DEPARTMENT OF ELECTRICAL ENGINEERING & ELECTRONICS

Final Year Project Final Report

'MOSAIC Immersion: Metaverse Geophysical Visualisation and Interaction - VR Generation'

Author: Zhikun Peng (201769395)

Project Supervisor: Waleed Al-Nuaimy

Project Assessor: Jason Ralph

Declaration of academic integrity

By submitting this work I confirm that I have read and understood the University's Academic Integrity Policy.
By submitting this work I confirm that I have acted honestly, ethically and professionally in conduct leading to assessment for the programme of study.
By submitting this work I confirm that the work I am submitting is my own. I have not commissioned production of the work from a third party or used artificial intelligence software in an unacceptable manner to generate the work*. I have not copied material from another person or source, nor committed plagiarism, nor fabricated, falsified or embellished data when completing the attached piece of work. I have not colluded with any other student in the preparation and/ or production of this work.
*Software applications include but are not limited to, ChatGPT, Bing Chat, DALL.E, Bard

SIGNATURE Zhikun Peng DATE 28. April. 2025

Abstract

Modern geophysical exploration faces increasing complexity and collaboration demands, while traditional methods limit spatial understanding and remote teamwork. This project builds on its predecessor by improving a web-based virtual reality (VR) platform to enhance the visualization of borehole data. To enhance the immersive visualisation, the platform plays pre-recorded VR videos generated from simplified cloud point models, alongside synchronized synthetic borehole videos. It employs A-Frame for VR rendering, HTML for page structure, CSS for styling, JavaScript for managing video playback, timeline synchronization, and annotation interactions It uses A-Frame for VR rendering, HTML for page structure, CSS for styling, and JavaScript to manage video playback, timeline synchronisation, and annotation interactions. And the entire server programme is hosted on Amazon Web Services (AWS). WebSocket facilitates real-time client-server communication for synchronization. This cross-platform design allows individual users to visualize subsurface data immersivity on various devices without specialized VR hardware. This work is built upon the foundational contributions of previous researchers, Dr. Nan Zong, who developed the initial conversion pipeline from 2D borehole videos to immersive 360° formats with basic 3D reconstruction, and Mr. Jiang Xu, who pioneered the integration of bullet chat annotation and real-time interaction into the VR Web environment, providing a robust platform for further innovation The simplified borehole cloud point models, combined with synchronized videos, improve the accessibility and visualization of geophysical data in a 3D environment, laying the groundwork for collaborative workflows and informed decision-making in future geophysical surveys.

Contents

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Dr. Waleed Al-Nuaimy, for his consistent guidance, encouragement, and invaluable advice throughout the entire duration of this project. His expertise and support have been crucial to the completion of my work.

My sincere thanks also go to Dr. Zong Nan, whose pioneering research on the conversion of 2D drilling videos and point cloud model reconstruction provided a solid technical foundation for this project. I also wish to thank Mr. Xu Jiang for his important contributions in bringing interactive bullet chat features and real-time web collaboration to the VR platform, which inspired further enhancements in multi-user interaction and system design.

Without their generous support and insightful guidance, this project would not have achieved its current progress and quality

List of Figures

1 Introduction

Virtual Reality (VR) has moved from theory to practice since the mid-20th century. With the development of computer graphics, sensors and display technologies, VR has gradually expanded from a simple visual experience to an immersive environment that includes sound and touch. Current VR devices feature high-resolution visuals, spatial audio, and precise motion tracking, allowing users to interact naturally in virtual spaces, driving their application in various industries such as education, healthcare, and engineering [1] [2]. VR is no longer limited to entertainment, but has also demonstrated a high degree of adaptability in specialised fields such as immersive learning, medical training, and complex engineering design.

In scientific fields such as geophysics, VR provides a unique way of visualising and interactively analysing complex geological data in three dimensions, bridging the gap between two-dimensional visualisations that are inadequate for the representation of spatial relationships and structural features. VR technology allows researchers to immerse themselves in the data from multiple perspectives, increasing the efficiency of their analysis and discovering new patterns, especially in drilling scenarios, where VR can transform 2D videos into immersive 3D environments, significantly enhancing the understanding and interpretation of subsurface structures [3].

In recent years, with the development of cloud computing and low-latency networks, server-based multiplayer collaborative environments have become an important feature of virtual reality applications. In this project, by deploying servers hosted on Amazon Web Services (AWS), geophysicists are able to directly access specific web pages and interact remotely in real time in the same virtual space, no longer limited by geographic and equipment conditions. Whether in the lab, in the office, users can join the collaboration from their desktop or mobile device. Their annotations, ideas and discussions can be immediately synchronised with other participants.

The integration of video and annotation management systems further enhances the real-time and interactive nature of collaboration, allowing experts to efficiently share insights, ask questions, and discuss important geologic structures in a virtual drilling environment. This not only improves the efficiency of teamwork and the accuracy of data interpretation but also creates a technological basis for remote knowledge sharing and joint decision-making between experts in the geophysical field.

The annotation feature in this project is inspired by the pop-up real-time commenting system that is popular on the BiliBili platform in China, as shown in the Figure. 1 [4], and this dynamic commenting method greatly promotes communication among users.

Figure 1. The VR video with Bullet Comments on the Chinese BiliBili Channel [4]

During the development of this project, thanks to the solid foundation of previous work, the team was able to progress and realize several key innovations. First of all, we would like to give special thanks to Dr. Nan Zong, as the first predecessor, who has already achieved the conversion of 2D drilling videos to 3D models, and completed the reconstruction of the point cloud model of the drilling geological structure by using the monocular SLAM 3D modelling technique. Secondly, we have to thank the second predecessor, Mr Xu Jiang, who first applied the bullet comment interaction system to the VR environment and realised a basic single-user interaction webpage supporting desktop and mobile.

It is on top of their basic innovations that this project addresses the shortcomings of the system, such as the lack of efficient multi-user remote synchronization, limitations in video content switching, and insufficient accuracy of the 3D point cloud model-by proposing and realizing multi-people online synchronization, dual video free switching and high-precision point cloud modelling. These improvements effectively solve the outstanding problems of traditional methods, such as passive viewing by users, insufficient interactivity. Ultimately, this project creates an immersive collaboration platform that will enhance the efficiency of geophysical data interpretation, the speed of team decision-making and the ability of cross-regional experts to link up, enhancing the application of geophysical information technology in the field of visualisation.

The remainder of this report is organized as follows: Section 2 presents the project specifications and objectives. Section 3 surveys related literature and background. Section 4 discusses the industrial relevance, real-world applicability, and societal impact. Section 5 details the architecture and implementation of the VR web system. Section 6 describes the processes for model optimization and the VR video production workflow. Section 7 covers the results and system testing. Section 8 provides a discussion and analysis of the findings and limitations. Section 9 concludes the report.

2 Project Specifications

2.1 Interactive 3D Visualisation

The project aims to transform traditional 2D borehole videos into an interactive 3D model within a virtual reality environment. Users can freely rotate the viewpoint to explore the borehole's internal structure in detail, thereby enhancing their understanding and analysis of subsurface geological features. Additionally, the project extends the visualisation experience by integrating pre-recorded VR videos of high-precision point cloud models derived from borehole data, offering users a richer and more immersive perception of the geological environment.

2.2 Annotation List (Replacement of Bullet Chat System)

While the initial plan set out to implement a real-time bullet chat commenting system, the final project introduces an Annotation List feature. Users can add time-synchronized annotations or comments linked to specific moments or locations within the VR video environment, which are then displayed in an organized list for easy reference and review. This replaces the original bullet chat concept, providing more structured and collaborative information sharing amongst users. The effectiveness of this objective is measured by the usability, synchronisation, and visibility of individual and collective annotations within the immersive environment.

2.3 Video List and Multi-Video Support (New Addition)

The final system introduces a Video List feature, enabling users to select and switch freely among multiple borehole VR videos and related geophysical visualisation resources. This supports dual video synchronisation and comparison, allowing users to efficiently analyse and cross-reference different data sets within the same interface. The inclusion of a Video List increases the flexibility and scalability of the platform and is a significant enhancement beyond the original project specification.

2.4 Cross-platform Compatibility

The VR viewer is developed to operate seamlessly on both desktop (Windows) and mobile platforms (iOS), without requiring a dedicated VR headset. This ensures that users can access geophysical visualisation tools on a variety of devices, supporting collaborative analysis regardless of hardware constraints.

2.5 User-friendly Interface

An objective of the project is the design of an intuitive and accessible user interface (UI). The UI has been optimised so that users, including those with little previous experience in VR technologies, can easily navigate, add or review annotations, and manage video resources. Clear interaction flows allow users to focus on their analytical tasks rather than technical complexities.

3 Literature Survey

3.1 Shortcomings of Traditional Geophysical Visualization

Two-dimensional visualisation restricts data interaction and immersion experience, making it difficult to help researchers gain deep insights into the deformation of strata, fractures and other subtle features. For example, in the current key scenarios such as drilling exploration, experts can only passively observe the structure with the help of flat images or traditional animations, and the spatial association relies on personal experience, which lacks the support of efficient intuitive tools.

In addition, traditional geophysical 3D visualisation methods rely on discrete and limited observational data, and are unable to comprehensively and meticulously restore complex geological structures. For example, Pazzi [5] mentioned that some traditional methods lead to difficulties in the representation of results, limited resolution, and missing structural details. And Olierook [6] pointed out that traditional modelling combines quantitative and qualitative methods, but it is difficult to comprehensively and accurately express 3D spatial correlation and deformation characteristics, and engineering practice relies heavily on manpower and experience, which affects spatial judgement and the depth of data interaction. In practice, this shortcoming of representation difficulties leads to longer project lead times and more misjudgements, further exacerbating the difficulties of model building in analytical geophysics.

3.2 Advances on Web-Based VR Visualisation in Geophysics

With the development of 3D virtual reality and web technologies, more and more fields have started to use virtual reality technology and have made significant progress. For example, GeospatialVR [7] is an open-source collaborative virtual reality framework that supports users to access real-world 3D environments on web platforms via desktop, mobile and VR devices. The framework dynamically generates terrain, infrastructure, and multiple types of information layers (e.g., hazards, sensors, traffic, weather) for real-time data visualisation, and supports multi-user remote collaboration and interaction through virtual characters, thus serving as a virtual command centre or conference room. GeospatialVR has been applied in a variety of domains such as flooding, mountain fires, and public safety, driving immersive geospatial visualisation and decision support systems.

Mr Xu Jiang's project [8], as an important cornerstone of this project, successfully implements a video player that supports VR bullet comment interaction based on the A-Frame framework and WebSocket technology. The system allows users to send bullet comments via keyboard or voice recognition in a 3D virtual video, effectively enhancing the sense of 3D visualisation immersion. The project fully demonstrates that A-Frame, as an open-source, HTML-based Web VR development framework, has a high degree of ease of use and cross-platform compatibility. Unfortunately, the project does not support automatic multi-user real-time synchronisation and lacks an intuitive UI, which greatly reduces its usefulness, and these are also the features that this project focuses on improving.

Compared with other VR frameworks, A-Frame [9] allows users to have an immersive VR experience on the web without the need to develop and install an app. Developers do not need to go deep into WebGL programming and can quickly build and extend complex 3D/VR scenes with simple HTML tags, which greatly reduces the development threshold [10]. which improves the accessibility and popularity of the project, and its ecosystem and community are active and innovative, which promotes the continuous progress of 3D visualisation and interactive media on the web.

In summary, the existing research results provide a solid technical and theoretical foundation for this project, proving that the combination of A-Frame and WebSocket can effectively promote the real-time interactive and immersive experience in the Web VR scenario, and enhance the effect of geophysical visualisation.

3.3 Points Cloud Module Optimisation and Surface Reconstruction

With the increasing requirements of geophysical exploration for the accuracy of 3D visualisation of underground structures, the acquisition and processing technology of point cloud data has gradually become a key link. The iterative Poisson point cloud surface reconstruction method proposed and improved by Lu and Chen [11] addresses the problems of error accumulation, loss of details, and surface discontinuities that exist in traditional reconstruction algorithms for dealing with complex geological structures, and efficiently filters and smoothes the point cloud data through the introduction of a multiple-iteration optimisation process for efficient filtering and smoothing of point cloud data. The method firstly enhances the uniformity of the sparse point cloud by using the density interpolation algorithm, which makes the point cloud data denser and reasonable in spatial distribution. Subsequently, the influence of measurement noise and anomalies is attenuated by the filtering algorithm, which effectively preserves the micro-features of the original geological structures. Finally, the improved Poisson reconstruction algorithm is able to automatically fill in the holes in the point cloud, improve the continuity of the surface reconstruction, and maintain a high degree of geometric reproduction at larger scales.

In addition, thanks to the support of Nan, our PhD student, we have obtained the original point cloud model data for this project. This provided a solid database for the subsequent point cloud optimisation and Poisson reconstruction work, enabling more accurate and realistic modelling of 3D geological structures. With the combination of the existing research and the support of the senior students, this project is able to carry out high-quality point cloud generation and surface reconstruction more effectively, which further improves the 3D visualisation effect and immersive interactive experience.

4 Industrial Relevance, Real-World Applicability and Scientific Societal Impact

4.1 Suitability for the Geophysical Industry

This project effectively serves the geophysical exploration industry, which is growing in complexity and collaboration needs. By introducing Web VR and point cloud modelling, the project promotes the geological data visualisation approach to cross over from static and experience-dependent to an information-based platform with high interactivity and collaboration. The solution based on A-Frame and WebSocket supports desktop and mobile, which improves the accessibility of the industry tools, allows to experience the 3D geological environment without the need of special VR equipment, conveniently access, annotate and discuss the geological data through the browser, and supports multi-user online synchronisation, which provides a unified communication and decision-making space for the exploration technician, scientific researcher, executives and other users with multiple roles. Remote expert collaboration, data interpretation and discussion are more efficient.

The previous passive one-way ‘viewing-interpretation’ mode is transformed into active, traceable and structured real-time interaction. This is of obvious significance in improving teamwork efficiency, cross-regional expert resource sharing and decision-making speed in geological exploration projects. It is suitable for remote consultation and cross-validation of enterprises, institutions and scientific research organisations after data collection, and provides a solid tool base for multi-location collaborative work, which is in line with the industry trend of remote access and real-time interaction of data under digital transformation.

This project addresses the need for high-precision 3D visualisation by adopting an optimised iterative Poisson point cloud reconstruction algorithm, which improves the coherence, detail and geological feature reduction of the point cloud model. This choice is a direct response to the core challenges in high-precision modelling and analysis of complex subsurface structures in earth science, engineering geology and other industries. The Poisson reconstruction of the point cloud improves the uniformity of the spatial distribution of the point cloud and automatically fills in some of the missing data areas to enhance geometric restoration and geological micro-feature retention.

These improvements make the database accumulated by the point cloud model more solid, which can bring more trustworthy 3D information support to the modelling, interpretation and decision-making processes in industries such as geological exploration, mineral resources assessment, tunnel and underground engineering design, and so on. Industry users can rely on this project to provide finer and more intuitive model representations in subsequent structural modelling, attribute analysis and even geological disaster risk prediction, reducing human subjective misjudgements and improving collaboration accuracy and decision-making efficiency.

4.2 Scalable Cloud Deployment and Extensible Web Architecture

This project focuses on adopting AWS cloud deployment and mainstream web technology stack (HTML, CSS, JavaScript) in the software architecture, which realises a full convergence with the current IT mainstream operation and maintenance, development and data management standards. Relying on AWS cloud services, not only does it realise reliable remote access to geoscientific data across regions and terminals, providing a highly available and elastic online platform for distributed research teams, but also the cloud-native architecture facilitates rapid scheduling. At the same time, because the Web front-end technology (A-Frame, WebSocket, etc.) is consistent with the mainstream habits of modern enterprises and research platforms in terms of front-end and back-end development, UI integration, and data visualisation, the project code has a high degree of readability and is easy to operate and expand, so that third-party or enterprise developers can carry out customised extensions, interface reconstruction, or functional iteration accordingly in the later stages.

This not only ensures the stability and operability of the industrial-grade system, but also provides an open interface and ecological foundation for universities, research institutes and industrial users to develop featured plug-ins on demand, integrate existing productivity tools, etc., thus promoting the continuous innovation and widespread promotion of geoscience informatisation tools.

4.3 Potential for Cross-Industry Application and Societal Benefits

In addition to the direct promotion of the geoscience industry, the project's platform and technological solutions have also demonstrated broad application potential in a number of other industries. Focusing on efficient, low-threshold, cloud-based VR collaboration without the need for specialised hardware, the remote access and real-time interaction of data achieved by the project responds to the common demand for digital transformation and remote working among global enterprises and research institutions. The education and training sectors will also benefit from the scene reproduction and real-time interaction in the immersive VR environment, which is not restricted by geography and equipment, and will significantly improve the fairness of teaching and the breadth and quality of knowledge dissemination.

More importantly, the overall architecture of the platform is highly modular and can be easily embedded into industries requiring large-scale 3D data collaborative analysis and remote diagnosis, such as healthcare, transportation, and construction, to promote the radiation of innovations from geosciences to cross-industry digitalisation, synchronisation and green development models. This is not only conducive to enhancing the efficiency of the use of social resources, but also accelerates the process of building a ‘smart society’ centred on low-cost, green and low-carbon development, and promotes equitable and sustainable development.

5 VR Web System

5.1 Web System Architecture

This section describes the structure and functional division of labour of the modules in the web system, combined with the architecture diagram (shown in Figure 2) and the actual implementation process:

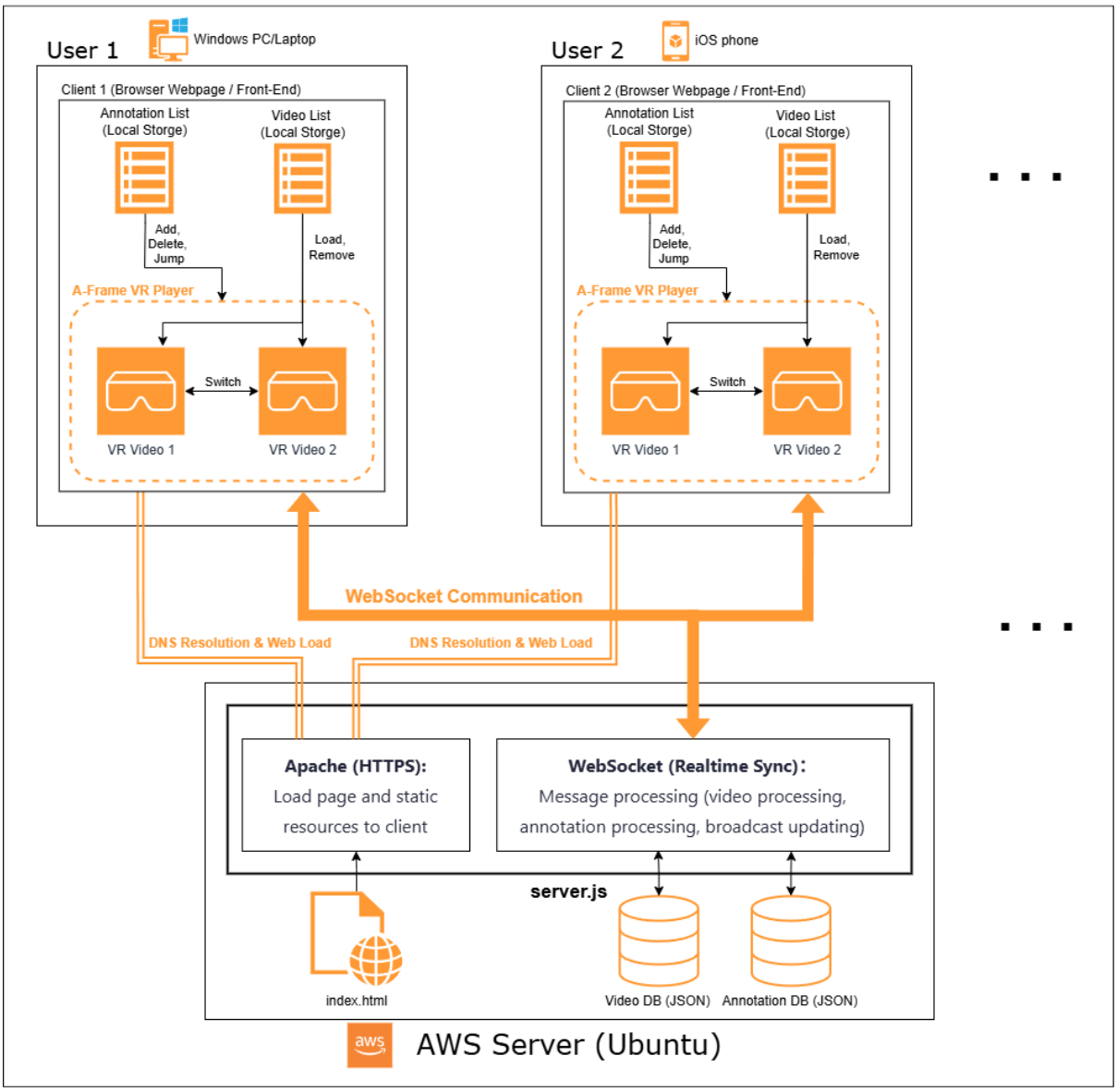

Figure 2. The Architecture of the Web System

5.1.1 Client Layer:

This layer is responsible for the direct interaction between the web page and the user. The user can interact with the elements of the web page. The whole client or front-end of the web page is written in HTML/CSS/ JavaScript.

