Electrical and Electronics Engineering
Semester Project
Bachelor Semester 5 - Autumn 2018

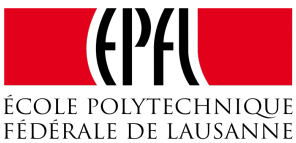
Quality assessment of 3D representations in Head Mounted Displays (HMDs)

Student: Peisen Xu

Supervised by: Evangelos Alexiou
Prof. Dr. Touradj Ebrahimi

January 13, 2019

Multimedia Signal Processing Group
EPFL



Abstract

3D technologies and virtual reality have shown an increasing popularity and potential over the last decade. Different kinds of 3D content representations such as point cloud have been generated and widely used in HMDs. However, the subjective and objective quality assessment of such 3D representations in head mounted displays is still an open question. In this project, we focus on developing the platform for subjective quality assessment of point clouds using HMDs. The platform will have the following features: adjustable point representation, adjustable virtual environment, navigation in the virtual space, supporting different subjective evaluation protocols (DSIS, ACR, etc.), recording the eye interaction of the subject with the object during assessment and post-processing and generating the gaze map on the object.

Contents

1 Introduction

High quality 3D contents are the key to enable immersive virtual environments. One of the alternatives for advanced 3D content representations is point clouds. In particular, point cloud offers a format to represent a 3D object by a set of data points with various attributes in space. It has the advantage of ease in capture and efficient processing. Point clouds can be used in different scenarios such as real-time 3D immersive telepresence, content VR viewing with Interactive Parallax, 3D free viewpoint sport broadcasting, geographic information systems, cultural heritage and autonomous navigation based on large-scale 3D dynamic maps [1]. In such systems, quality assessment is an essential step to identify the visual quality of the displayed contents, and can be exploited to enhance the user's experience or, potentially, the performance of the application.

There are two main categories for quality assessment methodologies: objective and subjective. Objective quality assessment is based on algorithms that aim to predict the perceived quality of distorted stimuli. Subjective quality assessment is based on human grades, and provides the ground truth for the visual quality of a content. In virtual reality, subjective quality assessment is an approach to obtain human judgments in an artificially generated, immersive environment. An important advantage of using virtual reality is that there is no need for setting up a physical experimentation environment. Thus, all the required environment conditions for the experiment (e.g., non-distractive room, diffused lighting) can be built in the virtual world, facilitating the usage across different users, places and platforms.

In this project, we design and develop a software to conduct subjective evaluation experiments with six degrees of freedom (6DoF) in virtual reality. The developed platform can be configured to different scenarios and viewing conditions required by the experiment conductor, from the point cloud rendering methods to the post-processing parameters, while special care was given to design an immersive application with high level of responsiveness. In our experiment, we use an eye-tracking hardware device in conjunction with the HTC Vive Pro head-mounted display (HMD) to record the eye behavior of the subjects while interacting with the contents under assessment. We design and conduct a task-dependent experiment to obtain the gaze information of the participants that correspond to the displayed stimuli. Finally, we devise and propose a methodology to post-process the recorded gazing data facing multiple challenges on our aim to extract accurate fixation density maps to indicate the areas of interest of the subjects.

The rest of the report follows the development process of the project analysed in corresponding sections:

1. State-of-the-art of point cloud capture, coding, display technology and subjective quality assessment.
2. Customization of our own importing and rendering methods for the experiment.
3. Structure of the developed experiment platform.
4. Introduction to the eye tracking and data recorder.
5. Description of the experiment and the post-processing of the recorded data.

2 State-of-the-art

2.1 Acquisition

The acquisition of point clouds can be performed by either indirect or direct methods. In indirect methods, point positions on the surface are determined by 2D images that don't contain 3D information. Direct methods determine the point position directly with the 3D information captured by the system [2].

Photogrammetry is an indirect acquisition method. For image acquisition, a large number of images are taken from all the effective viewpoints of the 3D model. During processing, similar distinct points on multiple overlapping RGB images taken from different perspectives are used to reconstruct the one point in the 3D space. The camera's position is either known or unknown. If unknown, the camera's position and angle relative to the point cloud is estimated at the same time as the point cloud is constructed. Knowing camera's position could help further classify the points in the images into points in space and create a denser cloud [2].

Figure 1: A monument in Nanyang Technology University: modelled with *Photoscan*

LiDAR, which stands for "Light Detection And Ranging", is often used as the direct acquisition method. It uses laser beams to scan the 3D objects and get 3D position information of the surface. The distance between the laser emitter and the surface point is calculated by the product of the speed of light and half the time of flight from laser emitting time to receiving time. Another similar type of LiDAR is the Time-of-Flight (ToF) camera. ToF camera can gather the 3D information of every pixel in the frame by knowing the time of travel [2].

Figure 2: Cultural heritage modelled with LiDAR [3]

In general, LiDAR has the advantage of capturing point cloud accurately with different types of surface and shapes, while photogrammetry can get the color information, but has difficulties in dealing with repetitive, or shiny surface.

2.2 Compression

Compression is essential for real-time display, streaming and storing large point clouds. The encoding techniques can be classified in two categories: (a) Progressive encoding, where different levels of details (LODs) of the same object are encoded progressively into a data structure, and (b) single-rate compression, in which the entire point cloud is directly encoded or decoded. Compression methodologies can be further divided into encoding techniques for static, or dynamic point clouds, which are analogous to image and video methodologies for 2D imaging. To encode the point cloud data, there are different approaches reported in the literature. In particular, octree-based, projection-based, and graph-based solutions have been proposed. The octree data structure is the most commonly adopted, especially for the geometry of the point cloud, as it offers an efficient data representation [4]. Combinations of these methodologies can be also used. For instance, the geometry of a point cloud can be encoded using an octree, while the color attributes can be compressed using projection-based methods from popular 2D imaging solutions such as JPEG [5] or H.264/AVC [6], or graph-based approaches [7].

2.3 Display Devices

The point cloud can be rendered and displayed through different devices. The most popular ones are traditional 2D computer monitors and more recently, the HMD devices with software rendering 3D graphics. Commercially, there are famous HMD products including HTC Vive, Oculus, Window Mixed Reality and Microsoft Hololens. Besides, point clouds can also be displayed via 3D display [8], lightfield display [9] or holographic representations [10].

2.4 Rendering

Several existing software or projects are popular for point cloud rendering, such as Point Cloud Library (PCL), MeshLab, CloudCompare and Matlab. There are also renderers based on web browsers such as MeshLab JS and Potree. For head-mounted displays, game engines like Unity and Unreal are famous for their cross platform support and large amount of extension or assets created by the community.

Unity 3D is the state-of-the-art cross-platform real-time engine. In our project, we choose Unity not only because it is easy to use and it fully supports virtual reality, but also because it is extensible, allowing people to improve parts of its core importing mechanics to read new data format. The shader scripts in CG/HLSL code also allow people to customize rendering methods.

As our implementation is based on the Unity 3D platform, below we refer to its rendering pipeline and existing software to display point clouds.

2.4.1 Point Cloud Rendering and Unity Shader

The Unity shader is designed to run on GPU. It normally draws triangles of the 3D models, but it can also be used to render 3D point clouds [11]. The rendering work flow in Unity is represented using the diagram in Figure 3. In our case, the material of the 3D model only uses shader to read the vertex position and the color information.

Unity3D supports surface shader and fragment plus vertex shader. Surface shader is normally for realistic lighting on the 3D model. It integrates and simulates how light is reflected on the surface by specifying simpler terms, such as albedo, normal, etc. Fragment and vertex shader does not naturally support lighting on the 3D model, but gives more control to the behaviour of each vertex and fragment. It takes vertex information on the model as input, and outputs the RGB color for each pixel rendered on the screen [11].

Furthermore, Unity also implements a method to support geometry shader [12]. It is a relative new shader supported from Direct3D 10 and OpenGL 3.2. Geometry takes vertices primitives as input, and outputs zero or more primitives that are then passed into the fragment shader [13].

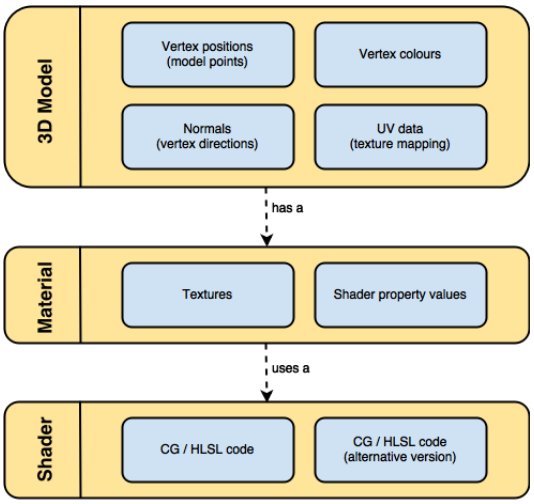


Figure 3: Rendering work flow in Unity [11]

However, to work with geometry shader, lighting calculation must be implemented manually as the surface shader does not allow geometry customization [14].

2.4.2 Existing Point Cloud Rendering Software in Unity

There are several existing free point cloud rendering software in Unity. The following charts are to compare their supported attributes and real-time performance with our custom point clouds.

Table 1: Features Comparison

Name	Format	Renderer Primitives	Features
Point Cloud Free Viewer	.off	Point	Load point cloud in run-time
Pcx Point Cloud Importer	.ply (binary little endian)	Point, disk	Create point cloud asset when importing
BA_Point Cloud	Potree	Point, square, interpolated square	Able to render large point cloud

Point Cloud Free Viewer is a run-time point cloud loader and renderer in Unity Asset Store. It can read only OFF format files, which contain the polygon description of the 3D object [15]. It also separates the point cloud data into several chunks and stores them in different Unity gameObjects to display them. On the shader side, it can only render the vertices into point primitives according to the input color [16].

Pcx Point Cloud Importer utilizes Unity's assets importer API to read and import PLY files. PLY is one of the most common formats to store data from 3D scanners. It stores data in either ASCII or binary format. *Pcx Importer* can only read from binary little endian format. It can not only render the vertex into point, but also into disks of size that can be specified. It uses geometry to split a single vertex into up to 13 vertices to form a screen facing disk [17].

BA_Point Cloud uses the point cloud data stored in potree octree data structure to render large point clouds. Potree converter will convert point cloud from LAS, LAZ, binary PLY, XYZ

or PTX to the potree octree data structure. This type of data structure contains point cloud information in different resolutions. Potree is developed to render large point cloud in web browser [18]. Using the *BA_Point Cloud* Unity project, the point cloud can be displayed in different levels of detail in different view-port. It can render very large point cloud with a point budge. However, since it's lossy when displaying lower level of details, it will add uncertain factors into the evaluation process [19].

Another interesting feature in *BA_Point Cloud* is that it can render a point as a paraboloid. The result of this implementation is that points of different depth, are "interpolared" before displayed onto the screen, so that the point that is further away from the camera won't be culled by the point that is closer to the camera, as shown in Figure 4. This technique will render the point cloud with more sharpened inner edge, as can be seen in Figure 5. This approach is good to avoid distortions of the details of a displayed object.

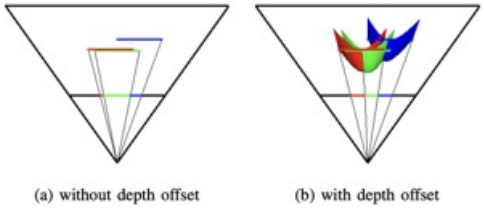


Figure 4: Paraboloid interpolation [20]

On the implementation side, quads are used as primitive elements. In the geometry shader, each vertex is split into four points whose positions are defined based on the point size. At the same time, each split point is also assigned a UV coordinate, and inversely projected into the view space. This is for the fragment shader to be able to modify the z value of the interpolated pixels on each quad in the view space according to the relative position of each pixel on the quad. Using the z value, each pixel is assigned by a distinct modified depth to be rendered on the screen.

(a) Non-interpolated (b) Interpolated

Figure 5: Interpolation comparison

Table 2 shows the number of points that the renderers are able to display in different frame per second (FPS). Note that for the first two renderers, the number of points represents the total number of vertices in the point cloud. For the *BA_Point Cloud*, a larger point cloud was loaded into the scene and the number of points represents the point budget we assigned to the script.

As we can see from the comparison, using points as primitives will increase the performance of each renderer. The number of points rendered in the *BA_Point Cloud* is higher for fixed FPS values, however, the frame rate will drop when changing the view-port, which is continuous happening in reality as a typical user constantly explores the virtual environment. Furthermore, the visual quality observed when using 2D primitives (disk, quad, etc.) is much better than

Table 2: Performance Comparison

Name	Shader	90FPS	45FPS	30FPS	Remarks
Point Cloud Free Viewer	Point	5m	10m	17m	Take a long time to load
Pcx Point Cloud Importer	Point	5m	10m	17m	Using disk shader may affect frame rate when camera is close
	Disk	2m	4m	8m	
BA_Point Cloud (Point Budget)	Point	10m	20m	>50m	Display large point cloud with lower level of details. Frame rate drop when moving camera
	VertexQuad	1m	5m	10m	
	GeoQuad	1m	5m	8m	

points, as they can fill the holes and make the 3D models look more realistic.

