PACEO: Innovating the Video-based Virtual Gathering Experience

Christopher Sastropranoto
University of Illinois at Urbana-Champaign

Jiawei Tang
University of Illinois at Urbana-Champaign

David Tran
University of Illinois at Urbana-Champaign

1 Introduction

The COVID-19 pandemic has been around the globe for a year now and yet most people are still restricted from face-to-face activities. People cannot enjoy offline activities as they fear contracting the virus. Online activities, therefore, are the new norm. Besides, with the communication technologies developing quickly, faster network speed and lower latency is approaching into our lives. Such improvements will enable us to find various new ways to have online social interaction. Under these circumstances, there is a booming need for online gaming, online social networking, and online communications. Consequently, there emerged some successful products that fairly addressed the need, such as Zoom. However, given more and more products are being developed to better socialize people online, it still doesn't seem to be enough. Applications like Gather.town seem to be creative on virtual interaction but the foundation of video chatting has barely changed. There needs to be a new way of online interaction that feels more fun, more real and more connected. Thus, we want to introduce PACEO (People Anywhere Connect Each Other). PACEO aims to be: an innovative online social interaction tool that offers realistic and fun interacting experiences. It targets everyone who wants to hang out with friends virtually. Not only that, the product can be used in formal gatherings, such as ceremonies or commencements. The application this paper proposes has the unique idea of attaching facial video streams to 3D models instead of merely showing the video feed as a 2D canvas somewhere on the screen. Users would then be able to interact with each other in a 3D world through their browser. In this sense, the interaction in this application follows real-world behaviors, such as positioning yourself to have a face-to-face chat with someone, and you can only chat with a proximity constraint.

Several hypotheses helped influence application underlying design. The first is that streaming applications through a centralized server will be sufficient to stream multiple users concurrently in the application. The second is to be able to render a 3D game in the browser without severely impacting performance.

This paper contains the intention for the following contributions:

- Function to retrieve user's video streams and streaming it to the central server.
- A centralized server that can handle requests to update video stream and update user position.
- Code layout to design and build a 3D world.
- Models for the users that will display their webcam and allow movement.
- Business plan for monetizing and promoting this application.
- Ideas for future additions to the project.

2 Related Work

As an example, Gather[2] and Chudo[8] demonstrate a fun way to get socialized online with more than a video chat. It simulates reality by creating a space and a user can interact with others abiding a proximity rule like in real life. There are multiple sets of scenes that have different purposes of interaction. For example, there is a private meeting room which allows only the users who move their avatars to that specific region to be able to chat privately together. Given that this app is a creative way to let people interact in a fun way and it does mimic some real-life features, it is still using the usual video chat for the main interaction among the users.

Inspired by an Instagram filter[3], a simple face detection can be done to wrap the image in the video stream of face to the 3D avatar model. In such a way, this provides a more intimate interaction among the users. Also, the filter itself has made it very efficiently so that it looks like a good solution to integrate people into the scene.

3 Challenges

There are definitely some challenges both technically and theoretically. First, such applications are not implemented to our knowledge so we are not sure how people will enjoy this type of interaction. Second, it is a resource demanding app that without good optimization, it might be hard to support a lot of users to use the app with a browser.

From the technical perspective of our approach, we will do some research and comparison to find the most efficient and compatible WebGL frameworks (PlayCanvas etc.) for our use case. For user interaction, a peer-to-peer system will be compared with a server-based video call solution. A P2P solution may not work well when there are many users connecting at the same time. A server-based solution may be adopted to ensure scalability[11].

Additionally, a face-detection module on the server will help locate the face, so that we can warp it to the 3D face model. But there can be some scalability issues as there are more users in the room, so maybe such face detection can be

done on the local browser to relieve the server. Face detection algorithms[9] could be costly, so there would need to be limitations set on when to detect the face and when to render it onto a new 3D facial model. Weboji[16] provides a useful base that we can implement to create 3D facial detection by requiring the developers to create 10 models, one for each expression.