HTML: Builds the structure and core elements of the web page.

CSS: Adds style, layout and visual beautification to the elements.

JavaScript: Implements user interaction, dynamic operation and data interaction with the server.

Moreover, the system supports Windows PC/Laptop and iOS on mobile, and the user accesses it through the browser without the need for a special VR headset. The following are the core elements of a web page:

- **Annotation List (browser local storage):**

  Users can add, delete, and skip to the location of annotations in the VR video. Annotations are initially saved locally and synchronised with the server in real time via WebSocket to ensure the consistency of collaborative data across multiple sites.

- **Video List (browser local storage):**

  Users can upload VR videos from local device to the server, or load, rename, or remove available VR videos from the server to facilitate multi-video management.

- **A-Frame VR Video Player:**

  The VR video player is developed based on the A-Frame framework and integrates video control buttons (play, pause, fast forward, rewind), timeline and video toggle buttons.

  The interior of the player is based on the 3D entity of the video sphere (implemented by A-Frame), and the panoramic video mapping is mapped on the inner wall of the sphere to create a three-dimensional VR space. Users are free to switch viewpoints via mouse (PC) or touch-swipe and gravity sensing (mobile) to get the feeling of VR space.

  Timeline control allows users to view any time point of the video for easy positioning and in-depth analysis. The player can directly switch between two selected videos to visually compare details of the same geological structure in different materials or renderings.

5.1.2 Network Communication Layer:

This layer is responsible for client-server communication.

- **DNS Resolution and Web Loading:**

  After the user accesses the domain name, it is pointed to the back-end server deployed on AWS via Domain Name System (DNS) resolution, and the front-end pages and static resources are pulled via HTTP(S).

- **WebSocket Real-Time Communication:**

  The front-end automatically establishes a WebSocket connection after the page is loaded to achieve real-time data synchronisation and two-way interaction for video, annotations and so on. Users' real-time operations (e.g. adding or deleting annotations, video synchronisation) are broadcast to all online users through this channel, ensuring collaborative consistency.

5.1.3 Server Layer (AWS Server):

AWS Server is both the hosting platform for the application and integrates a series of key back-end services such as static website publishing, real-time communication, data storage and security management, which is the basis for the operation of this system.

The operating system environment is Ubuntu, and AWS security group rules are used to restrict and allow traffic access from different sources, where the port security group is configured to open SSH (22), HTTP (80), HTTPS (443) and WebSocket (custom port) access. Encryption of all web communications is secured using SSL certificates (Let's Encrypt).

Key components included with the server:

- **Apache HTTP Server:**

  Used to provide external Web services, responsible for handling HTTP/HTTPS requests from clients and distributing static resources such as HTML/CSS/JavaScript.

  Support SSL/TLS, encrypted transmission to ensure data security, can achieve automatic boot and multi-port listening and other functions.

- **WebSocket Server (Node.js):**

  Built based on Node.js environment, using ws and other modules, it handles real-time two-way communication between the front-end and the back-end.

  Mainly responsible for annotations and other real-time message reception, processing and broadcasting, multi-user collaboration and data synchronisation.

  It supports HTTPS/SSL certificates to protect communication security, runs on a custom port, and works with the Apache server.

- **JSON Data Storage:**

  Includes video_info.json for managing video information and global_annotations.json for storing annotations. Supports video upload, information reading, writing and synchronisation for all front-end clients to access.

5.2 Server Deployment and Domain Name Redirection

5.2.1 AWS Server Deployment

The system back-end is deployed on EC2 virtual machines (Ubuntu OS) provided by Amazon Web Services (AWS). The main deployment process is as follows:

- **Instance Selection and Environmental Preparation:**

  Select a free t2.micro, install the Ubuntu system, and allocate the required storage space for the root volume. Create a security group and open the necessary ports (SSH: 22, HTTP: 80, HTTPS: 443, and WebSocket custom port: wss://vrdanmaku.uk:8080).

  Securely connect to the cloud host via SSH keys, update and upgrade the operating system environment, and ensure that the underlying software is the latest stable version.

- **Web Server Installation and Configuration:**

  Install Apache HTTP server for hosting static web pages and front-end resources (HTML, CSS, JS). Configure SSL support to ensure encrypted data transmission. Set Apache to start automatically with the host through system services, and allow HTTP/HTTPS traffic in and out through security groups and firewalls.

- **Front-End Deployment:**

  Deploy the front-end page code in the /var/www/html directory and adjust the file permissions as needed to ensure that the web server can read and distribute the page content correctly.

- **Back-End Real-Time Communication Services:**

  Use Node.js to write a WebSocket server, which is responsible for handling the sending and receiving and synchronisation of real-time information. The service is deployed and run with the pm2 process management tool and supports HTTPS/SSL certificates to enhance communication security.

  Data Storage:

  All video metadata (video_info.json) and annotations (global_annotations.json) are stored locally on the server in JSON file format for real-time access and operation by front-end and multiple users.

5.2.2 Domain Name Configuration and Resolution

To enhance access, convenience and trust, the system is configured with a custom domain name (vrdanmaku.uk). The A record for this domain name is pointed to the public IPv4 address of the AWS deployment instance, i.e. 18.135.231.122, through the domain name registration service provider. After the user enters the system domain name in the browser, DNS resolves the domain name to the actual server IP, enabling access to the page from the public network.

5.3 Web Features Design and Implementation

In order to enhance the practicality of the project, based on the previous web-based VR video player, video management, annotation management, and dual-video synchronous switching features have been introduced. The following section describes how the front-end interface (controls, events), the main JS logic snippets, and the back-end Node processing code work together to make the feature work.

5.3.1 Video Management Feature

Video management is one of the core components of the VR web platform, allowing users to fully manage VR videos. The main functions include:

- Uploading local videos to the server.
- Adding online video via Glitch link.
- Naming existing videos.
- Deleting videos on the server.
- Setting videos to tag vid1 or vid2 (for dual-video synchronous comparison).

Front-End Implementation (index.html):

- **User Interface Components:**

The front-end interface consists mainly of the following components:

```
<div id="video-list-container">
  <div id="video-list-header">
    <span>Videos List</span>
    <div class="header-actions">
      <button id="upload-button">⏫</button>
      <button id="link-button">🔗</button>
      <button id="video-list-toggle">▼</button>
    </div>
  </div>
  <div id="video-list-content">
    <table id="video-table">
      <thead>
        <tr>
          <th>Name</th>
          <th>Tag</th>
        </tr>
      </thead>
      <tbody id="video-table-body">
        <!-- The video list will be generated dynamically here -->
      </tbody>
    </table>
```

```
  </div>
</div>

<div id="video-action-popup" class="hidden">
  <button id="video-rename">🖉</button>
  <button id="video-delete">✕</button>
  <button id="video-select-vid1">1</button>
  <button id="video-select-vid2">2</button>
</div>
```

With the above HTML code, the UI of the videos list can be available as shown in Figures 3 and 4:

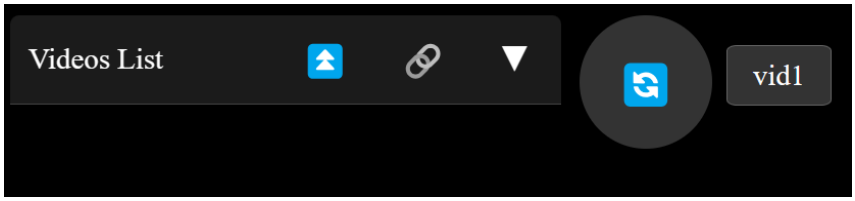

Figure 3. The UI of the Folded Videos List

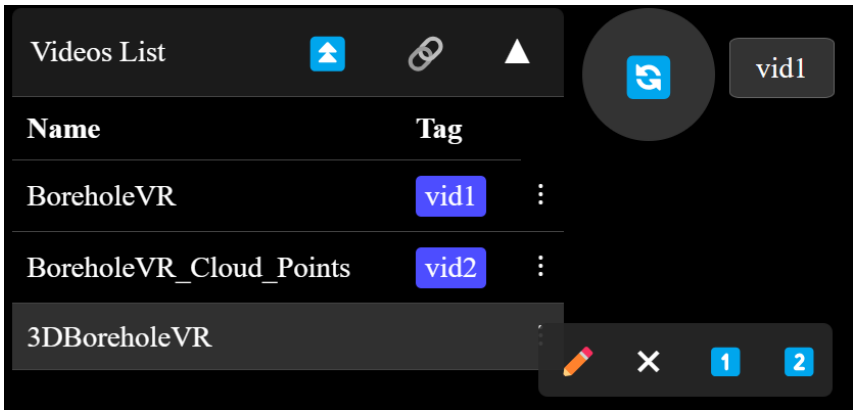

Figure 4. The UI of the Expanded Videos List with Pop-up Menu

In the header section of the list table are the upload button, the link button and the triangle tag indicating whether the table is collapsed or not. The list content includes the name of the video, and the Tag is used to indicate whether the current video is selected by the player or not. When the list element is clicked, a secondary window containing four buttons pops up to name an existing video, delete a video from the server, and set the video to vid1 or vid2.

- **Video Upload Implementation:**

The front-end handles local video uploads with the following code (Showing parts only, see Appendix 1 for details):

```
    uploadInput.addEventListener("change", function (e) {
      var file = e.target.files[0];
      if (!file) return;
      const formData = new FormData();
```

```
      formData.append('video', file);
      fetch('https://vrdanmaku.uk:8080/upload', { method: 'POST', body:
formData })
        .then(response => response.json())
        .then(data => {
          // Processing response after successful upload
          //...
        });
    });
```

This code listens for file selection changes, creates a FormData object to which the selected video file is added, and sends a POST request to the server's /upload endpoint via the fetch API.

- **Videos Lists and Actions**

The video list display function is implemented by the updateVideoTable() function:

```
    function updateVideoTable() {
      videoTableBody.innerHTML = '';
      videoList.forEach(video => {
        const row = document.createElement('tr');
        row.dataset.id = video.id;

        // Create tag elements
        let tagHtml = '';
        if (video.url === videos.vid1.src) {
          tagHtml += '<span class="video-tag">vid1</span>';
        }
        if (video.url === videos.vid2.src) {
          tagHtml += '<span class="video-tag">vid2</span>';
        }

        row.innerHTML = `
      <td>${video.name}</td>
      <td>${tagHtml}</td>
      <td>:</td>
    `;

        row.addEventListener('click', function (e) {
          showVideoActions(e.clientX, e.clientY, video.id);
        });

        videoTableBody.appendChild(row);
      });
    }
```

This function empties the existing table, iterates through the list of videos, creates a row for each video, marks whether it is currently set to vid1 or vid2, and adds a click event listener so that the action menu pops up when the user clicks on it.

Video Actions Menu Function

When clicking on a video line element, the action menu is displayed by the showVideoActions() function:

```
function showVideoActions(x, y, videoId) {
  selectedVideoId = videoId;
  videoActionPopup.style.left = x + 'px';
  videoActionPopup.style.top = y + 'px';
  videoActionPopup.classList.remove('hidden');
}
```

The pop-up UI effect is shown in Figure 4 and contains four different buttons for renaming, deleting, and tagging features.

Video Rename Function Implementation:

```
videoRenameBtn.addEventListener('click', function () {
  const video = videoList.find(v => v.id === selectedVideoId);
  if (video) {
    const newName = prompt('Enter new name:', video.name);
    if (newName && newName.trim() !== '') {
      ws.send(JSON.stringify({
        type: 'update_video_name',
        videoId: selectedVideoId,
        newName: newName.trim()
      }));
    }
  }
  videoActionPopup.classList.add('hidden');
});
```

When the Rename button is pressed, the corresponding naming window will pop up, as shown in Figure 5:

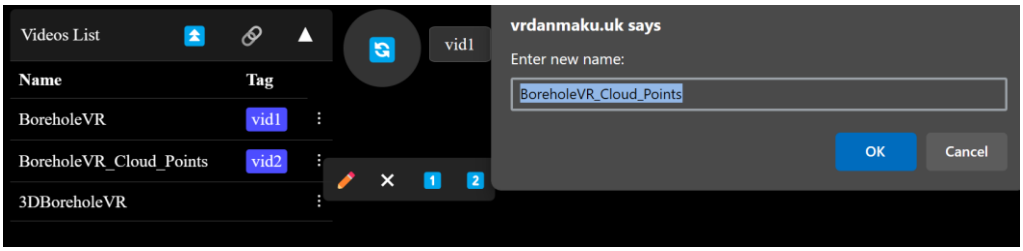

Figure 5. UI Triggered by Video Rename Event

The video delete function and the vid1/vid2 settings buttons similarly trigger WebSocket messages, informing the server to update the video tags. The detailed code can be found in Appendix 1.

Back-End Implementation (server.js):

- **Video File Upload Processing**

Formidable is a Node.js module for parsing form data, especially file uploads, to simplify server-side processing of files and form fields. The back-end handles file upload requests through the formidable library:

```
if (req.method === "POST" && parsedUrl.pathname === "/upload") {
  const form = new formidable.IncomingForm({
    uploadDir: VIDEOS_DIR,
    keepExtensions: true,
    maxFileSize: 2000 * 1024 * 1024, // Set video limit to 2GB
    multiples: true
  });

  form.parse(req, async (err, fields, files) => {
    // Process upload files and update video information
    const newVideo = {
      id: videoId,
      name: fields.name?.[0] || newFileName,
      url: videoUrl,
      tags: fields.tags?.[0] || "",
      uploadDate: new Date().toISOString()
    };

    videoInfo.videos.push(newVideo);
    fs.writeFileSync(VIDEO_INFO_FILE, JSON.stringify(videoInfo), "utf-8");

    broadcastVideoList();

    res.writeHead(200, { "Content-Type": "application/json" });
    res.end(JSON.stringify({ success: true, video: newVideo }));
  });
}
```

After a successful upload, the server adds the new video information to the videoInfo.videos array, updates the JSON file, and broadcasts the update to all clients via the broadcastVideoList() function.

- **Video Renaming Function**

The server receives and processes rename requests:

```
  else if (parsedMessage.type === "update_video_name") {
    //Update video name
    const { videoId, newName } = parsedMessage;
    const videoIndex = videoInfo.videos.findIndex(v => v.id === videoId);

    if (videoIndex !== -1) {
      videoInfo.videos[videoIndex].name = newName;
      fs.writeFileSync(VIDEO_INFO_FILE, JSON.stringify(videoInfo), "utf-8");
      broadcastVideoList();
    }
  }
```

- **Video Deletion Function**

The server receives and processes deletion requests:

```
  else if (parsedMessage.type === "delete_video") {
    // delete video
    const { videoId } = parsedMessage;
    const videoIndex = videoInfo.videos.findIndex(v => v.id === videoId);

    if (videoIndex !== -1) {
      const video = videoInfo.videos[videoIndex];
      const videoPath = path.join(__dirname, video.url);

      // Try to delete the video file
      try {
        if (fs.existsSync(videoPath)) {
          fs.unlinkSync(videoPath);
        }
      } catch (err) {
        console.error("Error deleting video file:", err);
      }

      // Remove video from the list
      videoInfo.videos.splice(videoIndex, 1);
      fs.writeFileSync(VIDEO_INFO_FILE, JSON.stringify(videoInfo), "utf-8");
      broadcastVideoList();
    }
  }
```

This code not only removes the video information from the video list but also attempts to delete the actual video file on the server, ensuring that resources are freed up.

- **Video List Broadcasting**

The server broadcasts the latest video list to all connected clients via the broadcastVideoList() function:

```
function broadcastVideoList() {
  wss.clients.forEach((client) => {
    if (client.readyState === WebSocket.OPEN) {
      client.send(
        JSON.stringify({ type: "video_list", data: videoInfo.videos })
      );
    }
  });
}
```

WebSocket Communications

The front and back ends communicate in real-time via WebSocket in both directions. When a client connects, the server sends the current video list, or it can actively request the latest video list. Any client action on a video is sent to the server via WebSocket, and the server processes it and broadcasts the update, ensuring that all client data is synchronised.

Summary

The video management feature is the basic module of this VR web platform, which enables comprehensive video resource management through close collaboration between the front-end and the back-end. The front-end provides an intuitive user interface and interactive experience, while the back-end ensures data persistence and multi-client synchronisation.

5.3.2 Annotation Management Feature

The annotation management function is one of the interactive modules of this VR video platform, which aims to provide a precise and efficient collaborative labelling and discussion mechanism for geological exploration data analysis. This function allows users to add annotations to any time point and spatial location of the video in the immersive VR environment, and provides a complete set of annotation management tools, including:

- Annotation List: All added annotations are displayed on the side of the interface in sequential order for easy access and management.
- Content View: visual display of the text content of each annotation and the corresponding video time point.
- Delete Function: allows users to remove annotations that are no longer needed.
- Time Jump: clicking on an annotation immediately jumps to the corresponding point in time in the video, quickly locating the area of interest.
- Real-Time Synchronisation: synchronised display of annotations between multiple users in real time, supporting remote collaborative analysis.

Front-End Implementation (index.html)

- **User Interface Structure**

The front-end interface of annotation management mainly consists of two parts: the annotation input area and the annotation list.

```
<div id="annotation-list-container">
  <div id="annotation-list-header">
    <span>Annotations List</span>
    <div class="header-actions">
      <button id="annotation-list-toggle">▼</button>
    </div>
  </div>
  <div id="annotation-list-content">
    <table id="annotation-table">
      <thead>
        <tr>
          <th>Time</th>
          <th>Text</th>
        </tr>
      </thead>
      <tbody id="annotation-table-body">
        <!-- The annotation table will dynamaically generated here -->
      </tbody>
    </table>
  </div>
</div>
```

```
<div id="annotation-action-popup" class="hidden">
  <button id="annotation-delete">✕</button>
  <button id="annotation-jump">🚀</button>
</div>
```

With the above HTML code, the UI of the annotations list can be available as shown in Figure and Figure :

Figure 6 . The UI of the Folded Annotations List

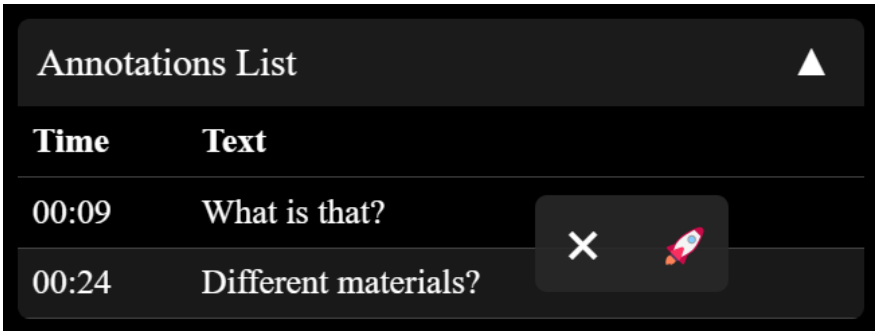

Figure 7 . The UI of the Expanded Annotations List with Pop-up Menu

In the title section of the list form, there is a triangle label that indicates whether the form is folded or not. The list content includes the time of the annotation, as well as its text content. Clicking on the list element pops up a menu containing 2 buttons for deleting the corresponding annotation and jumping to the corresponding time point.