Comparing *Point Cloud Free Viewer* to *Pcx Point Cloud Importer*, one drawback of the *Point Cloud Free Viewer* is that it takes a long time to load a new point cloud. Moreover, the *Point Cloud Free Viewer* crashes when trying to load point cloud with over 30 millions points. As a result, we decided to use *Pcx Point Cloud Importer* as the basis for this project, and try to integrate the interpolation feature that is provided by the *BA_Point Cloud* into it.

2.5 Subjective Quality Assessment

Subjective quality assessment has been extensively used in rating the visual quality of image, or video content. Based on the ITU-R Recommendation BT.500-13 [21], the subjective evaluation methodologies can be generally classified as Single-Stimulus, or Double-Stimulus, while the subjects are asked to rate either the level of impairment, or the visual quality of the displayed contents. The Double-Stimulus Impairment Scale (DSIS) is one of the most popular protocols, where the subjects compare the impaired content to the unimpaired one, and rate the impaired content based on an impairment scale. Absolute Category Rating (ACR) is used with Single-Stimulus viewing and denotes another methodology, where participants inspect only the impaired content, and rate according to a quality scale. Below, we provide a typical 5-grading impairment scale:

1. imperceptible
2. perceptible, but not annoying
3. slightly annoying
4. annoying
5. very annoying

and a 5-grade quality scale:

1. excellent
2. good

3. fair
4. poor
5. bad

In point clouds, a number of subjective evaluations have been recently reported in the literature. The most relevant study is reported in [22], where the experiment conductor used an HMD with augmented reality to display the content. The DSIS subjective evaluation protocol was adopted in this experiment. Another representative study is presented in [23], where a similar experiment was conducted in a typical desktop setup. Both the DSIS and the ACR protocols were used, and the subjective results were compared to the scores obtained by objective metrics. Zerman et al. [24] have released the most recent study on subjective evaluation of dynamic point clouds, where references to the related work on subjective and objective quality assessment can be found.

3 Point Cloud Importing and Rendering for Experiment

3.1 Point Cloud Rendering

3.1.1 Modified Shader

The original *Pcx Point Cloud Importer* renders a point into a disk. Performance wise, this is worse than splitting point into a quad, which only contains four vertices. And since the primitives will be later transformed to a screen faced paraboloid in the fragment shader, the visual difference between using quad and disk can be hardly noticed.

Since in both projects different C# scripts is associated with the shader to modify or assign certain properties, the scripts needs to be slightly modified to either decouple them from each other, or adapt to new variables added during integration.

Firstly in *Pcx Point Cloud Importer*, custom material editor is used for shader. Hence to edit properties in shader by editor, additional fields need to be added in the custom editor script as well as in shader script itself.

Secondly in *BA_ Point Cloud*, the transformation from clip space to view space is done by assigning the transformation matrix by a script attached to the camera. There are two drawbacks with this approach. One is the coupling between camera and each rendered point cloud in the scene. The second is this approach does not work well with off-game, or editor inspection of the point cloud. Unfortunately, Unity doesn't provide build-in shader variable in its shader script for inverse transformation from clip space to view space. Therefore, a generate inverse function for 4 by 4 shader is created in the shader script to get the inverse of one matrix.

3.1.2 Adaptive Point Size

At this point, the point cloud looks much better than with interpolation, especially for dense point clouds that the points are evenly distributed. However, the same size is use for each point in the point cloud, which will cause two issues. Firstly, for point clouds with different densities, different sizes need to be applied manually in the editor. Secondly, for some point clouds whose structure is not regular, overlapping in dense regions and holes in sparse regions will be observed. Therefore, we implemented an algorithm to assign an different point size to each point of the content. The point size property is computed based on local nearest neighbors of each point when importing the point cloud into Unity.

In particular, KDTree is used for calculating the nearest K neighbours. After finding the size of every point, vertices with extremely large or small point size is normalized to the global mean of the point size. Finally, the UV attribute is used as the place holder for the point size value, as

there is no build-in point size attribute in Unity mesh. In geometry shader, the adaptive point size is read from UV attribute and applied to the extend of each split points.

3.2 Point Cloud Importing

Point Cloud Importing is handled by *Pcx Point Cloud Importer*. On top of that, we added three new features to the importer to suit point clouds to the experiment better:

- The **first feature** is the adaptive point size calculation as mentioned above.
- The **second feature** is automatically fitting the transform. The point clouds generated from other software can be vastly different in scale and position when importing the point cloud, and this step is important to avoid manual size and position adjustments for each point cloud in the editor. Instead, we can get the bounding box information of the point cloud in unity world space. Using this information, we can either limit the size of the longest side of the bounding box, or transform it to a fixed volume, surface area, or perimeter. Additionally, if the point cloud is too tall after fitting the transform, it is automatically re-scaled to a shorter height. The position of the point cloud is adjusted so that its bounding box center is in the middle of the scene with height equal to the half of the height of the bounding box. As a result, point cloud is automatically in the center of the scene when imported into Unity. One limitation is that the point cloud is not rotated automatically. Thus, the point cloud must have the correct orientation with respect to horizontal plane before it is imported to Unity.
- The **third feature** is to automatically project and create a shadow texture. This is for enhancing the environment, which will be described in detail later in the next section.

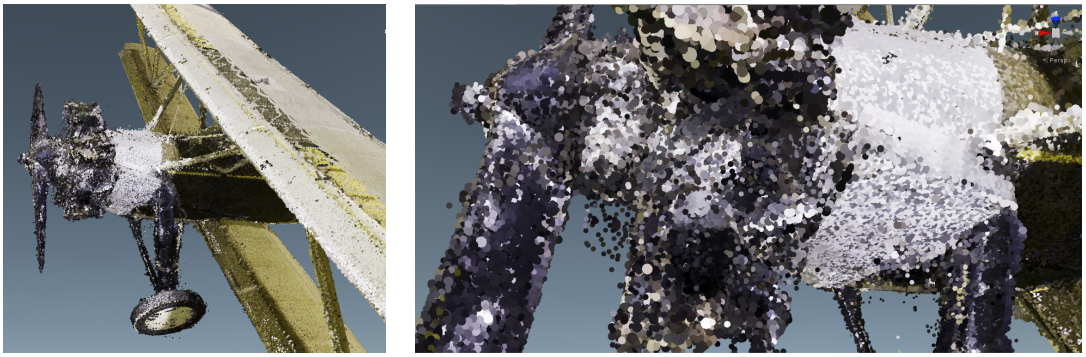
4 Experiment Platform

4.1 Environment and Lighting

The virtual environment should be non-distracting, following the recommendation ITU-R BT.500-13 [21]. Moreover, it should be responsive and allow comfortable viewing and interaction, in order to increase the level of immersiveness of the subject.

4.1.1 State-of-the-art

To achieve such realistic environment, lighting is a key factor. At the same time, optimization in the rendering pipeline is important, as users will feel sickness with low frame rates. Global



(a) biplane.ply (b) details

Figure 6: Adaptive point size

illumination is the most popular technique for realistic lighting conditions. In addition to direct light from the light source, it also takes into consideration the reflected light bouncing off in the scene. An example can be seen in Figure 7.

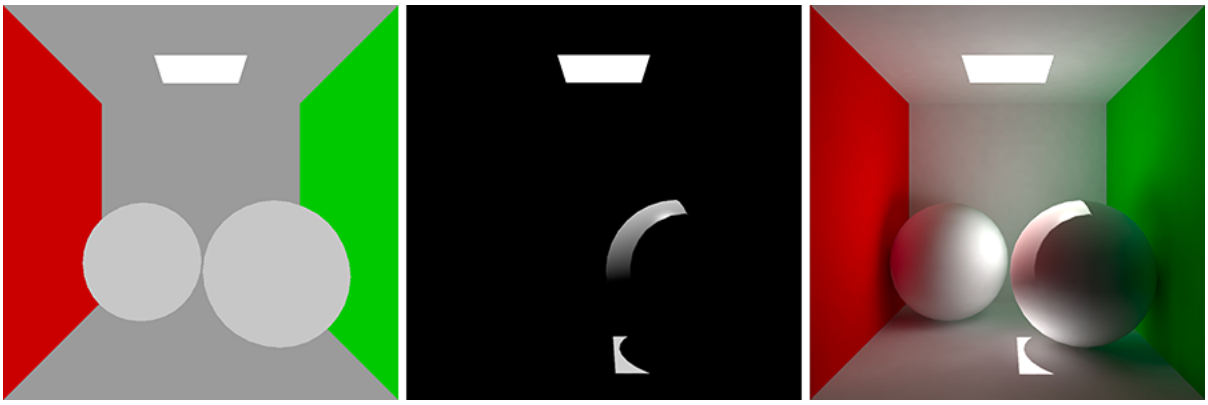


Figure 7: No Lighting. Direct Lighting. Global Illumination. [25]

Unity supports two kinds of global illumination (GI): (i) baked GI and (ii) pre-computed real-time GI. The baking process is to pre-compute the lighting effect on all the static objects in the scene, and generate light-maps overlay above them. The baking process can take a long time. However, this method has the best performance in real time [25]. Pre-computed real-time GI utilizes pre-computed environmental features and real-time lights to adapt to any real-time change in the application. This method introduces higher overhead, but it is better in adapting to any object or lighting change or movement in real-time [25].

4.1.2 Baked Lighting and ChangeLightmaps()

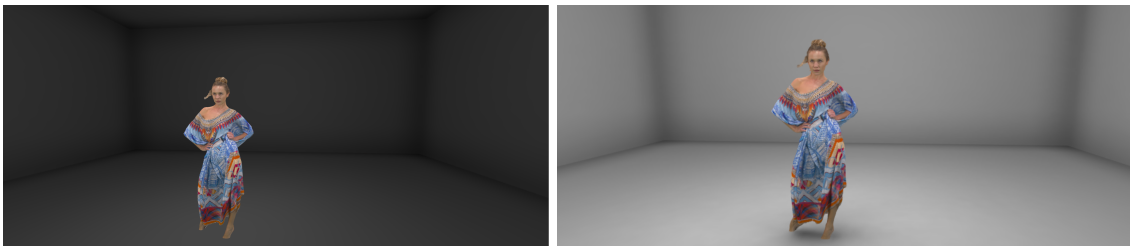
For high frame rate, we firstly implemented the baked lighting. Although during the experiment different point clouds need to be displayed in the scene, we could still bake multiple light-maps and use a script to change them in real-time. Before baking the light in Unity, each point cloud needs to be build into a separate mesh that is also imported into Unity. The meshing process can be done in Meshlab. To reduce the baking time, it's recommended to reduce the faces of the mesh as much as possible.

The baking is performed before running the experiment. The mesh is placed into the scene at the exact same location as the point cloud. The mesh property is changed to "lightmap static". The light in the scene is changed to "baked" lighting. "Baked global illumination" is turned on in the lighting setting tab. Then, "Generate Lighting" is clicked (if the auto generate is not enabled).

After baking, a folder of light-map data appears in the directory of the current scene. These light-maps inside the folder are copied to another folder named as the point cloud name for further re-use. Then, we change the mesh to correspond to the new point cloud, and we bake again. We repeat this process for every point cloud.

During the experiment, as the point cloud is changed, the lighting data in the scene also need to be changed. A script is attached to each of the point cloud so that when it was enabled in the scene, the corresponding light-map data will be automatically extracted from the folder and the current set of light-maps will be replaced.

Although good result can be achieved with this method, it is very intricate and time-consuming. It is not suitable for generalizing to other use-cases. Moreover, in case we want to change the transform of the point cloud, or create a stage to display small objects (which was done latter, as a refine step in the project), the whole sets of lighting data need to be re-baked.



