Pre-test interview

- What education do you have?
    - Medialogy for 2 years at AAU, dropped out.
    - Coded since he was 8 years old
    - Self taught programmer, almost no formal education.
    - Currently working at CodeCreator (non gaming industry firm).
- What work experience do you have in programming?
    - Rendering engines
    - Freelance programmer
    - Interceptor (2014) and Sumo Digital when in the game dev industry
    - Currently working at CodeCreator (non gaming industry firm)
- Which programming languages have you worked in?
    - C/C++ and a good bit of C#. Some visual basic. Debugging Assembly for consoles. HLSL, GLSL and a bit of Cg.
- How much experience do you have in graphics programming?
    - Which APIs have you utilized?
        - Direct3D 11 is very familiar, Direct3D 9 and 10 in addition.
        - Some OpenGL core. Close to OpenGL ES. So modern OpenGL.
        - Any experience with Vulkan or Direct3D 12?
            - Very little.
            - Have had some dialog about using them at earlier work.
            - Already understands the benefits of commands.

Tasks

Triangle

16:20
Starts by wanting to create a vertex buffer with a triangle.
Prompted to use GLM, already familiar with GLM.
Expects a default winding order. Told that we use counter clockwise.

Wants to create shaders
Asks how shaders are compiled, told that it happens at runtime.
"Why are you using a unique pointer?" Brings up that he may want more pointers to a single object. Could do shared pointers, but requires us to have a compiler with move semantics.
He stresses the importance of creating atomized objects as to increase portability.

Attempts to compile vertex shader before setting up parser.
Asked if it was called fragment or pixel shader (DX terminology)
Figures out without help he needs to setup a parser.
Has no confusion about the use of entry point in shader compile methods.

Figures that you need a global device, surprised that you can simply construct it as an object.
Looks for documentation on how to construct Device.
Relatively quickly finds enumerate device.
Notes that enumerate device is dangerous due to move semantics.
Figures that he first needs to define a surface, before enumerating devices.
Looks into source code for seeing what features and extensions are available for device.
Likes that it is easy to say what a device supports.

Knows what a swapchain is, surprised that it is not required.
Takes the first device returned by enumerateDevices.

(gets hold of device[0] in a interesting way, devices[0].get()->createSwapChain(...))
Figures that you need to create a swapchain with the device.
Talks about what format the API would rather want.
Had to be prompted to use tripple buffering over single buffering when creating swapchain.
Not quite sure what PresentMode::eMailBox is, has it explained and it makes sense.
Thought that framebuffer counts indicates how many render targets you have.
Thinks that using a size_t is very large for a framebuffer count. Maybe use an enum instead?

When a swapchain is available, he looks to the loop and wants to clear it.
Tries to clear swapChain without commandbuffers, directly from swapchain. sets application to debug, tries to run application and gets an error.
Sets application to release, still gets an error (string too long exception).
Stresses that we should focus on getting our error messages through.

Now wants to make a program.
Again comments on the unique pointer centrism of the API. (negatively)*(“what will happen with those unique pointers once we get out of scope?”*)
Moves are expensive, so a unique pointer may not be so good for the performance of a low-level API.

Went from his unique way of using device to the way the wiki presents it.
Comments on unique pointer centrism again. You’re not quite sure what kind of code would execute when moving unique/shared pointers. Claims it makes it harder to see what the CPU might be doing behind the scenes, it can damage performance.

Goes onto create a vertex buffer.
Seems to like createVertexBuffer function, because of generics.
However generics can also be dangerous when it comes to executable size.

Asks if index buffers are required, is told no, doesn’t.

Attempts to create commandbuffer as the next step.
Thinks that a renderpass is an entire command buffer.
Sees that he needs a renderpass, uses it.

Expects an offset for viewport, so that you can render parts of the screen.
Expects that format on renderpass should match the one on the swap chain.
Explores what the RenderPass can do (without looking at documentation), as he is curious.
Not the biggest wiki user, looks more into the header.

Creates a renderpass, which expects a depth buffer. Doesn't yet match with swap chain.
Guesses that it is in record that you bind the rendertarget and render pass.
Takes an interest in the lambda.

Got some prompts for how to setup the lambda for recording commands.
Happy to find the clear within the recording lambda.
Also clears depth buffer
Goes to execute, then finds out that the command buffer is for an entire scene. While the subcommand is more specialized to objects.

Then goes onto the subCommandBuffer.
Sets the vertex buffer first, then talks about lambdas and unique pointers, makes the lambda a call by reference instead after prompt.

Comments again on his dislikes with the unique pointer design. Easy to get out of scope issues with unique pointers.
Says he understand why we use "newer features" such as unique pointers, blames C++ for having become too complex over the years.

Points out that we are not very compatible with languages other than C++.
Difficult for others to use our DLL.

Asked what the purpose was for setDynamicIndex, was told it was used for dynamic buffers.
Figures out how to execute the subcommandbuffer in the commandbuffer.
Prompted to use initializer list for execute on primary command buffer.

