uint64_t errorcode

uint64_t result = vkGetErrorInfoEXT( VkResult result, std::string prefix )
{...}
```

Reporting Error Results, II

```cpp
uint64_t result = vkGetErrorInfoEXT( VkResult result, std::string prefix )
{...
    if (result != VK_SUCCESS)
    {
        ErrorCodes[errorcode].meaning = "Error Code " + std::to_string(errorcode);
        std::cerr << "Error Code " + std::to_string(errorcode) << " : " << ErrorCodes[errorcode].meaning << std::endl;
    }
}..."
```c
#define REPORT(s)               { PrintVkError( result, s );  fflush(FpDebug); }
#define HERE_I_AM(s)          if( Verbose )  { fprintf( FpDebug, "***** %s *****
", s );  fflush(FpDebug); }

bool Paused;
bool Verbose;

#define DEBUGFILE               "VulkanDebug.txt"

errno_t err = fopen_s( &FpDebug, DEBUGFILE, "w" );
```