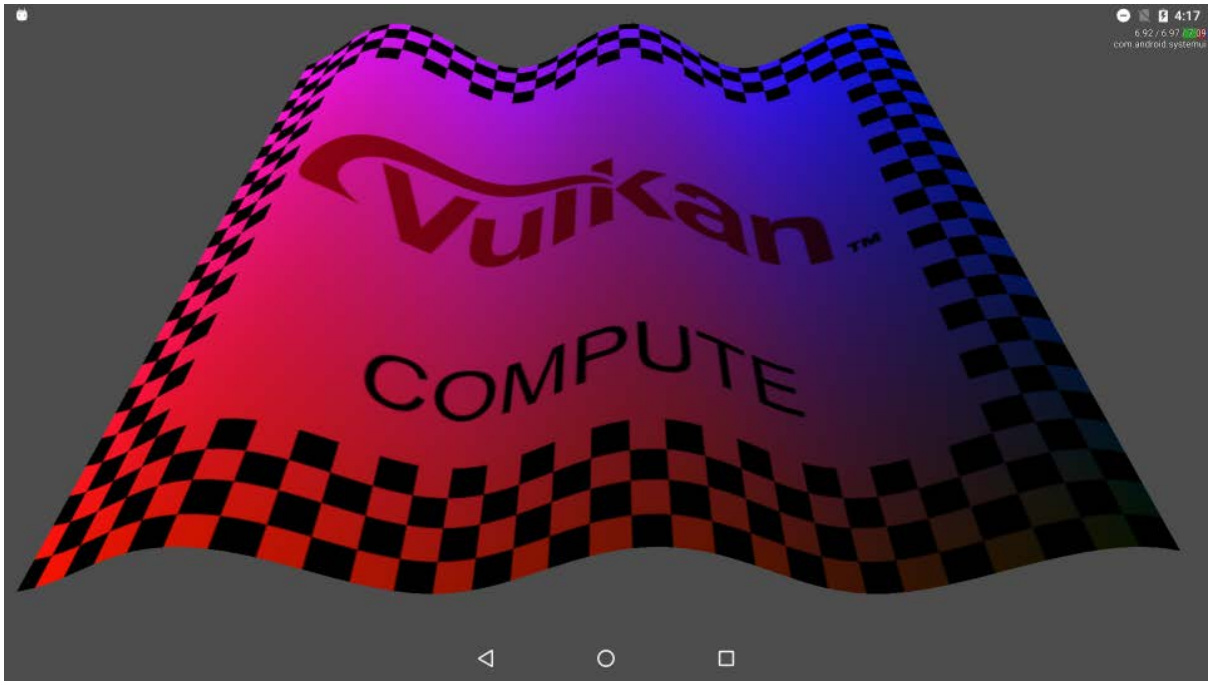
Sample Guide

Samples are located within the `samples/` directory and are laid out as Android Studio projects. The "Getting started with the Vulkan Graphics API" walkthrough document shows how to load the samples into the development environment.

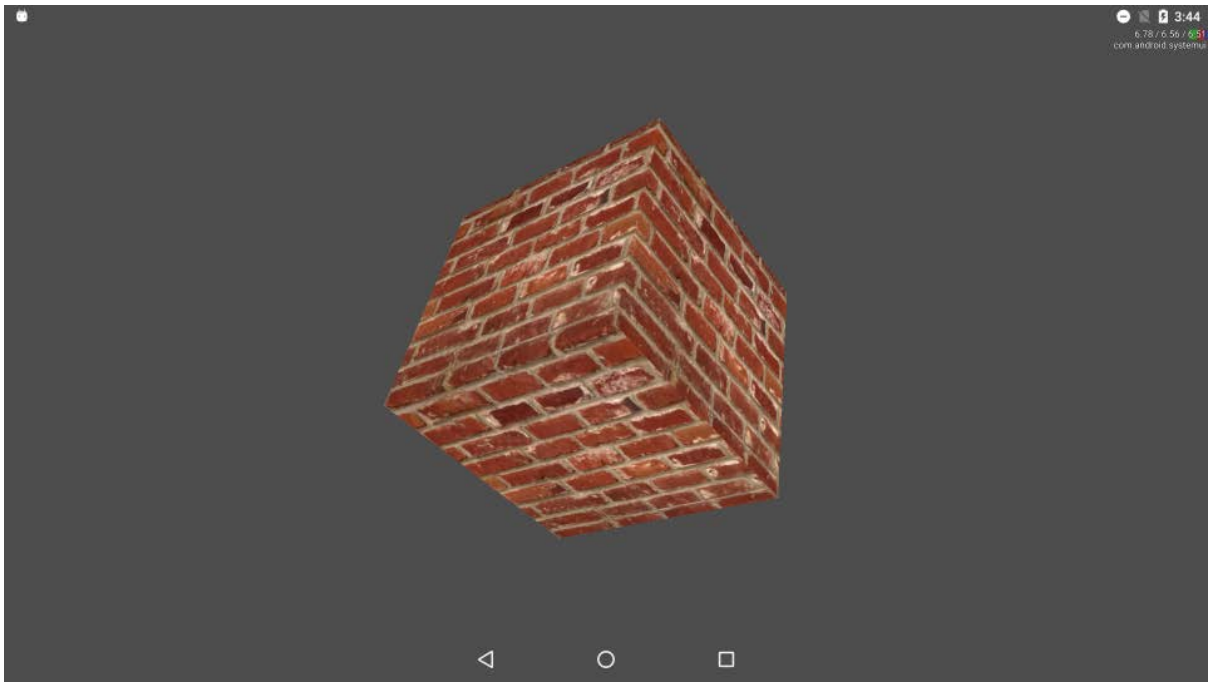
Within the samples directory, there is a `README.md` file containing the most recent details for that sample, including prerequisites such as NDK and Android Studio versions.

compute



This Vulkan example shows how to set up and dispatch compute shaders. A compute shader offsets a grid of verticies to introduce a waveform effect, which is then rendered with an overlaid texture.

