Technologies used

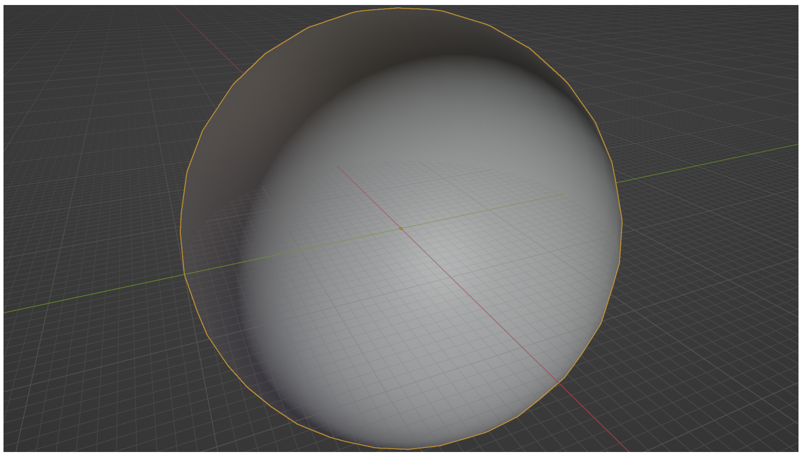
- Hardware: Vive Pro Eye w/ integrated 120Hz Tobii eye tracker
- Unity Version: 2020.3.X (X should be interchangable)
- OpenXR used for Gaze interaction and VR support
- ffmpeg used for video conversion
    - video compression formats used: MPEG4, VP8
- SRT subtitle format
- The latest SRAnipal runtime must be run to provide OpenXR Gaze Interaction data.

Usage instructions

1. Launch SteamVR
2. Launch a recent version of the SRAnipal runtime. You should see a red-eyed robot icon in the system tray.
3. Load the project into Unity 202.3.x
4. Ensure the scene `Assets/Scenes/SampleScene.unity` is open
5. Adjust settings on the "Basic Subtitler" script on the Subtitle Game Object (do not adjust the items above "Character Colors")
6. Press play to run the experiment.
7. Results will be recorded to a file named `out.csv` in the project's root directory. Ensure you rename this file and move it to a safe location before starting the program again.

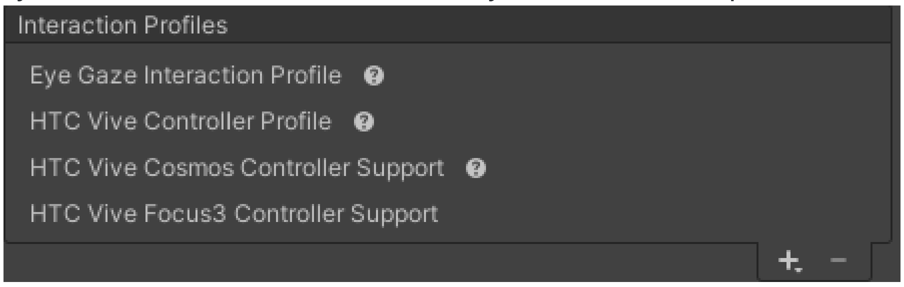
Notes: depending on the previous state of the project, you may need to enable OpenXR. To do this go to `Edit > Project Settings... > XR Plug-in Management` and check the box marked `OpenXR` . Also ensure that the checkmark labelled 'Testing' on the "Test Controller" script on the Main Camera object is unchecked.

Implementation details:

1. get video. Convert to VP8/vorbis video file with FFMPEG (only format unity supports on linux)
    1. Command: `ffmpeg -i ./holidays_2_s.mp4 -crf 10 -c:v vp8 -c:a libvorbis ./holidays_2_s.webm`
    2. **NOTE:** Video quality was poor, consider lower CRF in future. Also figure out if CRF is taken into account for VP8 encoding. Also note that VP8 encoding will always be slow due to no hardware support on AMD or NVIDIA.
    3. MPEG-4 does work well on the Windows version of Unity.
2. create inside-out sphere in blender, invert UV map x direction
    1. Import into unity.
    2. 
    3. This is used for easier third-person visualization, and does not appear to cause significant distortion on the video. 2- Create mouse-controlled test camera 1- click-and-drag = movement 2- only for testing on non-VR devices
3. use Video Player element to play video and apply to the inverse sphere as a texture. Loop the video using scripting.
4. Get subtitle parser from here https://github.com/AlexPoint/SubtitlesParser (MIT license)
    1. Note that subtitle files need to have the `.txt` extension to be properly recognized as a text file by Unity. Otherwise, it is considered a binary file, and is basically unusable in the interface.
5. modify parser to work with subtitle files from research project (these sub files had extra information, such as speaker index and subtitle x and y location)
    1. X and Y location are parsed to a Vector2. These correspond to pitch and yaw.
6. Create several placement methods for subtitles

1. Lock to cam: take difference in position as offset, for each frame, place subtitle at offset rotated by camera rotation
2. Lag to cam: similar, but instead of directly setting position and rotation of the subtitle, use slerp (spherical interpolation) - slerp prevents it from "cutting corners" on fast head movement
3. Appear and stick: similar to 1, but only do this when a new subtitle is available
4. 120° - get the rotation of the camera, and flatten to a 2D vector in the XZ plane. Get the angle from this vector to a static scene forward vector. Use dot product with global right vector to set the sign of this angle, and then round to the nearest 120° interval. Resulting subtitle position is generated like in `1`, but using this generated angle instead of the camera's 3D rotation.
5. Position in scene: use the X and Y parsed from SRT file as Euler X and Y in scene. Offsets are required (-90° for both X and Y). Z rotation of the Euler is always zero, and the subtitle is automatically mapped to look at camera.
6. Moving bubble: similar to Position in Scene, but the subtitle smoothly animates from one position to the next.
7. Comic bubble: placement is similar to "Lag to Cam," but this method adds a triangular "tail" to the subtitle that points towards the current speaker in the scene (a position as determined by the "position in scene" method). The tip of the tail is $\frac{1}{10}$ of the way to the speaker point, with a minimum length of 2 units. The based of the tail is attached to whichever of the four sides of the subtitle is closest to the speaker.

7. OpenXR is used for eye tracking

    1. Only get one vector, with position and rotation. (defined as a "Pose" in the OpenXR specification)
    2. Use Eye Gaze Interaction Profile in Unity, as well as HTC profiles
        1. 
        Interaction Profiles
        Eye Gaze Interaction Profile
        HTC Vive Controller Profile
        HTC Vive Cosmos Controller Support
        HTC Vive Focus3 Controller Support
    3. Eye gaze vector is recorded each frame as position and direction vector (x,y,z) and outputted to CSV with two time stamps (one for seconds since program start, and another for the video presentation time, which can be cross-referenced with the input SRT file).
    4. In order for this to work with the Vive Pro Eye, you need the *latest* version of the SRAnipal runtime, but you **do not** need the SRAnipal SDK in Unity.

        1. SRAnipal runtime needs to be run as Administrator.
    5. The gaze can be attached to a callback using the unity editor UI.
        1. A similar method is used for headset position and rotation.
    6. Directly using the SRAnipal SDK would not work for me, and caused frequent unity crashes.
    7. I added a simple raycast along the gaze vector to determine if the user is looking at the subtitle or elsewhere.

8. Implemented different text colors for different characters

    1. This is defined as a color list interface in the Unity Editor

    2. Each character is given a 0-indexed number in the SRT file. This is used to directly index into the color list.

    3. If the subtitle isn't indexed, or the index is outside of the list range, a default color is applied (white)

    4. this can be applied to any of the placement methods (via a checkbox in the Unity Interface).

9. Implemented Bionic Reading

    1. divide subtitle into words

    2. Each word has the first half bolded (by placing in `<b></b>` tags, as required by TextMeshPro)

    3. Words with an odd number of letters round the number of bolded letters down to the next lowest whole number.

    4. this can be applied (via a checkmark in the Unity Editor) to any of the placement methods. It can also be used on top of text colors for different characters.

