Pre-test interview

- What education do you have?
    - Bachelor in Mediaology and taking a master in VGIS. 10th semester.
- What work experience do you have in programming?
    - Programming on each project
- Which programming languages have you worked in?
    - C#, some C++ and Java.
- How much experience do you have in graphics programming?
    - Lecture on 7th semester and lots of graphics programming on the 10th semester in Cg with Unity.
    - Which APIs have you utilized?
        - OpenGl
        - Cg
        - Compute shaders in Unity
        - Any experience with Vulkan or Direct3D 12?
            - No

Tasks

Triangle

13:10
Starts thinking about vertex buffer and index buffer.
Says it's been a while he's worked in C++. Doesn't have experience with glm.
Prompted with C++ syntax (not part of our API)
Creates a vertex vector, goes on to create an index vector.

Now wants to pass this data to a vertex shader.
Is not that familiar with GLSL code.
Searches through the Wiki for how to use the vertex shader.
Copy pastes code from wiki to the project

Is prompted that he needs to construct a parser.
Familiar with entry point.
Seems a bit stressed/nervous about being tested.

Finds out he needs a device when trying to create a shader program from the shaders.
Heavy wiki user.
Is prompted to look at IDevice documentation more closely.

Unsure what Surface is used for, when enumerating devices.
Goes to look up how the surface is created.
Now has a shader program and wants to pass some input to them.

Finds IDevice::CreateRenderPass, sees that before he can use it he needs a color buffer.

Does not understand the purpose of the commandbuffer, facilitator explains it.
Has graphics queue explained.
Can see that commandbuffer needs to set vertex buffer, so creates that from device.
Succeeds and then creates an index buffer.

“What’s the difference between a command buffer and a sub command buffer?”
Starts by creating a regular commandbuffer, and is then prompted to look at the methods.

Thinks about plugging a vertex buffer into record.
Goes back to looking at how a render pass is created. Giving it window width and height.
Likes it when possible to copy paste from wiki.
Looks up how the Format enum works.

Has to figure out from prompt that he has to dereference some object in order to pass them in as object because they are wrapped as unique pointers.
Unique pointers gives issues when having to pass them around objects.
Has not used pointers since 3rd Semester.

From creating render pass, he figures out that he needs a swapchain.
Creates a color buffer at first, figures to use same format as in renderpass.
Thinks PresentMode is a global enum at first, Has to be promoted that it lies on IDevice, maybe because of lack of documentation.
Now has a swapchain and goes back to the command buffer.

Is told in detail how to use lambda expressions, as he is unfamiliar.
After a bit, understands the general idea behind lambda usage for record.

Subcommand buffers must contain sub commands.
Mismatch in documentation with ISubCommandBuffer and ICommandSubBuffer.
Feels that Papago is very low level.
Happy to find methods on IRecordingSubCommandBuffer.
Sets the vertex buffer, then index buffer and finally thinks a bit and then uses drawIndexed.

Then writes execute on the main command buffer, is prompted to use intializer list.
Figures that execute will draw the triangle, after putting it on the graphics queue.
Submits the commands after a bit. (submit commands example is a bit misleading).

Tries to run the application, gets syntax errors. (uses device before he has created it).

Unique Pointers giving issues in setVertex and Index buffer methods in subcommand buffer.
From error figures that indices need to be uint16_t instead of int16_t.

14:19 renders a triangle on the screen, outside of loop.
Is prompted to move graphicsqueue code to the render loop.

Triangle flikkers, he only records to one framebuffer.
Is prompted to move commandbuffer down to the loop.
14:22 renders triangle inside the loop.
Understands why the command recording has to happen in the loop.

Sequential Textured Cubes

14:23
Is prompted to switch triangle Shaders out with mvpTexShaders. Recognizes matrices.
Figures that he needs to pass in the view projection matrix.
Figures that he should pass shader parameters into the subCommandBuffer.

Wants to make a sampler first, looks to device.
Thinks an ISampler is equivalent to a sampler in GLSL. Familiar with filters and wrap modes.
(eMirrorClampToEdge will always crash, remember to fix)
Wonders where a texture needs to be loaded in.
Uses readPixels to get the data from the texture.
After a while looks to the image resource. Figures it for passing image data to and from the GPU.
Figures out to use IDevice::createTextureSampler.
Then figures that he needs to upload the data. Does it without problems.

Goes to make the model resources before binding the image.
But then figures out he has to bind on the render pass.
Readily binds by name.

Unique pointer dereferencing doesn't seem to be an issue anymore.
Does not seem familiar with interface blocks in GLSL.

Spends some time creating glm matrices and finally uploads to a uniform buffer.
Goes onto trying to upload model matrix to the same buffer, is prompted to create a new buffer.

Runs code without increasing the number of indices to draw. Renders a textured triangle as a result.
Figures out he gave the wrong number of indices to draw.
15:00 Gets a single cube drawn
Figures he needs to rebind the buffer at each frame.
Takes some time to get the cube to rotate
Rotates but does not clear screen.
Figures it doesn't get cleared.
“Have to do everything yourself with (low level) programming”
Clears framebuffers for each frame and successfully renders the rotating cube.