- **Annotation List Management**

The function of displaying, deleting the annotation list is implemented by the following code:

```
  function updateAnnotationTable() {
    annotationTableBody.innerHTML = '';
    annotationList.forEach((annotation, index) => {
      const row = document.createElement('tr');
      row.dataset.index = index;

      // format the time
      const timeFormatted = formatTime(annotation.time);

      row.innerHTML = `
    <td>${timeFormatted}</td>
    <td>${annotation.text.replace(/\n/g, ' ')}</td>
  `;
```

```
        row.addEventListener('click', function (e) {
          selectedAnnotationIndex = parseInt(this.dataset.index);
          showAnnotationActions(e.clientX, e.clientY);
        });

        annotationTableBody.appendChild(row);
      });
    }
```

```
    annotationDeleteBtn.addEventListener('click', function () {
      if (selectedAnnotationIndex >= 0 && selectedAnnotationIndex <
annotationList.length) {
        const annotation = annotationList[selectedAnnotationIndex];

        console.log("Deleting annotation:", annotation);

        // send delete annotation message to the server
        const deleteMessage = {
          type: "delete_annotation",
          time: annotation.time,
          text: annotation.text,
        };
        console.log("Sending delete message:", deleteMessage);
        ws.send(JSON.stringify(deleteMessage));

        // remove the annotation entity from the scene
        if (annotation.entity) {
          scene.removeChild(annotation.entity);
        }
        if (annotation.backgroundEntity) {
          scene.removeChild(annotation.backgroundEntity);
        }

        // remove the annotation from the list
        annotationList.splice(selectedAnnotationIndex, 1);
        updateAnnotationTable();

        // close the action popup
        annotationActionPopup.classList.add('hidden');
      }
    });
```

Similar to the video management feature, both additions and deletions, the annotation list are also real-time synchronised with the server to display the content.

- **Annotation Time Point Jump**

The feature of jumping to the corresponding time point is implemented by the following code:

```
   annotationJumpBtn.addEventListener('click', function () {
      if (selectedAnnotationIndex >= 0 && selectedAnnotationIndex <
annotationList.length) {
        const annotation = annotationList[selectedAnnotationIndex];
        const currentVid = currentVideoIndex === 1 ? vid1 : vid2;

        // jump to the annotation time
        currentVid.currentTime = annotation.time;

        // update the timeline
        updateAnnotationPositions();

        // close the action popup
        annotationActionPopup.classList.add('hidden');
      }
    });
```

The annotations correlate to specific points in time and spatial locations of the video, making the discussion more precise and focused.

- **Annotation Display and Position Update**

Annotation display in 3D space and updating with the video point in time is achieved with the following code:

```
    function updateAnnotationPositions() {
      var currentTime =
        currentVideoIndex === 1 ? vid1.currentTime : vid2.currentTime;

      annotationList.forEach(function (annotation) {
        if (!annotation.shown && currentTime >= annotation.time) {
          // creat the annotation background
          var backgroundEntity = document.createElement('a-plane');
          backgroundEntity.setAttribute('color', 'black');
          backgroundEntity.setAttribute('opacity', 0.5);
          backgroundEntity.setAttribute('width', 1.2);
          backgroundEntity.setAttribute('height', 0.3);
          backgroundEntity.setAttribute('position', annotation.position);
          backgroundEntity.setAttribute('look-at', '[camera]');
          scene.appendChild(backgroundEntity);

          annotation.backgroundEntity = backgroundEntity;

          // creat the annotation text
          var annotationEntity = document.createElement('a-entity');
          annotationEntity.setAttribute('text', {
```

```
            value: annotation.text,
            color: 'white',
            align: 'center',
            width: 1,
            wrapCount: 20
          });
          annotationEntity.setAttribute('position', annotation.position);
          annotationEntity.setAttribute('look-at', '[camera]');
          scene.appendChild(annotationEntity);

          annotation.entity = annotationEntity;
          annotation.shown = true;
        }

        if (annotation.shown) {
          var elapsedTime = currentTime - annotation.time;
          var distanceTraveled = elapsedTime * videoSpeed;

          var newPosition = {
            x: annotation.position.x + distanceTraveled,
            y: annotation.position.y,
            z: annotation.position.z
          };

          // update annotation position
          annotation.entity.setAttribute('position', newPosition);

          // update background position
          if (annotation.backgroundEntity) {
            annotation.backgroundEntity.setAttribute('position', newPosition);
            //annotation look at the camera
            annotation.backgroundEntity.setAttribute('look-at', '[camera]');
          }
        }
      });
    }
```

- **WebSocket Communication**

Real-time communication between the front-end and the server is achieved via WebSocket, ensuring synchronisation of comments between multiple users.

```
function connectWebSocket() {
      ws = new WebSocket("wss://vrdanmaku.uk:8080");

      ws.onopen = function () {
        console.log("Connected to WebSocket server");
        clearTimeout(wsReconnectTimer);
      ws.onclose = function () {
        console.log("Disconnected from WebSocket server, trying to
reconnect...");
        wsReconnectTimer = setTimeout(connectWebSocket, 2000);
      };

      ws.onerror = function (err) {
        console.error("WebSocket error:", err);
      };

      ws.onmessage = function (event) {
        try {
          var data = JSON.parse(event.data);
          if (data.type === "existing_annotation") {
            data.data.forEach((annotationMessage) => {
              addAnnotation(
                annotationMessage.text,
                annotationMessage.position,
                annotationMessage.time
              );
            });
          } else if (data.type === "new_annotation") {
            addAnnotation(data.data.text, data.data.position, data.data.time);
          }
        } catch (e) {
          console.error("Error parsing message", e);
        }
      };
    }
```

Backend Implementation (server.js)

- **Annotation Database Creation:**

A JSON database for storing global annotations can be created within the AWS server with the following code:

```
const GLOBAL_ANNOTATION_FILE = "global_annotations.json";   // Global
annotation file

// Load annotation library
function loadAnnotationLibrary() {
  const filepath = path.join(__dirname, GLOBAL_ANNOTATION_FILE);
  if (fs.existsSync(filepath)) {
    return JSON.parse(fs.readFileSync(filepath, "utf-8"));
  }
  return [];
}

// Save annotation library
function saveAnnotationLibrary(library) {
  const filepath = path.join(__dirname, GLOBAL_ANNOTATION_FILE);
  try {
    fs.writeFileSync(filepath, JSON.stringify(library, null, 2), "utf-8");
    console.log("Annotation library saved successfully");
  } catch (error) {
    console.error("Error saving annotation library:", error);
  }
}
```

- **WebSocket Message Processing**

The server performs different processing logic according to the type of information, mainly including adding and deleting annotations. Due to the similarity of the implementation method, the following only shows the processing code for adding annotations. Refer to Appendix 2 for the detailed code.

```
else if (parsedMessage.type === "new_annotation") {
      const annotationMessage = {
        text: parsedMessage.text,
        time: parsedMessage.time,
        position: parsedMessage.position,
        videoLink: parsedMessage.videoLink,
      };

      // Save the annotation message to the library
      const library = loadAnnotationLibrary();
      // Check if the annotation already exists
      const exists = library.some(anno =>
        anno.text === annotationMessage.text &&
```

```
        anno.time === annotationMessage.time &&
        anno.videoLink === annotationMessage.videoLink
      );
      // Add and broadcast only if it does not exist
      if (!exists) {
        library.push(annotationMessage);
        saveAnnotationLibrary(library);

        // Broadcast annotation messages to all clients
        wss.clients.forEach((client) => {
          if (client.readyState === WebSocket.OPEN) {
            client.send(
              JSON.stringify({ type: "new_annotation", data:
annotationMessage })
            );
          }
        });
      }
    }
```

Summary

The front-end achieves an immersive annotation display, and the back-end ensures multi-user real-time synchronisation via WebSocket. The overall design ensures the accuracy of professional analysis and enhances the convenience of team collaboration. The annotation management function is the core interactive module of this VR video platform, which transforms professional analyses from passive viewing to active participation and collaboration.

5.3.3 Dual Video Synchronous Switching Feature

Dual-video switching is one of the featured innovations of this web platform, which is designed to improve the efficiency of comparative analysis of geological exploration data. Users can select and mark two different videos as vid1 and vid2 in the video list, and then seamlessly switch between them during playback by using the switch button, while keeping the timeline of the videos synchronised. This feature is particularly useful for comparing the same geological structure rendered in different materials, or comparing the same area over time.

The procedure is as follows:

1. In the video list management interface, select a video and set it to vid1 or vid2.

2. Start playing the currently active video (default is vid1).

3. When you need to compare, click the video switch button, and the system will keep the current time point but switch to another video source.

4. Playback will continue, switching between the two sources will not interrupt the continuity of the video timeline.

Front-End Implementation (index.html):

This feature is only implemented based on videos already loaded on the client (front-end), when there are two or more available videos already loaded locally, switching the video source does not involve WebSocket communication and server processing.

- **Video Sphere and Material Setting:**

The core of the dual-video switching functionality is implemented through the A-Frame framework. In the HTML structure, two video elements are predefined and applied as material sources to the same video sphere:

```
    <a-assets>
      <video id="vid1" playsinline crossorigin="anonymous" loop="true"
style="display: none"></video>
      <video id="vid2" playsinline crossorigin="anonymous" loop="true"
style="display: none"></video>
    </a-assets>

    <a-entity id="video-sphere" geometry="primitive: sphere; radius: 3000;
segmentsWidth: 64; segmentsHeight: 64;"material="shader: flat; src: #vid1;"
scale="-1 1 1" visible="true">
    </a-entity>
```

When performing a switching operation, the code only needs to change the material source (material src attribute) of the video-sphere entity to instantly switch the display inside the video sphere from one video to another. The A-Frame automatically handles the material change and rendering update, making the switching process seamless.

- **Video Switch Button and Switching Logic:**

The video switch button on the interface (as shown in Figure 6) is responsible for triggering a switch between two videos:

```
<button id="switch-video-button"></button>
```

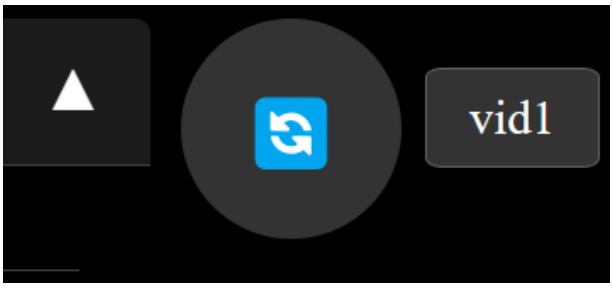

Figure 8. Video Switching Buttons on the Interface

The event listener for the toggle button implements the core toggle logic:

```
switchVideoButton.addEventListener("click", function () {
  currentVideoIndex = currentVideoIndex === 1 ? 2 : 1;

  // Update the video label
  videoLabel.textContent = `vid${currentVideoIndex}`;

  // Switch the video source
  videoSphere.setAttribute('material', 'src', `#vid${currentVideoIndex}`);

  // Update the annotation position
  updateAnnotationPositions();
});
```

This code works as follows:

Video Index Switching: When the user clicks the switch button, currentVideoIndex alternates between 1 and 2. This means that the system supports switching between two video sources.

Update Interface Label: update the video label text to show the video number (vid1 or vid2) that is currently playing.

Switching material source: switch the material source of the videoSphere from the current video to another video via the setAttribute method of the A-Frame. This is the key step to achieve visual switching.

Synchronise annotation positions: call the updateAnnotationPositions() function to make sure that all video-related annotation positions are also correctly updated with the video switch.

● **Tag Interaction Management**

Your code also includes interaction management for video tags:

```
    // Display video label
    videoLabel.addEventListener('blur', function () {
      if (this.textContent.trim() === '') {
        this.textContent = `vid${currentVideoIndex}`;
      }
    });
```

This code ensures that even if the user tries to clear the video tabs, the system automatically restores the correct tab display when it loses focus, maintaining the consistency and usability of the interface.

Summary

The dual-video switching feature is a purely front-end implementation of a feature that utilises the material system of the A-Frame framework to achieve seamless switching. The system switches between different videos by changing the material source properties of the video sphere, while keeping the timeline synchronised and the annotation position updated.

This design allows users to compare different video contents at the same point in time, which is especially suitable for geological exploration and other scenarios that require multi-angle and multi-processing comparative analysis.

6 Model Optimization and VR Video Production Workflow

MeshLab is an open-source 3D mesh and point cloud processing software, which supports various point cloud and 3D mesh formats (e.g. .obj, .ply), and can perform model simplification, normal correction, surface reconstruction, and texture mapping, which is an important tool for point cloud processing in scientific research and engineering.

Unity 3D engine is a cross-platform real-time 3D content development tool, widely used in game development, virtual reality (VR), augmented reality (AR), simulation, animation, architectural visualisation and other fields. Unity provides a rich development environment and intuitive interface to support the user through the visual operation and scripting (mainly C#) to create and manage three-dimensional scenes, models, lighting, physics, animation, and interactive content.

In this project, given the substantial size of the original point cloud model, direct import into the Unity engine was impractical due to memory and performance constraints. It is necessary to use MeshLab to simplify the model first, and then import the model with materials into Unity correctly for subsequent VR scene development.

6.1 Model Simplification and Texture Baking in MeshLab

- **Model Import and Material Inspection**

The first step is to import the point cloud model (usually in .obj format) into MeshLab. care should be taken during this step to preserve any existing vertex colours or textures. As shown in Figures 9 and 10, the point cloud model of a borehole is imported into MeshLab in two views, and point rendering is selected here.

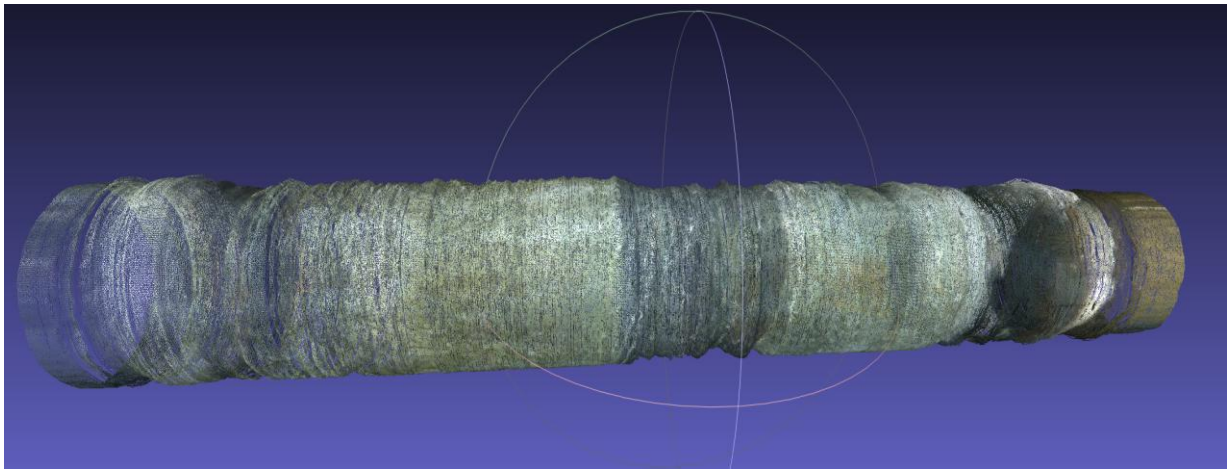

Figure 9. Points Cloud of Borehole Model in Points Rendering from an Outside Perspective

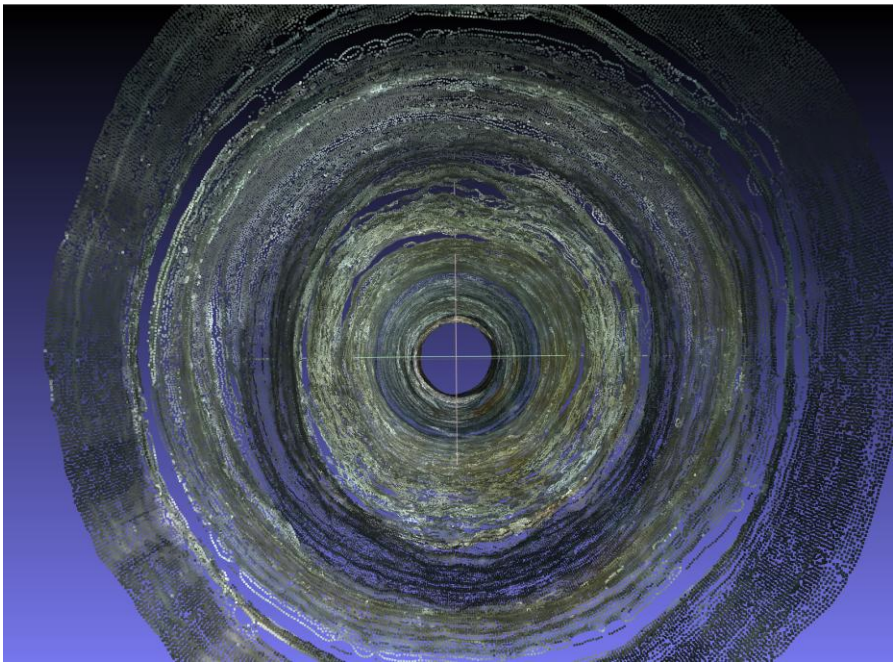

Figure 10. Points Cloud of Borehole Model in Points Rendering from an Inside Perspective

Adjusting the rendering mode to Vert rendering results in a more continuous 3D model, as shown in Figure 11:

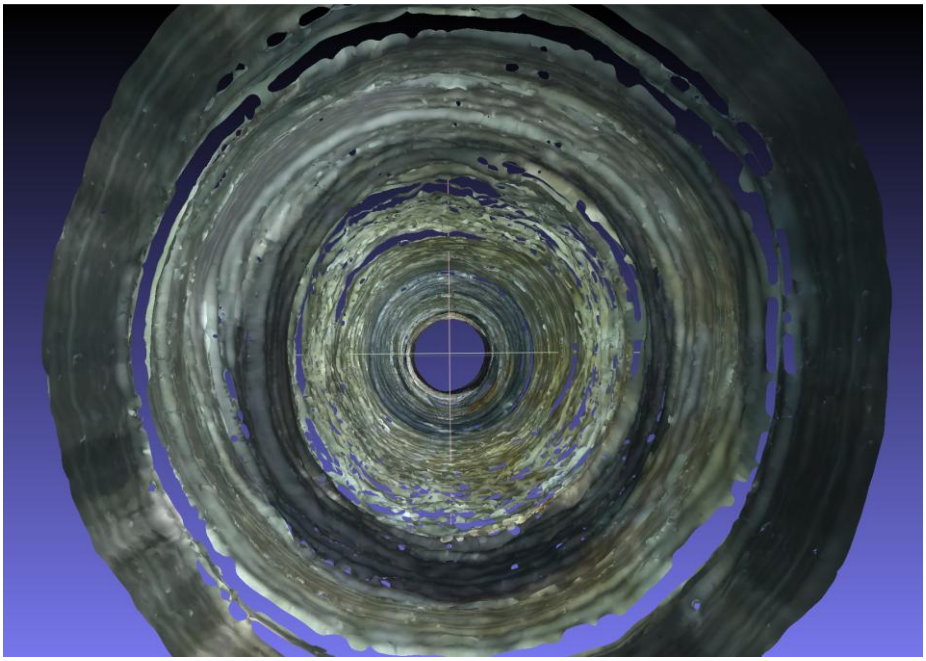

Figure 11. Borehole Model in Vert Rendering from an Inside Perspective

The source model contains only vertex colours, they must be converted to texture maps to ensure compatibility with the Unity render pipeline.

