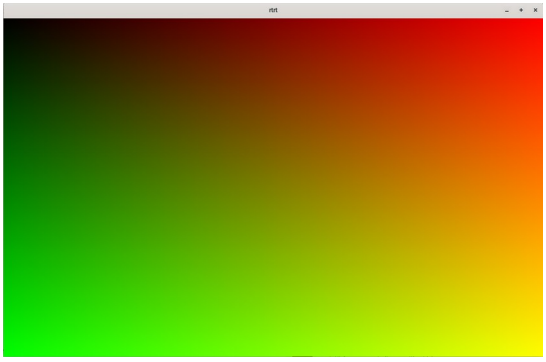
Project 1 First Light

Synopsis:

Let's call this "first light" – the first (boring) pixels displayed on the screen. The goal is to get through enough of the startup steps, and a few draw loop steps to get a minimal program displaying something (anything) on the screen.



Instructions:

Implement a succession of small procedures, and test each for correctness before moving on to the next. When the final procedure is in place, the picture on the right is the result.

So, one-by-one, starting at **createInstance** and ending at **createPostPipeline**, do:

- Complete the procedure according to the instructions provided in the code. Look for @@ comments for instructions. Instructions may include how to finish the code, how to verify its correctness, how to document your results in a separate document) maintained for submission and grading purposes, and how to cleanup/destroy your Vulkan objects before closing down the application.
- Add the required destroy command to procedure **destroyAllVulkanResources.**
- Uncomment the call to the procedure and build and run your program. You have completed this procedure when you get a successful run with no validation-layer errors.
- Go on to the next.

The **constructor** and **drawFrame** methods of **VkApp** in **vkapp.cpp** should look like this by the time you finish. The code for each of these functions, and a few extra auxiliary functions, can be found in **vkapp_fns.cpp**.

Submission

Submit a zip file containing

- Source code: *.cpp, *.h, all shader files.
  Do not include: Debug, **Release, .vs** or any other files maintained by Visual Studio.
- A report (in any documentation format) containing any output requested in the various @@ comments, and a screen capture of the final results, and a verification that your program runs with no validation-layer messages.

```
VkApp::VkApp(App* _app) : app(_app)
{
   createInstance(app->doApiDump);  // -> m_instance
   assert (m_instance);
   createPhysicalDevice();          // -> m_physicalDevice i.e. the GPU
   chooseQueueIndex();              // -> m_graphicsQueueIndex
   createDevice();                  // -> m_device
   getCommandQueue();               // -> m_queue

   loadExtensions();                // Auto generated; loads namespace of all known extensions

   getSurface();                    // -> m_surface
   createCommandPool();             // -> m_cmdPool

   createSwapchain();               // -> m_swapchain
   createDepthResource();           // -> m_depthImage, ...
   createPostRenderPass();          // -> m_postRenderPass
   createPostFrameBuffers();        // -> m_framebuffers

   // createScBuffer();             // -> m_scImageBuffer
   // createPostDescriptor();       // -> m_postDesc
   createPostPipeline();            // -> m_postPipelineLayout

   // Everything below here remains commented out for now.
   ...
}

void VkApp::drawFrame()
{
   prepareFrame();

   VkCommandBufferBeginInfo ...;
   beginInfo.flags = ...;
   vkBeginCommandBuffer(m_commandBuffer, &beginInfo);

   {   // Extra indent for code clarity
      //updateCameraBuffer();

      // Draw scene
      // if (useRaytracer) {
      //     raytrace();
      //     denoise(); }
      // else {
      //     rasterize(); }

      postProcess(); // output to swapchain image.


   }   // Done recording;  Execute!
   vkEndCommandBuffer(m_commandBuffer);

   submitFrame();  // Submit for display
}
```