(a) No lighting (b) With light and baked shadow

Figure 8: Room with longdress.ply

4.1.3 CreateShadowTexture()

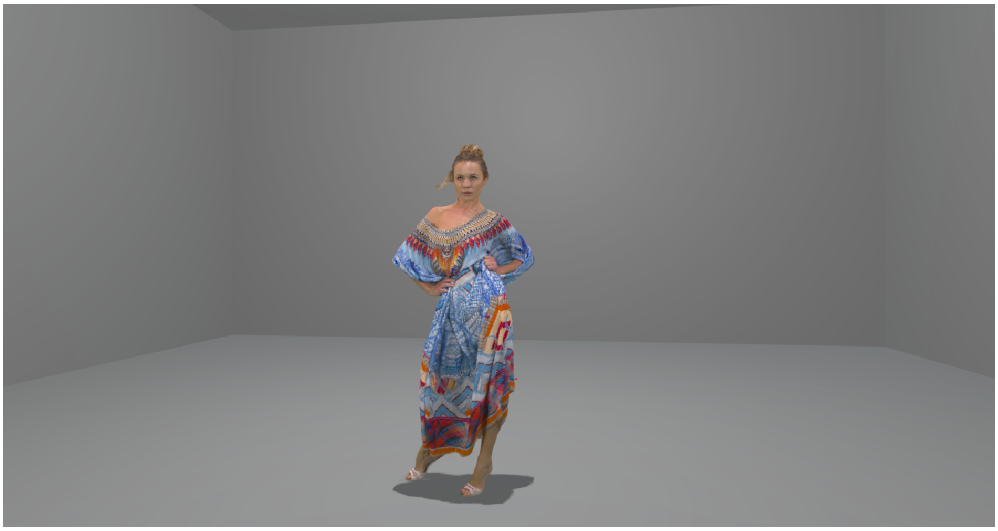
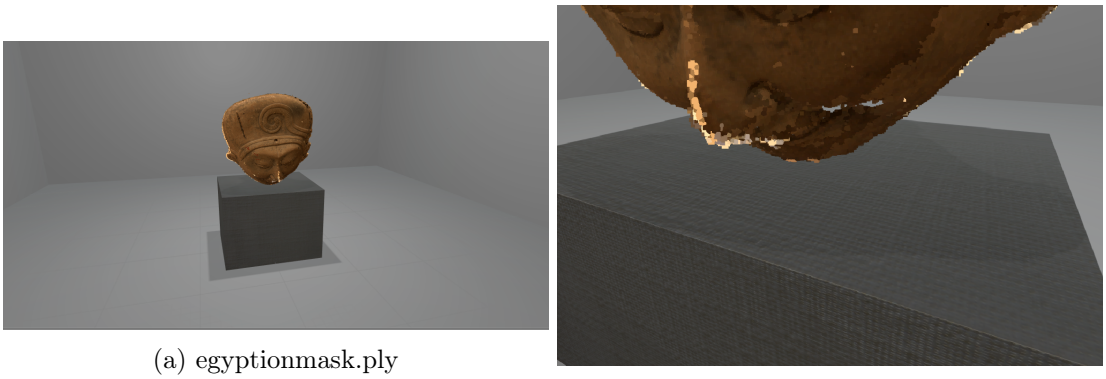


Figure 9: longdress.ply with shadow texture

Since the environment is simple, it is possible to simulate the shadow effect of the point cloud by a custom script. Assuming the point light is right above the object, the shadow of each point in the point cloud can be simulated by the extension of the connection between the light point and every point that belongs to the content. In this way, a shadow texture is created and applied to a quad, which is always attached at the underneath of the point cloud, as illustrated in Figure 9. Now, with the point cloud itself set to "unlit", it is rendered directly with its input vertex color multiplied by a uniform tint. At the same time, real-time lighting can be applied to the rest of the scene with little performance overhead.

4.1.4 Stage.cs

Although the transform of point cloud has been normalized to a global parameter during importing, some point cloud still need more accurate adjustment for more comfortable inspection. For objects that are supposed to be small, it is not natural to see them hover in the air, which could possibly affect the level of immersion. Therefore, considering the way an object is normally presented in real life, a stage is created to display smaller objects, as depicted in Figure 10a. To enforce the non-distracting principle, a bumped cloth material is applied to the stage so that the lighting on the stage is more uniform and diffused (i.e., see Figure 10b).



(a) egyptionmask.ply

(b) details

Figure 10: Object with stage and shadow texture

4.2 Point Cloud Displaying Sequence and Experiment Manager

The displaying sequence of point clouds varies among different subjective evaluation methodologies. In ACR, only impaired point clouds are displayed. In DSIS, for each 3D content, an impaired point cloud and an unimpaired point cloud need to be loaded simultaneously into the scene. They could be displayed next to each other, sequentially, or alternatively upon subject's commands.

4.2.1 ExperimentManager.cs

The experiment manager is responsible to load and display a sequence of point clouds based on the specified configuration. The script checks the current state (index, isDistorted) update on each frame, to receive the input from the experiment controller. If the state is updated, the experiment manager will unload the current point cloud and load the next one. Internal calibration (will be analysed in the next section) is also controlled by experiment manager.

4.2.2 data.json

The point cloud sequence and experiment configuration can be specified in Unity editor by assigning parameters into the public field of the experiment manager script. However, there are two drawbacks. Firstly, it might take unnecessary efforts for non-Unity users to understand. Secondly, the configuration of the experiment can no longer be changed in the build version of the project.

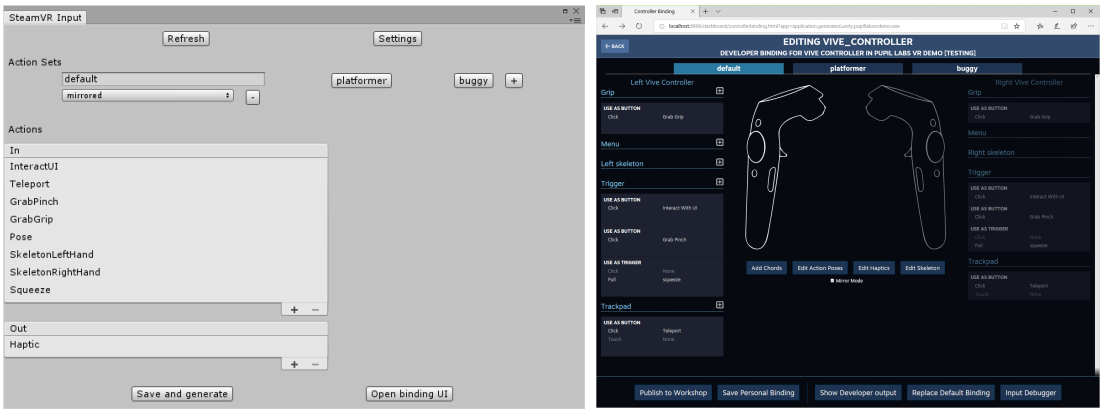
Therefore, instead of taking the input directly from the script, the experiment manager will read the configuration from an external JSON file. In the JSON file, each point cloud is with an unimpaired point cloud (or null, if the ACR mode is selected). The sequence in the JSON is the displaying sequence in the actual experiment. Furthermore, the scale of the point cloud can be specified again in the JSON file, as well as whether using a stage to display the point cloud. This is to enable precise control on the size and position of the point cloud, in order to adapt to special use cases. If not specified, the original size and position from the PLY importer will be used. Finally, the global parameters for material properties are also specified in the JSON file.

4.3 Experiment Controller

4.3.1 Control Scheme

Different from game-pad or keyboard and mouse, VR controller is a new interface for many users. It takes time to learn how to use different buttons properly. Therefore, for the experiment, we try to make the controlling scheme more intuitive so that subjects can learn it faster. To be

more specific, the pad button is for teleporting around the virtual world. The trigger button is for proceeding to the next content. The grab button is for turning around body in ACR mode and for switching between original and distorted content in DSIS mode.



(a) Steam VR input (b) Controller Binding

Figure 11: Control Scheme

4.3.2 SteamVR_Action_Boolean.GetState(inputSource)

Steam VR plugin in Unity is used for accessing the API of the controller. In the Steam VR input window, shown in Figure 11a, different actions can be defined for the controller. After save and generate the action class, one can bind each action to a physical input from the controller, which is depicted in Figure 11b. In the script, the state of input is requested from a certain action class of a certain input source.

5 Eye Tracking and Data Recording

To get more information on the interaction between the subject and the object, eye tracking technology is utilized to obtain the region of interest on the 3D contents. In this project, we used the *Pupil Labs* eye tracker. It is embedded into the HMD, with two cameras placed underneath the two eyes, and connected to the computer with a Type-C USB cable.

5.1 Pupil Capture

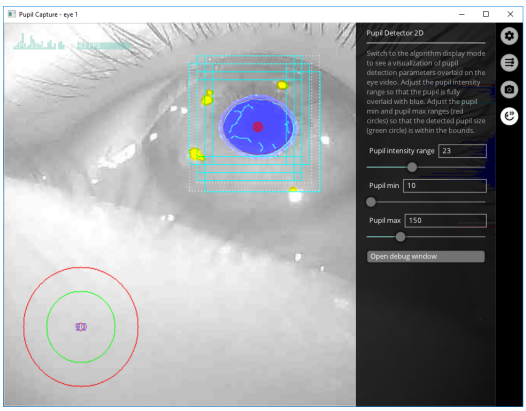
Pupil Capture is the software that reads the camera image and detects the pupil positions. It also handles gaze mapping, video recording, data streaming and events broadcast. Pupil Capture can be used with either Pupil Headset or the HMD integration [26].

5.2 Unity Integration

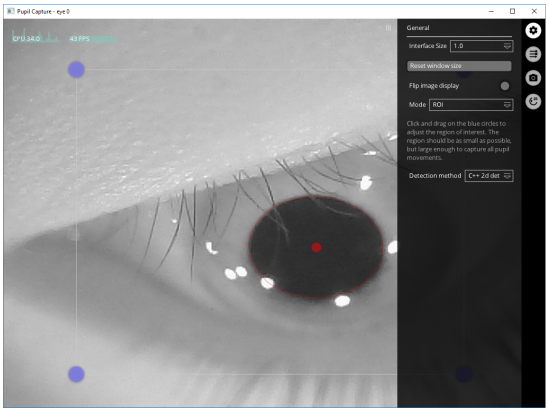
Note that although Pupil Service is the software intended to work with Unity integration, in the most recent release, it is recommended to use Pupil Capture instead [27]. Therefore, we use the Pupil Capture for our experiment. The HMD-eyes project in Unity is developed by *Pupil Labs* specifically for supporting the HTC Vive and Hololens integration of the hardware. It mainly serves the purpose of the initial calibration, so that the pupil position can be mapped to the gaze position in the virtual coordinates inside Unity. It also constantly communicates with Pupil Capture and gets the latest processed data for recording, or analysis inside Unity.

5.3 Eye Tracking Accuracy

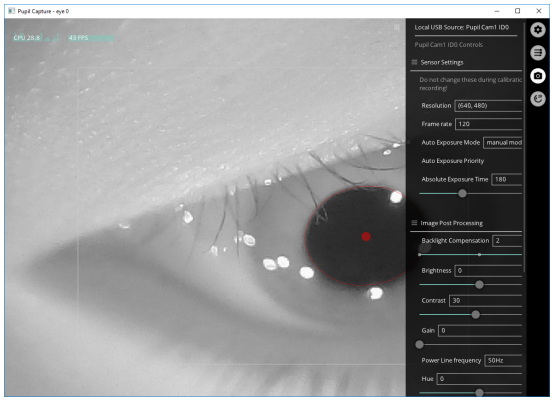
The eye tracking accuracy mainly depends on two factors. The first factor is the confidence of the pupil detection, which depends on the algorithm and parameters used in Pupil Capture software, while the second factor is the mapping of pupil position to the gaze point in the view-port.



(a) Algorithm View



(b) Region of Interests



(c) More parameters

Figure 12: Eye camera

Regarding the first factor, there are tools provided by the software coming with Pupil Labs eye-tracker that assist to inspect and increase the confidence of the detection of pupils in the recorded images. Below, we provide a brief description of them:

- In the Algorithm View of the eye camera window, the squares show the initial estimations of pupil region, as presented in Figure 12a. The blue regions are the "dark" regions detected by the algorithm, while the yellow regions are the spectral reflections to be excluded. The lines inside blue regions are the edges detected using the Canny edge detection algorithm. The red eclipse is the result of the pupil detection by fitting the Canny edges. The red dot in the middle shows the exact pupil position used in the current frame. At the bottom left, the two red circles are the limits set for the pupil size, while the green circle shows the current size of the pupil [28]. The accuracy can be visualized by the transparency of the middle red dot inside the pupil. The more opaque it is, the higher the confidence of the current detection. Moreover, the value of the confidence level is also displayed in real-time on the pupil capture window, as shown in the right part of the Figure 12a. The focal length of the physical cameras that record the eye movements might need adjustments in order to capture clear images and improve the confidence of the detection.

- The region of interest (ROI), as illustrated in Figure 12b, aims to limit the region of pupil detection. This parameter be adjusted to mitigate the effect of eyelashes and vignetting.

- The resolution of the image should be kept as high as possible to ensure no loss of information and higher level of details. However, by increasing the resolution of the image, more data are recorded and processed and, thus, the frame rate could be affected due to the processing overhead. The absolute exposure time is another option that can be set, and leads to changes in the brightness of the captured images. It can be useful to distinguish the pupil from the other areas. As presented in Figure 12c, there are additional parameters for image post processing, which is for more in depth analysis of the captured images. In our experience, changing these parameters won't affect the confidence values too much.

- Pupil min and max can limit the size of the detected region for the pupil. It is useful for limiting the error when detection is bad. Pupil intensity range is for adapting to different colours of pupil, as shown in Figure 12a.

The second factor is the mapping of pupil position to the gaze point in the view-port. Before recording the eye tracking in the virtual world, we have to first map each of the pupil positions in the camera screen to a gaze position in the view-port in the virtual camera. This process is called calibration. After each calibration, there is a fixed mapping from pupil position to the gaze position. However, in the case of virtual reality using an HMD, the relative position between the eye and the camera could change during the experiment due to HMD slippages. If this happens, the mapping will be inaccurate and a general offset will occur. This is a big limitation that will generate noisy data when conducting an experiment with HMD using an eye tracker. However, we can mitigate the effect by internal calibration.

5.4 Calibration

The HMD-eyes Unity project handles the external calibration to calculate the gaze mapping from pupil positions relative to eye camera to gaze positions in the virtual world. The external calibration will visualize the confidence level of both eyes by the colour of the calibration point. Based on [29], yellow means low confidence on left eye and pink means low confidence on right eye, while white color means high confidence on both eyes, as presented in Figure 13. After the external calibration, the experiment scene will be automatically loaded.

Figure 13: External calibration

After the external calibration, the information of a sample's time-stamp, gaze position, pupil position relative to the camera and confidence level of both eyes will be sent to Unity. However, there is no way to know the actual angular error between the predicted gaze and the real gaze in the virtual world. In general, we want to measure the difference between the subject's gaze and the predicted gaze after the external calibration. Therefore, internal calibration is implemented to obtain this information.