Relatively quickly figures out that he needs a graphics queue.
Submits the command buffers, then presents within the loop.

After failure to draw, is prompted that he needs a depth buffer in the swap chain.
Launches application again, gets a device lost exeception.

Doesn't use stencil often, so not too worried that we do not support it.
prompted on to change format to include the alpha channel. Runs application and gets a black screen.

Changes his vertices data, gets a triangle on screen but it flickers.
Gets told it happens cause of how he only records commands for one frame, moves commandbuffer to loop and launches application again
17:11 renders a triangle successfully

Sequential Textured Cube

17:12

Starts by changing out the triangle shader to the mvp shaders.
Thinks it's nice that we focus on compatibility with GLSL.
PGFx is mentioned for portability between APIs.

Guesses that our target user group are non-engine game programmers.

Asks about how we define our vertex input.
Talks about how you usually use a vertex definition in different APIs, which give more flexibility. A mismatch between vertex buffer and shader would be an issue, where the developer does not know what happens when they get an error. But does like that he has to do less as a developer.

Figures that he needs to create a uniform buffer.
But he actually starts off by creating a dynamic uniform buffer.
Facilitator explains the purpose of dynamicbuffer after participant asks what count is used for.
Didn't expect that we'd handle offset and that he did not have to calculate size.
Creates a dynamic uniform for both buffers.

Likes the handling of uniform buffers. Can reuse them.
Mentions that it'd be nice to resize these buffers.
At upload, likes the use of terminology in API *"We both use the same terminology"* .
Likes how upload and download of buffers further makes reuse possible.

(Hasn't looked at documentation for this assignment, asks Facilitator for assurance when in doubt)
Guesses that upload to buffers occurs before we send commands.
Creates the perspective matrix and then the world matrix.
Facilitator prompts to turn window width and height into float when creating perspective matrix.

At upload to dynamic buffer, thinks index is a count. Is prompted that he can use another overload to upload to the entire buffer at once.

Creates matrices and then tries to run without binding. Doesn't like that this crashes it.
Expects to bind on command buffer. Not quite sure where else to bind buffers.
Tries the uniforms themselves.
prompted to bind uniforms with the renderpass object.
Not at first sure what binding name is for, but at promt figures it out.

Surprised that we do not support multiple uniforms within an interface block.

Sees that we do not get the correct clearing color for the background even after calling.
(bug on our end, his implementation should have worked)

Doesn't get anything on screen.
Goes to use drawIndexed. But doesn't set the index buffer for usage. After failure figures to set the index buffer.

Is prompted to look at the shaders, and figures out that he needs a texture. Changes shader to confirm it before rotating.
Gets a cube on screen, but untextured, spends some time rotating it before going to use a texture.

Uses createTexture2D.
Figures that he needs a sampler in addition to a texture, goes to create that with no issue.

Figures that a sampler will function within the command buffer.
Doesn't think it'd be bound on renderpass at it'd then be the same sampler through all commands.
Think there might be an issue with having the same sampler through a render pass.

<u>17:46 successfully renders a cube with a texture (and it rotates, despite it not being part of the assignment).</u>
Begins to render more cubes.
Wants to create 1000 cubes.
Doesn't like the use of vectors since it forces the use of the std library.

After this, wants to resize uniform buffers for model.
Tries using draw indexed to draw several cubes.
Runs, gets only a single cube.

We do not have dynamic samplers yet, could be interesting.
Critiques on lack of error information.
Critiques on debug info in release mode (visual studio, not our API)
Figures out the crash happens due to setDynamicIndex. Tries to investigate why.
Critiques lack of access to source code

18:08 Code should work as is, but doesn't.
18:08 Currently test is halted, try to move test code over to client code.

Critiques lack of logging for debugging.
Suggests that we make our own logging system.
Had to move participant code to our old system.

18:21 Got 1000 cubes.
It created an issue to have the viewproj matrix be a dynamic uniform buffer instead of static.
Participant is back at the computer and test is resumed.

Parallel Textured Cubes

18:22
Is explained how to do parallelism by constructing subCommandBuffers in parallel.
Is prompted to use reference wrappers.

Dislikes the heavy use of std PAPAGO requires, belives it is too restrictive for developers. API should be closer to the language in order to support more language features, and thus give more freedom to developers in which datatypes they can use.

Asks for help how to use the ThreadPool.(not part of our API)
Mentions that we should be able to do parallel execution when we give a lambda to record.
Used (); at reference wrapper instead of ; fixed the compile error.
Tries to run application, gives a null pointer exception.
Gets around this by giving index as a parameter.

18:38 got it to work!

Shadow Mapping

Walks through our own example .
Expects first pass to get an off screen buffer.
Seems fine with it.
Memory requirements are on the GPU as to quickly load data into the registers. Mentions that if he doesn't know that this automatic padding is present in our API then he gets a performance hit. Could maybe pad memory on file.

Make it more clear for the user, where we are doing automatic things for them. Maybe update the documentation.

