VR Browser Project

Overview

- We can use voice commands to translate into words that can be used as search queries if user wants to use web browser through VR
- To render web browser, we have two options:
    - 1) build custom app based on Oculus Quest 2 Developer Platform
    - 2) build on top of VRChat, except this sounds fucking hard because it's possible that we are restricted to using only Udon, the language specific to VRChat itself, not sure if developers can use their own tools (e.g. Java, C++, C#)
    - The easiest way to do this is to replicate the Theater environment, and then working with the Chromium Web Browser API to render a fucking page, and then setting up STT, translate to text, then pass the string to query in google browser

Features

- Shutdown modes (define rgb intensity, time duration)
- Brightness control
- Web browser in VR; browser watch parties
- Portals

End Result

- Merge mindful/rest worlds with collaborative browsing; make mindfulness experience with social interaction
- Merge feature 1 and feature 2 (feature 1: create a rest world, feature 2: create webXR browser with oculus API wrapper; omitting many dependencies here of course)
- bringing the web to vr

Requirements

- [ ] define driver/browser/oculus app *endpoints*
- [ ] *simplify* down technical approach/implementation after *knowing what endpoints to use* and *mapping the actions of the user* to the *respective API endpoints of the Oculus Browser SDK*
- [ ] Implement *spec code*
    - [ ] Shell code
    - [ ] Logic code
    - [ ] Api code
    - [ ] Configure base code with Oculus Quest

System Implementation

Backend

- Mixins/Utils
    a. WebVRConfig : configure to headset
    b. OculusConfig : configure to core API
    c. VRInput : read operation where we: configure headset movement, reading movement, etc
    d. VROutput: write operation where we: configure output of movement and write that behavior into an action inside the *Portal* in *Session* and enter *Theater*
        - These classes are used to read in behavior/actions done in the vr environment.
- Core
    a. Browser
        - Browser object integrate webVR and chromium to render a browser into the quest app
    b. UserSession
    c. Menu
    d. UserAvatarConfig
    e. UserAvatar
    f. Chat
    g. Audio
    h. WebVRConfig
    i. Queue stores options for portal type
    j. Portal
        - RestingPortal
            - Users can leave the portal that is wrapped around the Session.
            - Restricted to 1 user per portal.
            - *Settings*
                - This would define and store the variable states
                    - We need to register users, authenticate with Oculus SDK, then auth our code and setup perms/users
        - WatchpartyPortal
            - Restricted to 4 users per portal e.g. max_users : int = 4
    k. SessionState
        - TerminationState
            - User who has admin permissions can start and end the state of the Session.
        - OwnerState
            - Tracks the state of the permissions, admin, owner information.
            - transferOwnership()
        - Session Storage
            - Store metadata/config of the session in a temporary storage file that gets flushed when the port gets closed e.g. the theater shuts down when there are no users in the session.
    l. BrowserPermissions
        - Define admin after >2 players join a session.
    m. SessionQueue
    n. UserSettingConfig
        - User defines settings. User creates new Portal and defines their Settings. Track this with Oculus-Native Mem Store
    o. VRController
    p. Users

User's Experience

- User enters app
- User sees two options, landscape background (minecraft vibes)
- options=["Resting Mode", "Collaborative Mode"]

Dependencies

- easyRTC
- JanusVR
- Networked Aframe
- WebRTC
- WebXR
- https://developer.oculus.com/unreal/

Technical Complexities

- webRTC server-client setup; gRPC and protobufs? How would Oculus take care of this problem?
- Renders, shaders, and creating virtual scenes/environments for games (akin to minecraft world, are there open source libraries that generate this for us for general vr apps, can we build a web browser into vrchat? That'd be awesome but we need VRChat developer access / trust … so unsure about how this process would work
- Keyboard mapping, formalizing the actions of the user, understanding the workflow of what actions they would take
    - I click options to choose what game mode I want to join (resting mode or collaborative mode)
- Merge understanding of WebVR api and Oculus Quest API for movement/controls and specify how the html/js can read in this behavior (onClick() → vr conjugate)
- The endpoints relevant, and porting Oculus Browser features for our own Oculus App
- Work backwards from functionality to technical implementation e.g. relevant endpoints and workflow e.g. control structure

Defining Multiplayer

- Have a theater with webRTC
- Avoid fancy multithreading and multiprocessing things relating to syncing data structures relating to user metadata; setup async functions for *voting, decisions, changing session state, transferring ownership, etc*

User Input

- Translating user input and finding endpoints for Oculus Browser API.