cube



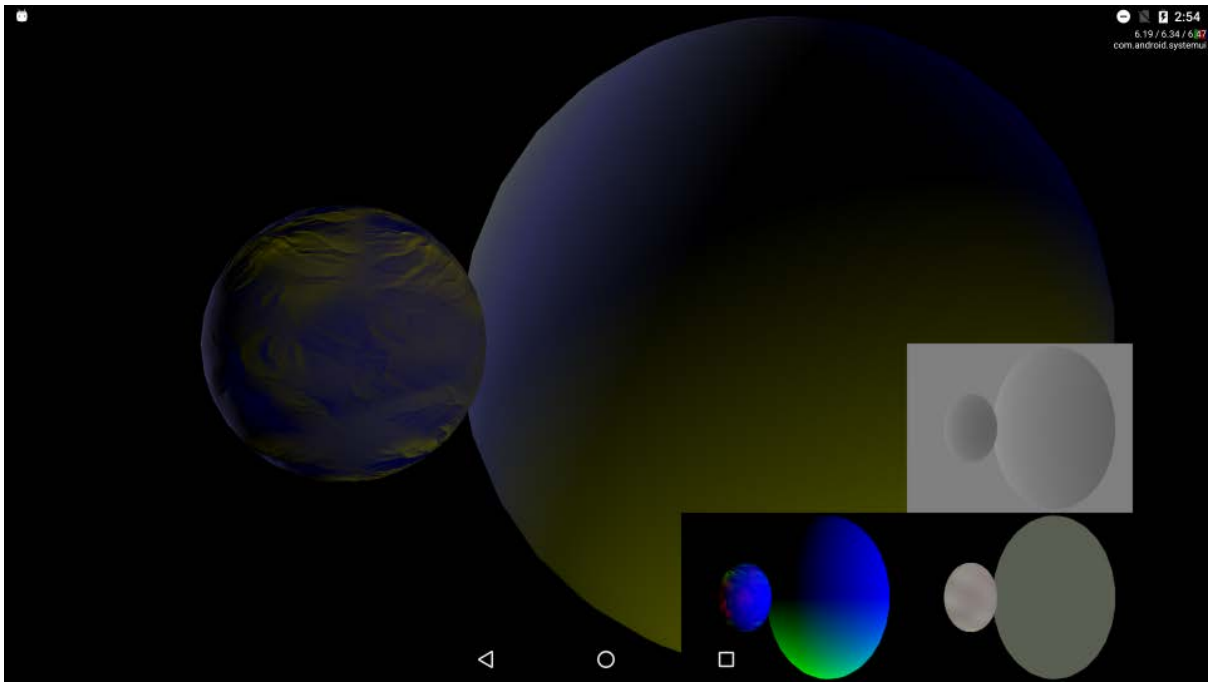
This Vulkan example shows how to set up state to render a rotating, textured cube. The ModelViewProjection matrix is provided to the vertex shader as a uniform, and UVs and Vertex colours forwarded to the fragment shader, which is a basic textured shader.

This sample is built on top of the triangle sample, without a framework. You can difference the source files against the triangle sample ones to see easily how uniform buffers and descriptor sets are handled. Textures are handled by a utility class, TextureObject.

cubemap

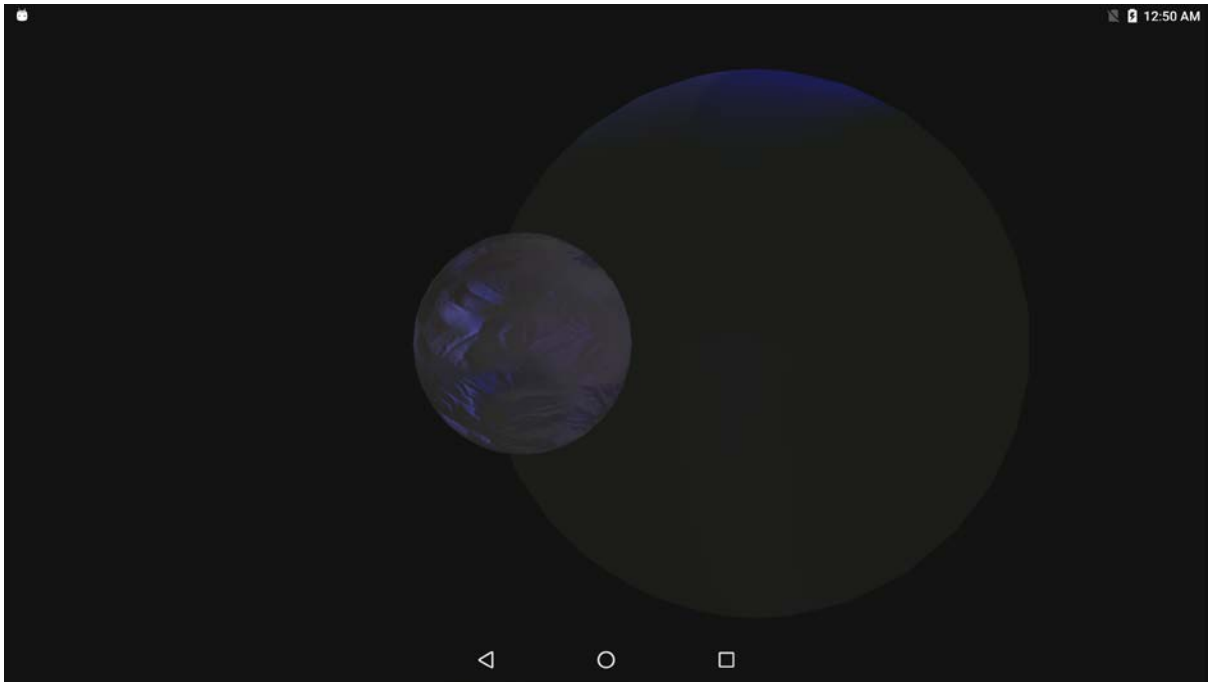
This Vulkan example shows how to use cubemaps in a Vulkan scene. A cubemap texture with 6 faces is used to define the cubemap and two shaders are used that sample the texture. Two pipelines are created: one to render the background cube and one to render the foreground teapot object. Both the cube and teapot are animated by rotating them together to show that the cubemap is seamless.