- **Model Simplification via Quadratic Edge Collapse Decimation**

To reduce the model's complexity while retaining critical geometric details, the Quadratic Edge Collapse Decimation filter was utilized. This involved setting a target number of faces, usually a significant reduction to 50% of the original face count. Options such as "Preserve Normal" and "Preserve Topology" were enabled to maintain the model's visual coherence and spatial structure after decimation.

The simplified result is shown in Figure 12, where the overall number of mesh faces is greatly reduced, which can effectively reduce the computational overhead of subsequent rendering and interaction. The triangular facets of the simplified model are more obvious, which is a common mesh feature change of the algorithm, and some of the tiny details have been smoothed, but the main geometric features and texture information of the drilled hole structure are still well preserved in general.

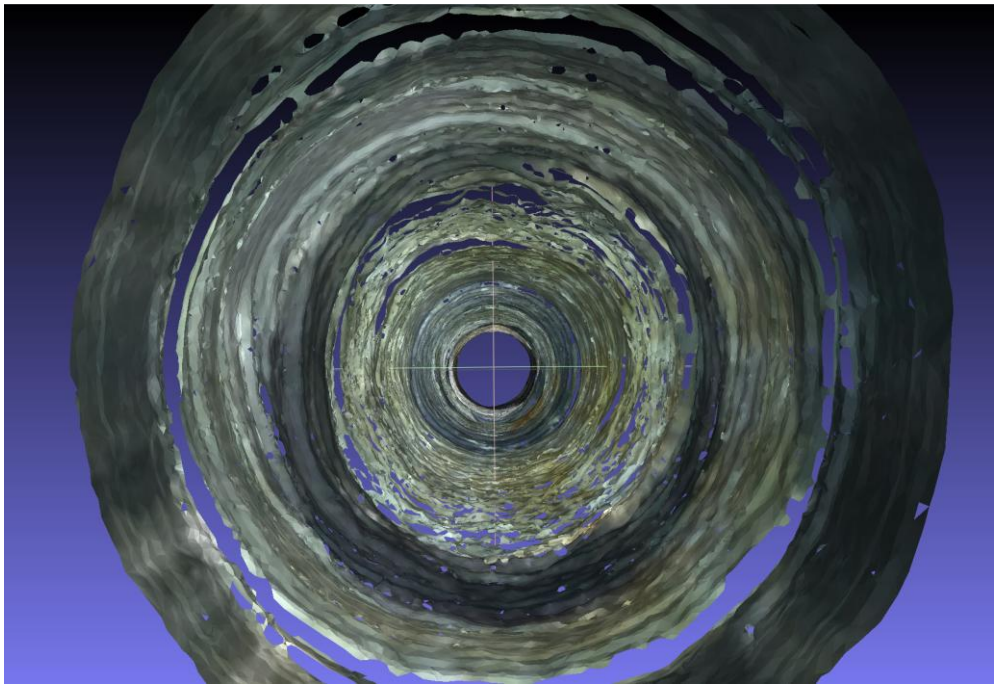

Figure 12. Borehole Model in Vert Rendering after Simplification

- **UV Unfolding and Texture Baking**

To ensure that the model's colour information is correctly identified in Unity, first, perform a UV Unfold operation in MeshLab using the 'Parametric + Texture from Mesh' filter. The method chosen was 'Trivial Per-Triangle Mapping' to achieve reasonable UV coverage and minimal distortion.

After UV mapping, the Vertex Colour to Texture filter is applied to bake the vertex colours into a high-resolution texture of 4096×4096 pixels, which is automatically assigned to the mesh. The generated high-resolution texture is shown in Figure 13. The generated texture mapping not only preserves the appearance of the original data but also contributes to efficient rendering in Unity.

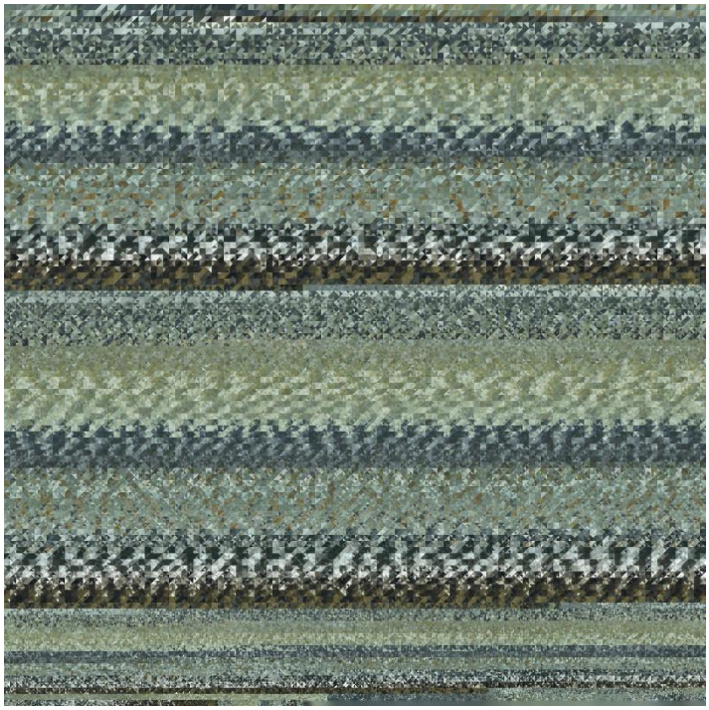

Figure 13. High-Resolution Textures Corresponding to Borehole Models

- **Model and texture export**

The final step was to export the optimised model as an .obj file containing the generated material (.mtl) and texture (.jpg) files.

- **Summary**

Through this workflow, the initially large amount of point cloud data was successfully converted into a streamlined textured 3D model. This greatly improves subsequent VR visualisation in Unity without sacrificing basic visual fidelity.

6.2 VR Video Generation in Unity

The purpose of this section is to introduce how to reconstruct the geological borehole VR scene and record high-quality video on the Unity 3D platform based on the optimised point cloud/surface model processed in MeshLab, so as to achieve the data visualisation and virtual roaming of 3D geological structures.

- **Model import and scene construction**

Firstly, the point cloud model (in OBJ format and its corresponding MTL and texture map files), which has been simplified in MeshLab with material maps, is imported into the Unity project, as shown in Figure 14. To ensure that the model is accurately represented in the scene, the position, scale, and orientation of the model are adjusted appropriately.

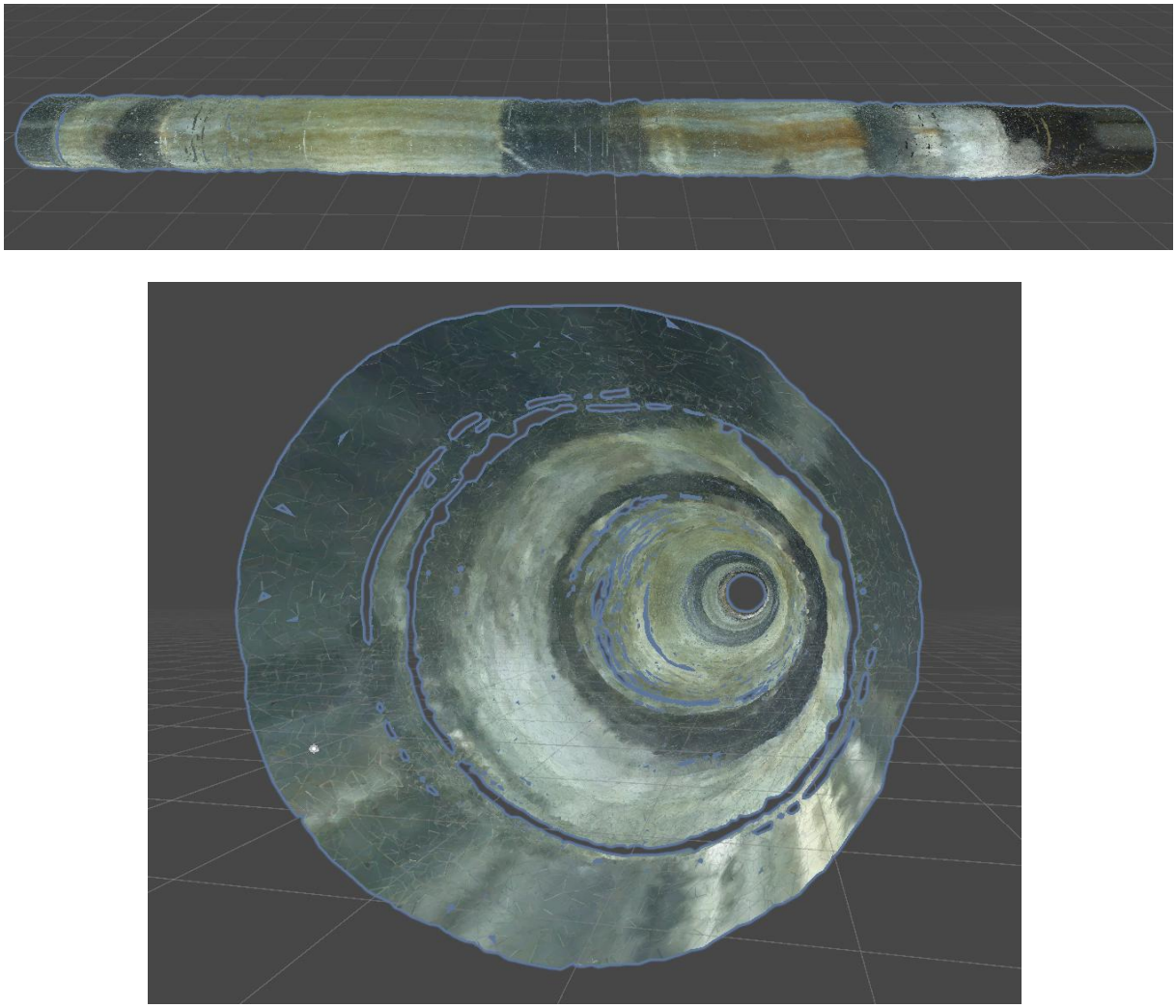

Figure 14. Simplified Borehole Model Rendered in Unity

● **Camera Trajectory Settings**

In order to realistically reproduce the perspective experience of drilling exploration, it is necessary to set the main camera's movement trajectory in the Unity scene. In this project, a camera motion script is written to smoothly advance along the borehole axis, simulating the observation process of actual drilling equipment and facilitating the automated recording of VR videos.

● **VR Support and Recording Configuration**

The project adopts Unity's built-in VR support module, XR Plugin Management, which can be activated to complete stereoscopic and immersive rendering in the virtual scene. During the recording process, the output resolution (such as 4K), frame rate (such as 60fps) and other parameters can be set according to the actual needs, and the Unity Recorder plug-in is used to achieve high-quality export of automatic roaming video of the VR scene, and ultimately generate the standard high-definition video file (MP4 format).

The effect of the recorded VR video of the simplified point cloud model in the local PotPlayer player, as shown in Figure 15:

Figure 15. VR Video in Local PotPlayer Player

● **Summary**

This process not only effectively integrates the segmentation of point cloud data modelling, 3D visualization and automated recording, but also lays a solid foundation for the subsequent multimodal display and geomorphological analysis on the Web side, and achieves a high degree of reproducibility in the migration and dissemination of real geological information to the virtual environment.

7 Results

7.1 System Functionality Testing

The overall function of the web client is stable, the VR video playback is smooth, the perspective control is smooth, and there is no abnormality in the core operations such as video and annotation management, video switching, and progress control. UI adaptation for major browsers such as Safari, Chrome, Edge, etc.

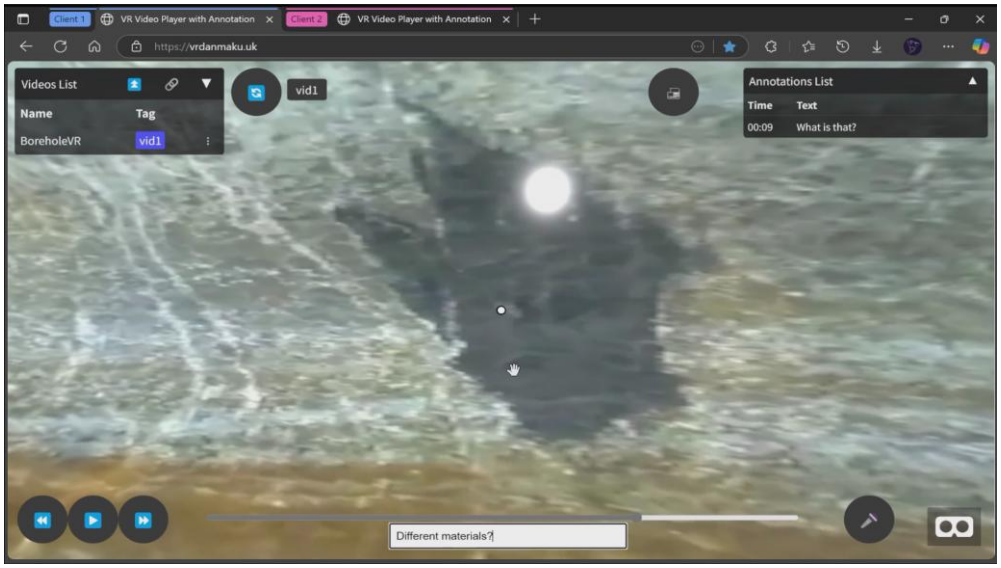

Figure 16. Adding New Annotation on Windows Explorer

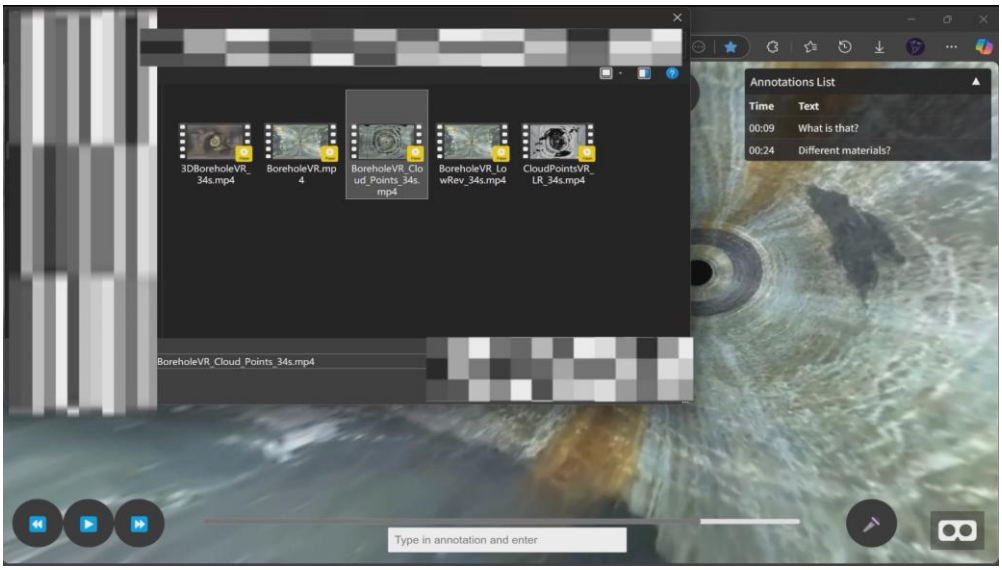

Figure 17. Uploading Local VR Video

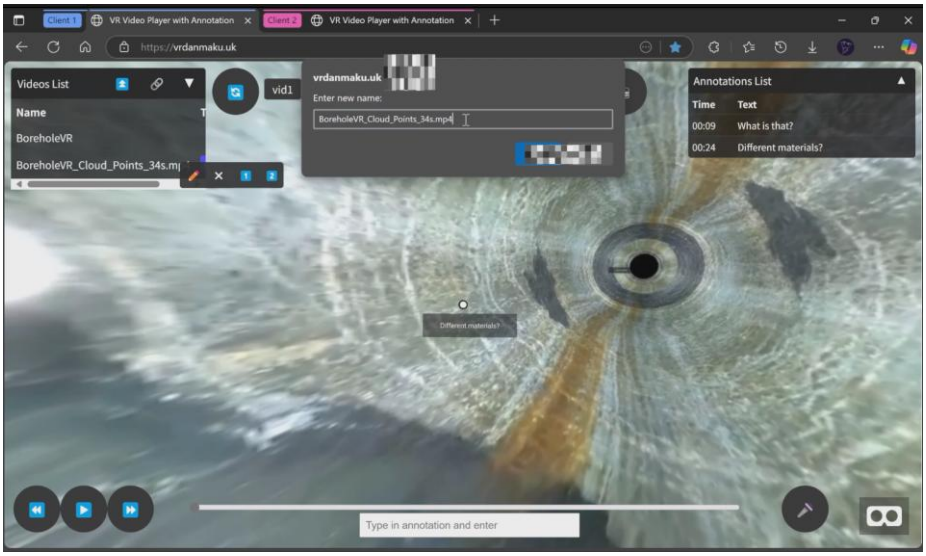

Figure 18. Changing the Name of VR Videos on Windows' Browser

The information synchronisation between multiple users and multiple is good, and different users (including different terminals) can achieve real-time synchronisation of video progress, annotations and other functions.

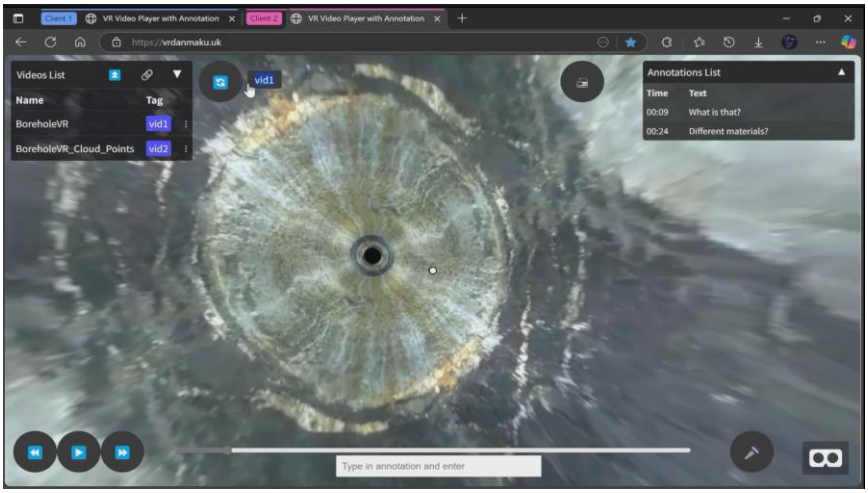

Figure 19. List Information Synchronised within Another Client on Windows

Figure 20. List Information Synchronised within the Client on iOS

As shown in Figure 21, users can switch geological videos with different renderings or materials under the same timeline, which is convenient for visual comparison and analysis. For example, vid1 and vid2 show the visualisation of the same borehole under different treatments, respectively.