The internal calibration starts after the inspection of each point cloud. There are five calibration cubes placed at the top left, top right, middle, bottom left, and bottom right corner of the screen respectively. As the used stares at each of these cubes, several data are being recorded

and logged into the debug window in Unity Editor, including the angular error of left and right eyes, the confidence of left and right eyes and the angular error of the average gaze position of the two eyes. When a subject is looking at a cube, the experiment conductor is externally noticed about the accuracy of the gaze position in real-time. An example is shown in Figure 14, where the angular error is also displayed.

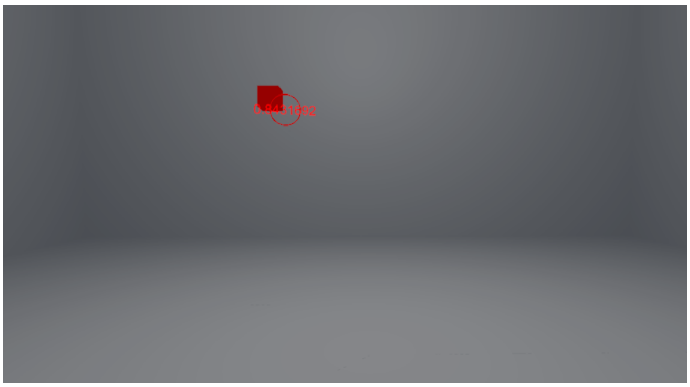


Figure 14: Internal calibration (red circle is for debugging)

5.5 Recording the Data

During the experiment, data is constantly being sent to Unity from the external Pupil Capture software.

5.5.1 PupilTools.cs

PupilTools.cs is a class for handling the state of data receiving in Unity. In addition, it also handles the calibration and part of the data recording. Using the *PupilTools.SubscribeTo(string topic)* method, the data from the corresponding topic can be received.

5.5.2 Connection.cs

Connection.cs uses sockets to handle the communication between Pupil Capture and Unity, and the interpretation of message from Pupil Capture. The messages received contain dictionaries, which are de-serialized using *MessagePackSerializer.Deserialize()* [29].

The values corresponding to different keys in the dictionary can be find by *PupilTools*'s get value functions. In *Connection.cs*, we get the data from "gaze" topic immediately after the dictionaries are de-serialized, and then store them into several static field in *ExpDataRecorder.cs*

5.5.3 PupilDemo.cs

PupilDemo.cs is the script that was developed to receive data from the "pupil" topic. However, since data from "pupil" topic and data from "gaze" topic are stored in two different streams, the synchronization between them is essential. Instead of storing each data from the "pupil" data into a single value static field that would overwrite the previous with new data, we add the new data into a static list in *ExpDataRecorder.cs*. Then, *ExpDataRecorder.cs* tries to match the data in the list from the "pupil" topic to the single value static data from the "gaze" topic.

5.5.4 ExpDataRecorder.cs

In *ExpDataRecorder.cs*, data are recorded once per *Update()* function. This way, data are recorded with the effective rendering frame rate, which corresponds to what people actually

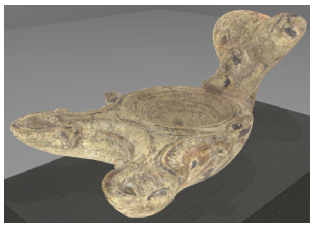
visualized. The data from "pupil" topic and "gaze" topics are matched by comparing the time-stamps of the two streams. The time-stamp from the "gaze" topic is used to find the closest time-stamp in the list of data from "pupil" topic. In addition, the recorded "gaze" topic sample will wait for the "pupil" topic samples to append samples in its list for 1 second, before searching for the matching time-stamp. This is for the pupil samples that appear after the current gaze sample to match the gaze sample. Based on our experience, with this method about half of the gaze samples are able to match with corresponding pupil samples based on their time-stamps. One possible way to improve this result is by using another keyword that should be contained in the dictionary which is returned from the "gaze" topic, named "base_data". For the matched samples, one can reproduce the gaze positions from the recorded pupil positions using a different mapping matrix during post-processing.

Data are also recorded during the internal calibration. The data from internal calibration are recorded in a separate file and contain the following properties: point cloud index, accuracy of the average of two gaze position, accuracy of left eye, confidence of left eye, accuracy of right eye, confidence of right eye, calibration cube index.

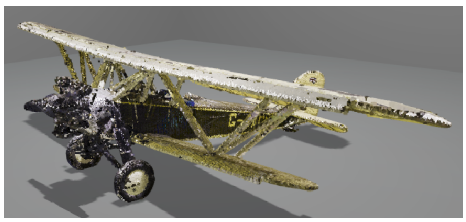
6 Conducting the Experiment and Data Post-processing

6.1 Setup

Two sets of point clouds were created for the experiment. One set contained six point clouds that represent objects, presented in Figure 15, while the other set contained six point clouds that represent people, depicted in Figure 16. The point cloud *Head.ply* was used as the training content.



(a) romanoilamp.ply



(b) biplane.ply

(c) egyptian.ply

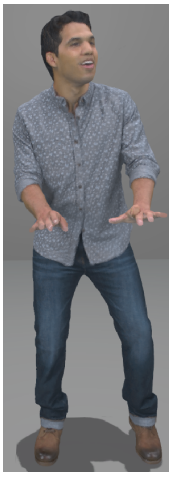
(d) statue.ply

(e) amphoriskos.ply

(f) shiva.ply

Figure 15: First set of point clouds showing objects

(a) longdress.ply



(b) loot.ply

(c) hhiulliwegner.ply

(d) redandblack.ply

(e) the20sMaria.ply

(f) soldier.ply

Figure 16: Second set of point clouds showing people

The training content was always displayed first to the participants. The order of the other point clouds in each set was randomized for each subject. The height and the width of the models was specified manually in the configuration file, in order to scale them accordingly for more comfortable inspection. For the first set, stages were used to display some of the models (i.e., can be seed in Figure 15). All the point clouds were rendered after multiplying the size of each point by an identical scalar (on top of the adaptive point size) and tint color (on top of the original vertex colors).

6.2 Methodology

In this experiment, we used task-dependent viewing to conduct the subjective quality assessment. The point cloud sequence was designed similarly to the ACR subjective evaluation protocol. Instead of letting the subjects to grade each point cloud, we let them order all the point clouds in each set according to their own preferences. We also asked the subjects what is the criterion for their selected ordering sequence.

The test began with measuring the eye distance of each subject. The experiment conductor adjusted the eye distance in the HMD for the subject. Then, the subject was familiarized with the controllers and the naming of each button in order to be able to communicate easier.

Then, we helped the subject to wear the headset, one conductor would look at the eye camera display on the monitor while the other would adjust the position of the headset to make sure that the pupils were in the near center of the eye cameras. After that, we fastened the headset on subject's head, making sure it was both stable and comfortable for the subject to navigate.

From this point, we told the subject that he shouldn't touch the HMD anymore, as the eye

tracking would be very sensitive to the HMD slippage on the head. Before moving into the external calibration, we would adjust the eye camera parameters to make sure the detection confidence was high.

After the external calibration, the subject would be able to see the training contents. He/she was instructed to move around the scene using either physical movements, or through locomotion using the controller, and inspect the content. After he/she got familiarized with the environment and the interface, we would instruct him/her to press the trigger button to proceed into the internal calibration. In the internal calibration, the next calibration cube was shown each time the subject pressed the trigger button. If the angular error of their gaze was lower than 2 for more than 2 calibration cubes, we would instruct them to move on to the next content. Otherwise, we would restart the external calibration to remap their gaze. Then the experiment would continue from the next point cloud.

For each content, the subject could inspect as much as he wanted to. After finishing each set, we would let them take off the HMD and order the contents using a 2D picture reference to remind them what they had just seen.

After the experiment, we gave each subject a questionnaire for them to provide feedback for our experiment platform. We firstly asked them how familiar they were with the HMDs. The questionnaire let them rate their overall experience with three questions:

1. How much immersed did you feed?
2. How much discomfort did you feel?
3. How much tired did you feel?

The tests were done in 3 days with 11 participants (7 males, 4 females). The average age of participants was 24.36, median was 24, minimum was 20 and maximum was 28.

6.3 Statistical Analysis

The rank of each point cloud is given as the average order from the 11 results, as shown in Table 3a.

Although participants generally had different criteria for their ordering, the following keywords appeared more than one time, as shown in Table 3b.

Table 3: Statistical Analysis

(a) Score of point cloud

	Name	Rank
"Objects"	biplane	3.545
	amphoriskos	2.455
	egyptian	4.636
	romanoilamp	3.273
	shiva	3.909
	statue	3.182
"People"	hhiulliwegner	3.000
	longdress	3.727
	loot	3.545
	redandblack	3.091
	soldier	3.727
	the20sMaria	3.909

(b) Keywords in the criteria

	Keyword	Appearance time
"Objects"	Color	3
	Interesting	2
	Beauty	2
	Realistic	2
	Size/Scale	2
"People"	Pleasantness	4
	Realistic	3
	Quality	3
	Clothes	2

For the first set of point clouds of objects, despite the wide range of criteria used by subjects, "color" had slightly more appearance times according to our records. Correspondingly, *ampho-*

riskos.ply, which has higher color variance, seems to be in higher preference and *egyptian.ply*, whose color is more monochromatic, is in a lower rank.

For the second set of point clouds of people, "hospitality" was stated 4 times among 11 subjects. While the average rank of the second set has lower variance the first set, *hhiulliwegner.ply* has a slightly better performance than others.

Based on the feedback provided by the subjects, the immersiveness of the environment had an average rate of 4.1 out of 5 (5 indicates the highest level of immersion), while the discomfort had only a rate of 1.4 (5 indicates the highest level of discomfort). The rate of tiredness was 2 out of 5 (5 indicates the highest level of tiredness). In our opinion, this was mostly due to the repeating calibration steps.

6.4 Eye Tracking Data Post-processing

The recorded experiment data contain noise resulting from headset slippage, low pupil detection confidence, blinks and random un-fixated eye gazes. Therefore, to get the accurate representation of the gaze map, we need to filter the data by internal calibration information and calculate the fixation points.

6.4.1 DataFilter.cs

The internal calibration is conducted after the inspection of each point cloud. Therefore, the combination of the internal calibrations before (if available) and after the inspection of the current point cloud gives us the information of the accuracy of the gaze samples on the current point cloud. Moreover, the recorded confidence on both eyes show us how well each pupil is detected in the respective region.

With each internal calibration, we get a representative eye on each region (top left, bottom right, etc.), and also their corresponding angular error. Therefore, for each region, the recorded gaze samples are first filtered with a confidence threshold and outlier angular threshold. In the meantime, we keep counting the number of valid samples for both eyes, as well as for only right and left eye.

If the count of samples that both eyes are valid is over the minimum valid samples count threshold, then the representative eye in this region is recorded as "both", and we use the average angular error of left and right eye in the valid samples as the accuracy information for the region.

If not, the eye with valid sample count larger than 1.2 times of the other eye is recorded as the current eye type. If the valid sample count of two eyes are similar, the one with lower average angular error is chosen as the representing eye.

After processing the data, a modifier is created for each region on each object of each subject. An example is indicated in Figure 17. Each modifier contains the information of point cloud index, gaze region, representing eye and the average angular error.

6.4.2 EyeDataProcess.cs

LoadNextLine() Lines representing the sample data are loaded one after the other, after the program starts. As a first step, a sample is filtered by a confidence threshold to eliminate blinks and low confidence data. Only the samples with at least one high confidence detection (left, right or both) are qualified for further processing.

The calibration modifiers are then applied to each sample. Each sample is classified to one region using a Voronoi diagram, as shown in Figure 17. Then, using the corresponding modifier in the modifiers list, the sample determines which gaze (left, right or both) is used for calculating the fixation point. Note that if the gaze type has low confidence, the sample is discarded.

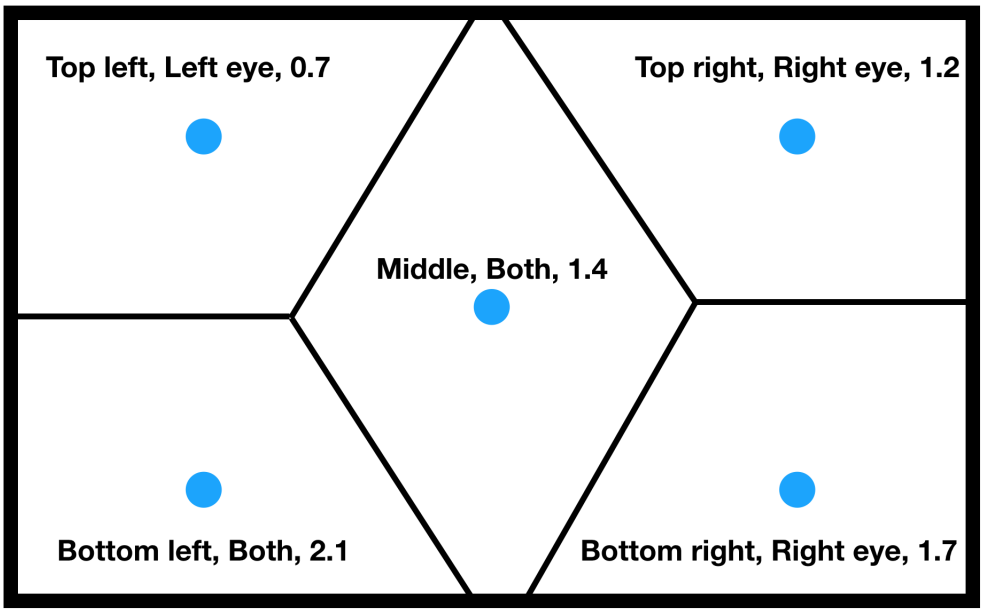


Figure 17: Calibration modifier

CalculateFixation() The calculation of a fixation point is shown by the diagram of Figure 18.

