Opens

* Headset & funding:
  + Prospectus to be send out by EOD, and ask for hardware and funding needed (Brad)
* Update on existing hardware:

**Notes about VR and such:**

* The application sits on top of an API
* Physical VR headset
  + Compositor component - runs as a service on the service
    - Runs on the system connected to the VR headset
    - This is what sends the frames to the headset
    - Makes use of graphics driver
    - Custom display drivers
    - It’s job is to take incoming frames and push them over to the headset, depending on how fancy the compositor can simply push frames or do “time warp”
      * Time warp is when compositor gets frame that is rendered and it was rendered with an assumption as to where the headset was pointing
        + Usually not accurate because of delay

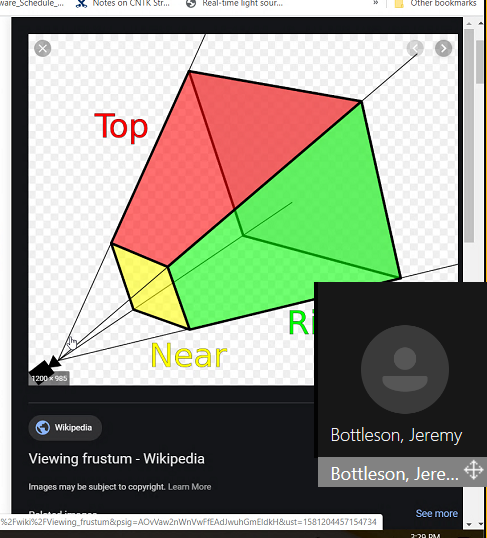
16 ms time slice, the game that is rendering at the start unity polls the compositor and gets the left/right eye frame, then it renders and push it out

Compositor wants to push frames to headset at last possible moment so that it’s most reminiscent to where the headset is looking on the screen

Poll sensors again see where head is looking, calculating delta and will warp the rendered frames its been given to make them look like the area where the actually head is looking

Makes frame that you render larger than the field of view

* + - * + Simple translations you can correct for using time warp
    - This is what STEAM VR does for you
    - What calls the compositor service?
      * The application (ex. unity) would connect to your compositor service
        + Unity provides frames for application and unity and rendering right eye and left eye and pushing these to the compositor service
        + Provides objects and physics for left/right eye view
    - The compositor is what talks to the headset and tells the headset what it needs
    - There are hooks both in unity and in the compositor service
      * It talks to the HMD to get where the headset is looking at
        + With headsets that don’t track eye movement, there is one value that is recorded
* **Frustrum** - when you have a camera or an eye, effectively there is a cone that projects out from the camera or eye and everything within that frustrom is ultimately what gets projected back to the eye



* With VR, you have 2 frustrums that you are rendering, a left eye view and a right eye view
  + Vive: has an IPD setting → the separation distance between the two virtual cameras for the left and right eye
* Two frustums would be rendered but with respect to where each eye is looking
  + The compositor service would take care of looking at those two images and merging them
    - The compositor may be able to give left and right eye frustums
    - The compositor knows about IPD
    - We need to intercept between unity and compositor service to get at these frustums to tweak based on the gaze vectors and how they’re representing
      * **Steam VR is our compositor service!!**
        + This may not be something that we can crack open
      * Google search: SteamVr API FOV
        + There is a VR wiki that exposes a lot of this to find where you can modify this value
  + We still want the two parallel frustums, so we should render the two parallel frustums, and maybe we warp the rendered image so it recenters to the location on the screen where the eye is pointing
    - Think Timewarp
    - We could perform our own warp on the rendered image
    - There is a spot where we could adjust the frustum on a per headset basis
      * We could have some file with our headset definition ← handled by compositor
        + Each version of the headset has very specific optics and this determines what filters need to be applied
    - We need to convince SteamVR to treat us as a Vive Pro Eye
    - Suggestion: adding in another layer “custom compositor layer” that goes in between the normal compositor and unity and this layer would handle talking to the compositor & getting the location of the HMD etc and tellng unity what to render
      * What needs to happen is: each frame is rendered with a wider field of view
        + Apply a rotation warp on those returning frames based on where the eye is looking and crop the appropriate field of view and then send that to the compositor
  + The pixels are on an x-y coordinate system
    - The frustum itself is in x-y-z coordinates
* **Foveated Rendering** - the idea that our eyes only really see detail in the very middle of our retina which only maps to a small % of our FOV
  + If you are aware of that, when you render you don’t have to try to render everything at super high resolution
    - Foveal area = high res
    - Blend region = mid res
    - Peripheral region = low res

How do unity and steamVR community?

* Unity works as a unity plugin to the compositor service
  + Write a standalone module that talks to the compositor, and then unity talks to the module
  + Suspecting that the application will create a link to unity most likely,
  + Or maybe the application makes a link to the strabismus correction wrapper
* Google search: SteamVR Unity Plugin
* Based on eye information, perform some change behavior -- add extra post processing step into the rendering
  + Study foveated rendering to see how these frames are being rendered

Research:

* Look into how foveated rendering is done
* SteamVR API -- look through documentation and what are the upward facing API calls that are exposed
* Looking at the Unity plugin for SteamVR to see what are the upward and downard calls that it does (check the API)