10. Began implementation of word-based gaze detection.

    1. Each character in the Text Mesh Pro instance can return locations for each of its four extreme corners.

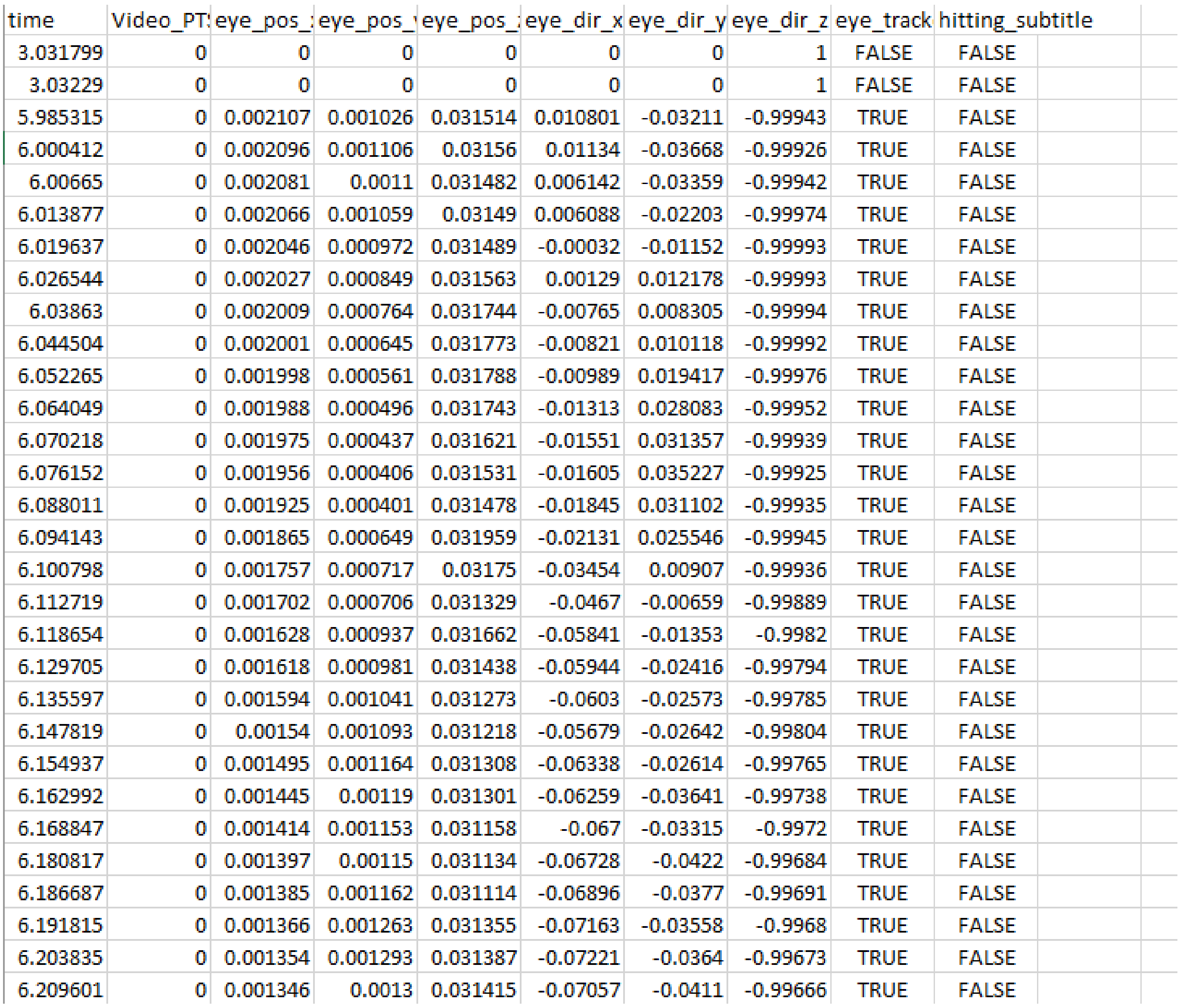
    2. This can be used (via maximum and minimum) to find the bounding box for each word.