deferred



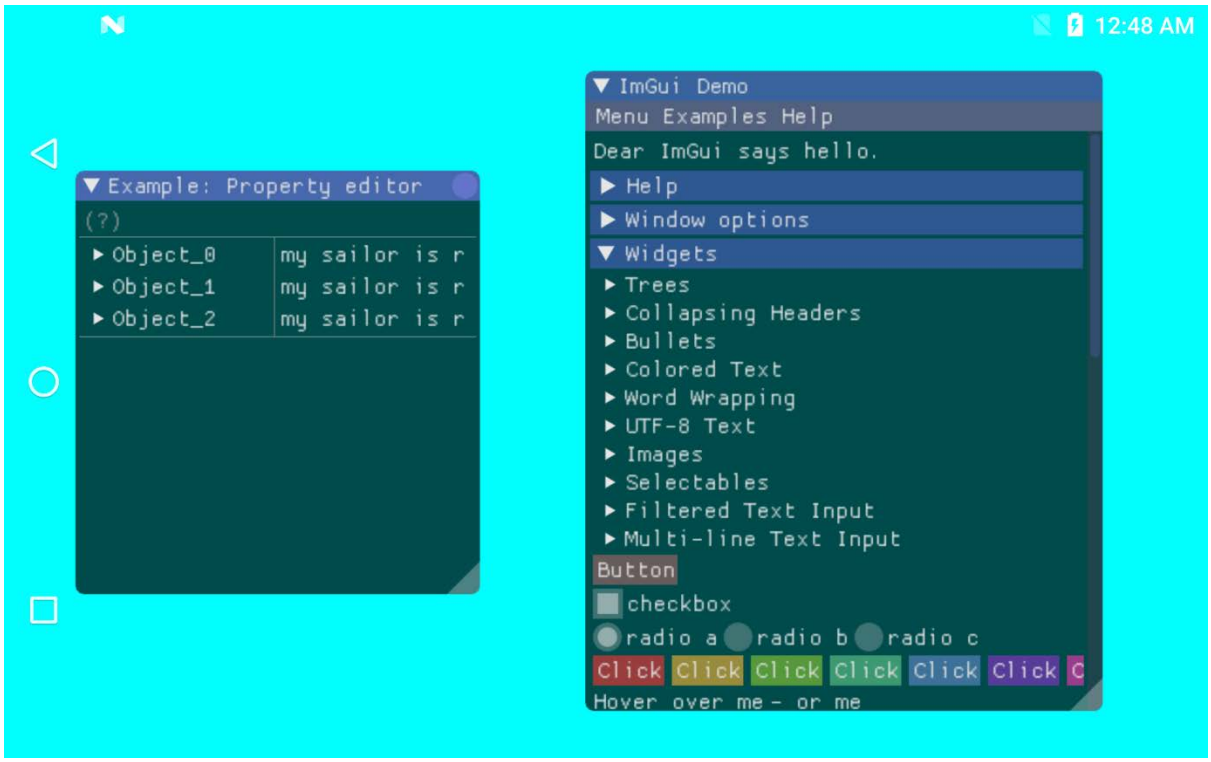
This Vulkan example shows how implement deferred rendering. Geometry is rendered and three image targets are output, for position, normal, and color. These image targets are copied into texture objects, which can then be sampled from. A further full-screen quad is rendered where these three G-buffers are merged with lighting information into a final scene. A small debug output display shows a representation of the G-buffers.

deferred multipass



This Vulkan example shows how to set up a renderpass object with multiple attachments, and passes. The attachments represent the G-Buffers which are generated to in the first pass, and combined in the second pass.