Two parameters define a fixation point, one is the minimum time interval of the fixation window, the other is the maximum dispersion threshold to discard the samples that are already registered for fixation detection in the *tempList*. If a fixation point is detected, the current gaze direction is calculated by the average gaze point world position minus the average camera position in the fixation *tempList*.

RegisterPoints() The gaze direction is classified again by the regions of the internal calibration to get the corresponding angular error of the current gaze. With the angular error and the gaze direction, a gaze cone is formed to detect a region of points on a point cloud. These points are then filtered by a manually set depth value, in order to exclude points that are occluded. For the final list of points on the point cloud that is classified as current gaze, we give each of them a gaze importance value with different methods. For this scope, multiple alternatives have been tested:

1. The first method is to give every point the importance that is inversely proportional to the angular error, or the square angular error. This will result in more evenly distributed gaze importance, as can be seen in Figure 19a.

2. The second one is the importance inversely proportional to the number of points classified as the current gaze. An example of the result of this method can be seen in Figure 19b.

3. The third one, as proposed by [30], assigns the importance according to Gaussian distribution. The result can be seen in Figure 19c.

Note that in Figure 19, we present the results of every tested method for the same content, using the gaze samples that were recorded by the 11 subjects that participated in our experiment. The only step that differs, is the methodology applied to obtain importance values for every point.

WritePointCloud() After all the gaze samples obtained by every subject for a specific object have been processed, a text file containing a list of vertex positions with their corresponding gaze importance values is created for visualization.

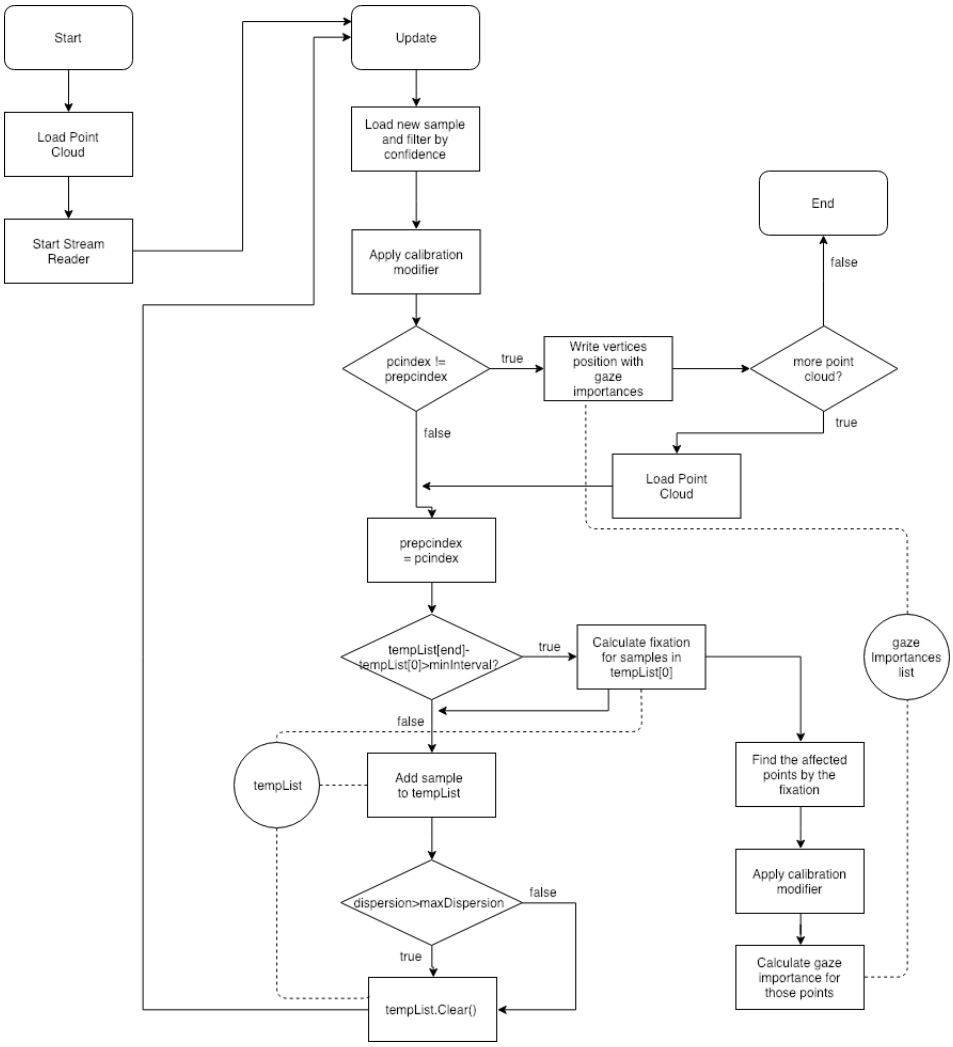


Figure 18: Eye data process

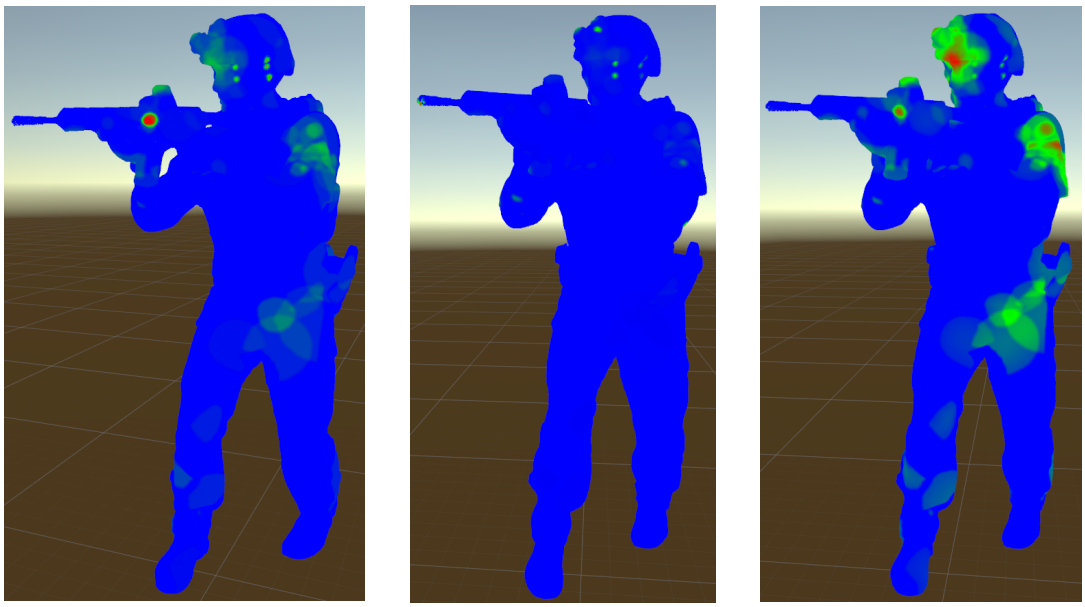
ProcessMultipleFolders.cs Since a new folder is created each time the experiment starts or restarts, it's much more convenient to process multiple folders one after another without manually changing the post-processing target.

6.4.3 EyeDataVisualise.cs

Main Thread In the main thread, the point cloud name to be visualized and the source files to be combined are specified. Furthermore, after assigning a color to each point of the point cloud and applying Gaussian blur (optionally), the main thread will handle creating the mesh and game object to display the result in the scene.

DataVisualiseJob.cs *DataVisualiseJob.cs* is a thread processing the visualization data as fast as possible in parallel with the main Update() function of Unity. It combines the gaze importance values of the same point cloud in different text files. At the same time, it also sends the amount of lines that have been processed to the main thread for debugging purposes.

GaussianBlurJob.cs After the gaze importance are combined, 3D Gaussian blur can be applied to get better visualization results, similarly to [30]. For each vertex, Gaussian blur is applied on its nearest K neighbours that are specified using a KD-tree. Finally, to speed up the process, the point cloud is split into several chunks and processed in parallel in different threads.



(a) Using angular error (b) Importance add to 1 (c) Gaussian distribution

Figure 19: Different methods on assigning importance (all with Gaussian blur)

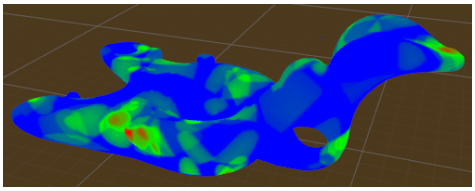
6.4.4 Results

The results were generated using the following parameters:

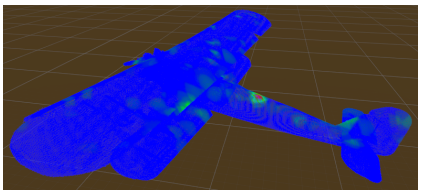
1. Internal calibration confidence threshold: 0.75.
2. Internal calibration outlier threshold: 5 degrees.
3. Internal calibration valid sample count threshold: 50.
4. Experiment data confidence threshold: 0.6.
5. Minimum interval for fixation: 0.3 sec.
6. Maximum dispersion for fixation: 3 degrees.
7. Gaze importance calculation method: Gaussian distribution.
8. Apply Gaussian blur: Yes.

Table 4 shows the number and the percentage of valid samples after filtered by confidence threshold and modifiers. In the first column, different numbers represent different subjects. "A" and "B" represent the first and the second set of models, respectively. Note that for subject 01 and 02, the internal calibration data were not separated by region and for subject 03, the data were not separated by eye type. Therefore, different modifier models were applied for those samples.

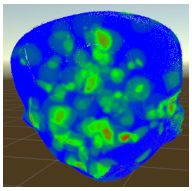
In Figure 20, we present our results that show the estimated fixation density maps for our experiment.



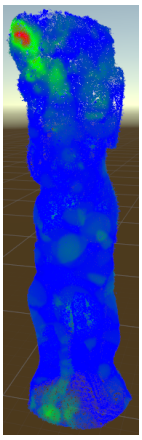
(a) romanoilamp.ply



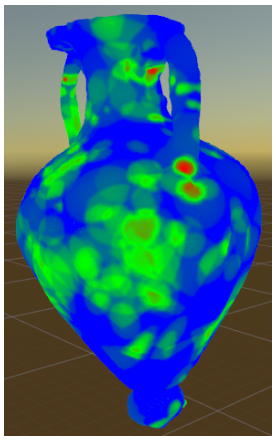
(b) biplane.ply



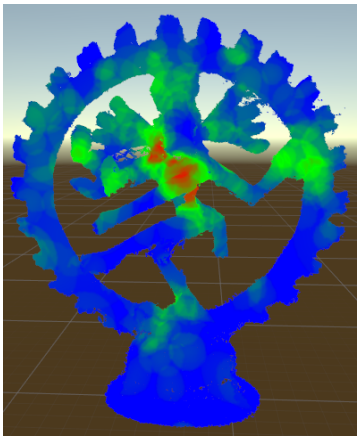
(c) egyptian.ply



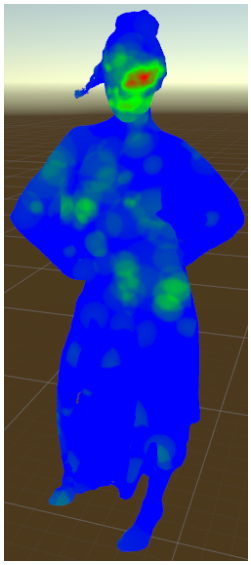
(d) statue.ply



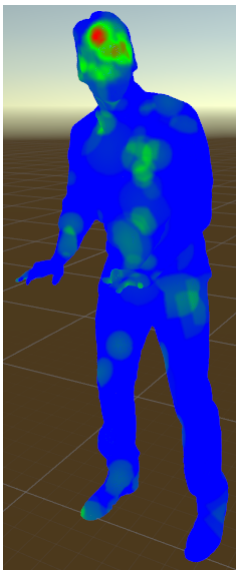
(e) amphoriskos.ply



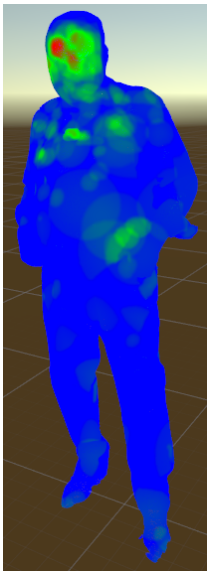
(f) shiva.ply



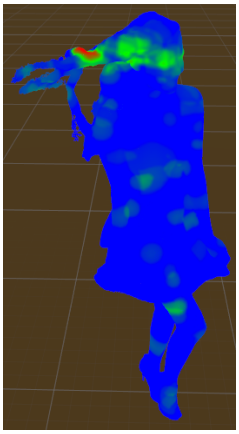
(g) longdress.ply



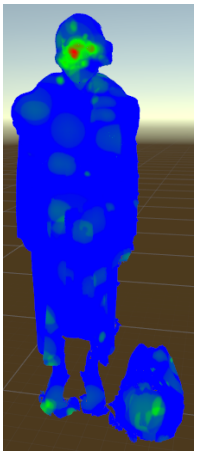
(h) loot.ply



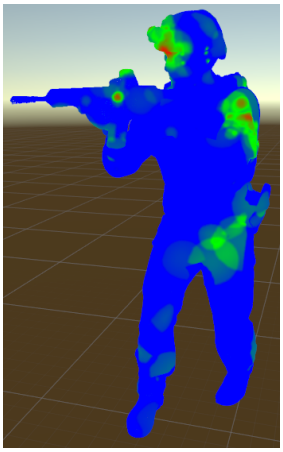
(i) hhiulliwegner.ply



(j) redandblack.ply



(k) the20sMaria.ply



(l) soldier.ply

Figure 20: All saliency maps

Table 4: Valid Sample Rate

Subject_Set	Total	Valid	Percentage
01A	70769	55430	78.33%
01B	36007	25842	71.77%
02A	46004	34379	74.73%
02B	22622	15445	68.27%
03A	42184	23399	55.47%
03B	20797	9117	43.84%
04A	47007	17875	38.03%
04B	18628	7876	42.28%
05A	49685	13729	27.63%
05B	19798	3226	16.29%
06A	51974	25518	49.10%
06B	30330	11251	37.10%
07A	44596	27184	60.96%
07B	16113	13682	84.91%
08A	35384	14288	40.38%
08B	18197	13997	76.92%
09A	48557	30696	63.22%
09B	35842	26946	75.18%
10A	29669	2799	00.09%
10B	12289	6	00.00%
11A	70810	42802	60.45%
11B	35263	25002	70.90%