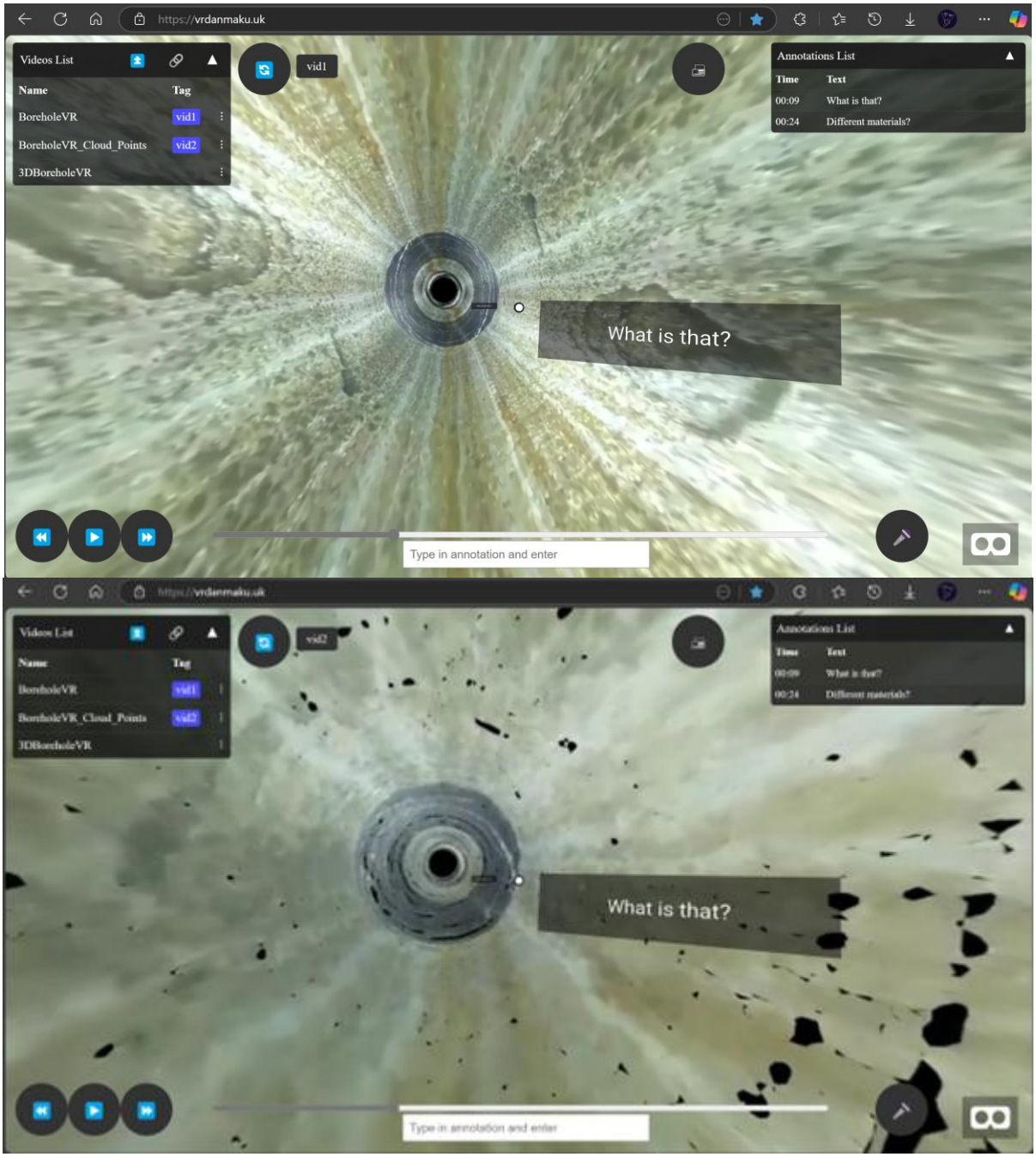

Figure 21. Synchronised Comparison of Different Rendered Borehole Videos

7.2 Functional Exceptions and Compatibility Issues

7.2.1 Anomaly in Annotation Function

When adding new annotations, there are occasional repetitions (2 or 3 times) in the annotation list; and the new annotations are always fixed in the user's current viewpoint.

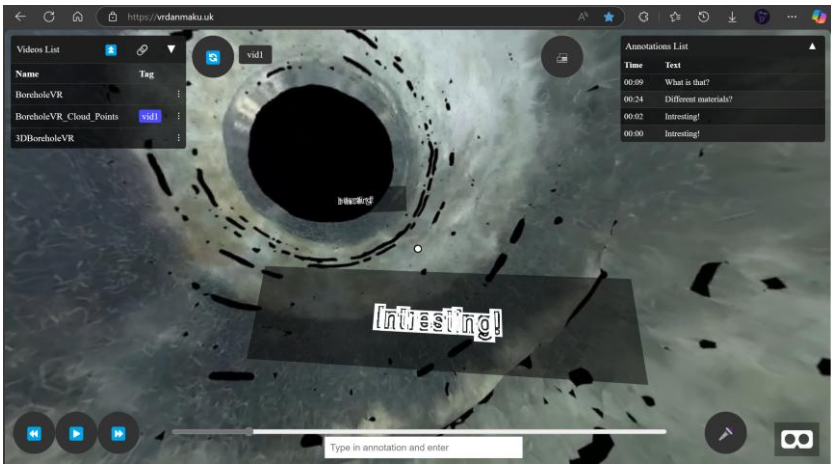

Figure 22. Error-Displaying Added Annotation

At this time, the synchronisation and display of annotations on other terminals are still normal, and the problem is confined to the current operating terminal, which is suspected to be related to the local storage of the front-end or the multiple triggering of events.

7.2.2 Video Fluency and UI Problems on Mobile Client

- The mobile phone terminal can only run smoothly when playing a single video, but when selecting the function of comparing two videos, the lag is obvious and almost unusable, presumably related to the hardware and rendering performance of the device.
- There are transparency and display errors at the bottom of the video list UI on the mobile phone side, resulting in the user's operation experience being affected. As shown in Figure 23:

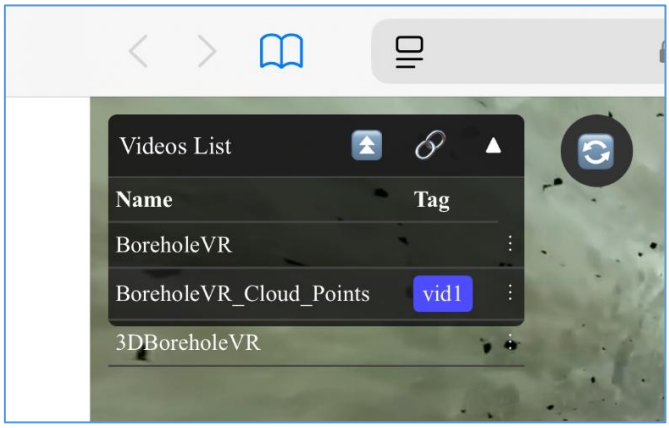

Figure 23. UI for Transparent Lists on Mobile

7.3 VR Visualisation of Point Cloud Model

The simplified point cloud model can completely retain the colour and overall spatial structure, which basically achieves the design objective.

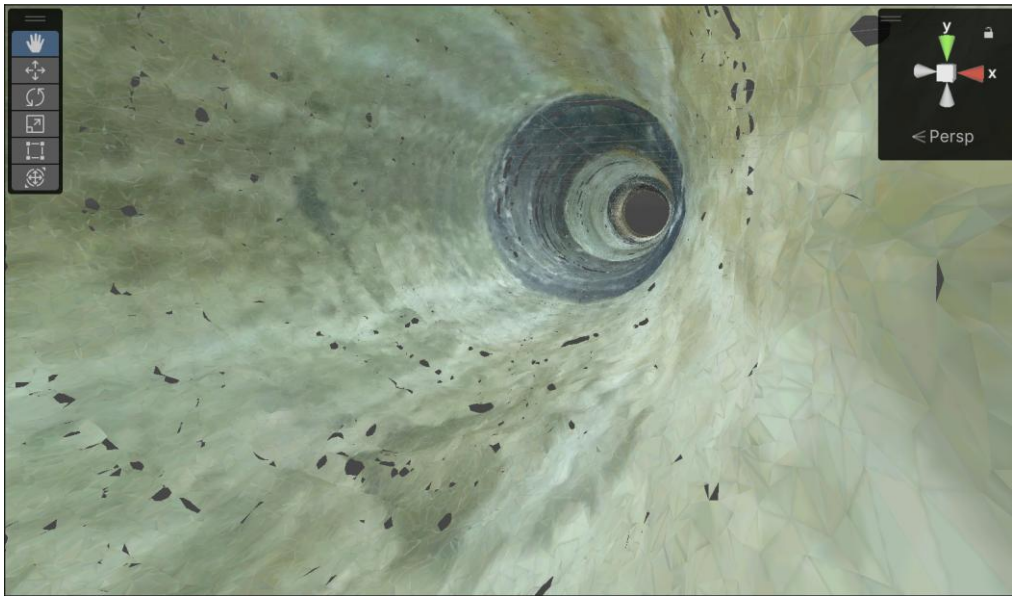

Figure 24. Simplified Points Cloud Module Rendering in Unity

However, in the presentation of the model (Figure 25), a number of irregular holes and voids can be clearly seen distributed on the surface. These holes are mainly manifested in the absence of some areas of the model's surface, which appear as black ‘holes’ in the form of rings or broken pieces, resulting in the disruption of the overall surface continuity.

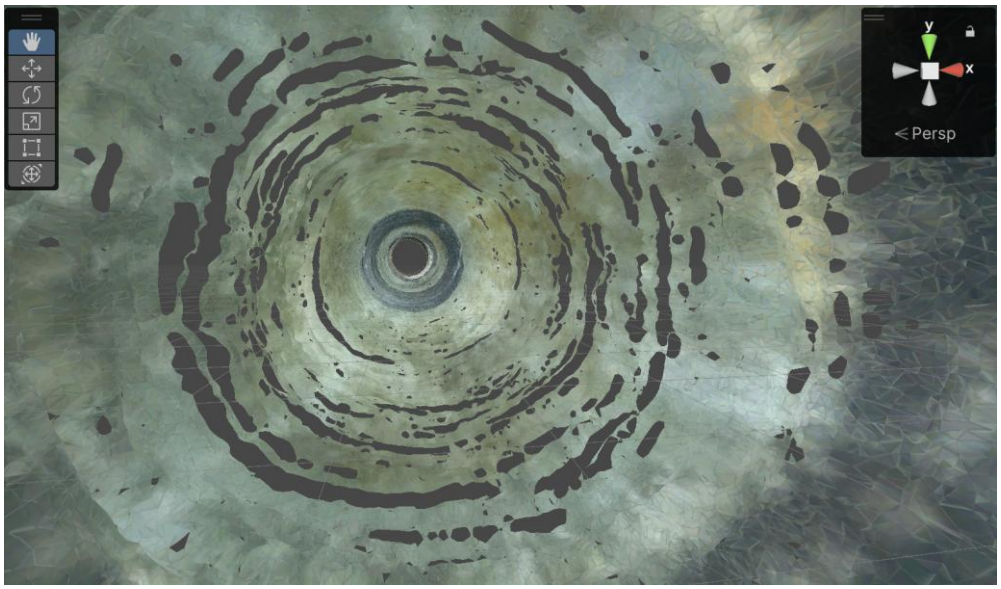

Figure 25. Borehole Model with Broken Holes on Surface

It is clearly observed by the comparison of Figures 26 and 27, geological features, especially complex three-dimensional structures (e.g. protuberances, fissures, laminations) are not obvious enough after the VR video conversion, and the three-dimensional details are not enough, which affects the accuracy of immersive interpretation.

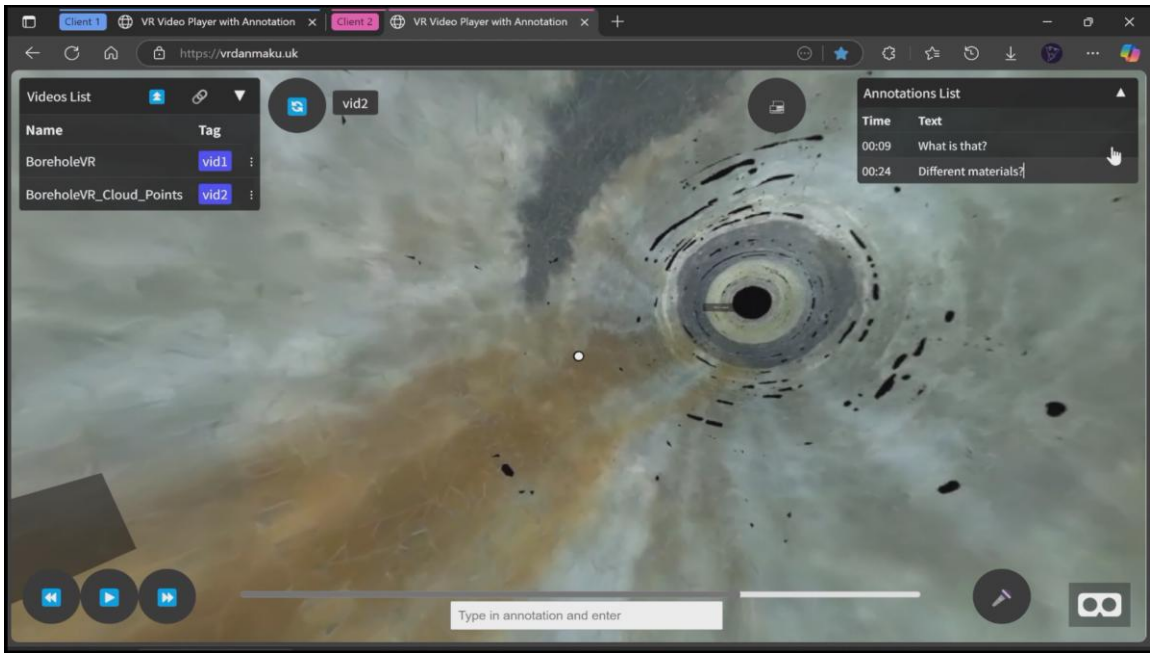

Figure 26. Simplified Point Cloud Module in VR Video

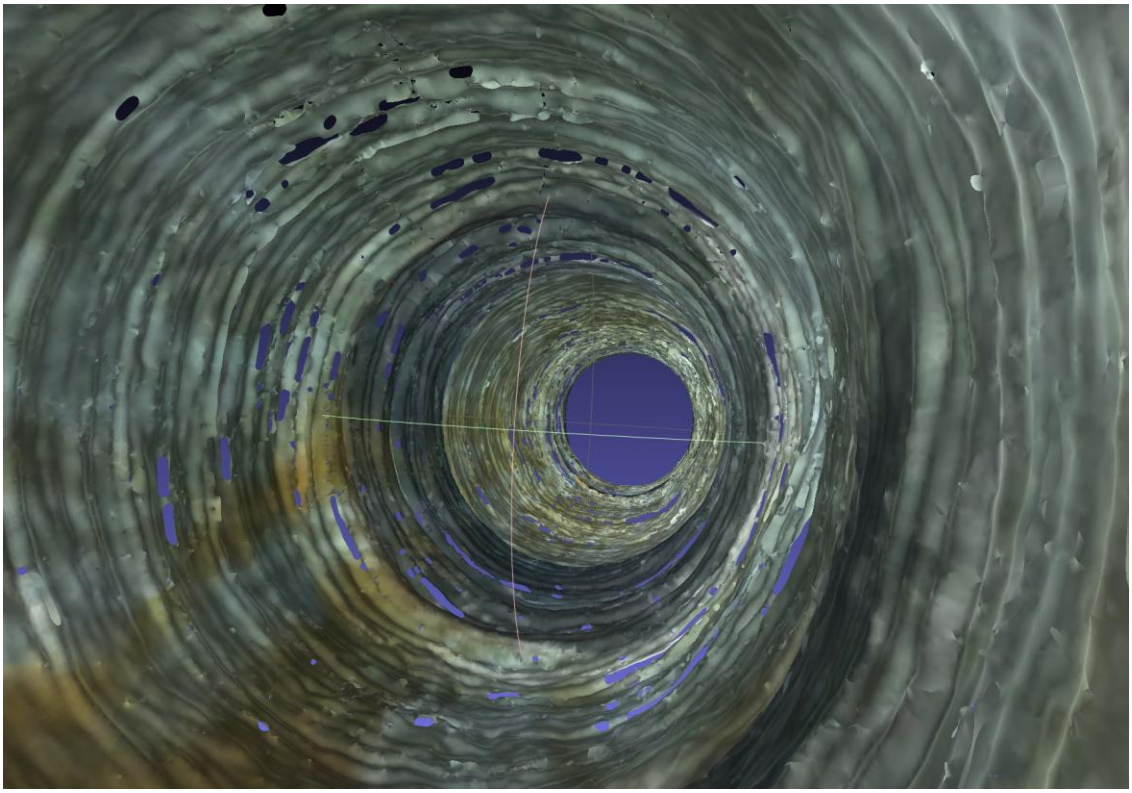

Figure 27. The Original Point Cloud Model

7.4 Summary of Results

This section summarises the main test results of the platform. Through comprehensive testing on multiple mainstream browsers of Windows PC terminal and iOS mobile terminal, the system demonstrates high cross-platform compatibility and functional stability. pc terminal has complete functions, smooth operation experience, and good synchronisation of information between multiple people and multiple terminals, which fully meets the demand for multi-user collaboration of geological data. iOS terminal basically realises the main functions after adaptation, but the dual video playback and however, some UI details still need to be optimised.

Some technical bottlenecks and anomalies were found during the test, such as annotation repeat display, double video lag and model hole, which limit the engineering applicability of some advanced functions. In particular, the loss of 3D details and geological feature information during point cloud simplification and model reconstruction is of concern. Overall, the platform has achieved the core objectives of ‘ready-to-use, guaranteed collaboration, compatible with multiple ends’, which provides a solid technical foundation for promoting the popularity and collaborative application of visualisation tools in the geophysical industry, and at the same time clarifies the direction of the subsequent optimisation and function expansion.

8 Discussion

8.1 Comparison and Evaluation of Results and Specification

- **Interactive 3D Visualisation and Multi-Data Support**

  The core objective of the project is to transform the traditional 2D geological borehole video into an interactive 3D virtual reality scene, and to support free switching between multiple videos and comparative analysis. Practical tests show that the system successfully realises VR viewpoint browsing and immersive experience. In addition, the added video list function allows users to switch, synchronise and compare data from different drill holes, exceeding the limitations of the original 'single-video visualisation' and enhancing the flexibility and scalability of the platform.

- **Annotation System and Multi-User Collaboration**

  According to the specification, the system needs to be equipped with annotations based on time or space synchronisation, in order to replace the instantaneous bullet comments and achieve a more orderly collaborative interaction. The finalised annotation list not only allows users to add or view annotations at different nodes of the VR video, but also the synchronisation of the annotations, helping team members to review and discuss the historical viewpoints and decision-making basis. Although there are problems, such as anomalies or duplicates in the display of annotations in complex scenarios, the overall usability and synchronisation criteria are met.

- **Cross-platform compatibility and performance**

  The system in the mainstream Windows desktop major browser performance is stable, complete functions, multi-user, and the collaboration experience is good; iOS terminal achieves the main functions, but in the multi-video playback, UI details of the adaptation still have room for improvement. Besides, this project lacks Android browser adaptation. This shows that the goal of 'seamless multi-end adaptation' has basically been achieved, but further optimisation is still necessary.

- **User Interface (UI) Design Evaluation**

  In terms of UI, the project clearly emphasises the design of a concise and intuitive interface for non-professional VR users. Most users were able to successfully complete core tasks such as navigation, annotation, adding and managing video resources. However, there is still room for improving the aesthetics and convenience of some of the mobile UI in terms of small-screen adaptation and touch gesture interaction.

8.2 Critical Analysis of Emerging Issues and Limitations

- **Compatibility and Performance Bottlenecks**

In iOS and other mobile terminals, there are still obvious limitations in compatibility. For example, for dual-video synchronous playback, mobile devices are more susceptible to insufficient processing power and memory to support high-load 3D rendering and multi-video operations, thus affecting the smoothness of interaction. Furthermore, the browser on the Web standards, especially WebSocket and other implementation differences, brings the technical challenges of cross-platform development.

- **Limitations of Annotation and Collaboration Functions**

The annotation system suffers from the problem of duplicate displays, especially in high-concurrency scenarios. Causes include server-side message queues and front-end synchronisation mechanisms that have not yet been fully optimised, as well as display entity loading errors caused by user jump times.