gui

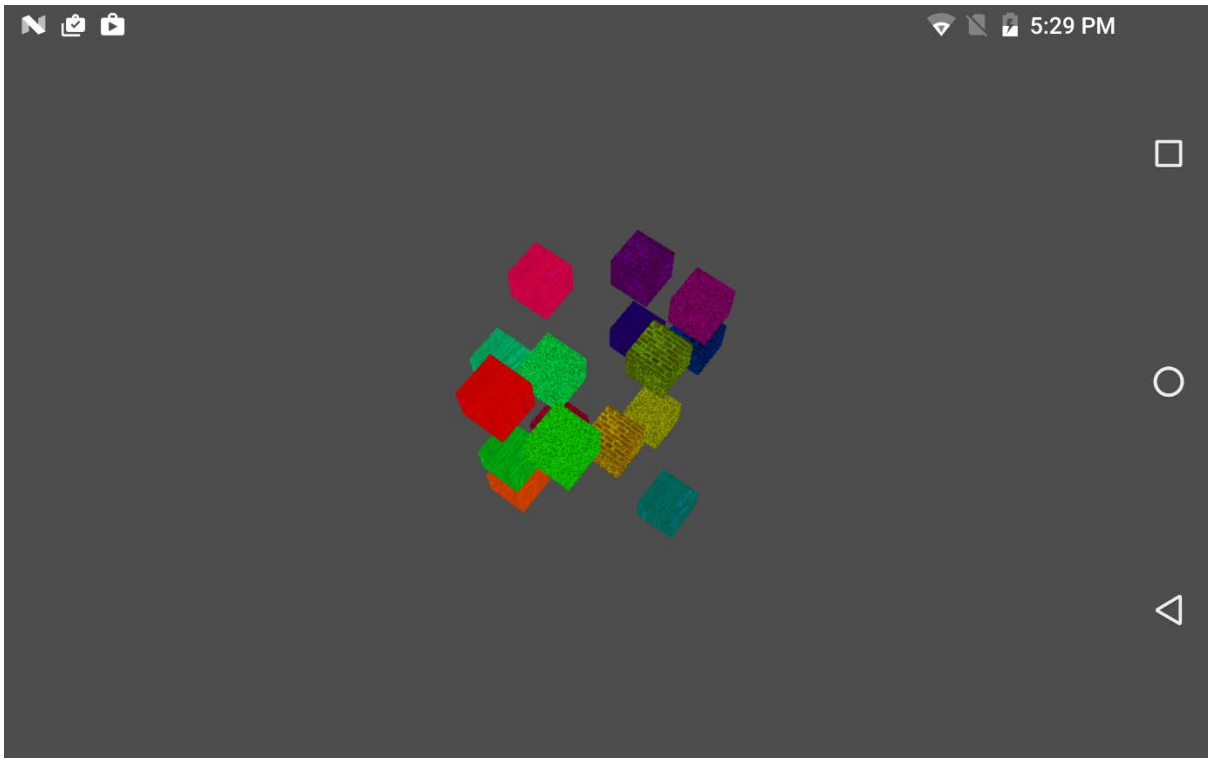


This Vulkan example shows how to create an immediate mode gui (graphics user interface). We've ported the ImGui library to Vulkan and allowed it to be used in this SDK. ImGui is a popular C++ user interface library with a rich set of widgets that are rendered immediately in your application. The sample renders the ImGui TestWindow which shows off many of the possible interface options. Refer to https://github.com/ocornut/imgui for additional information and ImGui and the Developer Guide document for ImGui's license and permission notice.

imageeffects

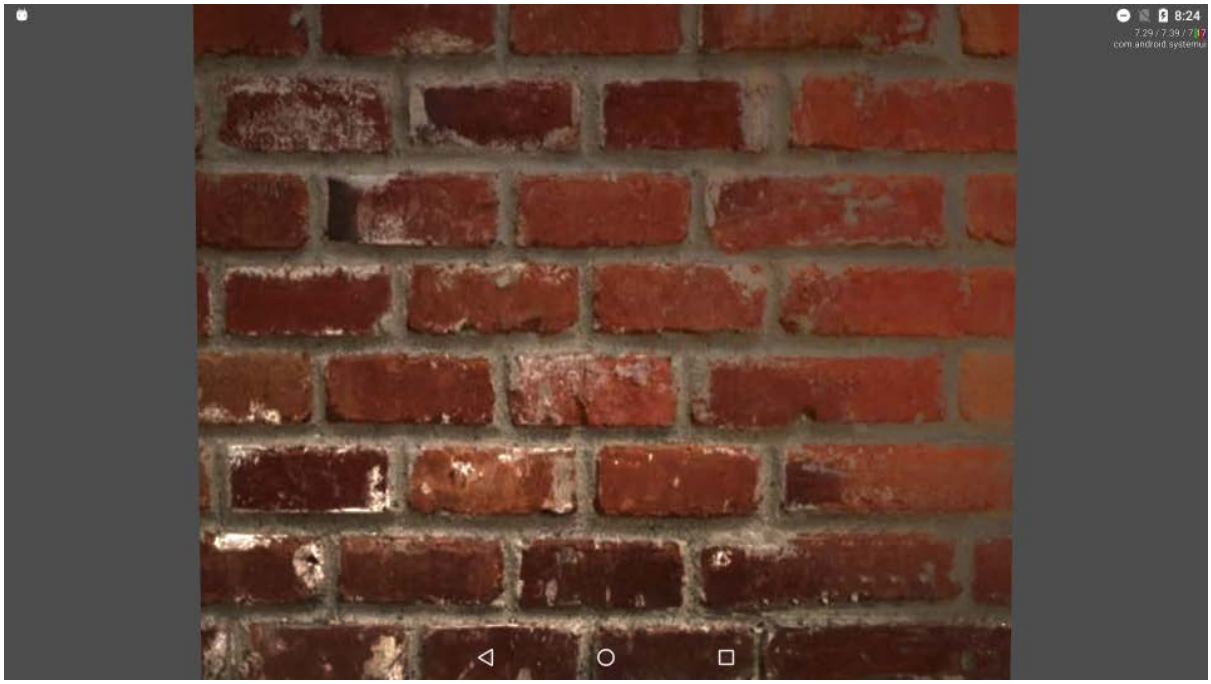
This Vulkan example shows how to set up multiple pipelines for operating different shaders. It also shows how to manage Descriptor Sets when working with dynamic uniforms, image samplers and how the layouts are intitialized. Additionally, the texture changes for each shader example after a period of time, showing updates to descriptor sets and re-recording of command buffers.

multithreading



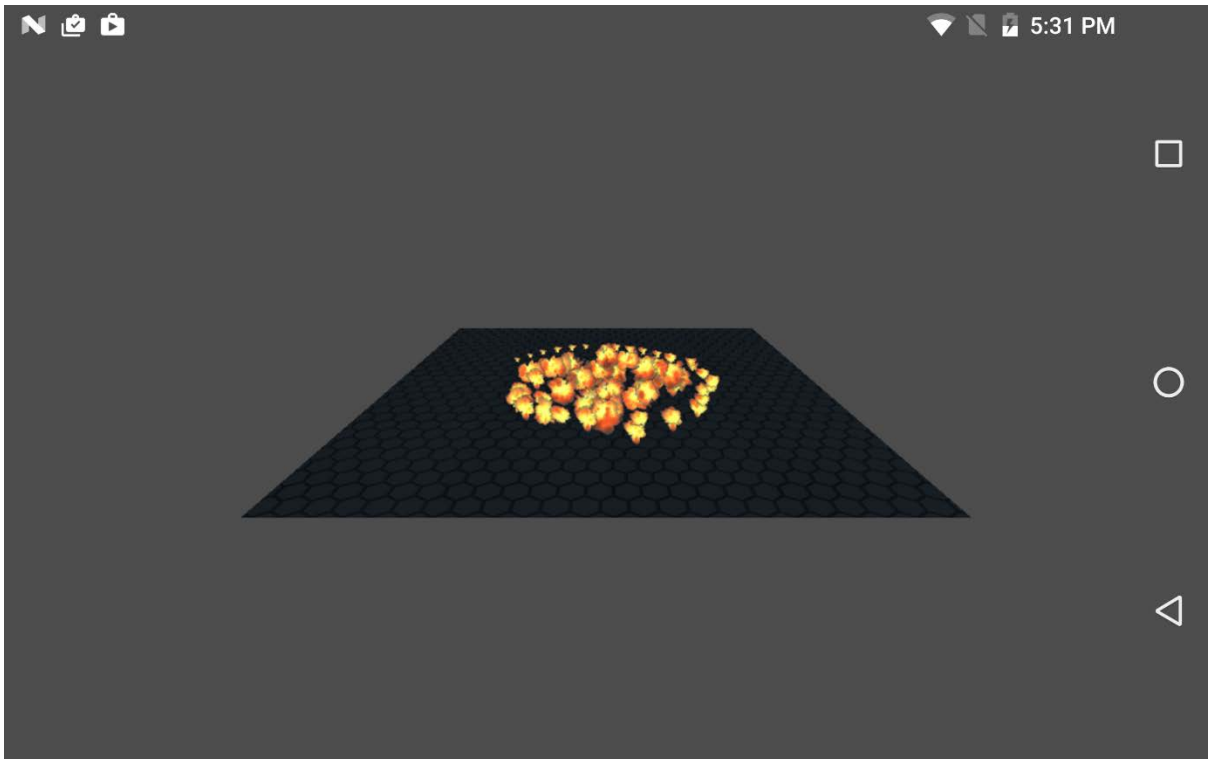
This Vulkan example shows how to use multiple threads to update separate secondary command buffers in parallel. Each thread creates and updates it's own descriptor set, pipeline, vertex buffer, and command buffer. The final rendering shows a colored and textured cube rendered by each thread, called by a primary command buffer.