7 Conclusion

In this project, we initially compared the different state-of-the-art methodologies for point cloud rendering in virtual reality, and adopted the one with the better visual quality in our software. Then, we designed a configurable virtual environment to conduct subjective evaluations of point cloud contents in 6DOF with the aforementioned rendering technique. Custom methods were applied to explicitly improve the efficiency of the application, the immersiveness levels and the intuitive control of the interactivity between the user and the content under assessment.

The platform can now be used to repeatedly host the same experiment, or experiments with different protocols, displaying methods, different virtual environments or point cloud set-ups. However, we will keep improving the adaptiveness after our project is being used in more scenarios.

The eye tracking part of the experiment needs to be tested with other eye trackers in the

consumer market. However, with Pupil Labs, improvements can also be done by using 3D eye tracking, recording more data types for post-processing or other methods for detection of slippage.

When post-processing the data, we proposed our own method to analyze the internal calibration data to apply a modifier to each sample. For detection of fixation points, we used the window size and dispersion to filter the eye gazes, as proposed in [31]. For visualization, referring to [30], the Gaussian distribution was used to spread the error, and Gaussian blur was applied to the whole object at the end, in order to smooth the regions of interest. However, more methods for detection of fixation points and visualization can be tested.

Finally, our results show the common considerations when users are inspecting the point cloud. These could be used in further research or utilized when developing or processing new 3D contents.

References

[1] C. Tulvan, R. Mekuria, Z. Li, and S. Laserre, "Use cases for point cloud compression (pcc)," 2016.

[2] S. Perry, L. Cruz, M. Pereira, A. Pinheiro, E. Dumic, G. Lavoué, E. Alexiou, and T. Ebrahimi, "Overview of point cloud technology," 2019.

[3] D. S. Sinton, "Lidar key tool for preserving cultural heritage," Sep 2017. [Online]. Available: https://www.directionsmag.com/article/6807

[4] J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz, and E. Steinbach, "Real-time compression of point cloud streams," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 778–785.

[5] I. Xu, U. Cizmeci, and C. Steinbach, "Point-cloud-based model-mediated teleoperation," in *Haptic Audio Visual Environments and Games (HAVE), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 69–74.

[6] R. Mekuria, K. Blom, and P. Cesar, "Design, implementation, and evaluation of a point cloud codec for tele-immersive video," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 4, pp. 828–842, 2017.

[7] D. Thanou, P. A. Chou, and P. Frossard, "Graph-based compression of dynamic 3d point cloud sequences," *IEEE Transactions on Image Processing*, vol. 25, no. 4, pp. 1765–1778, 2016.

[8] T. Kawai, "3d displays and applications," *Displays*, vol. 23, no. 1-2, pp. 49–56, 2002.

[9] M. Levoy and P. Hanrahan, "Light field rendering," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 1996, pp. 31–42.

[10] R. H.-Y. Chen and T. D. Wilkinson, "Computer generated hologram from point cloud using graphics processor," *Applied optics*, vol. 48, no. 36, pp. 6841–6850, 2009.

[11] A. Zucconi, "A gentle introduction to shaders in unity3d," Jun 2015. [Online]. Available: http://www.alanzucconi.com/2015/06/10/a-gentle-introduction-to-shaders-in-unity3d/

[12] J. Liu, "Unity3d introduction to geometry shader," Jan 2018. [Online]. Available: https://jayjingyuliu.wordpress.com/2018/01/24/unity3d-intro-to-geometry-shader/

[13] Wikipedia, "Shader," Oct 2018. [Online]. Available: https://en.wikipedia.org/wiki/Shader#cite_note-5

[14] Keijiro, "Standard geometry shader," Nov 2017. [Online]. Available: https://github.com/keijiro/StandardGeometryShader

[15] Wikipedia, "Off(file format)," November 2018. [Online]. Available: https://en.wikipedia.org/wiki/OFF_(file_format)

[16] G. Llorach, "Point cloud free viewer," Apr 2018. [Online]. Available: https://assetstore.unity.com/packages/tools/utilities/point-cloud-free-viewer-19811

[17] Keijiro, "Pcx point cloud importer/render for unity," Jan 2019. [Online]. Available: https://github.com/keijiro/Pcx

[18] M. Schütz, "Potree: Rendering large point clouds in web browsers," Master's thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, Sep. 2016. [Online]. Available: https://www.cg.tuwien.ac.at/research/publications/2016/SCHUETZ-2016-POT/

[19] S. M. Fraiss, "Rendering large point clouds in unity," Favoritenstrasse 9-11/186, A-1040 Vienna, Austria, Sep. 2017. [Online]. Available: https://www.cg.tuwien.ac.at/research/publications/2017/FRAISS-2017-PCU/

[20] M. Schütz and M. Wimmer, "High-quality point based rendering using fast single pass interpolation," in *Proceedings of Digital Heritage 2015 Short Papers*, Sep. 2015, pp. 369–372. [Online]. Available: https://www.cg.tuwien.ac.at/research/publications/2015/SCHUETZ-2015-HQP/

[21] R. I.-R. BT, "Methodology for the subjective assessment of the quality of television pictures," 2002.

[22] E. Alexiou, E. Upenik, and T. Ebrahimi, "Towards subjective quality assessment of point cloud imaging in augmented reality," IEEE 19th Interational Workshop on Multimedia Signal Processing. Luton, UK: IEEE, Oct 2017.

[23] E. Alexiou and T. Ebrahimi, "On the performance of metrics to predict quality in point cloud representations," in *Applications of Digital Image Processing XL*, vol. 10396. International Society for Optics and Photonics, 2017, p. 103961H.

[24] E. Zerman, P. Gao, C. Ozcinar, and A. Smolic, "Subjective and objective quality assessment for volumetric video compression," 01 2019.

[25] U. Technologies, "Introduction to lighting and rendering." [Online]. Available: https://unity3d.com/learn/tutorials/topics/graphics/introduction-lighting-and-rendering?playlist=17102

[26] P. Labs, "Pupil docs." [Online]. Available: https://docs.pupil-labs.com/#pupil-capture

[27] ——, "Hmd-eyes releases," Oct 2018. [Online]. Available: https://github.com/pupil-labs/hmd-eyes/releases

[28] M. Kassner, W. Patera, and A. Bulling, "Pupil: An open source platform for pervasive eye tracking and mobile gaze-based interaction," April 2014. [Online]. Available: http://arxiv.org/abs/1405.0006

[29] P. Labs, "Hmd-eyes developer.md," Mar 2018. [Online]. Available: https://github.com/pupil-labs/hmd-eyes/blob/master/Developer.md

[30] G. Lavoué, F. Cordier, H. Seo, and M.-C. Larabi, "Visual attention for rendered 3d shapes," *Comput. Graph. Forum*, vol. 37, pp. 191–203, 2018.

[31] A. T. Duchowski, "Eye tracking methodology," *Theory and practice*, vol. 328, 2007.

8 Appendix

8.1 Square Shader

```
// Pcx - Point cloud importer & renderer for Unity
// https://github.com/keijiro/Pcx
// Modified by Xu Peisen, adding interpolation feature.

#include "UnityCG.cginc"
#include "Common.cginc"

half4 _Tint;
half _PointSizeScale;
float _PointSize;
float4x4 _Transform;
//float4x4 _InverseProjMatrix;

#if _COMPUTE_BUFFER
StructuredBuffer<float4> _PointBuffer;
#endif

int _UseInterpolation;
int _UseAdaptivePointSize;



// Vertex input attributes
struct Attributes
{
  float4 uv: TEXCOORD1;
#if _COMPUTE_BUFFER
    uint vertexID : SV_VertexID;
#else
    float4 position : POSITION;
    half3 color : COLOR;
#endif
};

// Fragment varyings
struct Varyings
{
    float4 position : SV_POSITION;
  float4 uv: TEXCOORD1;
#if !PCX_SHADOW_CASTER
    half3 color : COLOR;
    UNITY_FOG_COORDS(2)
#endif
};



struct VertOut
{
  float4 position : SV_POSITION;
  float4 viewposition: TEXCOORD1;
  float2 uv : TEXCOORD0;
  float size : POINTSIZE;
#if !PCX_SHADOW_CASTER
  half3 color : COLOR;
  UNITY_FOG_COORDS(2)
#endif
};
```

```
struct FragOut {
  half4 color : SV_Target;
  float depth : SV_Depth;
};

float4x4 inverse(float4x4 input)
{
#define minor(a,b,c) determinant(float3x3(input.a, input.b, input.c))

  float4x4 cofactors = float4x4(
    minor(_22_23_24, _32_33_34, _42_43_44),
    -minor(_21_23_24, _31_33_34, _41_43_44),
    minor(_21_22_24, _31_32_34, _41_42_44),
    -minor(_21_22_23, _31_32_33, _41_42_43),

    -minor(_12_13_14, _32_33_34, _42_43_44),
    minor(_11_13_14, _31_33_34, _41_43_44),
    -minor(_11_12_14, _31_32_34, _41_42_44),
    minor(_11_12_13, _31_32_33, _41_42_43),

    minor(_12_13_14, _22_23_24, _42_43_44),
    -minor(_11_13_14, _21_23_24, _41_43_44),
    minor(_11_12_14, _21_22_24, _41_42_44),
    -minor(_11_12_13, _21_22_23, _41_42_43),

    -minor(_12_13_14, _22_23_24, _32_33_34),
    minor(_11_13_14, _21_23_24, _31_33_34),
    -minor(_11_12_14, _21_22_24, _31_32_34),
    minor(_11_12_13, _21_22_23, _31_32_33)
    );
#undef minor
  return transpose(cofactors) / determinant(input);
}



// Vertex phase
Varyings Vertex(Attributes input)
{
    // Retrieve vertex attributes.
  half3 col;

#if _COMPUTE_BUFFER
  float4 pt = _PointBuffer[input.vertexID];
  float4 pos = mul(_Transform, float4(pt.xyz, 1));
  col = PcxDecodeColor(asuint(pt.w));
#else
  float4 pos = input.position;
  col = input.color;
#endif


#if !PCX_SHADOW_CASTER
    // Color space convertion & applying tint
    #if UNITY_COLORSPACE_GAMMA
        col *= _Tint.rgb * 2;
    #else
        col *= LinearToGammaSpace(_Tint.rgb) * 2;
        col = GammaToLinearSpace(col);
    #endif
#endif
```

```
123
    // Set vertex output.
125     Varyings o;
    o.position = UnityObjectToClipPos(pos);
127   o.uv = input.uv;
#if !PCX_SHADOW_CASTER
129     o.color = col;
    UNITY_TRANSFER_FOG(o, o.position);
131 #endif
    return o;
133 }

135 // Geometry phase
[maxvertexcount(4)]
137 void Geometry(point Varyings input[1], inout TriangleStream<VertOut> outStream)
{
139   float size;
  if (input[0].uv.x == 0 || _UseAdaptivePointSize < 0.5) {
141     size = 0.001;
  }
143   else {
    size = input[0].uv.x;
145   }

147     float4 origin = input[0].position;
  float2 extent = abs(UNITY_MATRIX_P._11_22 * size * _PointSize *
    _PointSizeScale);
149   float4x4 _InverseProjMatrix = inverse(UNITY_MATRIX_P);

151     Varyings o = input[0];
  VertOut vo;
153   vo.size = size;
#if !PCX_SHADOW_CASTER
155   vo.color = o.color;
  UNITY_TRANSFER_FOG(vo, o.position);
157 #endif

159     vo.position.y = origin.y + extent.y;
  vo.position.x = origin.x - extent.x;
161     vo.position.zw = origin.zw;
  vo.uv = float2(-1.0f, 1.0f);
163   vo.viewposition = mul(_InverseProjMatrix, vo.position);
    outStream.Append(vo);
165
  vo.position.y = origin.y + extent.y;
167   vo.position.x = origin.x + extent.x;
  vo.position.zw = origin.zw;
169   vo.uv = float2(1.0f, 1.0f);
  vo.viewposition = mul(_InverseProjMatrix, vo.position);
171   outStream.Append(vo);

173   vo.position.y = origin.y - extent.y;
  vo.position.x = origin.x - extent.x;
175   vo.position.zw = origin.zw;
  vo.uv = float2(-1.0f, -1.0f);
177   vo.viewposition = mul(_InverseProjMatrix, vo.position);
  outStream.Append(vo);
179
  vo.position.y = origin.y - extent.y;
181   vo.position.x = origin.x + extent.x;
  vo.position.zw = origin.zw;;
183   vo.uv = float2(1.0f, -1.0f);
  vo.viewposition = mul(_InverseProjMatrix, vo.position);
```

```
185   outStream.Append(vo);

187     //outStream.RestartStrip();
}
189


191
FragOut Fragment(VertOut input)
193 {
  FragOut o;
195   half4 c;
#if PCX_SHADOW_CASTER
197   c = 0;
#else
199     c = half4(input.color, _Tint.a);
    UNITY_APPLY_FOG(input.fogCoord, c);
201 #endif


203
  if (_UseInterpolation > 0.5) {
205     float uvlen = input.uv.x*input.uv.x + input.uv.y*input.uv.y;
    input.viewposition.z += (1 - uvlen) * input.size * _PointSize *
     _PointSizeScale;
207   }


209
  float4 pos = mul(UNITY_MATRIX_P, input.viewposition);
211   pos /= pos.w;
  //float4 pos = input.position;
213
  o.color = c;
215   o.depth = pos.z;
  return o;
217 }
```