    3. This would then be used to generate collision boxes for each word, which can then be ray-traced.

4. This requires much more infrastructure to implement than I have time to build for the remainder of the class.

Data example

The following is an example of data collected from this program:



time	Video_PT	eye_pos_	eye_pos_	eye_pos_	eye_dir_x	eye_dir_y	eye_dir_z	eye_track	hitting_subtitle
3.031799	0	0	0	0	0	0	1	FALSE	FALSE
3.03229	0	0	0	0	0	0	1	FALSE	FALSE
5.985315	0	0.002107	0.001026	0.031514	0.010801	-0.03211	-0.99943	TRUE	FALSE
6.000412	0	0.002096	0.001106	0.03156	0.01134	-0.03668	-0.99926	TRUE	FALSE
6.00665	0	0.002081	0.0011	0.031482	0.006142	-0.03359	-0.99942	TRUE	FALSE
6.013877	0	0.002066	0.001059	0.03149	0.006088	-0.02203	-0.99974	TRUE	FALSE
6.019637	0	0.002046	0.000972	0.031489	-0.00032	-0.01152	-0.99993	TRUE	FALSE
6.026544	0	0.002027	0.000849	0.031563	0.00129	0.012178	-0.99993	TRUE	FALSE
6.03863	0	0.002009	0.000764	0.031744	-0.00765	0.008305	-0.99994	TRUE	FALSE
6.044504	0	0.002001	0.000645	0.031773	-0.00821	0.010118	-0.99992	TRUE	FALSE
6.052265	0	0.001998	0.000561	0.031788	-0.00989	0.019417	-0.99976	TRUE	FALSE
6.064049	0	0.001988	0.000496	0.031743	-0.01313	0.028083	-0.99952	TRUE	FALSE
6.070218	0	0.001975	0.000437	0.031621	-0.01551	0.031357	-0.99939	TRUE	FALSE
6.076152	0	0.001956	0.000406	0.031531	-0.01605	0.035227	-0.99925	TRUE	FALSE
6.088011	0	0.001925	0.000401	0.031478	-0.01845	0.031102	-0.99935	TRUE	FALSE
6.094143	0	0.001865	0.000649	0.031959	-0.02131	0.025546	-0.99945	TRUE	FALSE
6.100798	0	0.001757	0.000717	0.03175	-0.03454	0.00907	-0.99936	TRUE	FALSE
6.112719	0	0.001702	0.000706	0.031329	-0.0467	-0.00659	-0.99889	TRUE	FALSE
6.118654	0	0.001628	0.000937	0.031662	-0.05841	-0.01353	-0.9982	TRUE	FALSE
6.129705	0	0.001618	0.000981	0.031438	-0.05944	-0.02416	-0.99794	TRUE	FALSE
6.135597	0	0.001594	0.001041	0.031273	-0.0603	-0.02573	-0.99785	TRUE	FALSE
6.147819	0	0.00154	0.001093	0.031218	-0.05679	-0.02642	-0.99804	TRUE	FALSE
6.154937	0	0.001495	0.001164	0.031308	-0.06338	-0.02614	-0.99765	TRUE	FALSE
6.162992	0	0.001445	0.00119	0.031301	-0.06259	-0.03641	-0.99738	TRUE	FALSE
6.168847	0	0.001414	0.001153	0.031158	-0.067	-0.03315	-0.9972	TRUE	FALSE
6.180817	0	0.001397	0.00115	0.031134	-0.06728	-0.0422	-0.99684	TRUE	FALSE
6.186687	0	0.001385	0.001162	0.031114	-0.06896	-0.0377	-0.99691	TRUE	FALSE
6.191815	0	0.001366	0.001263	0.031355	-0.07163	-0.03558	-0.9968	TRUE	FALSE
6.203835	0	0.001354	0.001293	0.031387	-0.07221	-0.0364	-0.99673	TRUE	FALSE
6.209601	0	0.001346	0.0013	0.031415	-0.07057	-0.0411	-0.99666	TRUE	FALSE

The `Video_PTS` value is 0 for the duration of this sample, as the video doesn't start for a few seconds while the program starts up.