Post-test interview

- What did you think of the tasks?
    - Found it interesting.
    - Doesn't like unique pointers, would prefer raw pointers or handles
    - Needs some better error handling.
    - Keep the API close to the language, not too much std.
    - Could do intrusive referencing, to have the objects release themselves.
    - Very easy to use. But may not use our API as it only functions on top of Vulkan. Mentions he wishes the API was more platform agnostic.
    - Level of difficulty?
        - Not too difficult, shadow mapping would have taken a longer while.

        - Don’t use exceptions to the user, as they are expensive. Casting exceptions between dlls is not very performant. (mentions how the PS4 compiler doesn't have it due to industry standards)
            - Would usually have every return have some kind of error value.
- What’s your first impression of the Papago API?
    - Seems easy to use, but what if I wanted to compile for mac?
    - Wants more cross-platform support.
    - Suggests using cmake for building, if we wanna deploy to something like android.
    - Suggests that we compile shaders at compile time.
    - Mentions something about reflection on Spir-v.
    - VGFX has a Vulkan wrapper.
    - Wanted to see more structure over the API via documentation or CMake.
    - Does like the Wiki, but finds it slightly lacking
    - Dislike no access to source code.
- Would you like to keep using Papago or use another API?
    - For now he would use something else. As we tie in tightly to Vulkan.
- What was good in Papago?
    - Resource handling was simple
    - You don’t need to do much to get something up on the screen.
    - Expects the resource binding to function asynchronously.
- What should have been better in Papago?
    - We’re bound too much up on the standard library.
    - More cross platform support.
    - Templates increases executable size.
- Cognitive Dimensions Questionnaire:
    - Abstraction level
        - Did the API provide adequate control? Too much?
            - Had adequate control.
            - Doesn’t think that geometry and tesselation shaders would take a lot of time to include. Also would like compute shaders. Compute shaders are more portable than OpenCL and CUDA.
        - What was easiest to do?
            - Very easy to make a triangle.
        - What was most difficult?
            - A bit difficult to understand dynamic index. How to render several instances.
    - Learning style
        - Would you be able to use this API without documentation?
            - Yes, because he knows some of the base principles. What is most important is what is a renderpass? what is a graphics queue?
        - Do you think you could learn it by exploration alone?
            - yes, I think so. Still missing logging feature though.
        - What was the most useful resource to learn the API?

            - Intellisense, write the comments first then you can create a wiki from that.
- Working framework
    - Did you have to think about too many different elements at once?
        - Not too much. The terms come from the other APIs so it makes sense.
        - Figuring out what methods takes some objects can be difficult.
        - Maybe have a graph explaining dependencies.
    - How many elements did you feel you needed to keep in mind during coding? Which elements?
        - Not really. It’s a small test case, but if it was bigger he would most likely split out the code a bit.
- Work-step unit
    - How much work did you have to do in order to enact a change in the program?
        - Very little had to be changed.
        - It was cool that we treated resources in the same manner.
        - Was this a fitting amount of work?
            - Yes
- Progressive evaluation
    - Did you feel that you could execute the application during coding in order to evaluate your progress?
        - No, lacking of logging feature during failures of the application. Expects logging from debug version of library.
    - What did you think of the feedback given through the API?
        - Not good
- Premature commitment
    - What did you think of the dependency between API objects?
        - Hard to tell, kind of confused that texture had to be tied to a render pass. Relationship between renderpass and command buffer was not explicit.
    - Did you ever have to make a decision before all information was available?
        - Yes, such as when I had to find out how to bind the resources.
- Penetrability
    - How easy was it to explorer the API features?
        - Easy thanks to intellisense.
        - Very 1 to 1 with other APIs, still low level.
        - Can’t do more or less than Vulkan, just makes it simpler.
    - How easy was the API to understand?
        - Pretty easy
    - How did you go about retrieving what you needed to solve the tasks?
        - Mostly intellisense, and otherwise wiki.
        - Virtual method calls on our interfaces are a performance hazard.
- API elaboration

        - Any features, which were missing?
            - Happy to have the thread pool available as part of the API. Also likes that it is kept seperate. Best that users give the number of threads.
        - How would you expand upon the API?
            - More portability and usability from something else than Visual studio.
    - API viscosity
        - How difficult was it to make changes to the system, when parts of it were already coded?
            - Yes
    - Consistency
        - After drawing a triangle, could you infer the remaining features of the API? How?
            - Yes
    - Role expressiveness
        - Was it ever difficult to tell what the role of different classes and methods were? Which?
            - Yes, Commandbuffer, renderpass and graphicsqueue could be confused with one another.
    - Domain Correspondence
        - Were you previously familiar with the terminology of the API (commands, command buffers, queues etc.)?
            - All of them but dynamicIndex and subCommandBuffer, grapcshiqueue and renderpass.
        - Does the naming of these elements make sense?
            - Yes
        - Would you like to change any of the names? Which and why?
            - Renderpass to Pipelinestate
            - Dynamic Index, not quite sure about the difference between static and dynamic. Dynamic sounded like the way it was stored, needing to be updated often. MultiBuffer or something would be better.

19:45
35 min overtime.