**3D Human Modelling:**

**WebRTC:**

WebRTC, which stands for Web Real-Time Communication, is a free, open-source project that offers real-time communication capabilities via basic APIs to web browsers and mobile applications (Application Programming Interfaces). With WebRTC, peer-to-peer data, audio, and video sharing is possible between browsers and devices without the need for additional plugins or software installations. WebRTC uses encryption and it offers secure communication, and it supports a variety of media formats, including audio, video, and data.

WebRTC is necessary because it offers a quick and effective method for real-time communication between web browsers and mobile applications without the need for additional plugins or software installations. Real-time communication between web browsers and devices was frequently difficult before WebRTC was developed and required the use of third-party plugins, which were frequently unreliable and presented security risks.

**Index.js:**

Then in the next part a signaling server is established for WebRTC. To accomplish this, the JavaScript file will create a HTTP server utilizing the express.js framework and Socket.IO library to handle WebSocket connections. The script is responsible for serving static files found in the public directory, logging messages to a client console when they connect to the server and defining two events called “message” and “create or join” to handle incoming messages from clients. Whenever the server receives a message event from a client, it logs the message to the console and transmits the message to all other clients in the same room as the sender. When the server gets create or join event from a client, it verifies the number of clients in the specified room and takes the appropriate action based on the number of clients.

If the room is vacant the client who triggered the event is added to the room and emits a created event. If there is only one client in the room, the client who initiated the event joins the room and emits a joined event. If two clients already exist in the room, the server transmits a full event to the client. The server also retrieves the IP address of the server and transmits it to the client when it receives an ipaddr event from a client. Lastly, when the server receives a bye event from a client, it logs a message to the server console.

**Config.js:**

The WebRTC application can use this JavaScript object to define a list of ICE servers for use in creating a peer-to-peer connection between two clients.

Two objects are present in the ice Servers array. A STUN server, represented by the first object, enables clients to learn their public IP address and other network details. If a direct peer-to-peer connection cannot be made, the second object represents a TURN server that enables the clients to relay their media traffic through a different server.

Xirsys provides the TURN server, which needs to be authenticated with a username and credential. The URLs for the various transport protocols (UDP, TCP, and TLS) and ports that can be used to connect to the TURN server are specified in the urls array of the second object.

Overall, even when the clients are protected by firewalls or network address translation (NAT) devices, this configuration object provides the information required for the WebRTC application to establish a dependable and secure peer-to-peer connection between two clients.

**Main.js:**

This JavaScript code demonstrates how WebRTC is implemented for video conferencing. It creates a peer-to-peer connection between two users in a room and displays their local and remote streams on a webpage.

A few global variables, including isChannelReady, isInitiator, isStarted, localStream, pc, remoteStream, and turnReady, are first set by the code. The TURN/STUN server's configuration data is then initialized into the pcConfig variable.

The user is then prompted for a room name, which is used to establish a connection with the signaling server using Socket.io. When a user enters a room name, the signaling server creates or joins the specified room after receiving a create or join event from the client.

Next, some event handlers for the signaling server are defined in the code, including created, full, join, joined, and log. When the server creates a room, when a room is full, when a peer wants to join the room, when the current peer joins the room, and when the server logs a message, these handlers are used to log messages to the console.

Then, using the signaling server, the code defines a sendMessage function that sends a message to each client present in the same room.

The local video stream is then configured by the code by invoking getUserMedia with the localStreamConstraints parameter. Once the local stream has been acquired, the code calls the gotStream function, which then sets the localStream variable, renders the local stream on the website, and uses the sendMessage function to send a got user media message to every client in the same room.

When the client receives the got user media message, the maybeStart function is invoked. The client creates a new RTCPeerConnection object, adds the local stream to it, sets the isStarted variable to true, and then calls the doCall function if it is the initiator and hasn't already started the call.

The onicecandidate, onaddstream, and onremovestream events are handled by the new RTCPeerConnection object created by the createPeerConnection function.

A client-generated ICE candidate is handled by the handleIceCandidate function. With the help of the sendMessage function, it distributes the candidate information to all users present in the same space.

The doCall function uses the createOffer function to create an offer for the remote peer and the sendMessage function to send the offer to every client in the same room.

The doAnswer function uses the createAnswer function to create an answer to the offer, and then uses the sendMessage function to send the answer to every client in the same room.

The RTCPeerConnection object's local description is set by the setLocalAndSendMessage function, and all clients sharing the same room are sent the session description via the sendMessage function.

When the remote stream is added to the RTCPeerConnection object, the handleRemoteStreamAdded function is called. The remote stream is shown on the webpage after the remoteStream variable has been set.

The code then creates a window. When a user closes a window, an onbeforeunload event handler uses the sendMessage function to send a good-bye message to all clients in the same room.

**Screenshots:**

Graphical user interface, application

Description automatically generated