15:10
Goes on to attempt rendering more cubes.

Copies the subcommand buffer.
Mentioned at first he should make another renderpass.
At prompt figures out that he needs to change the model matrix.
Looks at wiki for guidance.
Prompt from facilitator: “You have to use something you havent used yet” (referring to dynamic uniform buffer)
Finds the dynamic uniform buffer, figures that it is used for creating an array of different model matrices.
Guesses that size is per object and that count is the number of objects.
Uploads buffers per index within the main loop.

Replaces the old bound uniform buffer with the dynamic buffer.
Tries to run before setting dynamic index, gets a single cube.
Prompted to look at where commands are submitted and uses SetDynamicIndex.

15:24 successfully renders 2 textured cubes.

Parallel Textured Cubes

15:26
Has tried some parallelism with compute shaders.
Figures that parallelism has something to do with the commands. One subCommandBuffer per cube.

Makes a copy of subcommandbuffer.
Starts to try and enqueue a subcommand recording onto the threadpool.
Writes two full enqueues.
Not familiar with std::futures, is explained how they work and then waits on them.
15:39

Shadow Mapping

Post-test interview

- What did you think of the tasks?
    - Good enough standard tasks.
    - Level of difficulty?
        - Well enough, some trouble remembering C++ things.
- What’s your first impression of the Papago API?
    - It makes sense when you get into it. Recognizes stuff from OpenGL.
- Would you like to keep using Papago or use another API?
    - He thinks so yes. But what is the purpose of the API?
        - Being told about CPU-boundness he sees the idea.
- What was good in Papago?

    - Holistically everything makes sense. Recognizes shader usage and some type of parameter binding.
    - Command buffers were cool.
- What should have been better in Papago?
    - Not really used to pointers, so perhaps being able to code in a language like C#.
- Cognitive Dimensions Questionnaire:
    - Abstraction level
        - Did the API provide adequate control? Too much?
            - Yes certainly. But of course it's low level, used to mainly shader programming.
        - What was easiest to do?
            - Subcommand buffers made sense along with bind resource. Used to binding per variable name.
        - What was most difficult?
            - Starting up by making shaders and creating the render pass took a while to get into.
    - Learning style
        - Would you be able to use this API without documentation?
            - It would be difficult. Did not know all the terms, but a lot of names also made sense.
        - Do you think you could learn it by exploration alone?
            - Yes, of course command buffers were new. But could perhaps google a solution.
        - What was the most useful resource to learn the API?
            - Thinking to the next method and the input required for it.
    - Working framework
        - Did you have to think about too many different elements at once?
            - No, at first there was a lot to handle. But now sitting back. Maybe a bit much to handle when setting up shaders.
        - How many elements did you feel you needed to keep in mind during coding? Which elements?
            - Handling uploads with the different matrices.
    - Work-step unit
        - How much work did you have to do in order to enact a change in the program?
            - Not a lot of work required. Other vertices, indices and then the resources. Makes sense.
            - Very little difference when enacting multithreading, but he is of course unfamiliar with some of it.
            - 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?
            - Yes. Good as sometimes you just gotta try and see.

        - What did you think of the feedback given through the API?
            - Most of the feedback is visual.
    - Premature commitment
        - What did you think of the dependency between API objects?
            - Yes. Could pick a function and then go backwards from there.
            - Sub commands for a command buffer made sense.
        - Did you ever have to make a decision before all information was available?
            - Usage of command buffers for draw calls was new and confusing at first.
    - Penetrability
        - How easy was it to explorer the API features?
            - The wiki was fine, some cross-referencing could be nice.
            - Could also use intellisense fine.
        - How easy was the API to understand?
            - A bit difficult for a start. Had to get use to defining everything yourself.
            - Names made sense mostly.
        - How did you go about retrieving what you needed to solve the tasks?
            - Using functions and going backwards.
    - API elaboration
        - Any features, which were missing?
            - Can't think of anything specific.
            - 3D textures
        - How would you expand upon the API?
            - Dynamic was a bit weird. MultiBuffer resource would be a better name.
    - API viscosity
        - How difficult was it to make changes to the system, when parts of it were already coded?
            - Already noted
    - Consistency
        - After drawing a triangle, could you infer the remaining features of the API? How?
            - Already noted
    - Role expressiveness
        - Was it ever difficult to tell what the role of different classes and methods were? Which?
            - Command buffers and recording was new. Graphics queue as well. Renderpass seemed to make sense as a name.
    - Domain Correspondence
        - Were you previously familiar with the terminology of the API (commands, command buffers, queues etc.)?
            - Most of them. Command buffers and dynamic, maybe queue.
        - Does the naming of these elements make sense?
            - Yes mostly

- Would you like to change any of the names? Which and why?
    - Already noted.