8.2 Stage.cs

```
\\Created by Xu Peisen
2
using System.Collections;
4 using System.Collections.Generic;
using UnityEngine;
6 using System.IO;

8 public class Stage : MonoBehaviour {

10     // Use this for initialization
    private Bounds bound;
12     private GameObject shadowQuad;
    private GameObject shadowStageQuad;
14     private GameObject stage;
    public Vector2 quadPos;
16     public Vector2 quadScale;
    private Vector3 lightPos;
18     private bool enableStageShadow = true;
    private bool enableGroundShadowForStagedObj = false;
20     private float centerHeight = 0.8f;
    private float maxHeight = 1.7f;
22     private GameObject teleportLocked;
```

```
24      private ExperimentManager expManager;

26      private static StreamWriter sw;

28      [System.Serializable]
        public class PointCloudConfig
30      {
            public PointCloud[] pointClouds;
32      }

34      [System.Serializable]
        public class PointCloud
36      {
            public float height = 0f;
38          public float width = 0f;
            public int useStage = -1;
40      }


42
        private void OnEnable()
44      {
            expManager = FindObjectOfType<ExperimentManager>();
46
            bound = GetComponent<MeshRenderer>().bounds;
48
            string filePath = Application.dataPath + "/StreamingAssets/" +
    expManager.configFile + ".json";
50
            PointCloudConfig pcConfig = new PointCloudConfig();
52          if (File.Exists(filePath))
            {
54              string dataAsJson = File.ReadAllText(filePath);
                pcConfig = JsonUtility.FromJson<PointCloudConfig>(dataAsJson);
56          } else
            {
58              Debug.LogError("Config not found!");
                Application.Quit();
60          }


62



64
            if (pcConfig.pointClouds[ExperimentManager.index].height > 0f)
66          {
                float scale = (float)pcConfig.pointClouds[ExperimentManager.index].
    height / (bound.extents.y * 2f);
68              transform.localScale = transform.localScale * scale;
            }
70
            if (pcConfig.pointClouds[ExperimentManager.index].width > 0f)
72          {
                float length = Mathf.Max(bound.extents.x * 2f, bound.extents.z * 2f)
    ;
74              float scale = (float)pcConfig.pointClouds[ExperimentManager.index].
    width / length;
                transform.localScale = transform.localScale * scale;
76          }


78


80          bound = GetComponent<MeshRenderer>().bounds;

82          Vector3 horiOffset = new Vector3(-bound.center.x, 0f, -bound.center.z);
```

```
        transform.position += horiOffset;
84
        if (pcConfig.pointClouds[ExperimentManager.index].useStage > -1)
86      {
            if(pcConfig.pointClouds[ExperimentManager.index].useStage == 1)
88          {
                if (bound.extents.y * 2f < maxHeight)
90              {
                    float off = 0.5f+bound.extents.y-bound.center.y;
92                  //float off = maxHeight - (bound.center.y + bound.extents.y)
    ;
                    transform.position += Vector3.up * off;
94                  GameObject go = Resources.Load<GameObject>("Stage");
                    stage = Instantiate(go);
96                  //float offset = maxHeight - bound.extents.y * 2f;
                    stage.transform.localScale = new Vector3(bound.extents.x *
    (2f + 0.2f), 0.5f, bound.extents.z * (2f + 0.2f));
98                  stage.transform.position = Vector3.up * 0.5f / 2.0f;
                    //stage.transform.localScale = new Vector3(bound.extents.x *
     (2f + 0.2f), offset, bound.extents.z * (2f + 0.2f));
100                 //stage.transform.position = Vector3.up * offset / 2.0f;
                    if (enableGroundShadowForStagedObj)
102                 {
                        shadowQuad = Instantiate(Resources.Load<GameObject>("
    ShadowQuad"));
104                     MeshRenderer quadRenderer = shadowQuad.GetComponent<
    MeshRenderer>();
                        Texture tex = Resources.Load<Texture>("Materials/" +
    GetComponent<MeshFilter>().sharedMesh.name + "_shadowTexture");
106                     quadRenderer.material.mainTexture = tex;
                        shadowQuad.transform.localScale = new Vector3(bound.
    extents.x * 2f, bound.extents.z * 2f, 1f);
108                     shadowQuad.transform.position = new Vector3(0, 0.0001f,
    0);

110                 }
                    if (enableStageShadow)
112                 {

114                     shadowStageQuad = Instantiate(Resources.Load<GameObject
    >("ShadowStageQuad"));
                        MeshRenderer stageQuadRenderer = shadowStageQuad.
    GetComponent<MeshRenderer>();
116                     Texture stageTex = Resources.Load<Texture>("Materials/"
    + GetComponent<MeshFilter>().sharedMesh.name + "_shadowTexture");

118                     stageQuadRenderer.material.mainTexture = stageTex;

120                     shadowStageQuad.transform.position = new Vector3(0, 0.5f
     + 0.001f, 0);
                        //shadowStageQuad.transform.position = new Vector3(0,
    offset + 0.001f, 0);
122                     shadowStageQuad.transform.localScale = new Vector3(bound
    .extents.x * (2f + 0.2f), bound.extents.z * (2f + 0.2f), 1f);
                    }
124             } else
                {
126                 float offset = bound.extents.y - bound.center.y;
                    transform.position += Vector3.up * offset;
128                 shadowQuad = Instantiate(Resources.Load<GameObject>("
    ShadowQuad"));
                    MeshRenderer quadRenderer = shadowQuad.GetComponent<
    MeshRenderer>();
```

```
                    Texture tex = Resources.Load<Texture>("Materials/" +
GetComponent<MeshFilter>().sharedMesh.name + "_shadowTexture");
                    quadRenderer.material.mainTexture = tex;
                    shadowQuad.transform.localScale = new Vector3(bound.extents.
x * 2f, bound.extents.z * 2f, 1f);
                    shadowQuad.transform.position = new Vector3(0, 0.0001f, 0);
                }
            } else
            {
                float offset = bound.extents.y - bound.center.y;
                transform.position += Vector3.up * offset;
                shadowQuad = Instantiate(Resources.Load<GameObject>("ShadowQuad"
));
                MeshRenderer quadRenderer = shadowQuad.GetComponent<MeshRenderer
>();
                Texture tex = Resources.Load<Texture>("Materials/" +
GetComponent<MeshFilter>().sharedMesh.name + "_shadowTexture");
                quadRenderer.material.mainTexture = tex;
                shadowQuad.transform.localScale = new Vector3(bound.extents.x *
2f, bound.extents.z * 2f, 1f);
                shadowQuad.transform.position = new Vector3(0, 0.0001f, 0);
            }
        }
        else
        {
        if (bound.center.y < centerHeight)
            {
                float offset = centerHeight - transform.position.y;
                transform.position += Vector3.up * offset;
                GameObject go = Resources.Load<GameObject>("Stage");
                stage = Instantiate(go);
                stage.transform.localScale = new Vector3(bound.extents.x * (2f +
 0.2f), offset, bound.extents.z * (2f + 0.2f));
                stage.transform.position = Vector3.up * offset / 2.0f;
                if (enableGroundShadowForStagedObj)
                {
                    shadowQuad = Instantiate(Resources.Load<GameObject>("
ShadowQuad"));
                    MeshRenderer quadRenderer = shadowQuad.GetComponent<
MeshRenderer>();
                    Texture tex = Resources.Load<Texture>("Materials/" +
GetComponent<MeshFilter>().sharedMesh.name + "_shadowTexture");
                    quadRenderer.material.mainTexture = tex;
                    shadowQuad.transform.localScale = new Vector3(quadScale.x,
quadScale.y, 1f);
                    shadowQuad.transform.position = new Vector3(quadPos.x,
0.0001f, quadPos.y);

                }
                if (enableStageShadow)
                {
                    lightPos = FindObjectOfType<Light>().transform.position;
                    float lightHeight = lightPos.y;
                    float multiplier = (lightHeight - offset) / lightHeight;
                    shadowStageQuad = Instantiate(Resources.Load<GameObject>("
ShadowStageQuad"));
                    MeshRenderer stageQuadRenderer = shadowStageQuad.
GetComponent<MeshRenderer>();
                    Texture stageTex = Resources.Load<Texture>("Materials/" +
GetComponent<MeshFilter>().sharedMesh.name + "_shadowTexture");
                    Vector2 stageQuadPos = quadPos * multiplier;
                    Vector2 stageQuadScale = quadScale * multiplier;
                    stageQuadRenderer.material.mainTexture = stageTex;
```

```
178                     float tileX = Mathf.Min(bound.extents.x * (2f + 0.2f) /
    stageQuadScale.x, 1f);
                    float tileY = Mathf.Min(bound.extents.z * (2f + 0.2f) /
    stageQuadScale.y, 1f);
180
                    stageQuadRenderer.material.SetTextureScale("_MainTex", new
    Vector2(tileX, tileY));
182                 shadowStageQuad.transform.position = new Vector3(0, offset +
     0.001f, 0);
                    shadowStageQuad.transform.localScale = new Vector3(bound.
    extents.x * (2f + 0.2f), bound.extents.z * (2f + 0.2f), 1f);
184                 }

186             }
                else
188             {
                    shadowQuad = Instantiate(Resources.Load<GameObject>("ShadowQuad"
    ));
190                 MeshRenderer quadRenderer = shadowQuad.GetComponent<MeshRenderer
    >();
                    Texture tex = Resources.Load<Texture>("Materials/" +
    GetComponent<MeshFilter>().sharedMesh.name + "_shadowTexture");
192                 quadRenderer.material.mainTexture = tex;
                    shadowQuad.transform.localScale = new Vector3(quadScale.x,
    quadScale.y, 1f);
194                 shadowQuad.transform.position = new Vector3(quadPos.x, 0.0001f,
    quadPos.y);
                }
196         }

198         teleportLocked = GameObject.FindGameObjectWithTag("TeleportLocked");
            teleportLocked.transform.localScale = new Vector3(bound.extents.x * (2f
    + 0.2f),1f, bound.extents.z * (2f + 0.2f));
200
            StartCoroutine(RecordTransform());
202
        }
204
        IEnumerator RecordTransform()
206     {
            yield return new WaitForSeconds(0.5f);
208     ExpDataRecorder.RecordModifiedTransform(ExperimentManager.index, transform.
    position, transform.localScale.x);
        }
210


212     private void OnDisable()
        {
214         if (stage != null)
            {
216             Destroy(stage);
            }
218         if (shadowQuad != null)
            {
220             Destroy(shadowQuad);
            }
222         if(shadowStageQuad != null)
            {
224             Destroy(shadowStageQuad);
            }
226     }

228 }
```

