# Pre-test interview

- What education do you have?
    - Datamatiker. On 4th semester of Computer science. Not a lot of experience before that. (a tiny bit of programming in high school)
- What work experience do you have in programming?
    - With internship at UCN.
- Which programming languages have you worked in?
    - C#, then C++.
    - Also a bit of Java, web languages like PHP and JS.
- How much experience do you have in graphics programming?
    - It's a hobby. Used it for a project on Datamatiker.
    - Which APIs have you utilized?
        - OpenGL 3.0.
        - Any experience with Vulkan or Direct3D 12?
            - No.

# Tasks

## Triangle

Start 13:09
What do I need for beginning?
IDevice seems important from documentation.
Not sure if Vertex data should be defined in init.  This is the first point of attack.
Haven't heard of index buffer before.

Needed help to figure out that we need a device.
Using examples heavily for setup.
Been a while since C++, so needs some reminders. Not part of our API though.

Thinks that there is boilerplate present, like in OpenGL.
Surprised at our use of unique pointers.

Recognizes glm.
Had to remember where the origin was in coordinate system, when creating the vertex data. (next time include a triangle mesh)

Vertex buffer was easy to make. What now?
Looks at shaders, natural next step.
Then looks at IShader, followed quickly by parser.
Thinks that source is a path.
Used to writing a shader and then linking together.
Figures out the use of entry point, maybe have main be a default?

Mistook function definition for a code example.
Can't remember if compile path is also required in OpenGL.
Entry point was written as "main()".

Then onto the fragment shader.
Wants to look more at documentation before continuing coding.
Remembers buffers being something you write images to. (in the context of OpenGL)
Then goes onto renderpasses.
Thinks that constructor is on the renderpass. (description wrong, in renderPass)

Reminded that Renderpass can be created from Device.
Here figures out that a shader program contains shaders.

Thoroughly reads the documentation, really wants to understand what is going on.
Doesn't know if he will be using the shaderProgram later.

CreateRenderpass, at first not sure what width and height are.
Unsure what a RenderPass is, participants assumes it is like a frame.
A facilitator explains what a RenderPass is.

Starts creating a render pass.
Halts when he needs to pick a format. Thinks about picking a stencil format for render pass.
He thinks that format becomes relevant when you begin to work with buffers that need to be presented on screen.
Expects format to be...Well he thinks it has something to do with the color of the fragment shader.

Prompted on an earlier comment that he needs to create an image.
Was unfamiliar with swapchain as a term, but did know about double buffering.
Looks at creating a swapchain, then at the vertex buffer.

Thinks that creating a swapchain it will have some methods on it that are important.
What is a presentmode?
Figures then that the swapchain will be used in some other context.
Figures that the swapchain format should match renderpass.
Has only ever used single buffering. Needed to be told we only support triple buffering.

Looks at IDevice, thinking he may need to do more setup before moving on.
Facilitator confirms setup for IDevice is complete for now.
Misses the draw method, wanted to work backwards.

Looking at the graphics queue after a prompt, figures the necessity of command buffer.
Sees the different kinds of command buffers. Looks to ICommandBuffer first.
ICommandBuffer documentation does not explain what commands are.
A facilitator explains the relation between CommandBuffer and Recording CommandBuffer.

Figures that an ICommandBuffer is the first thing we need to create.
Tries to create an IRecordingCommandBuffer with device.
Sees it doesn't work, instead uses the ICommandBuffer.

Surprised at the use of lambda expression for recording.
Needed help with recording commands.
Notes he has used lambdas before in C#.

Thinks that it would be important in loop to use commands.
Facilitator explains command re-use is beneficial, and why it does not have to be done in the loop.
Figures that we should clear buffer and then execute.
Figures correctly that he should give the background color to clear buffer.
Wants to create sub commands within the lambda, but runs into some problems with unique command buffers and goes on to create sub commands outside.
Thinks that the command concept with resuage is cool however.
Thought it wasn't possible to get a reference to the recorded commands, facilitator explained the CommandBuffer is the reference to them.

Has to be prompted to fill the sub command buffer.
Expects renderpass to be used within the lambda.
Happy to finally see the draw command in the IRecordingSubCommandBuffer.

After a prompt, participant attempts to bind the vertex buffer to the GPU.
Confusion that vertex buffer is an IBufferResource. But then figures to use the vertex buffer.

After recording, looks up methods on the command buffer.

Facilitator needed to explain the relation between swapchain and graphicsqueue, in order for the participant to understand how rendering functions.
Returns to graphics queue.

Spends some time setting up private fields on the testing class.
Accurately guesses that present will present the image.

14:56 Participant successfully rendered a triangle. (In first try!)

Facilitator asks if participant likes how command recording rfunctions:
Thinks that commands are good.
Understands the usage for parallel. Has done parallel programming, but did not get to try out the parallel works.