Because of these issues regarding computational blockages, there could be a reduction of performance. As the system scales with users and computations, there would have to be graceful degradation in place to ensure that the quality of the server is not lost. In terms of graceful degradation, there would be a set order of applications that would terminate sequentially. At the first step of an overloaded network, the system will stop the user's video feed. This would still allow the users to walk around to view other users' video as well as still allow the text chat feature. If there is still an issue with performance, the user would not be able to move their character, but will still have the ability to use the text chat. Once the network issues reach the maximum, then the user would be kicked from the session. With a graceful degradation process as stated above, users can still enjoy the atmosphere that this application brings without going through major issues when there are connection problems.

4 Implementation

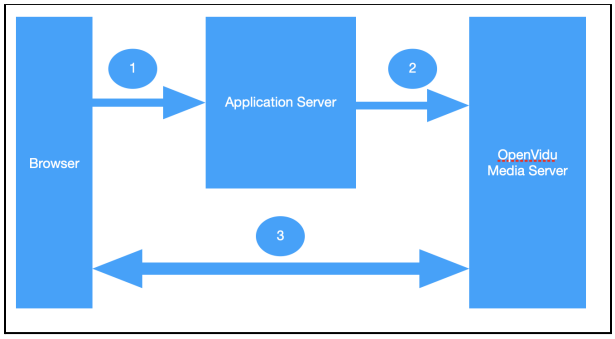
The goal is to push the innovation of online social interaction even further. PACEO will not only allow users to communicate through video chat. Instead, users will be displayed as 3D avatars of themselves where each avatar's face is a video stream of the user's face. This requires the implementation to blend together various components in order to successfully integrate video streaming with the 3D world in a performant manner. Once generated into the 3D world, users can move around using keyboard controls to interact with others.

4.1 General Architecture

PACEO's architecture is made up of three main components. The first is the application server, which is responsible for authenticating users, keeping track of in-game players as well as others in game logic such as player positions and moves. The server is implemented using Java and the SprintBoot framework. The second is the React frontend, which will render the application in the browser. Both of these components are packaged together into a single docker image and deployed using Heroku. The final component is the dedicated OpenVidu Media server, which is used for streaming user camera feeds. This is currently hosted on an AWS c5.xlarge EC2 instance. Diagram 1 illustrates the general flow of the application. Whenever a user joins a game, the browser will first make an API request to the application

server to join the game as shown in circle 1. The application server will then make a request to the OpenVidu [17] media server to fetch a media server token and return it to the browser. This is illustrated in step 2. Once the browser has received the token from the application server, it then uses it to open a web sockets connection to the media server as illustrated in step 3. Clients then create new video streams in the browser whenever it detects a new stream created by the media server.

Diagram 1



4.2 Application Server

The application server is mainly responsible for handling client authentication with the OpenVidu media server along with tracking user movements. Internally, it tracks in game users along with their positions using a dictionary stored in an in memory cache.

Sample User Positions Dictionary

```
{
 "userOne": {
   "x": 10,
   "y": 5
 },
 "userTwo": {
   "x": 40,
   "y": 2
 }
}
```

In terms of x and y from above, these would be functions of x and z in the application because y represents the up-down plane. Utilizing an x and y coordinate system would depict the system as if it was viewed from above.

Furthermore, the application server also exposes a set of REST and websocket APIs. These APIs are used by the clients so that players can communicate with each other.

Endpoint	Type	Response	Description
POST /session	REST	JoinResponseObject	Called whenever a new user joins a game. This endpoint is responsible for setting up a new connection to the OpenVidu server as well as setting a user's initial x and y positions It then creates an *OpenViduServerData* object, which is passed along to the OpenVidu media server. The media server then relays this to the client through the *onStreamCreated* event. In addition to this, the new user will be added to the dictionary of in game users.
POST /leave	REST	None	Called whenever a new user leaves the game. The endpoint will remove a user from the dictionary of in game users.
/pos	Websocket	PositionMessage	Clients write messages to this topic whenever they move in the 3D world. They also listen to this topic to detect any updates to a user's position.