normalmapping



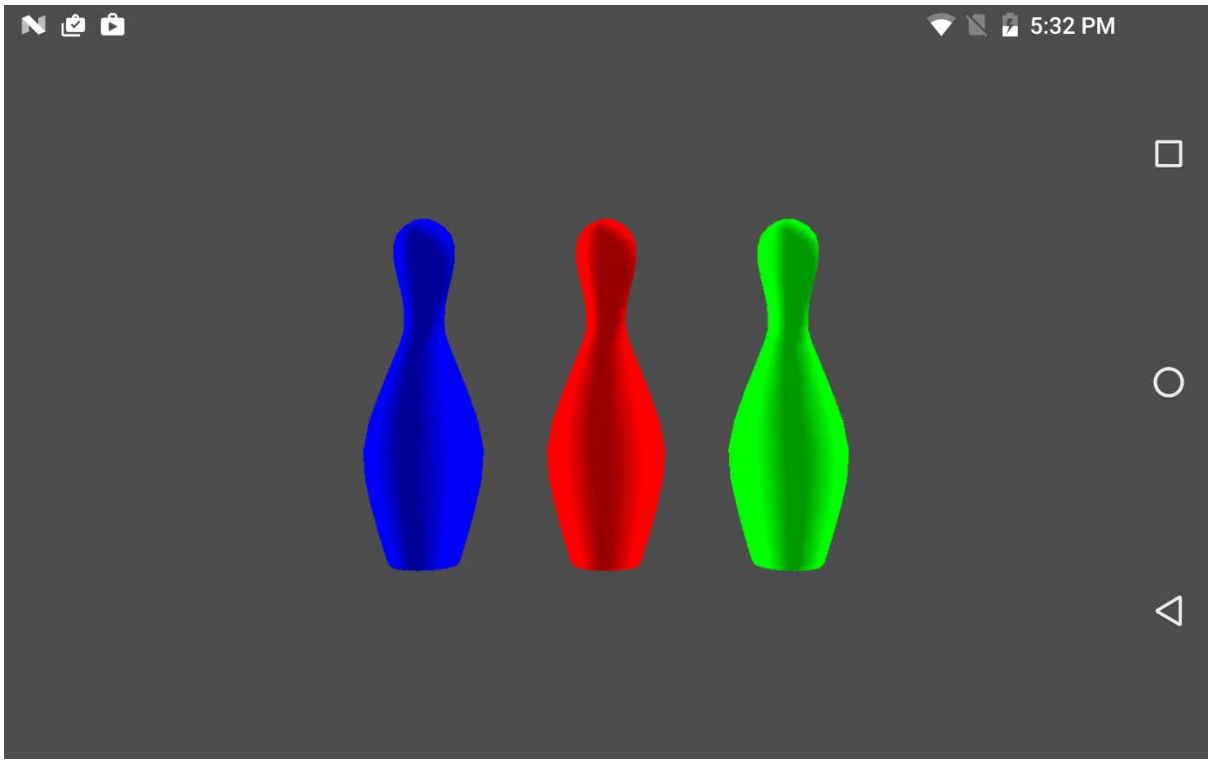
This Vulkan example shows how to expand upon the 'cube' source to enable simple normal mapping via additions to the descriptor sets and vertex attributes for larger shader programs.