- **Distortion of Point Cloud Simplification and 3D Reconstruction**

In point cloud optimisation and surface reconstruction, the platform has implemented a simplified solution for the pursuit of browsing efficiency, but this has resulted in the loss of details in some complex geological structures, as well as the emergence of ‘voids’ in the model, which has weakened the realism of the 3D geological data to a certain extent.

8.3 Directions for Improvements and Future Work

- **Compatibility and Performance Optimisation**

To overcome the current compatibility and performance limitations on iOS and other mobile devices, on-demand loading, lazy loading, or model chunking techniques can be introduced to significantly reduce the transient rendering load on mobile devices and improve the smoothness when synchronising multiple VR videos.

Strengthen compatibility testing of key APIs such as WebSocket, build unified adaptation and degradation logic to mitigate the occurrence of bugs under different platforms.

- **Collaboration Enhancement**

Given the current annotation function of repeated display, timing disorder and other problems, adjust the annotation loading update trigger, introduce better message queue management logic, and adopt the transaction consistency guarantee scheme to ensure the reliable synchronisation and sequential restoration of the annotation events of each client.

Adjust the triggering of annotation loading and updating, introduce better message queue management in the logic, and adopt a transaction consistency guarantee scheme to ensure the reliable synchronisation and sequential restoration of annotation events in each client.

Furthermore, explore the integration of multi-user real-time voice call function, drawing on the mechanism of ‘multi-person chat room’ to further enhance the efficiency and immersion of off-site collaboration.

- **3D Model Fidelity and Smart Data Processing**

In order to improve the model reproduction and geomorphological features expression ability, we provide optional loading and real-time switching of point clouds of different accuracies for different needs, taking into account the efficiency and detail display.

Advanced point cloud sampling and machine learning algorithms are applied to automatically fill in and repair the ‘voids’ and missing details in the simplification process.

- **Other Innovative Directions**

Add a user account login and permission management mechanism to ensure security and operational compliance of collaboration data.

Image recognition and deep learning models can be integrated to achieve automatic tagging and intelligent searching of geological anomalies, ore bodies or fault features in drillhole video and point cloud data, to improve the efficiency of geoscientific research.

Based on the modular architecture, the platform can be gradually extended to industries requiring complex 3D environment collaboration, such as education, construction, and healthcare, so as to realise the cross-industry value extension of the platform.

8.4 Summary of the Main Contributions

The project has made progress in immersive visualisation and collaborative analysis of geophysical borehole data. The main contributions are listed below:

- **Video and Annotation Management**

The project introduces a video and annotation management system that improves the web-based VR platform to better support real-time, multi-user collaboration. Storing VR videos and annotations via a server improves user convenience and thus the efficiency of geophysical interpretation.

- **Dual Video Synchronised Switching**

A key innovation in this work is the implementation of dual-video synchronisation. These improvements not only allow the flexibility to switch and compare different borehole videos but also significantly improve visualisation fidelity and accuracy of subsurface geological interpretation.

- **Point Cloud Simplification and Texture Mapping**

Due to the huge volume of high-precision point cloud model data collected, directly importing it into Unity will easily cause performance bottlenecks. Therefore, the project carried out model simplification and texture baking in MeshLab, which greatly facilitates the VR visualisation process in Unity while effectively preserving the colour and spatial features.

- **VR Video Production Workflow**

The project is scripted to set the virtual camera to move autonomously along the borehole axis to maximise the restoration of the actual observation experience. The whole VR video is automatically recorded and output in standard format, which is easy to be called and played by the front-end of the webpage, and also provides convenience for data archiving and offline analysis.

9 Conclusion

The project aims to build a web-based virtual reality (VR) platform for visualising geophysical borehole data, overcoming spatial limitations of traditional 2D tools and supporting remote collaborative analysis. Using A-Frame and AWS, the platform provides immersive 3D scenes without special equipment. Real-time communication and annotations via WebSocket improve remote teamwork and efficiency. The dual-video switching function makes geological detail observation more convenient. The system works well on Windows PC and has good iOS compatibility, supporting the promotion of geoscience data visualisation. Point cloud model simplification and VR video workflows were successfully implemented in Unity.

As a technical upgrade of earlier research, the project supports multiplayer collaboration, synchronised resources, and annotation management, solving previous single-user and single-video limitations. The intuitive interface enables even non-experts to efficiently manage and navigate data. High-precision point cloud VR videos enhance the 3D geological structure representation.

Limitations remain: some annotation sync scenarios show abnormalities, iOS and other mobile adaptation needs improvement (especially for dual video and UI), and the system depends on network and hardware performance. Some geological detail is lost during point cloud simplification.

The project also enabled significant learning: gaining skills in Web design, real-time communication (WebSocket), cloud deployment and VR image processing, as well as cross-disciplinary engineering problem-solving. Project management, self-learning, and teamwork abilities were improved, providing valuable experience for future projects.

In summary, this project promotes 3D visualisation and remote collaboration of geophysical borehole data, enhances the immersion experience and teamwork, and provides new tools for industry data interpretation and decision-making. The platform enables cross-end adaptation, real-time annotation, and multi-video switching, improving the lack of interaction in traditional methods. Although there is still room for improvement in mobile and point cloud modelling, the scalability of the overall framework and industry applications has been proven. High-precision modelling and intelligent recognition will be introduced in the future to further expand the application and enhance the experience. This project lays the foundation for the promotion of geophysical visualisation tools and the building of a digital society, and also supports cross-industry innovative applications.

Reference List

[1] M. Holly, J. Pirker, S. Resch, S. Brettschuh, and C. Gütl, "Designing VR Experiences-Expectations for Teaching and Learning in VR," *Educational Technology & Society*, vol. 24, no. 2, pp. 107-119, 2021, doi: 10.30191/ETS.202104_24(2).0009.

[2] Q. Yuan *et al.*, "Exploring Healthcare Students' Intention to Use Virtual Reality Simulations in China: A Cross-sectional Study Applying the Technology Acceptance Model," *Computers, informatics, nursing*, vol. 43, no. 3, 2025, doi: 10.1097/CIN.0000000000001224.

[3] G. H. Alene *et al.*, "Virtual reality visualization of geophysical flows: A framework," *Environmental Modelling and Software*, Article vol. 177, 2024, Art no. 106063, doi: 10.1016/j.envsoft.2024.106063.

[4] D. Jiaqian. [6K panoramic VR video] 360° immersive scene in Iceland without filter. Available: https://www.bilibili.com/video/BV1nU4y1D7TR/.

[5] V. Pazzi, S. Morelli, and R. Fanti, "A Review of the Advantages and Limitations of Geophysical Investigations in Landslide Studies," *International Journal of Geophysics*, vol. 2019, no. 1, p. 2983087, 2019, doi: https://doi.org/10.1155/2019/2983087.

[6] H. K. H. Olierook *et al.*, "Bayesian geological and geophysical data fusion for the construction and uncertainty quantification of 3D geological models," *Geoscience Frontiers*, vol. 12, no. 1, pp. 479-493, 2021/01/01/ 2021, doi: https://doi.org/10.1016/j.gsf.2020.04.015.

[7] Y. Sermet and I. Demir, "GeospatialVR: A web-based virtual reality framework for collaborative environmental simulations," *Computers & geosciences*, vol. 159, p. 105010, 2022, doi: 10.1016/j.cageo.2021.105010.

[8] X. Jiang, "SUBMERGE: Subterranean Unity-Based Metaverse Experience for Geological Exploration " MSc dissertation, Dept. Electronic & Elec. Eng, University of Liverpool, 2024.

[9] K. N. Diego Marcos, Don McCurdy. "A-Frame – Make WebVR." https://aframe.io/ (accessed 22nd April 2025).

[10] P. Catanzariti, "A-Frame: The Easiest Way to Bring VR to the Web Today," in *sitepoint*, ed, 2024.

[11] C. H. Lu and X. H. Chen, "Improved iterative Poisson point cloud surface reconstruction," in *2023 3rd International Conference on Digital Society and Intelligent Systems (DSInS)*, 10-12 Nov. 2023, pp. 382-385, doi: 10.1109/DSInS60115.2023.10455616.

Appendix 1 Client-Side (Front-End) Source Code – index.html:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8" />
  <title>VR Video Player with Annotation</title>
  <script src="https://aframe.io/releases/0.6.0/aframe.min.js"></script>
  <script src="https://unpkg.com/aframe-look-at-component@0.6.0/dist/aframe-look-at-component.min.js"></script>
  <style>
    #link-confirm,
    #switch-video-button,
    #print-button,
    #rewind-button,
    #play-button,
    #pause-button,
    #forward-button,
    #mic-button {
      position: fixed;
      font-size: 1.5rem;
      width: 3em;
      height: 3em;
      background-color: #333;
      border-radius: 50%;
      z-index: 10;
      color: #fff;
      border: none;
    }

    #upload-input {
      display: none;
      position: absolute;
      width: 0;
      height: 0;
      opacity: 0;
      overflow: hidden;
    }

    #link-input {
      position: fixed;
      top: calc(50% + 2em);
      left: calc(50% - 10em);
      width: 20em;
      font-size: 1rem;
      padding: 0.5em;
      z-index: 10;
      display: none;
```

```
}

#link-confirm {
  top: calc(50% + 2em);
  left: calc(50% + 10em);
  display: none;
}

#switch-video-button {
  top: 10px;
  left: 320px;
}

#video-label {
  position: fixed;
  top: 26px;
  left: 400px;
  width: 3.5em;
  height: 2em;
  background-color: #333;
  color: #fff;
  border-radius: 10%;
  display: flex;
  justify-content: center;
  align-items: center;
  z-index: 10;
  font-size: 1rem;
  cursor: pointer;
  outline: none;
  border: 1px solid #666;
}

/*annotation list*/
#annotation-list-container {
  position: fixed;
  top: 10px;
  right: 10px;
  width: 350px;
  background-color: rgba(0, 0, 0, 0.7);
  color: white;
  border-radius: 5px;
  z-index: 100;
  overflow: hidden;
  max-height: 50vh;
  display: flex;
  flex-direction: column;
}
```

```css
#annotation-list-header {
  padding: 8px;
  background-color: rgba(30, 30, 30, 0.9);
  display: flex;
  justify-content: space-between;
  align-items: center;
  cursor: pointer;
}

.header-actions {
  display: flex;
  gap: 10px;
}

#annotation-list-toggle {
  background: none;
  border: none;
  color: white;
  font-size: 16px;
  cursor: pointer;
}

#annotation-list-content {
  overflow-y: auto;
  max-height: 300px;
  display: none;
}

#annotation-table {
  width: 100%;
  border-collapse: collapse;
}

#annotation-table th,
#annotation-table td {
  padding: 6px;
  text-align: left;
  border-bottom: 1px solid #444;
  font-size: 0.9rem;
  overflow: hidden;
  text-overflow: ellipsis;
  white-space: nowrap;
}

#annotation-table tr:nth-child(even) {
  background-color: rgba(50, 50, 50, 0.5);
}
```

```
#annotation-table tr:hover {
  background-color: rgba(60, 60, 60, 0.8);
  cursor: pointer;
}

#annotation-table th:first-child,
#annotation-table td:first-child {
  width: 20%;
  /* Time column */
}

#annotation-table th:last-child,
#annotation-table td:last-child {
  width: 80%;
  /* Text column */
}

#annotation-action-popup {
  position: absolute;
  background-color: rgba(40, 40, 40, 0.9);
  border-radius: 5px;
  padding: 5px;
  display: flex;
  gap: 10px;
  z-index: 101;
}

#annotation-action-popup button {
  background: none;
  border: none;
  color: white;
  font-size: 16px;
  width: 30px;
  height: 30px;
  border-radius: 50%;
  cursor: pointer;
}

#annotation-delete {
  background-color: #e74c3c;
}

#annotation-jump {
  background-color: #3498db;
}

.hidden {
  display: none !important;
```

```
}

#print-button {
  top: 10px;
  right: calc(30%);
}

/* Video List */
#video-list-container {
  position: fixed;
  top: 10px;
  left: 10px;
  width: 300px;
  background-color: rgba(0, 0, 0, 0.7);
  color: white;
  border-radius: 5px;
  z-index: 100;
}

#video-list-header {
  padding: 10px;
  display: flex;
  justify-content: space-between;
  align-items: center;
  background-color: rgba(30, 30, 30, 0.9);
  border-bottom: 1px solid #444;
  cursor: pointer;
  border-radius: 5px 5px 0 0;
}

#video-list-content {
  max-height: 300px;
  overflow-y: auto;
}

#video-table {
  width: 100%;
  border-collapse: collapse;
}

#video-table th,
#video-table td {
  padding: 8px;
  text-align: left;
  border-bottom: 1px solid #444;
}

#video-table tr:hover {
```

```
    background-color: rgba(60, 60, 60, 0.8);
    cursor: pointer;
  }

  .header-actions button {
    background: none;
    border: none;
    color: white;
    cursor: pointer;
    font-size: 1.2em;
    margin-left: 5px;
  }

  #video-action-popup {
    position: absolute;
    background-color: rgba(40, 40, 40, 0.9);
    border-radius: 5px;
    padding: 5px;
    display: flex;
    gap: 10px;
    z-index: 101;
  }

  #video-action-popup button {
    background: none;
    border: none;
    color: white;
    font-size: 16px;
    width: 30px;
    height: 30px;
    border-radius: 50%;
    cursor: pointer;
  }

  #video-rename {
    background-color: #3498db;
  }

  #video-delete {
    background-color: #e74c3c;
  }

  #video-select-vid1 {
    background-color: #2ecc71;
  }

  #video-select-vid2 {
    background-color: #f39c12;
```

```
}

.hidden {
  display: none !important;
}

.video-tag {
  display: inline-block;
  padding: 2px 5px;
  background-color: #4d4dff;
  border-radius: 3px;
  margin-right: 5px;
}

/* Bottom */
#rewind-button {
  bottom: 1em;
  left: calc(1%);
}

#pause-button {
  bottom: 1em;
  left: calc(1% + 3em);
  display: none;
}

#play-button {
  bottom: 1em;
  left: calc(1% + 3em);
}

#forward-button {
  bottom: 1em;
  left: calc(1% + 6em);
}

#video-controls {
  position: fixed;
  bottom: 50px;
  left: 50%;
  transform: translateX(-50%);
  z-index: 10;
  width: 60%;
}

#video-controls input[type="range"] {
  width: 100%;
}
```

```
#annotation-input {
  position: fixed;
  bottom: 1em;
  left: calc(50% - 10em);
  width: 80em;
  font-size: 1rem;
  padding: 0.5em;
  z-index: 10;
}

#mic-button {
  bottom: 1em;
  right: calc(6em);
}

#annotation-input {
  position: fixed;
  bottom: 1em;
  left: calc(50% - 10em);
  width: 20em;
  font-size: 1rem;
  padding: 0.5em;
  z-index: 10;
}

#annotation-container {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  pointer-events: none;
  z-index: 10;
}

.center-dot {
  position: fixed;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  width: 10px;
  height: 10px;
  background-color: white;
  border-radius: 50%;
  z-index: 10;
}
```

```css
@media screen and (max-width: 932px) {
  /* Mobile Device Size e.g iPhone 14 */

  #link-confirm,
  #switch-video-button,

  #print-button,

  #rewind-button,
  #play-button,
  #pause-button,
  #forward-button,
  #mic-button {
    width: 2.5em;
    height: 2.5em;
    font-size: 1rem;
  }

  #switch-video-button {
    top: 10px;
    left: 260px;
  }

  #print-button {
    top: 10px;
    right: calc(35%);
  }

  #video-list-container {
    width: 230px;
    max-height: 30vh;
  }

  #video-list-header {
    padding: 6px;
    font-size: 0.85rem;
    background-color: rgba(30, 30, 30, 0.9);
    border-bottom: 1px solid #444;
  }

  #video-list-header>span {
    font-size: 0.85rem;
  }

  .header-actions button {
    font-size: 1em;
    margin-left: 3px;
    background: none;
```

```
    border: none;
    color: white;
    cursor: pointer;
    width: 22px;
    height: 22px;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
  }

  #video-table th,
  #video-table td {
    padding: 4px;
    font-size: 0.8rem;
    text-align: left;
    border-bottom: 1px solid #444;
    overflow: hidden;
    text-overflow: ellipsis;
    white-space: nowrap;
  }

  #video-table tr:nth-child(even) {
    background-color: rgba(50, 50, 50, 0.5);
  }

  #video-label {
    position: fixed;
    top: 13px;
    left: 345px;
    width: 3.5em;
    height: 1.5em;
    background-color: #333;
    color: #fff;
    border-radius: 10%;
    display: flex;
    justify-content: center;
    align-items: center;
    z-index: 10;
    font-size: 1rem;
    cursor: pointer;
    outline: none;
    border: 1px solid #666;
  }

  #annotation-list-container {
    width: 230px;
    max-height: 30vh;
```

```
}

#annotation-table th,
#annotation-table td {
  padding: 4px;
  font-size: 0.8rem;
}

#link-input {
  position: fixed;
  top: calc(50% + 2em);
  left: calc(50% - 9em);
  width: 15em;
  font-size: 1rem;
  padding: 0.5em;
  z-index: 10;
  display: none;
}

#video-controls {
  position: fixed;
  bottom: 2.5em;
  left: 50%;
  transform: translateX(-50%);
  z-index: 10;
  width: 60%;
}

#annotation-input {
  position: fixed;
  bottom: 0.5em;
  left: calc(50% - 9em);
  width: 15em;
  font-size: 1rem;
  padding: 0.5em;
  z-index: 10;
}

#switch-video-button,
#print-button,
#rewind-button,
#play-button,
#pause-button,
#forward-button,
#mic-button {
  display: flex;
  justify-content: center;
  align-items: center;
```

```
        padding: 0;
        line-height: 1;
      }

      #switch-video-button span,
      #print-button span,
      #rewind-button span,
      #play-button span,
      #pause-button span,
      #forward-button span,
      #mic-button span {
        display: block;
        text-align: center;
      }
    }
  </style>
</head>

<body>
  <button id="play-button">▶️</button>
  <button id="link-confirm">✔</button>
  <button id="mic-button">🎤</button>
  <button id="rewind-button">⏪</button>
  <button id="pause-button">⏸️</button>
  <button id="forward-button">⏩</button>
  <button id="print-button">🖨️</button>
  <button id="switch-video-button">🔄</button>
  <div id="video-label">vid1</div>
  <input type="file" id="upload-input" accept="video/*" />
  <input type="text" id="link-input" placeholder="Type in video link and
enter" />
  <input type="text" id="annotation-input" placeholder="Type in annotation and
enter" />
  <div id="annotation-container"></div>
  <div class="center-dot"></div>

  <div id="annotation-list-container">
    <div id="annotation-list-header">
      <span>Annotations List</span>
      <div class="header-actions">
        <button id="annotation-list-toggle">▼</button>
      </div>
    </div>
    <div id="annotation-list-content">
      <table id="annotation-table">
        <thead>
          <tr>
            <th>Time</th>
```

```
            <th>Text</th>
          </tr>
        </thead>
        <tbody id="annotation-table-body">
          <!-- The annotation table will dynamaically generated here -->
        </tbody>
      </table>
    </div>
  </div>

  <div id="annotation-action-popup" class="hidden">
    <button id="annotation-delete">✕</button>
    <button id="annotation-jump">🚀</button>
  </div>

  <div id="video-list-container">
    <div id="video-list-header">
      <span>Videos List</span>
      <div class="header-actions">
        <button id="upload-button">⏫</button>
        <button id="link-button">🔗</button>
        <button id="video-list-toggle">▼</button>
      </div>
    </div>
    <div id="video-list-content">
      <table id="video-table">
        <thead>
          <tr>
            <th>Name</th>
            <th>Tag</th>
          </tr>
        </thead>
        <tbody id="video-table-body">
          <!-- The video list will be generated dynamically here -->
        </tbody>
      </table>
    </div>
  </div>

  <div id="video-action-popup" class="hidden">
    <button id="video-rename">🖉</button>
    <button id="video-delete">✕</button>
    <button id="video-select-vid1">1</button>
    <button id="video-select-vid2">2</button>
  </div>