JoinResponseObject. The openViduServerData field is an example OpenViduServerData object.

```
{
 "token": "placeholder-open-vidu-token",
 "openViduServerData": {
   "username": "userOne",
   "initialX": 10,
   "initialY": 20
 }
}
```

PositionMessage

This is just the user positions dictionary with the updated user positions. The client then uses this to update their local user position dictionary.

```
{
 "userOne": {
   "x": 8,
   "y": 1
 },
 "userTwo": {
   "x": 6,
   "y": 2
 }
}
```

4.3 User Movements

User positions are currently tracked using a dictionary mapping unique usernames to a user's x and y positions in the 3D world. The application server stores the main version of this dictionary and is the only component in the application that writes to it. In addition to this, each client also stores a local copy of the map so that they are able to render the models in the correct positions. Whenever a client moves, the browser transmits a new message to the */pos* topic in the application server. This message contains the username of the moving user along with their updated x and y positions. The application server then parses this message and updates the centralized version of the user position map. Once it has completed this, it then forwards the updated dictionary to all other clients in the game. The client then updates it's local state using the updated positions.

***/pos* topic message**
Format:
`{USERNAME}_{X_POS}_{Y_POS}_{SENT_AT_EPOCH_MS}`
Sample:
`UserOne_10_15_1620495594588`

4.4 Video Streaming

PACEO uses OpenVidu to implement the video streaming feature. The OpenVidu project provides a set of server-side and client-side libraries that can facilitate user to user video communication using WebRTC. Upon successfully authenticating with the application server and receiving an OpenVidu server token, clients start communicating with the OpenVidu media server by creating a websockets connection to it. Clients then publish the user's camera feed to the server. In addition to this, the client also starts listening to the *onStreamCreated* event provided by OpenVidu. This event is fired whenever a new user successfully joins in the game and contains a message for the client to process. As previously mentioned, this message is created by the application server, passed along to OpenVidu and then finally forwarded to any receiving clients. The message body is the *OpenViduServerData* object that was first created in the */session* endpoint when the user first joined the game. It contains the new user's metadata including the username and initial x and y positions. The client then uses the incoming stream to display the joining user's camera feed by creating a new HTML video element and rendering it on a new 3D model.

4.5 Three Dimensional Rendering

For the front-end portion of this application, we tested various ways on displaying a user's face and generating a 3D model to add to our system's world. These methods are still being tested currently to determine the best use which include a 3D pre-modeled face rendering with a webcam wrapped onto it, a 3D cube with the webcam wrapped onto one

face, and multiple 3D models using a face detection system that decides which model to be displayed utilizing Jeeliz's Weboji[16]. The first method uses a premade obj file that contains a three-dimensional model of a human from the shoulders up. When wrapping the 2D webcam plane onto this model directly, it would reduce clarity of the user's face. The second method entails creating a 3D cube using THREE JS's built-in cube and box functions[7]. To integrate the user's webcam onto only one side of the cube, a materials array was created with the one side being the webcam render and the other sides being a standard solid color. Using a cube allows for the webcam to be more visible to other users since the side face does not have any raised or descended points. The final method is to display the user's face via various rendered models and utilizing a face detection software to display the model that matches the user's expressions. This method would be built on top of Jeeliz's Weboji[16] and requires designing models for each possible expression and a texture mapping. Using Weboji for the user's face display would accurately map the user's expressions, but loses the webcam display so every user would have the same face model, unless we construct every possible user faces as models.

Experimentally, we have tested two of the above methods for the model that would display in the 3D face model in open world experience. The first option displays a proper human face model, but lacks in showing the user's facial expressions (no movement of eyes, mouth, eyebrows, etc). Testing this design in the world proved to be effective at showcasing that a user exists at that point, but does not feature any personal details such as facial expressions or webcam. Further modifications allow for a webcam to be texture mapped directly onto the face model, but the webcam would display problematically because of the bumps and ridges, such as the nose and mouth. For this method to succeed, we would have to design a face detection system that would clip the user's eyes, mouth, ears, etc. and properly display them at the designated location on the human model. The second option to add the user to this 3D world would be to use a 3D cube and display the entire webcam onto one side of the cube. This option proves successful because the user's face and features are clearly visible onto the flat surface. However, the problem lies in the webcam itself where display depends on webcam quality and lighting, which would in turn affect how the user's face is displayed and facial expressions are harder to detect on a straight 2D webcam image. A cube does not mimic the appearance of a human head onto a world, but can be solved by merging options 1 and 2 where a cube would be placed in the place of the human head model onto a model with neck and shoulders. This would create the illusion of a human head with standard shoulders and neck, but with a cube where the face would be.