## Sequential Textured Cubes

15:02

Facilitator helps participant with c++ syntax in the context of using the cube mesh (not part of our api)
After updating vertex data the participant runs program, gets a black screen, and an exception shortly afterwards.

After a prompt, participant tries to use different shaders to draw the cube.
Participant spends time inspecting the cube shaders, trying to understand how they work.

Figures from the shaders that we need to set some uniforms.
Figures it has to be done via the parser.
Remembers that you use strings to get shader parameter locations in OpenGL.

Gets a prompt to create a buffer resource before attempting binding one.
Gets a prompt to use device to create a buffer resource.
Asks what's the purpose for creating a buffer resource.

"Why have you made it so that you can upload several times?"
Participants gets help in constructing matrices (not part of the api)
Tries to upload matrix directly. Is prompted to upload it through a vector.

bindResource, "Name must be the name in the shader".
After the first resource bind, it makes sense how to bind the next matrix.

He guessed that he could reuse a variable 'mat'.
Guesses that he should use a texture 2D after being told about how a sampler in GLSL relates to an image.
(need a method for finding width and height of texture image being loaded in).
Recognizes filters within samplers as well as wrap mode.

Needed a prompt for setting up index buffer, participant never used index buffer before.
Prompted to change the name from vertexcountDraw to indexedDraw.

15:40 Cube in first try.

### Parallel Textured Cubes

### Shadow Mapping

## Post-test interview

- What did you think of the tasks?
    - Good 'hello world' type tasks.
    - Level of difficulty?
        - If I was alone I would have figured it out in due time.
- What's your first impression of the Papago API?

    - Some well known terms were used, also some new ones.
- Would you like to keep using Papago or use another API?
    - If I looked into it further that would be a possibility. However for my own needs I like the simplicity in OpenGL, which is supported by my own helper functions.
- What was good in Papago?
    - Less magic strings and things like in OpenGL, that was nice.
- What should have been better in Papago?
    - Not quite, not quite sure you can remove the boilerplate.
    - Needed some more steps in how to present instructions.
    - Maybe get a function for a cube?
- Cognitive Dimensions Questionnaire:
    - Abstraction level
        - Did the API provide adequate control? Too much?
            - Didn't see how to draw more cubes. No troubles. Felt he had control.
        - What was easiest to do?
            - Arranging vertex data made most sense. Didn't notice that he had to bind the vertex buffer.
        - What was most difficult?
            - Command system was difficult. How did they function together. Lamdas. Made sense when you were finished.
    - Learning style
        - Would you be able to use this API without documentation?
            - Would probably give up without documentation.
        - Do you think you could learn it by exploration alone?
            - Same answer as above
        - What was the most useful resource to learn the API?
            - He liked having everything placed on device. Nice resource.
            - Documentation and hints were important.
    - Working framework
        - Did you have to think about too many different elements at once?
            - There were a lot of elements to work with.
            - Objects flowed into each other.
            - Didn't have to think too much about things made a long time ago.
        - How many elements did you feel you needed to keep in mind during coding? Which elements?
            - Same as above.
    - Work-step unit
        - How much work did you have to do in order to enact a change in the program?
            - Faster than expected to go from triangle to cubes. Positive experience.
            - Was this a fitting amount of work?
                - Same as above

- Progressive evaluation
  - Did you feel that you could execute the application during coding in order to evaluate your progress?
    - No, I didn't expect it. But if you have your setup then you could probably evaluate progressively.
  - What did you think of the feedback given through the API?
    - Very happy that things ran at first attempt.
    - Like in OpenGL you're on your own. It is to be expected.
- Premature commitment
  - What did you think of the dependency between API objects?
    - On the face of it no, but expects that this is because something from Vulkan. But can't remember exact problems.
  - Did you ever have to make a decision before all information was available?
    - (missing)
- Penetrability
  - How easy was it to explore the API features?
    - It might have been easier without someone looking over your shoulder.
  - How easy was the API to understand?
    - Took some time to get the idea, but it started to make sense.
  - How did you go about retrieving what you needed to solve the tasks?
    - The trick was to use the documentation to look at the class in question or constructor on device.
- API elaboration
  - Any features, which were missing?
    - None
  - How would you expand upon the API?
    - API is fine as is.
- API viscosity
  - How difficult was it to make changes to the system, when parts of it were already coded?
    - It was relatively painless.
- Consistency
  - After drawing a triangle, could you infer the remaining features of the API? How?
    - Could, with some prompts, infer binding of shaders after drawing a triangle.
- Role expressiveness
  - Was it ever difficult to tell what the role of different classes and methods were? Which?
    - Could be difficult. Swapchain, Renderpass, Commands and resources.
- Domain Correspondence
  - Were you previously familiar with the terminology of the API (commands, command buffers, queues etc.)?

        - Has some OpenGL features. Shaderprogram as an example, GLSL he recognizes.
- Does the naming of these elements make sense?
    - New names make sense now.
- Would you like to change any of the names? Which and why?
    - No. Can’t come up with something better.