  <div id="video-controls">
    <input type="range" id="timeline" value="0" max="100" />
```

```
  </div>

  <a-scene id="vr-scene" vr-mode-ui="enabled: true">
    <a-assets>
      <video id="vid1" playsinline crossorigin="anonymous" loop="true"
style="display: none"></video>
      <video id="vid2" playsinline crossorigin="anonymous" loop="true"
style="display: none"></video>
    </a-assets>

    <a-entity id="video-sphere" geometry="primitive: sphere; radius: 3000;
segmentsWidth: 64; segmentsHeight: 64;"
      material="shader: flat; src: #vid1;" scale="-1 1 1" visible="true"></a-
entity>

    <a-entity id="camera" camera look-controls position="0 1.6 0">
      <a-cursor fuse="true" fuse-timeout="2000"></a-cursor>
    </a-entity>
  </a-scene>

  <script>
    var vid1 = document.getElementById("vid1");
    var vid2 = document.getElementById("vid2");
    var currentVideoIndex = 1;
    const videos = {
      vid1: document.getElementById('vid1'),
      vid2: document.getElementById('vid2')
    };
    const videoSphere = document.getElementById('video-sphere');

    var uploadButton = document.getElementById("upload-button");
    var uploadInput = document.getElementById("upload-input");
    var linkButton = document.getElementById("link-button");
    var linkInput = document.getElementById("link-input");
    var linkConfirm = document.getElementById("link-confirm");
    var switchVideoButton = document.getElementById("switch-video-button");
    var videoLabel = document.getElementById('video-label');
    var printButton = document.getElementById("print-button");

    var playButton = document.getElementById("play-button");
    var pauseButton = document.getElementById("pause-button");
    var rewindButton = document.getElementById("rewind-button");
    var forwardButton = document.getElementById("forward-button");
    var timeline = document.getElementById("timeline");
    var annotationInput = document.getElementById("annotation-input");
    var micButton = document.getElementById("mic-button");

    var scene = document.getElementById("vr-scene");
```

```
    var currentVideoLink = "";
    var currentInput = "";
    var annotationList = [];
    var lastTouchTime = 0;
    var touchCount = 0;
    var recognitionActive = false;

    // Set the speed at which the video moves (assuming units of metres per
second)
    var videoSpeed = 1.0; // Adjusts to the actual movement speed of the video

    // Connecting to a WebSocket Server
    //var ws = new WebSocket("wss://localhost:8080"); //for local testing
    var ws;
    var wsReconnectTimer;

    function connectWebSocket() {
      ws = new WebSocket("wss://vrdanmaku.uk:8080");

      ws.onopen = function () {
        console.log("Connected to WebSocket server");
        clearTimeout(wsReconnectTimer);

        // Request a video list after successful connection
        ws.send(JSON.stringify({
          type: 'request_video_list'
        }));
      };

      ws.onclose = function () {
        console.log("Disconnected from WebSocket server, trying to
reconnect...");
        wsReconnectTimer = setTimeout(connectWebSocket, 2000);
      };

      ws.onerror = function (err) {
        console.error("WebSocket error:", err);
      };

      ws.onmessage = function (event) {
        try {
          var data = JSON.parse(event.data);
          if (data.type === "existing_annotation") {
            data.data.forEach((annotationMessage) => {
              addAnnotation(
                annotationMessage.text,
                annotationMessage.position,
                annotationMessage.time
```

```
            );
          });
        } else if (data.type === "new_annotation") {
          addAnnotation(data.data.text, data.data.position, data.data.time);
        }
      } catch (e) {
        console.error("Error parsing message", e);
      }
    };
  }

  // Initial connection
  connectWebSocket();

  function requestDeviceMotionPermission() {
    if (typeof DeviceMotionEvent.requestPermission === "function") {
      DeviceMotionEvent.requestPermission()
        .then((permissionState) => {
          if (permissionState === "granted") {
            console.log("DeviceMotion permission granted.");
          } else {
            alert("DeviceMotion permission denied.");
          }
        })
        .catch(console.error);
    }
  }

  playButton.addEventListener("click", function (e) {
    requestDeviceMotionPermission();
    const currentVid = currentVideoIndex === 1 ? vid1 : vid2;
    if (!currentVid.src || currentVid.src === window.location.href) {
      alert("Please upload the video or a link to the video first!");
      return;
    }
    this.style.display = "none"; pauseButton.style.display = "block";

    // exitButton.style.display = "block";

    currentVid.play();
    startAnnotationMovement(); // Starting annotation movement
  },
    false
  );

  annotationInput.addEventListener("keypress", function (e) {
    if (e.key === "Enter") {
```

```
        var text = formatAnnotationText(annotationInput.value); // Formatting
annotation text
        var camera = document.getElementById("camera"); // Get the a-camera
element
        var position = getWorldPosition(camera);
        const currentVid = currentVideoIndex === 1 ? vid1 : vid2;

        annotationInput.value = "";
        // Sending a annotation to a WebSocket server
        ws.send(
          JSON.stringify({
            type: "new_annotation",
            text: text,
            position: position,
            videoLink: currentVideoLink,
            time: currentVid.currentTime,
          })
        );
      }
    });

    // Video synchronization controller
    class VideoSyncController {
      constructor(videos) {
        this.videos = videos;
        this.timeline = timeline;
        this.currentVideoIndex = currentVideoIndex;
        this.currentTime = 0;
        this.isPaused = true;
        this.setupSync();
        this.setupTimelineControl();
      }

      setupSync() {
        // sychronize play status
        Object.values(this.videos).forEach(video => {
          video.addEventListener('play', () => this.syncPlay());
          video.addEventListener('pause', () => this.syncPause());
          video.addEventListener('timeupdate', () => this.syncTime(video));

          // synchronize video duration
          video.addEventListener('loadedmetadata', () => {
            console.log(`Video ${video.id} duration:`, video.duration);
            // if current video loaded, update timeline
            if (video === this.getCurrentVideo()) {
              this.updateTimeline();
            }
          });
```

```
        });
      }

      //Set timeline control
      setupTimelineControl() {
        // timeline control
        this.timeline.addEventListener("input", () => {
          const time = (this.timeline.value * this.getCurrentVideo().duration)
/ 100;
          this.currentTime = time;

          // synchronize time for all videos
          Object.values(this.videos).forEach(video => {
            video.currentTime = time;
          });

          // if annotation need update the position
          if (typeof updateAnnotationPositions === 'function') {
            updateAnnotationPositions();
          }
        });
      }

      getCurrentVideo() {
        return this.videos[`vid${this.currentVideoIndex}`];
      }

      setCurrentVideoIndex(index) {
        this.currentVideoIndex = index;
        this.updateTimeline();
      }

      syncPlay() {
        if (this.isPaused) {
          Object.values(this.videos).forEach(v => v.play());
          this.isPaused = false;
        }
      }

      syncPause() {
        if (!this.isPaused) {
          Object.values(this.videos).forEach(v => v.pause());
          this.isPaused = true;
        }
      }

      syncTime(sourceVideo) {
        // synchronize time
```

```
        const newTime = sourceVideo.currentTime;
        if (Math.abs(this.currentTime - newTime) > 0.1) {
          this.currentTime = newTime;
          Object.values(this.videos).forEach(v => {
            if (v !== sourceVideo && Math.abs(v.currentTime - newTime) > 0.1)
{
              v.currentTime = newTime;
            }
          });

          // update timeline
          this.updateTimeline();
        }
      }

      // Update the timeline based on the current video's time
      updateTimeline() {
        const currentVid = this.getCurrentVideo();

        // Make sure the video is loaded and has a valid duration
        if (currentVid && currentVid.duration && !isNaN(currentVid.duration)
&& currentVid.duration > 0) {
          // Calculate the percentage of progress (ensure that it is in the
range 0-100)
          const percentage = Math.min(100, Math.max(0, (100 /
currentVid.duration) * this.currentTime));

          // update the timeline value
          this.timeline.value = percentage;

        }
      }
    }

    // Video synchronization controller instance
    const syncController = new VideoSyncController(videos);

    // video load event
    function handleVideoLoad(videoId, url) {
      // Handle relative paths to ensure the use of complete urls
      if (url.startsWith('/videos/')) {
        url = 'https://vrdanmaku.uk:8080' + url;
      }

      // Set up video source
      if (videoId === 'vid1') {
        vid1.src = url;
        vid1.load();
```

```
      currentVideoIndex = 1;
    } else {
      vid2.src = url;
      vid2.load();
      currentVideoIndex = 2;
    }

    currentVideoLink = url;
    playButton.style.display = 'block';

    // Send video loading event to the server
    ws.send(JSON.stringify({
      type: 'new_video',
      videoLink: url,
      videoIndex: videoId === 'vid1' ? 1 : 2
    }));

    updateVideoTable();
  }

  // Handling touch screen events
  document.body.addEventListener("touchstart", handleTouchStart, false);

  function handleTouchStart(e) {
    if (
      e.target.id === "mic-button" ||
      e.target.id === "rewind-button" ||
      e.target.id === "pause-button" ||
      e.target.id === "forward-button" ||
      e.target.id === "print-button"
    )
      return; // If the microphone button is clicked, skip it

    var currentTime = new Date().getTime();
    if (currentTime - lastTouchTime < 800) {
      // Modify the time window to accommodate quadruple hits
      touchCount++;
    } else {
      touchCount = 1;
    }
    lastTouchTime = currentTime;

    setTimeout(() => {
      const currentVid = currentVideoIndex === 1 ? vid1 : vid2;
      if (touchCount === 1) {
        // do nothing
      }
      touchCount = 0; // Reset Touch Count
```

```
      }, 1000); // Delayed processing of combos to avoid false triggers
    }

    function togglePlayPause() {
      const currentVid = currentVideoIndex === 1 ? vid1 : vid2;
      if (currentVid.paused) {
        currentVid.play();
      } else {
        currentVid.pause();
      }
      updateAnnotationPositions(); // Updates annotation positions during
pause or playback
    }

    function toggleMic() {
      if (!recognitionActive) {
        if (recognition) {
          recognition.start();
          recognitionActive = true;
          micButton.style.backgroundColor = "green"; // Turns green when
recording
        }
      } else {
        if (recognition) {
          recognition.stop();
          recognitionActive = false;
          micButton.style.backgroundColor = "red"; // Turns red when recording
stops
        }
      }
    }

    function getWorldPosition(camera) {
      var direction = new THREE.Vector3();
      camera.object3D.getWorldDirection(direction);
      direction.multiplyScalar(-2); // Set distance
      var position = camera.object3D.position.clone().add(direction);
      return { x: position.x, y: position.y, z: position.z };
    }

    function addAnnotation(text, position, time) {
      // Check if this annotation already exists in the list
      const alreadyExists = annotationList.some(anno =>
        anno.text === text &&
        anno.time === time
      );

      if (!alreadyExists) {
```

```
      // Add to local list
      annotationList.push({
        text: text,
        position: position,
        time: time,
        shown: false
      });

      // update the annotation table if it is displayed
      if (annotationListContent.style.display === 'block') {
        updateAnnotationTable();
      }

      // get the current video index
      const currentVid = currentVideoIndex === 1 ? vid1 : vid2;

      // send annotation to the WebSocket server
      ws.send(
        JSON.stringify({
          type: "new_annotation",
          text: text,
          position: position,
          videoLink: currentVideoLink,
          time: currentVid.currentTime,
        })
      );
    }
  }

  function updateAnnotationPositions() {
    var currentTime =
      currentVideoIndex === 1 ? vid1.currentTime : vid2.currentTime;

    annotationList.forEach(function (annotation) {
      if (!annotation.shown && currentTime >= annotation.time) {
        // creat the annotation background
        var backgroundEntity = document.createElement('a-plane');
        backgroundEntity.setAttribute('color', 'black');
        backgroundEntity.setAttribute('opacity', 0.5);
        backgroundEntity.setAttribute('width', 1.2);
        backgroundEntity.setAttribute('height', 0.3);
        backgroundEntity.setAttribute('position', annotation.position);
        backgroundEntity.setAttribute('look-at', '[camera]');
        scene.appendChild(backgroundEntity);

        annotation.backgroundEntity = backgroundEntity;

        // creat the annotation text
```

```
          var annotationEntity = document.createElement('a-entity');
          annotationEntity.setAttribute('text', {
            value: annotation.text,
            color: 'white',
            align: 'center',
            width: 1,
            wrapCount: 20
          });
          annotationEntity.setAttribute('position', annotation.position);
          annotationEntity.setAttribute('look-at', '[camera]');
          scene.appendChild(annotationEntity);

          annotation.entity = annotationEntity;
          annotation.shown = true;
        }

        if (annotation.shown) {
          var elapsedTime = currentTime - annotation.time;
          var distanceTraveled = elapsedTime * videoSpeed;

          var newPosition = {
            x: annotation.position.x + distanceTraveled,
            y: annotation.position.y,
            z: annotation.position.z
          };

          // update annotation position
          annotation.entity.setAttribute('position', newPosition);

          // update background position
          if (annotation.backgroundEntity) {
            annotation.backgroundEntity.setAttribute('position', newPosition);
            //annotation look at the camera
            annotation.backgroundEntity.setAttribute('look-at', '[camera]');
          }
        }
      });
    }

    function startAnnotationMovement() {
      function move() {
        const currentVid = currentVideoIndex === 1 ? vid1 : vid2;
        if (!currentVid.paused) {
          updateAnnotationPositions();
        }
        requestAnimationFrame(move); // Loop call to make annotation move
during playback
      }
```

```
      move(); // Initiate annotation movement
    }

    // Format annotation text with line breaks every three words
    function formatAnnotationText(text) {
      var words = text.split(" ");
      for (var i = 3; i < words.length; i += 4) {
        words[i] = words[i] + "\n";
      }
      return words.join(" ");
    }

    // Handling Web Speech API Speech Recognition
    var recognition;
    if ("webkitSpeechRecognition" in window) {
      recognition = new webkitSpeechRecognition();
      recognition.continuous = false;
      recognition.interimResults = false;
      recognition.lang = "en-US";

      recognition.onresult = function (event) {
        var transcript = formatAnnotationText(event.results[0][0].transcript);
// Formatting Speech Recognition Text
        var camera = document.getElementById("camera");
        var position = getWorldPosition(camera);
        const currentVid = currentVideoIndex === 1 ? vid1 : vid2;
        addAnnotation(transcript, position, currentVid.currentTime);
        ws.send(
          JSON.stringify({
            type: "new_annotation",
            text: text,
            position: position,
            videoLink: currentVideoLink,
            time: currentVid.currentTime,
          })
        );
      };

      recognition.onerror = function (event) {
        console.error("Speech recognition error", event);
      };
    }

    // Controlling the behaviour of the microphone button
    micButton.addEventListener("click", function () {
      toggleMic();
    });
```

```
    // Print annotation message
    printButton.addEventListener("click", function () {
      var output = "Annotation Information:\n\n";
      annotationList.forEach(function (annotation) {
        output += `Text: ${annotation.text}, Position:
(${annotation.position.x}, ${annotation.position.y},
${annotation.position.z}), Time: ${annotation.time}s\n`;
      });
      var blob = new Blob([output], { type: "text/plain;charset=utf-8" });
      var link = document.createElement("a");
      link.href = URL.createObjectURL(blob);
      link.download = "annotation_info.txt";
      document.body.appendChild(link);
      link.click();
      document.body.removeChild(link);
    });

    // Bind button function
    rewindButton.addEventListener("click", function () {
      const currentVid = currentVideoIndex === 1 ? vid1 : vid2;
      currentVid.currentTime -= 0.2;
      updateAnnotationPositions();
    });

    pauseButton.addEventListener("click", function () {
      togglePlayPause();
      this.style.display = 'none';
      playButton.style.display = 'block';
    });

    forwardButton.addEventListener("click", function () {
      const currentVid = currentVideoIndex === 1 ? vid1 : vid2;
      currentVid.currentTime += 0.2;
      updateAnnotationPositions();
    });

    switchVideoButton.addEventListener("click", function () {
      currentVideoIndex = currentVideoIndex === 1 ? 2 : 1;

      // Update the video label
      videoLabel.textContent = `vid${currentVideoIndex}`;

      // Switch the video source
      videoSphere.setAttribute('material', 'src', `#vid${currentVideoIndex}`);

      // Update the annotation position
      updateAnnotationPositions();
    });
```

```
    // Display video label
    videoLabel.addEventListener('blur', function () {
      if (this.textContent.trim() === '') {
        this.textContent = `vid${currentVideoIndex}`;
      }
    });

    // Annotation List functionality
    const annotationListContainer = document.getElementById('annotation-list-
container');
    const annotationListHeader = document.getElementById('annotation-list-
header');
    const annotationListToggle = document.getElementById('annotation-list-
toggle');
    const annotationListContent = document.getElementById('annotation-list-
content');
    const annotationTableBody = document.getElementById('annotation-table-
body');
    const annotationActionPopup = document.getElementById('annotation-action-
popup');
    const annotationDeleteBtn = document.getElementById('annotation-delete');
    const annotationJumpBtn = document.getElementById('annotation-jump');

    let selectedAnnotationIndex = -1;

    // switch the display of annotation list
    annotationListHeader.addEventListener('click', function (e) {
      if (annotationListContent.style.display === 'block') {
        annotationListContent.style.display = 'none';
        annotationListToggle.textContent = '▼';
      } else {
        annotationListContent.style.display = 'block';
        annotationListToggle.textContent = '▲';
        updateAnnotationTable(); // update the annotation table
      }
    });

    // update the annotation table
    function updateAnnotationTable() {
      annotationTableBody.innerHTML = '';
      annotationList.forEach((annotation, index) => {
        const row = document.createElement('tr');
        row.dataset.index = index;

        // format the time
        const timeFormatted = formatTime(annotation.time);
```

```
        row.innerHTML = `
      <td>${timeFormatted}</td>
      <td>${annotation.text.replace(/\n/g, ' ')}</td>
    `;

        row.addEventListener('click', function (e) {
          selectedAnnotationIndex = parseInt(this.dataset.index);
          showAnnotationActions(e.clientX, e.clientY);
        });

        annotationTableBody.appendChild(row);
      });
    }

    // format time to MM:SS
    function formatTime(seconds) {
      const mins = Math.floor(seconds / 60);
      const secs = Math.floor(seconds % 60);
      return `${String(mins).padStart(2, '0')}:${String(secs).padStart(2,
'0')}`;
    }

    // show the annotation actions popup
    function showAnnotationActions(x, y) {
      annotationActionPopup.classList.remove('hidden');
      annotationActionPopup.style.left = `${x - 40}px`;
      annotationActionPopup.style.top = `${y}px`;

      // close the popup when clicking outside
      document.addEventListener('click',
closeAnnotationActionsOnOutsideClick);
    }

    // close the annotation actions popup
    function closeAnnotationActionsOnOutsideClick(e) {
      if (!annotationActionPopup.contains(e.target)
&& !e.target.closest('#annotation-table tr')) {
        annotationActionPopup.classList.add('hidden');
        document.removeEventListener('click',
closeAnnotationActionsOnOutsideClick);
      }
    }

    // delete the selected annotation
    annotationDeleteBtn.addEventListener('click', function () {
      if (selectedAnnotationIndex >= 0 && selectedAnnotationIndex <
annotationList.length) {
        const annotation = annotationList[selectedAnnotationIndex];
```

```
        console.log("Deleting annotation:", annotation);

        // send delete annotation message to the server
        const deleteMessage = {
          type: "delete_annotation",
          time: annotation.time,
          text: annotation.text,
        };
        console.log("Sending delete message:", deleteMessage);
        ws.send(JSON.stringify(deleteMessage));

        // remove the annotation entity from the scene
        if (annotation.entity) {
          scene.removeChild(annotation.entity);
        }
        if (annotation.backgroundEntity) {
          scene.removeChild(annotation.backgroundEntity);
        }

        // remove the annotation from the list
        annotationList.splice(selectedAnnotationIndex, 1);
        updateAnnotationTable();