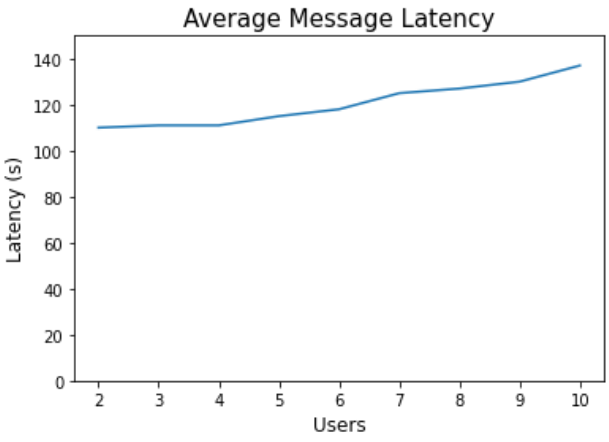
5 Experiments

We have conducted several experiments on each part of the PACEO to evaluate the overall experience of using the application and how well the application performs in different scenarios.

5.1 Performance

The application was deployed using an AWS EC2 instance and Heroku. It's performance was evaluated by viewing it in Chrome and obtaining performance metrics through Chrome DevTools. Initial experiments have yielded promising results as the browser drops less than 3% of frames despite having multiple concurrent users. Additionally, the application is able to maintain a frame rate of 60fps, which is considered acceptable by most gamers.

Figure 5.1

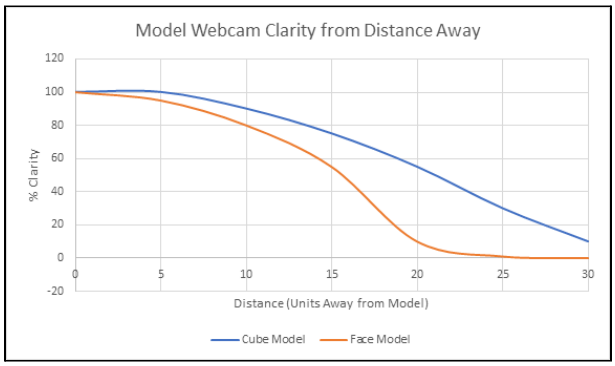


Since the movements of each user is synchronized by broadcasting position values for each update of the positions, it is important to measure the latency of each message being transmitted and how the latency is affected by the number of users in the game. From Figure 5.1, we can see that the latency of the messages increases as the number of users increases. This is expected due to the total number of messages transmitted for each move of the position. Due to the fact that we only move one user at a time during the testing, it shows a linear relationship between latency and number of users. In practice, the relationship can be undetermined and dependent on the pattern (frequency) of users moving.

5.2 Webcam Clarity

Since there are different models that can be used, the webcam clarity varied between each model over a given distance away. As shown below in Figure 5.2, the 3d rendered face model lost its clarity as the user went further away from the model. This is due to the facial features that are constructed on top of the model that warps the webcam.

Figure 5.2



With this information, it would be better to simply render 3D cubes to use as faces compared to having full models with features.

6 Business Plan

Video chat applications, such as Zoom and Skype, are related to the problem of online communication experience. During this time related to the COVID environment, users have been working remotely and may not have the chance to personally meet or build relationships with their colleagues. Using an app such as Zoom that only uses 2D flat webcam projections does not generate the same personalized connection that meeting a colleague in person would have. Our system is designed to develop a better team collaborative environment using a fun and interactive method.