particles



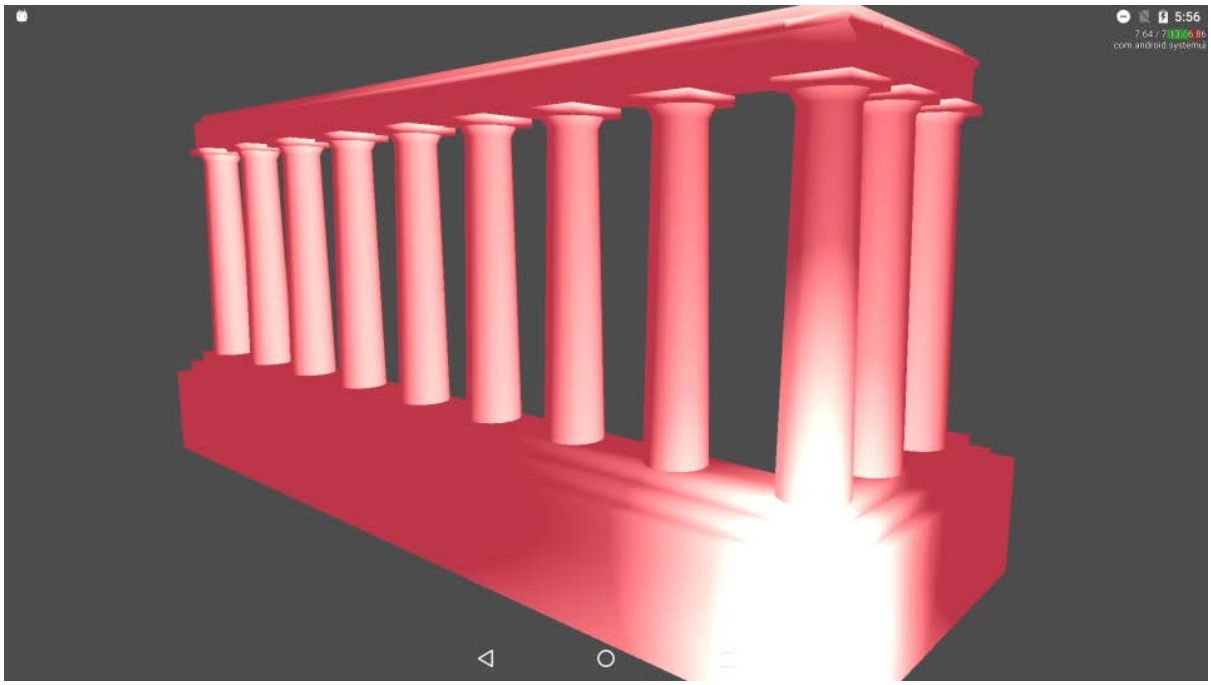
This Vulkan example shows one way to calculate and render particles. The particle is rendered using a texture mapped screen facing quad. The texture contains several animation tiles which are cycled through. Every frame the particles position and tile are computed and they are sorted based on their distance from the camera to allow for correct alpha blending.

pipeline



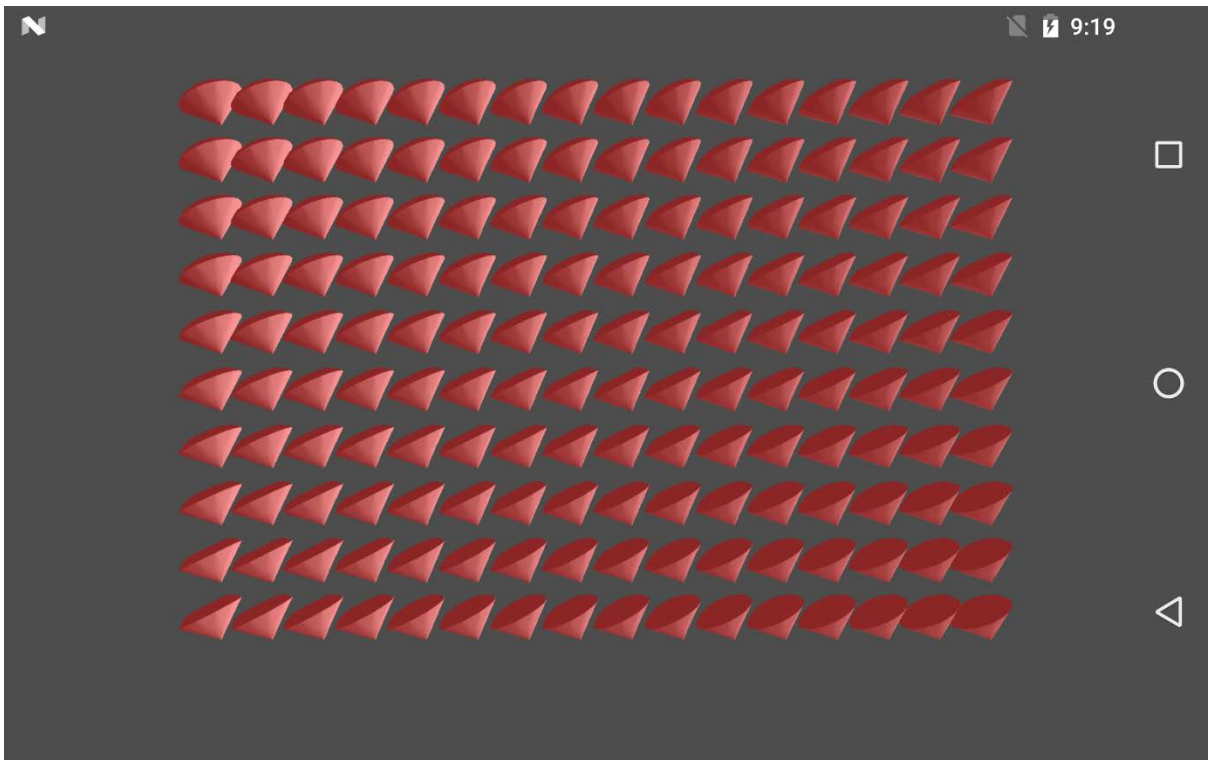
This Vulkan example shows how to derive pipelines from other pipelines, and how to both create a pipeline cache file on disk and use that cache file to load pipeline from.