        // close the action popup
        annotationActionPopup.classList.add('hidden');
      }
    });

    // jump to the selected annotation
    annotationJumpBtn.addEventListener('click', function () {
      if (selectedAnnotationIndex >= 0 && selectedAnnotationIndex <
annotationList.length) {
        const annotation = annotationList[selectedAnnotationIndex];
        const currentVid = currentVideoIndex === 1 ? vid1 : vid2;

        // jump to the annotation time
        currentVid.currentTime = annotation.time;

        // update the timeline
        updateAnnotationPositions();

        // close the action popup
        annotationActionPopup.classList.add('hidden');
      }
    });

    // Video List functionality
```

```
    const videoListContainer = document.getElementById('video-list-
container');
    const videoListHeader = document.getElementById('video-list-header');
    const videoListToggle = document.getElementById('video-list-toggle');
    const videoListContent = document.getElementById('video-list-content');
    const videoTableBody = document.getElementById('video-table-body');
    const videoActionPopup = document.getElementById('video-action-popup');
    const videoRenameBtn = document.getElementById('video-rename');
    const videoDeleteBtn = document.getElementById('video-delete');
    const videoSelectVid1Btn = document.getElementById('video-select-vid1');
    const videoSelectVid2Btn = document.getElementById('video-select-vid2');
    const videoListUploadBtn = document.getElementById('video-list-upload');
    const videoListLinkBtn = document.getElementById('video-list-link');

    let videoList = [];
    let selectedVideoId = null;

    // Switch video list display
    videoListHeader.addEventListener('click', function (e) {
      if (e.target.tagName === 'BUTTON') return; // Ignore button click

      if (videoListContent.style.display === 'block') {
        videoListContent.style.display = 'none';
        videoListToggle.textContent = '▼';
      } else {
        videoListContent.style.display = 'block';
        videoListToggle.textContent = '▲';
        updateVideoTable();
      }
    });

    // Display video operation pop-up menu
    function showVideoActions(x, y, videoId) {
      selectedVideoId = videoId;
      videoActionPopup.style.left = x + 'px';
      videoActionPopup.style.top = y + 'px';
      videoActionPopup.classList.remove('hidden');
    }

    // Hide video operation pop-up menu
    document.addEventListener('click', function (e) {
      if (!videoActionPopup.contains(e.target) &&
        !e.target.closest('#video-table tr')) {
        videoActionPopup.classList.add('hidden');
      }
    });

    // Update video table
```

```
function updateVideoTable() {
  videoTableBody.innerHTML = '';
  videoList.forEach(video => {
    const row = document.createElement('tr');
    row.dataset.id = video.id;

    // Create tag elements
    let tagHtml = '';
    if (video.url === videos.vid1.src) {
      tagHtml += '<span class="video-tag">vid1</span>';
    }
    if (video.url === videos.vid2.src) {
      tagHtml += '<span class="video-tag">vid2</span>';
    }

    row.innerHTML = `
  <td>${video.name}</td>
  <td>${tagHtml}</td>
  <td>:</td>
`;

    row.addEventListener('click', function (e) {
      showVideoActions(e.clientX, e.clientY, video.id);
    });

    videoTableBody.appendChild(row);
  });
}

// Rename the video
videoRenameBtn.addEventListener('click', function () {
  const video = videoList.find(v => v.id === selectedVideoId);
  if (video) {
    const newName = prompt('Enter new name:', video.name);
    if (newName && newName.trim() !== '') {
      ws.send(JSON.stringify({
        type: 'update_video_name',
        videoId: selectedVideoId,
        newName: newName.trim()
      }));
    }
  }
  videoActionPopup.classList.add('hidden');
});

// Delete the video
videoDeleteBtn.addEventListener('click', function () {
  if (confirm('Are you sure you want to delete this video?')) {
```

```
      ws.send(JSON.stringify({
        type: 'delete_video',
        videoId: selectedVideoId
      }));
    }
    videoActionPopup.classList.add('hidden');
  });

  // Choose the video as vid1
  videoSelectVid1Btn.addEventListener('click', function () {
    const video = videoList.find(v => v.id === selectedVideoId);
    if (video) {
      handleVideoLoad('vid1', video.url);
      videoActionPopup.classList.add('hidden');
    }
  });

  // Choose the video as vid2
  videoSelectVid2Btn.addEventListener('click', function () {
    const video = videoList.find(v => v.id === selectedVideoId);
    if (video) {
      handleVideoLoad('vid2', video.url);
      videoActionPopup.classList.add('hidden');
    }
  });

  uploadButton.addEventListener("click", function (e) {
    uploadInput.click();
  },
    false
  );

  uploadInput.addEventListener("change", function (e) {
    var file = e.target.files[0];
    if (!file) return;

    // Create a FormData object to send the file
    const formData = new FormData();
    formData.append('video', file);
    formData.append('name', file.name);

    // Send upload request -with detailed debugging information
    fetch('https://vrdanmaku.uk:8080/upload', {
      method: 'POST',
      body: formData,
      signal: AbortSignal.timeout(60000), // 60s timeout
      credentials: 'include', //Cross-domain credential support
      mode: 'cors' // Clearly set cross-domain mode
```

```
    })
      .then(response => {
        console.log("Response status:", response.status);
        if (!response.ok) {
          throw new Error(`HTTP error: ${response.status}`);
        }
        return response.text().then(text => {
          try {
            return JSON.parse(text);
          } catch (e) {
            console.error("JSON parse error:", e, "Response was:", text);
            throw new Error("Server response was not valid JSON");
          }
        });
      })
      .then(data => {
        console.log("Upload success:", data);
        // Request the video list to ensure UI is updated
        if (ws.readyState === WebSocket.OPEN) {
          ws.send(JSON.stringify({ type: 'request_video_list' }));
        }
        if (data.success) {
          handleVideoLoad('vid1', data.video.url);
        }
      })
      .catch(error => {
        console.error('Upload error:', error);
        alert('Upload failed: ' + error.message);
      });

    this.value = ''; // Reset the input box
  }, false);

  linkButton.addEventListener("click", function (e) {
    linkInput.style.display = "block";
    linkConfirm.style.display = "block";
    linkInput.focus();
  },
    false
  );

  linkInput.addEventListener("keypress", function (e) {
    if (e.key === 'Enter') {
      const url = linkInput.value;
      handleVideoLoad(`vid${currentVideoIndex}`, url);
    }
  });
```

```
  linkConfirm.addEventListener(
    "click",
    function (e) {
      var url = linkInput.value;
      if (currentVideoIndex === 1) {
        vid1.src = url;
        vid1.load();
      } else {
        vid2.src = url;
        vid2.load();
      }
      currentVideoLink = url;
      playButton.style.display = "block";
      linkInput.style.display = "none";
      linkConfirm.style.display = "none";

      ws.send(
        JSON.stringify({
          type: "new_video",
          videoLink: url,
          videoIndex: currentVideoIndex,
        })
      );
    },
    false
  );

  // Process video loading
  function handleVideoLoad(videoId, url) {
    // Set up video source
    if (videoId === 'vid1') {
      vid1.src = url;
      vid1.load();
      currentVideoIndex = 1;
    } else {
      vid2.src = url;
      vid2.load();
      currentVideoIndex = 2;
    }

    currentVideoLink = url;
    playButton.style.display = 'block';

    // Send video loading event to the server
    ws.send(JSON.stringify({
      type: 'new_video',
      videoLink: url,
      videoIndex: videoId === 'vid1' ? 1 : 2
```

```
      }));

      updateVideoTable();
    }

    // WebSocket processing
    ws.onmessage = function (event) {
      try {
        var data = JSON.parse(event.data);
        if (data.type === "existing_annotation") {
          annotationList = []; // Clear existing annotations before adding new
ones
          // Add annotations sent by the server
          data.data.forEach((annotationMessage) => {
            // Add to local list only
            annotationList.push({
              text: annotationMessage.text,
              position: annotationMessage.position,
              time: annotationMessage.time,
              shown: false
            });
          });
          updateAnnotationTable();
        } else if (data.type === "new_annotation") {
          // New annotations received from the server are added to the local
list only
          annotationList.push({
            text: data.data.text,
            position: data.data.position,
            time: data.data.time,
            shown: false
          });
          updateAnnotationTable();
        } else if (data.type === "annotation_deleted") {
          annotationList.splice(data.data.index, 1); // Delete items in the
local annotation list
          updateAnnotationTable();
          updateAnnotationPositions();
        } else if (data.type === "video_list") {
          videoList = data.data;
          updateVideoTable();
        } else if (data.type === "video_loaded") {
          // Handle videos loaded by other clients
          const videoId = data.videoIndex === 1 ? 'vid1' : 'vid2';

          // Automatically load only if the video is not currently played
          if (!vid1.src && !vid2.src) {
            handleVideoLoad(videoId, data.videoLink);
```

```
          }
        }
      } catch (e) {
        console.error("Error parsing message", e);
      }
    };

    // Request a list of videos when the page loads
    window.addEventListener('load', function () {
      ws.send(JSON.stringify({
        type: 'request_video_list'
      }));
    });

  </script>
</body>

</html>
```

Appendix 2 Server-Side (Back-End) Source Code – server.js:

```
const fs = require("fs");
const https = require("https");
const WebSocket = require("ws");
const path = require("path");
const url = require("url");
const formidable = require("formidable");// need to install formidable package

// Certificates obtained using Let's Encrypt
const server = https.createServer({
  cert: fs.readFileSync("/etc/letsencrypt/live/vrdanmaku.uk/fullchain.pem"),
  key: fs.readFileSync("/etc/letsencrypt/live/vrdanmaku.uk/privkey.pem"),
});

const wss = new WebSocket.Server({ server });

// Video storage directory
const VIDEOS_DIR = path.join(__dirname, "videos");
// Make sure the video directory exists
if (!fs.existsSync(VIDEOS_DIR)) {
  fs.mkdirSync(VIDEOS_DIR);
}
// Files that store video information
const VIDEO_INFO_FILE = path.join(__dirname, "video_info.json");
// If the video information file does not exist, create an empty one
if (!fs.existsSync(VIDEO_INFO_FILE)) {
  fs.writeFileSync(VIDEO_INFO_FILE, JSON.stringify({ videos: [] }), "utf-8");
}

// Load video information
let videoInfo = JSON.parse(fs.readFileSync(VIDEO_INFO_FILE, "utf-8"));

// Http request processing
server.on("request", (req, res) => {
  const parsedUrl = url.parse(req.url, true);
  res.setHeader('Access-Control-Allow-Origin', 'https://vrdanmaku.uk');
  res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT,
DELETE');
  res.setHeader('Access-Control-Allow-Headers', 'Content-Type,
Authorization');
  res.setHeader('Access-Control-Allow-Credentials', 'true');

  // Handle options preflight requests
  if (req.method === 'OPTIONS') {
    res.writeHead(200);
    res.end();
    return;
  }
```

```
  // Process video upload
  if (req.method === "POST" && parsedUrl.pathname === "/upload") {
    const form = new formidable.IncomingForm({
      uploadDir: VIDEOS_DIR,
      keepExtensions: true,
      maxFileSize: 2000 * 1024 * 1024, // Set video limitation 2GB
      multiples: true
    });

    form.parse(req, (err, fields, files) => {
      if (err) {
        console.error("Detail upload error:", err);
        // The error message is correctly sent back to the client
        res.writeHead(500, { "Content-Type": "application/json" });
        res.end(JSON.stringify({ success: false, message: err.message }));
        return;
      }

      try {
        // Make sure files.video exists and is an array
        if (!files.video || !Array.isArray(files.video) || files.video.length
=== 0) {
          console.error("No video file found in request");
          res.writeHead(400, { "Content-Type": "text/plain" });
          res.end("No video file in request");
          return;
        }

        const file = files.video[0];
        console.log("Uploaded file:", file); // Debugging information

        // Use the correct file name
        const newFileName = file.originalFilename ||
`video_${Date.now()}${path.extname(file.filepath)}`;
        const newPath = path.join(VIDEOS_DIR, newFileName);

        // Correctly rename uploaded files
        fs.renameSync(file.filepath, newPath);
        console.log(`File renamed from ${file.filepath} to ${newPath}`);

        // Add video information
        const videoId = Date.now().toString();
        const videoUrl = `https://vrdanmaku.uk/videos/${newFileName}`;

        const newVideo = {
          id: videoId,
          name: fields.name?.[0] || newFileName, // formidable 3.x
```

```
          url: videoUrl,
          tags: fields.tags?.[0] || "",
          uploadDate: new Date().toISOString()
        };

        videoInfo.videos.push(newVideo);
        fs.writeFileSync(VIDEO_INFO_FILE, JSON.stringify(videoInfo), "utf-8");

        broadcastVideoList();

        console.log("Upload success, sending response:", { success: true,
video: newVideo });
        res.writeHead(200, { "Content-Type": "application/json" });
        res.end(JSON.stringify({ success: true, video: newVideo }));
      } catch (error) {
        console.error("Error processing upload:", error);
        res.writeHead(500, { "Content-Type": "application/json" });
        res.end(JSON.stringify({ success: false, message: error.message }));
      }
    });
    return;
  }

  // Get a video list
  if (req.method === "GET" && parsedUrl.pathname === "/videos") {
    res.writeHead(200, { "Content-Type": "application/json" });
    res.end(JSON.stringify(videoInfo));
    return;
  }
});

// Broadcast video list to all clients
function broadcastVideoList() {
  wss.clients.forEach((client) => {
    if (client.readyState === WebSocket.OPEN) {
      client.send(
        JSON.stringify({ type: "video_list", data: videoInfo.videos })
      );
    }
  });
}

const GLOBAL_ANNOTATION_FILE = "global_annotations.json";   // Global
annotation file

// Load annotation library
function loadAnnotationLibrary() {
  const filepath = path.join(__dirname, GLOBAL_ANNOTATION_FILE);
```

```
  if (fs.existsSync(filepath)) {
    return JSON.parse(fs.readFileSync(filepath, "utf-8"));
  }
  return [];
}

// Save annotation library
function saveAnnotationLibrary(library) {
  const filepath = path.join(__dirname, GLOBAL_ANNOTATION_FILE);
  try {
    fs.writeFileSync(filepath, JSON.stringify(library, null, 2), "utf-8");
    console.log("Annotation library saved successfully");
  } catch (error) {
    console.error("Error saving annotation library:", error);
  }
}

wss.on("connection", (ws) => {
  console.log("New client connected");
  // Send the video list to the new client
  ws.send(
    JSON.stringify({
      type: "video_list",
      data: videoInfo.videos
    })
  );

  // Send existing annotations to the new client
  const library = loadAnnotationLibrary();
  console.log("Sending existing annotations to new client");
  ws.send(JSON.stringify({
    type: "existing_annotation",
    data: library
  }));

  ws.on("message", (message) => {
    const parsedMessage = JSON.parse(message);
    console.log("Received message type:", parsedMessage.type);
    if (parsedMessage.type === "new_video") {
      const library = loadAnnotationLibrary();
      // Send existing annotation to the client
      ws.send(JSON.stringify({ type: "existing_annotation", data: library }));
    }
    else if (parsedMessage.type === "new_annotation") {
      const annotationMessage = {
        text: parsedMessage.text,
        time: parsedMessage.time,
        position: parsedMessage.position,
```

```
        videoLink: parsedMessage.videoLink,
      };

      // Save the annotation message to the library
      const library = loadAnnotationLibrary();
      // Check if the annotation already exists
      const exists = library.some(anno =>
        anno.text === annotationMessage.text &&
        anno.time === annotationMessage.time &&
        anno.videoLink === annotationMessage.videoLink
      );
      // Add and broadcast only if it does not exist
      if (!exists) {
        library.push(annotationMessage);
        saveAnnotationLibrary(library);

        // Broadcast annotation messages to all clients
        wss.clients.forEach((client) => {
          if (client.readyState === WebSocket.OPEN) {
            client.send(
              JSON.stringify({ type: "new_annotation", data:
annotationMessage })
            );
          }
        });
      }
    }
    else if (parsedMessage.type === "delete_annotation") {
      // Load existing annotations
      const library = loadAnnotationLibrary();
      console.log("Received delete request:", parsedMessage);
      console.log("Current library before deletion:", library);

      // Find and delete the specified annotation
      const EPSILON = 0.001; //Permitted time error range
      const annotationIndex = library.findIndex(a =>
        Math.abs(a.time - parsedMessage.time) < EPSILON &&
        a.text === parsedMessage.text
      );

      console.log("Found annotation at index:", annotationIndex);

      if (annotationIndex !== -1) {
        library.splice(annotationIndex, 1);//Remove annotation from the
library
        saveAnnotationLibrary(library);//Save updated annotation library
        console.log("Library after deletion:", library);
```

```
      // Broadcast delete events to all clients
      wss.clients.forEach((client) => {
        if (client.readyState === WebSocket.OPEN) {
          client.send(
            JSON.stringify({
              type: "annotation_deleted",
              data: { index: annotationIndex }
            })
          );
        }
      });
    }else {
      console.log("Annotation not found for deletion:", parsedMessage);
    }
  }
  else if (parsedMessage.type === "update_video_name") {
    //Update video name
    const { videoId, newName } = parsedMessage;
    const videoIndex = videoInfo.videos.findIndex(v => v.id === videoId);

    if (videoIndex !== -1) {
      videoInfo.videos[videoIndex].name = newName;
      fs.writeFileSync(VIDEO_INFO_FILE, JSON.stringify(videoInfo), "utf-8");
      broadcastVideoList();
    }
  }
  else if (parsedMessage.type === "delete_video") {
    // delete video
    const { videoId } = parsedMessage;
    const videoIndex = videoInfo.videos.findIndex(v => v.id === videoId);

    if (videoIndex !== -1) {
      const video = videoInfo.videos[videoIndex];
      const videoPath = path.join(__dirname, video.url);

      // Try to delete the video file
      try {
        if (fs.existsSync(videoPath)) {
          fs.unlinkSync(videoPath);
        }
      } catch (err) {
        console.error("Error deleting video file:", err);
      }

      // Remove video from the list
      videoInfo.videos.splice(videoIndex, 1);
      fs.writeFileSync(VIDEO_INFO_FILE, JSON.stringify(videoInfo), "utf-8");
      broadcastVideoList();
```

```
      }
    }
    else if (parsedMessage.type === "request_video_list") {
      // Send video list
      ws.send(
        JSON.stringify({
          type: "video_list",
          data: videoInfo.videos
        })
      );
    }
  });

  ws.send(
    JSON.stringify({
      type: "welcome",
      data: "Welcome to the WebSocket server!",
    })
  );

});

server.listen(8080, () => {
  console.log("WebSocket server is running on wss://vrdanmaku.uk:8080");
});
```

Appendix 3 Project Gantt Chart

		Semester 1												Semester 2										
Tasks		W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11
Project Plan	Risk Assessment																							
	Ethical Approval																							
	Academic Integrity																							
	Project Plan/Specification Report Submission																							
Converting 2D to 360° Panoramic Video	Texture mapping																							
	Camera Setup																							
Generating 3D models using Photogrammetry	3D Model & Surface Details to the Cylinder																							
	Enhancing Interactivity (Lighting, Movement)																							
	Optimize and refine																							
	A-Frame VR Framework Setup																							
Basic interactions on Windows	Keyboard and mouse control																							
	Design the interface layout																							
	UI implementation and integration																							
	Exporting & building for Windows																							
Basic interactions on Android & iOS	Touch control																							
	Design the interface layout																							
	UI implementation and integration																							
	Exporting & building for Android & iOS																							
Crowdfunding Pitch	Submission																							
Adding Multi-Users and Pop-up Systems	AWS Server framework construction																							
	Implementation of Bullet Chats system																							
	Multi-user interaction testing																							
System Testing and Optimizing	Optimise the responsiveness of user operations																							
	Reduce the delay																							
Project Abstract	Submission																							
Bench Inspection	Poster Production																							
	Presentation																							
Final Report	Submission																							