We have several plans of monetization. First, there can be an option for premium features. Users would subscribe to a monthly premium subscription with a reasonable price to experience some advanced features, such as the customization of the scenes, longer sessions length, and the ability to host a larger room with more capacity. Second, we can provide product placement for companies to advertise products to the users. These advertisements are not going to be like floating elements in HTML, but are very subtle and not affect the overall gameplay experience. Lastly, another way to monetize would be more of a stretch goal, which is to allow users to purchase customizations for their model. This would include things like clothes, hair, shoes, as well as selling upgrades such as faster movement or flight. It can be similar to a minified version of SimCity.

To start our business, we will try to connect with Universities or any organizations or companies to persuade them to try our products. We are certain that they would probably never experience applications like our product and it will leave them an impression. Our application will be first used by the organizations and surely they need to buy our license to use it. Then we can scale out our application even more to welcome everyone domestically. Slowly, it should be accessible by everyone around the world. Scaling out would be a hard process as the architecture may even change due to the fact that a lot more factors need to be considered to build a reliable cloud-based communication system. We may partner up with a cloud communication solutions provider to speed up the process if the day comes.

References

[1] 2011. J. Uberti and P. Thatcher. Vidyo WebRTC. https://vidyo.io/platform/webrtc/

[2] 2020. Gather Presence, Inc. *Gather*. https://gather.town

[3] 2019. D. OReilly. Filter: *It's Always You*. http://www.davidoreilly.com/its-always-you

[4] 2004. Salman A. Baset and Henning G. Schulzrinne, An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol, http://www1.cs.columbia.edu/~salman/publications/skype1_4.pdf

[5] 2011, Peer-to-peer Live Streaming and Video On Demand Design Issues and it's Challenges, Hareesh and Manjaiah, https://arxiv.org/pdf/1111.6735.pdf

[6] 2017. Bart Jansen, Timothy Goodwin, Varun Gupta, Fernando Kuipers, and Gil Zussman. 2018. Performance Evaluation of WebRTC-based Video Conferencing. SIGMETRICS Perform. Eval. Rev. 45, 3 (December 2017), 56–68. DOI:https://doi.org/10.1145/3199524.3199534

[7] 2010. ThreeJS: JavaScript Library to Create 3D Computer Graphics Utilizing WebGL. https://threejs.org/docs/

[8] 2018. Magic Unicorn, Inc. Chudo Virtual Spaces. https://chu.do/

[9] 2020. H. Klym and I. Vasylchyshyn. Face Detection Using an Implementation Running in a Web Browser. https://ieeexplore-ieee-org.proxy2.library.illinois.edu/abstract/document/9238754

[10] 2014, Ryan Shea, Jiangchuan Liu, Edith C.-H. Ngai and Yong Cui. Cloud Gaming: Architecture and Performance https://www.sfu.ca/~rws1/papers/Cloud-Gaming-Architecture-and-Performance.pdf

[11] 2014. Kwok-Fai Ng, Man -Yan Ching, Yang Liu, Tao Cai, Li Li, and Wu Chou. A P2P-MCU Approach to Multi-Party Video Conference with WebRTC. http://www.ijfcc.org/papers/319-W002.pdf

[12] 2011. Asim Kadav, Chelsea Wanta, Nai-Wen Claire Yu, Kyung Lee, Enid Montague. Improving the Online Video Chat Experience. https://link.springer.com/chapter/10.1007/978-3-642-21716-6_21

[13] 2019. G. Rauch. SocketIO: JavaScript Library for Real-Time Connections Between Web Client and Server. https://socket.io/

[14] 2014. K. Chaturvedi. Web based 3D analysis and visualization using HTML5 and WebGL. http://essay.utwente.nl/84311/1/chaturvedi.pdf

[15] 2011. Luca De Cicco, Saverio Mascolo, Vittorio Palmisano. Skype Video congestion control: An experimental investigation. https://www.sciencedirect.com/science/article/abs/pii/S1389128610002914

[16] 2018. Jeeliz. Weboji, expression recreation using 3D models. https://github.com/jeeliz/jeelizWeboji

[17] 2021. openvidu, OpenVidu, OpenVidu Platform main repository. https://github.com/OpenVidu/openvidu