push_constants



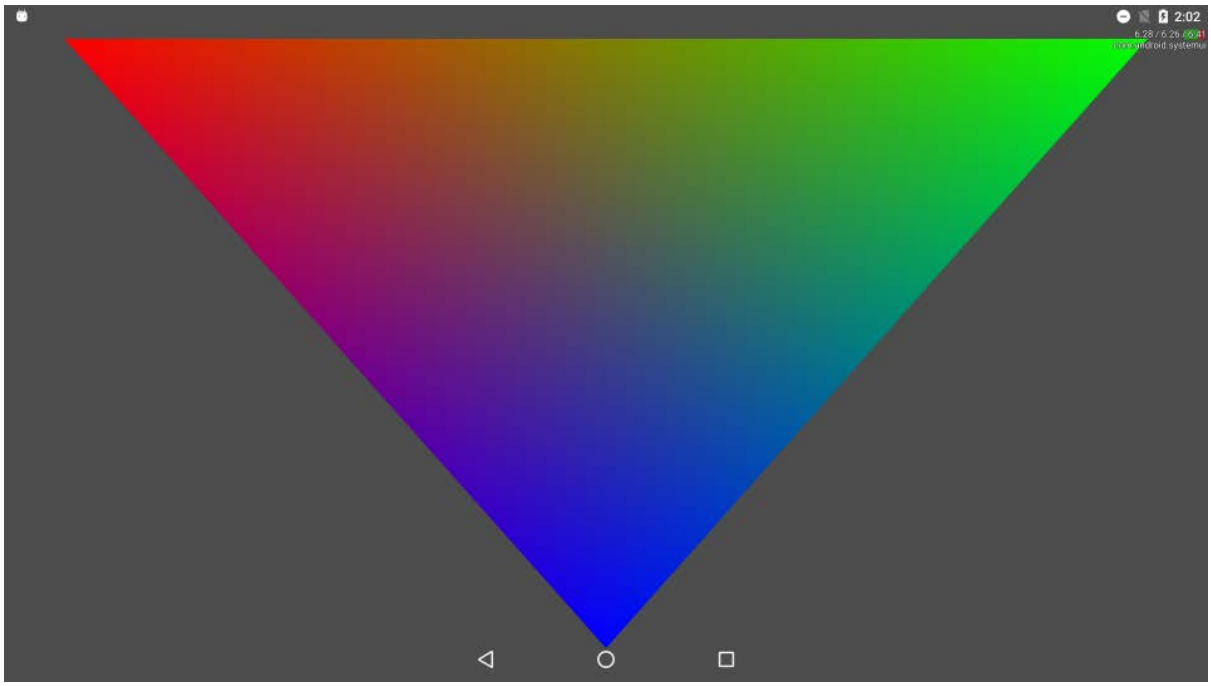
This Vulkan example shows how to use push constants to provide shader stage data directly from command buffers. The mesh color and light position is randomized each run of the sample.

suballocation



This Vulkan example shows how to suballocate memory to minimize the total number of memory allocations made by a Vulkan application. Efficient memory allocation in Vulkan can be accomplished by using a suballocation technique. You can minimize the cost and overheads of using the vkAllocMemory API by allocating just a few large allocations and then suballocating memory requests from them. The MemoryAllocator class in the framework implements a simple suballocation scheme. This example renders a set of red diamonds whose memory is suballocated from a single large allocation.

triangle

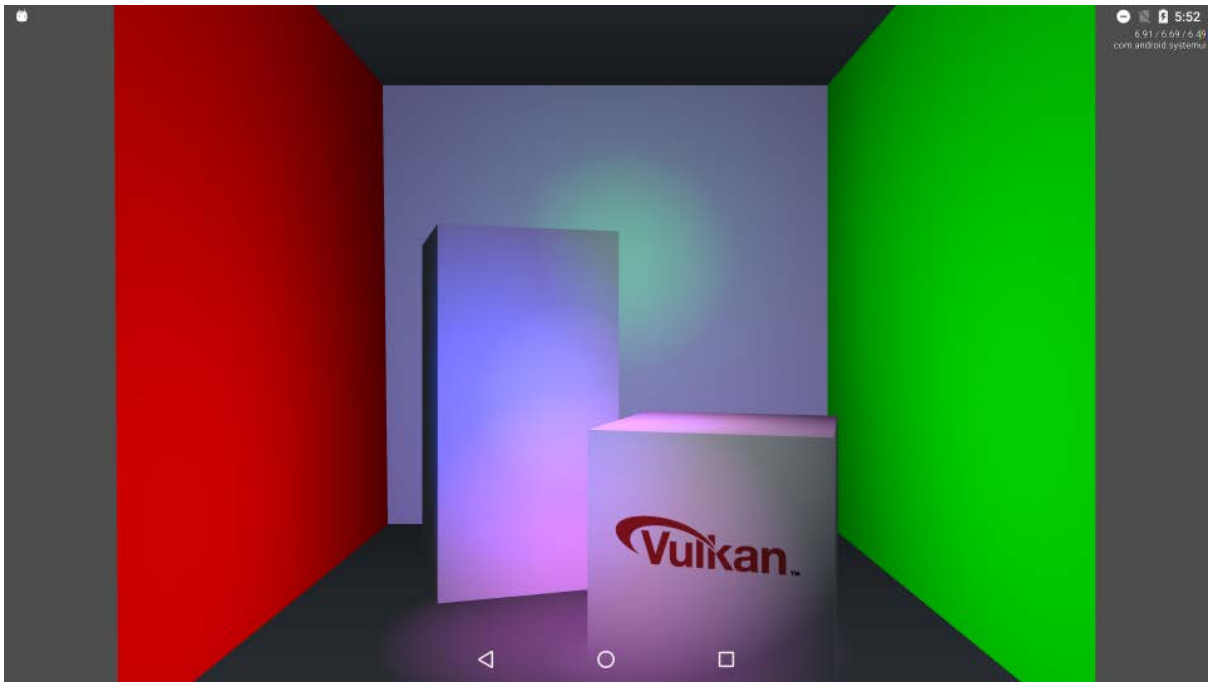


This Vulkan example shows how to set up basic device state, and draw a triangle with vertex colours.

The sample is self-contained and does not rely on any framework. All code can be seen within the sample.cpp and sample.h files.