- VRTrigger
- ControllerConfig
    - Walk
    - Global
    - Custom
    - Rotate
    - Menu
    - Right Click
    - Left Click
    - Scroll
    -

Mode 1: Rest

- **Configure Settings:** Single user configures settings to define their:
    - *compute_night_shift(brightness : float, client_lattitude : long, client_longitude : long):* access user display and adjust temperature on Oculus Quest
        - **Precondition: we know the parameter data about the user's geolocation when using this function/**
        - *@param brightness*
        - *@param client_lattitude*
        - *@param client_longitude*
        - *Compute: est_client_location → compare against a list of various time zones to edit the temperature of their vr environment as such*
        - *@param quest_video_driver - it's probably not needed but useful to check if certain methods to edit temperature don't translate to the CPU cores that render the graphics on the headset*
        - *@client_geolocation*
        - *Compute: when is it sunset for the client in question (iter)*
        - *Compute: shift the color of the user's display*
    - *Session_time : int*
        - We can track how long we'd maintain the session time for.
        - Store the user input and pass as argument to *time.sleep(self.session_time)*
- **Map Rendering:** Prefetch a set of maps or scenes to render, similar to calm virtual worlds in VRChat, but ported over for the purpose of some scenery if the user wants scenery
- **Purpose of Feature:** User can reflect and recuperate, as well as fixing their eyes a bit with some helpful temperature changes they defined in the settings page that they browser upon clicking the "Resting Mode" option.
    - Challenge: figure out how to stitch these workflows together and remove all the unnecessary parts, as well as understanding our MVP (end product we want in the simplest way possible) and our MVA (minimum viable approach that works)
        - This implies that we cut out unnecessary complexities and features that are <10% optimizations and not base features.

Mode 2: Watchparty Portal & Theater

- Preliminaries
    - We need to get read the data from the owner
    - We need to find a way to get the requests from the other members of the server/session/portal
    - We can setup a voting mechanism where users can vote for certain shows, videos, websites to navigate/use
    - There can be a live voice chat as well as a text chat, though who'd use a text chat on VR?
- *Precondition:* The owner will make sure to authenticate their google / saml / sso before they create a session (the first and last time when they are a new user); this way we can enter netflix without there being privacy problems, waiting, slow queues, etc
- User *creates* new server
- *Create new session* and *render portal (render both the queue for the session e.g. waiting room and the map, possibly a dark room)*
- The user who creates the session is by default the owner and admin of the session
    - Session is run in a server; portal is just another word to represent the "multiplayer" session
- Owner/admin of session must be able to invite their friends so we need to use *oculus.user.getUserFriendsList()* and then process this *client* e.g. *user* into the session. Track all *clients*, *client_id* per user, *session_id* to link invitee and inviter (admin)
- When the owner chooses to close the portal (similar to krunker where games get full) the session will become closed and then the browser/queue moves into the next state.
    - Loading feature that primes people similar to when a map is being loaded (generally computers are fast enough to immediately render these maps, but they purposely make it slower so that it processes behavioral cue; we can do this by having a delay and rendering a preset graphic)
- Instantiate browser object (with janus possibly, but tentative)
- Instantiate preset workflows (akin to how tesla has their dashboard for popular apps to use)

Conclusion and Future Work

- General API for Redshift/Flux for Virtual Worlds and General Oculus Quest Apps

Metahuman for Pseudonymous Identity

- latent video generation with metahuman to generate videos

Notes

Oculus Browser

- Based on the open source Chromium rendering engine used by some of the most popular browsers
- Optimized for WebXR and WebGL to enable fully immersive and 3D experiences
- Enhanced for media playback, including both traditional flat video and immersive 180/360/3D video formats
- Tailor-made for Oculus hardware and experiences with fast performance and better battery life
- Enriched with featured VR experiences to discover the best WebXR experiences and immersive media from the web and your friends on Facebook

References

- https://developer.oculus.com/documentation/native/android/mobile-vrapi/?device=QUEST
- https://vr-browser.herokuapp.com/
- https://docs.readyplayer.me/avatar-specification/avatar-types/vr-avatar
- https://docs.readyplayer.me/integration-guides/unity
- https://www.frozenmountain.com/developers/blog/archive/using-webrtc-to-create-multi-party-vr-experiences
- https://developer.oculus.com/documentation/native/pc/dg-render/
- https://immersive-web.github.io/webxr/#xrboundedreferencespace-interface
- https://developer.nvidia.com/nvidia-omniverse-platform
- https://en.wikipedia.org/wiki/Idoru
- https://github.com/gruberdev/vr-resources
- https://developer.oculus.com/manage/applications/5669897879748560/
- https://developer.oculus.com/documentation/unreal/
- https://developer.oculus.com/documentation/unreal/unreal-ide-guide-android/
- https://developer.oculus.com/documentation/unreal/unreal-platform-tool/
- https://developer.oculus.com/documentation/tools/odh-media/#open-web-page-in-headset
- https://developer.oculus.com/documentation/tools/tools-perf-opt-mobile/
- https://assetstore.unity.com/v2/3d
- https://securecdn.oculus.com/sr/platform-1.12/