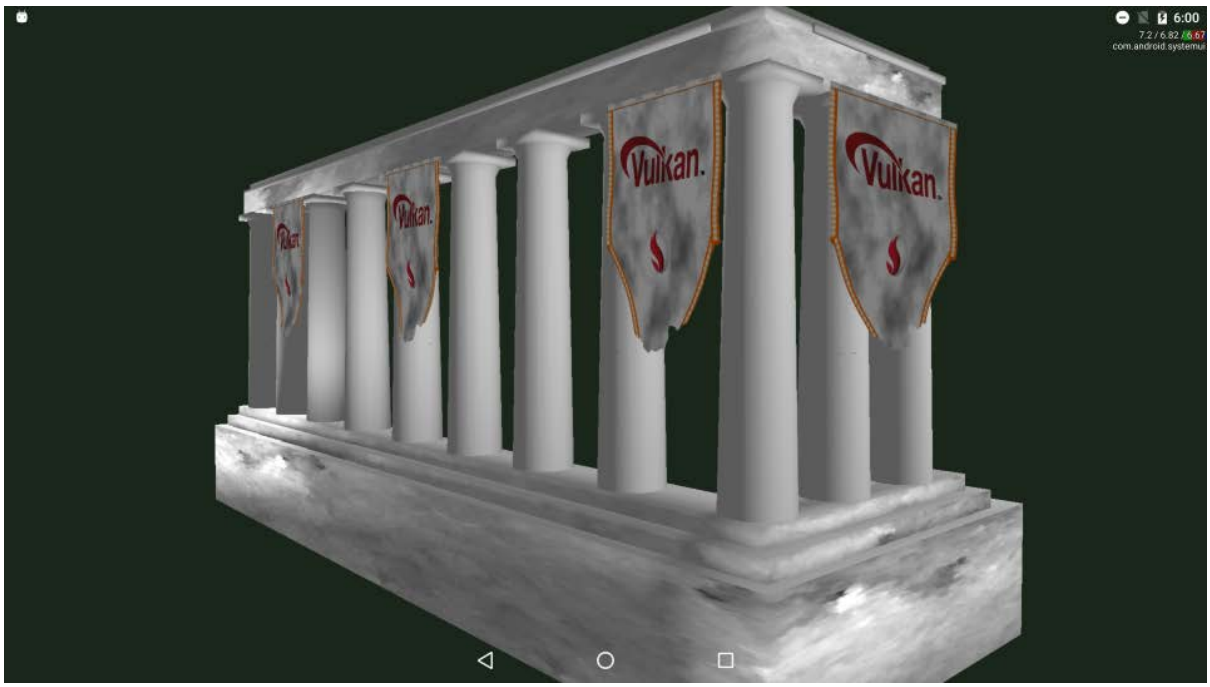
This sample has an in-depth walkthrough of the code available, in the *"Getting started with the Vulkan Graphics API – the first triangle"* document.

tut_cornelllights



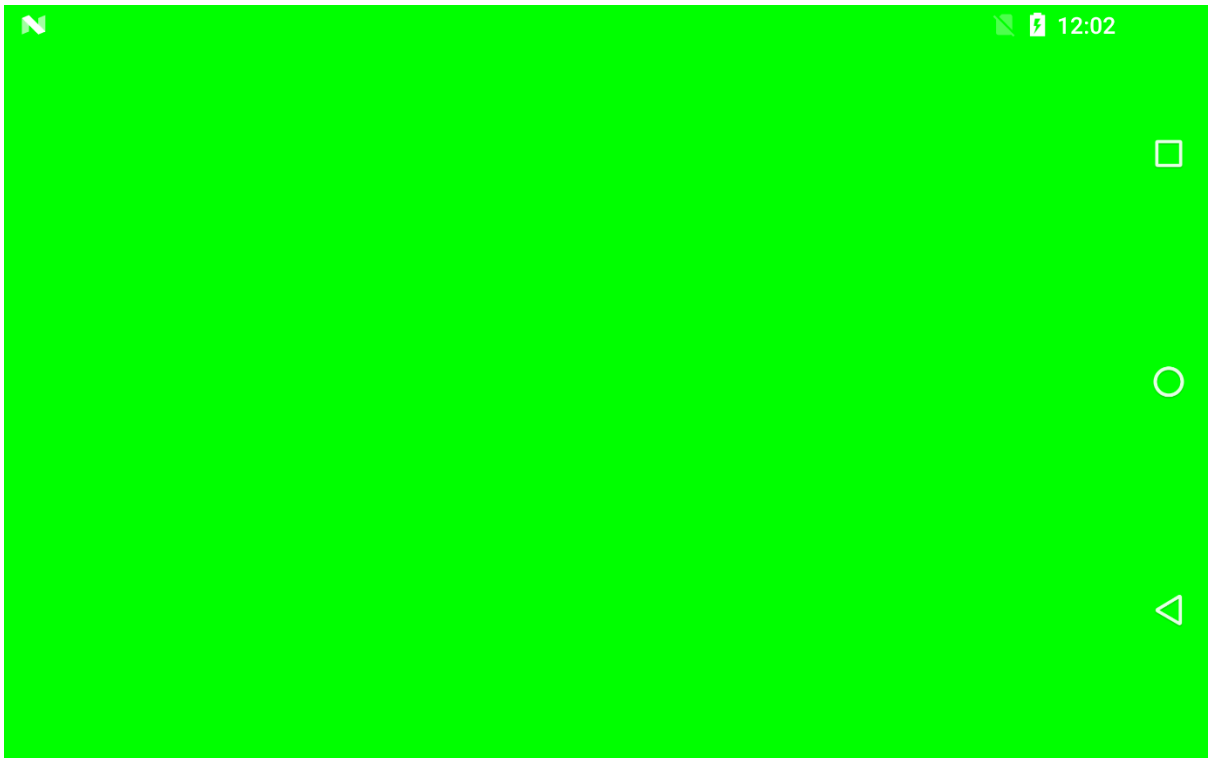
This Vulkan example shows a cornell box scene with simple point lighting. It is accompanied with a walk-though tutorial on its implementation. In this tutorial, the lights are implemented using a fragment shader that iteratively adds brightness and diffuse coloring depending on light positions and colors in world space. The fragment shader has a uniform containing a number of point light locations and colors. There is also a texture sampler, and this texture is mapped to a side of the small box using correct texture coordinates in the mesh object.

tut_monument



This example renders a simple monument mesh and overlays textured banner meshes on top, dynamically modified each frame by a compute shader. The tutorial shows using multiple descriptor sets, managing compute buffers and the synchronization required when using compute shader output as graphics pipeline input. This example also shows how to load in ASTC compressed textures.

validation



Validation is a useful feature provided by various Vulkan layers. Enabling validation in our framework will provide you with numerous warnings and errors that will help detect errors and allow you to write cleaner Vulkan code. In our framework, you can enable validation by calling SetUseValidation(true) in the sample constructor. The framework takes care of detecting and loading the appropriate external layers which implement the validation. Warnings and errors are caught in the DebugReportCallback function in the framework and are logged. This example renders a sold green frame by simply setting the clear color to green.