8.3 ExperimentManager.cs

```
\\Created by Xu Peisen
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using System.IO;

public class ExperimentManager : MonoBehaviour {


    public string configFile;
    private string dataPath = "/StreamingAssets/";
    private PointCloudConfig pcConfig;
    private GameObject currentObject;
    public static int index;
    private int preindex;
    public static bool isDistorted;
    private bool preisdistorted;
    private int pointCloudCount;
    //public GameObject[] allColliders;
    public GameObject[] calibrationCube;
    public static bool internalCalibration;
    private int calibrationIndex;
    public string calibrationSceneName;
    private Camera cam;
    public string emptyRoomLightMapFolderName = "PointCloudExperiment_Empty";
    private List<Texture2D> lightDir = new List<Texture2D>();
    private List<Texture2D> lightmap = new List<Texture2D>();
    //public float accuracyMaxDist;
    private ExpDataRecorder exp;

    public static bool intCaliTriggered;

    public ExperimentMode currentMode;

    public enum ExperimentMode
    {
        single, dsis
    }

    [System.Serializable]
    public class PointCloud
    {
        public string original;
        public string distorted;
    }


    [System.Serializable]
    public class PointCloudConfig
    {
        public PointCloud[] pointClouds;
        public string expMode;
        public bool useInterpolation;
        public float pointSize;
        public byte[] tint;
```

```
57      }
59      private void OnEnable()
        {
61          cam = Camera.main;
            intCaliTriggered = false;
63      }
        // Use this for initialization
65      void Awake() {
            dataPath = dataPath + configFile + ".json";
67          string filePath = Application.dataPath + this.dataPath;
69          if (File.Exists(filePath))
            {
71              string dataAsJson = File.ReadAllText(filePath);
                pcConfig = JsonUtility.FromJson<PointCloudConfig>(dataAsJson);
73              if (pcConfig.expMode != null)
                {
75                  if (pcConfig.expMode.StartsWith("dsis"))
                    {
77                      currentMode = ExperimentMode.dsis;
                    }
79                  else
                    {
81                      currentMode = ExperimentMode.single;
                    }
83              } else
                {
85                  currentMode = ExperimentMode.single;
                }
87              MatConfig.useInterpolation = pcConfig.useInterpolation;
                MatConfig.pointSize = pcConfig.pointSize;
89              MatConfig.tint = pcConfig.tint;
                currentObject = null;
91              if (PupilManager.startNew)
                {
93                  index = 0;
                }
95              else
                {
97                  index = PlayerPrefs.GetInt("pcindex", 0);
                }
99              preindex = index;
                isDistorted = false;
101             preisdistorted = isDistorted;
                pointCloudCount = pcConfig.pointClouds.Length;
103             internalCalibration = false;
            }
105         else
            {
107             Debug.LogError("Config not found!");
                Application.Quit();
109         }
            CheckIndexUpdate();
111         Application.targetFrameRate = -1;
        }
113
        private void Start()
115     {
            //LoadLightMap();
117         exp = FindObjectOfType<ExpDataRecorder>();
        }
119
```

```
    // Update is called once per frame
    void Update() {

        if (Input.GetKeyDown(KeyCode.D)|| intCaliTriggered)
        {
            intCaliTriggered = false;
            if (!internalCalibration)
            {
                internalCalibration = true;
                currentObject.SetActive(false);
                calibrationIndex = 0;
            }
            if(calibrationIndex >= calibrationCube.Length)
            {
                internalCalibration = false;
                index++;
                Destroy(currentObject);
                isDistorted = false;
                CheckIndexUpdate();
                DisableAll();
            }
            if (internalCalibration)
            {
                calibrationCube[calibrationIndex].SetActive(true);
                ExpDataRecorder.internalCalibrationIndex = calibrationIndex;
                DisableOthers(calibrationCube[calibrationIndex]);
                calibrationIndex++;
            }
        }

        if (Input.GetKeyDown(KeyCode.A))
        {
            if (internalCalibration)
            {
                calibrationIndex--;
                if(calibrationIndex == 0)
                {
                    currentObject.SetActive(true);
                    calibrationCube[0].SetActive(false);
                    internalCalibration = false;
                } else
                {
                    calibrationCube[calibrationIndex-1].SetActive(true);
                    ExpDataRecorder.internalCalibrationIndex = calibrationIndex
- 1;
                    DisableOthers(calibrationCube[calibrationIndex-1]);
                }
            }
        }

        if (internalCalibration)
        {
            CheckCalibrationAccuracy(calibrationCube[calibrationIndex-1]);
        } else
        {
            CheckIndexUpdate();
        }


        if (Input.GetKeyDown(KeyCode.R))
        {
            this.gameObject.SetActive(false);
            PlayerPrefs.SetInt("pcindex", index + 1);
```

```
            SceneManager.LoadScene(calibrationSceneName, LoadSceneMode.Single);
        }
    }


    void CheckCalibrationAccuracy(GameObject cube)
    {
        Vector2 middlePos = exp.oneEyeCalibration_Right ? ExpDataRecorder.
    rightGazePos : (ExpDataRecorder.leftGazePos + ExpDataRecorder.rightGazePos)
    / 2f;
        Vector3 rightPoint = ExpDataRecorder.rightGazePos;
        Vector3 leftPoint = ExpDataRecorder.leftGazePos;
        Vector3 viewPortPoint = new Vector3(middlePos.x, middlePos.y, 2f);
        Vector3 viewPortPointRight = new Vector3(rightPoint.x, rightPoint.y, 2f)
    ;
        Vector3 viewPortPointLeft = new Vector3(leftPoint.x, leftPoint.y, 2f);
        Vector3 cubeWorldPos = cube.transform.position;
        if (cam != null)
        {
            Vector3 cubeVector = cubeWorldPos - cam.transform.position;
            Vector3 viewPortWorldPoint = cam.ViewportToWorldPoint(viewPortPoint)
    ;
            Vector3 viewPortWorldPointLeft = cam.ViewportToWorldPoint(
    viewPortPointLeft);
            Vector3 viewPortWorldPointRight = cam.ViewportToWorldPoint(
    viewPortPointRight);

            Vector3 gazeVector = viewPortWorldPoint - cam.transform.position;
            float angle = Mathf.Abs(Vector3.Angle(cubeVector, gazeVector));

            gazeVector = viewPortWorldPointLeft - cam.transform.position;
            float leftAngle = Mathf.Abs(Vector3.Angle(cubeVector, gazeVector));

            gazeVector = viewPortWorldPointRight - cam.transform.position;
            float rightAngle = Mathf.Abs(Vector3.Angle(cubeVector, gazeVector));

            ExpDataRecorder.internalCalibrationAccuracy = angle;
            ExpDataRecorder.internalCalibrationAccuracyLeft = leftAngle;
            ExpDataRecorder.internalCalibrationAccuracyRight = rightAngle;
            Debug.Log("middle:"+angle+", left:("+leftAngle+","+ExpDataRecorder.
    leftGazeConfidence+"), right:(" + rightAngle + "," + ExpDataRecorder.
    rightGazeConfidence +")");
        }
        else
        {
            Debug.LogError("camera not assigned");
        }
    }

    void DisableAll()
    {
        foreach (GameObject o in calibrationCube)
        {
            o.SetActive(false);
        }
    }
    void DisableOthers(GameObject obj)
    {
        foreach(GameObject o in calibrationCube)
        {
            if(o != obj)
            {
                o.SetActive(false);
```

```
            }
        }
    }

    void CheckIndexUpdate()
    {
        if (currentObject == null || index != preindex || preisdistorted != isDistorted)
        {
            if (currentObject != null)
            {
                Destroy(currentObject);
            }
            if (index >= pointCloudCount)
            {
                index = 0;
            }
            GameObject currentObjectRef = Resources.Load<GameObject>(isDistorted ? pcConfig.pointClouds[index].distorted : pcConfig.pointClouds[index].original);
            currentObject = Instantiate(currentObjectRef) as GameObject;
        }
        preindex = index;
        preisdistorted = isDistorted;
    }

}
```

8.4 ExpDataRecorder.cs

```
//Created by Xu Peisen

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO;

public class ExpDataRecorder : MonoBehaviour {

    private StreamWriter sw, swInternalCali;
    private static StreamWriter swModifiedTransform;
    private Camera mainCamera;
    private Vector2 gazePoint;
    private int sampleIndex;
    public bool oneEyeCalibration_Right;


    public static List<PupilPos> leftPupil = new List<PupilPos>();
    public static List<PupilPos> rightPupil = new List<PupilPos>();
    public static float leftTimeStamp;
    public static float rightTimeStamp;
    public static Vector2 leftGazePos;
    public static Vector2 rightGazePos;
    public static float leftGazeConfidence;
    public static float rightGazeConfidence;

    public static float internalCalibrationAccuracy;
    public static float internalCalibrationAccuracyLeft;
    public static float internalCalibrationAccuracyRight;
    public static int internalCalibrationIndex;
```

```
32      private const float epsilon = 0.01f;

34      private ExperimentManager expManager;

36      private string FolderName;
        private DirectoryInfo di;
38

40      public struct PupilPos
        {
42          public Vector2 value;
            public float timeStamp;
44      }

46      private void Awake()
        {
48          expManager = FindObjectOfType<ExperimentManager>();
            FolderName = "Experiment_" + expManager.configFile + "_" + System.
        DateTime.Now.ToString("yyyyMMddHHmmss");
50          di = Directory.CreateDirectory(Application.dataPath + "/Resources/
        RecordedData/" + FolderName);
            string path = di.FullName + "/" + FolderName + "_ModifiedTransform.txt";
52          swModifiedTransform = new StreamWriter(path, true);
            swModifiedTransform.WriteLine("pcIndex pos.x pos.y pos.z scale
        isDistorted");
54          swModifiedTransform.Flush();
        }
56

58      void OnEnable()
        {
60          mainCamera = Camera.main;

62          if (PupilTools.IsConnected)
            {
64              PupilTools.IsGazing = true;
                PupilTools.SubscribeTo("gaze");
66          }
        }
68      // Use this for initialization
        void Start () {
70

            string path = di.FullName + "/" + FolderName + "_EyeDataOutput.txt";
72          string pathInternalCalibration = di.FullName + "/" + FolderName + "
        _InternalCalibration.txt";
            //string path2 = di.FullName + "/" + FolderName + "_ExperimentOutput.txt
        ";
74          sampleIndex = 1;
            sw = new StreamWriter(path, true);
76          swInternalCali = new StreamWriter(pathInternalCalibration, true);
            swInternalCali.WriteLine("pcIndex middleAcc leftAcc leftCon rightAcc
        rightCon caliIndex");
78          swInternalCali.Flush();
            //sw2 = new StreamWriter(path2, true);
80          sw.WriteLine("FieldOfView " + mainCamera.fieldOfView + " TargetEye "+
                (oneEyeCalibration_Right?"Right":"Both") + " Mode "+ expManager.
        currentMode.ToString());
82          sw.WriteLine("SampleIndex PCIndex Timestamp CamPos.x CamPos.y CamPos.z
        CamRot.x CamRot.y CamRot.z LeftGazePos.x LeftGazePos.y RightGazePos.x
        RightGazePos.y LeftPupilPos.x LeftPupilPos.y RightPupilPos.x RightPupilPos.y
         LeftConfidence RightConfidence");
            sw.Flush();
```

```
    }

// Update is called once per frame
void Update () {
      int expIndex = ExperimentManager.index + 1;
      int pcIndex = ExperimentManager.isDistorted ? expIndex * 2 : expIndex *
 2 - 1;
      Vector2 rightGazePosNew = rightGazePos;
      Vector2 leftGazePosNew = leftGazePos;
      if (oneEyeCalibration_Right)
      {
          leftGazePos = new Vector2(-1f, -1f);
      }
      if (!ExperimentManager.internalCalibration)
      {
          StartCoroutine(RecordData(sampleIndex, pcIndex, Time.
 realtimeSinceStartup, mainCamera.transform.position, mainCamera.transform.
 eulerAngles, leftGazePosNew, rightGazePosNew, leftGazeConfidence,
 rightGazeConfidence, leftTimeStamp, rightTimeStamp));
          sampleIndex++;
      } else
      {
          swInternalCali.WriteLine(pcIndex + " " + internalCalibrationAccuracy
  + " " + internalCalibrationAccuracyLeft + " " + leftGazeConfidence + " " +
 internalCalibrationAccuracyRight + " " + rightGazeConfidence + " " +
 internalCalibrationIndex);
      }
    }



    IEnumerator RecordData(int sampleIndex, int pcIndex, float timestamp,
 Vector3 camPos, Vector3 camRot, Vector2 leftGazePoint, Vector2
 rightGazePoint,
      float leftConfidence, float rightConfidence, float leftTimeStamp, float
 rightTimeStamp)
    {
      yield return new WaitForSeconds(1f);
      Vector2 leftPupilPos = new Vector2(-1f, -1f);
      Vector2 rightPupilPos = new Vector2(-1f, -1f);
      for(int i = 0; i < leftPupil.Count; i++)
      {
          PupilPos pp = leftPupil[i];
          if(Mathf.Abs(pp.timeStamp-leftTimeStamp) < epsilon)
          {
              leftPupilPos = pp.value;
              leftPupil.RemoveRange(0, i + 1);
              break;
          }
      }

      for(int i = 0; i < rightPupil.Count; i++)
      {
          PupilPos pp = rightPupil[i];
          if (Mathf.Abs(pp.timeStamp - rightTimeStamp) < epsilon)
          {
              rightPupilPos = pp.value;
              rightPupil.RemoveRange(0, i + 1);
              break;
          }
      }
```

```
136         sw.WriteLine(sampleIndex + " " + pcIndex + " " + timestamp + " " +
    camPos.x + " " + camPos.y + " " + camPos.z + " " +
                camRot.x + " " + camRot.y + " " + camRot.z + " " +
138             leftGazePoint.x + " " + leftGazePoint.y + " " + rightGazePoint.x
     + " " +
                rightGazePoint.y + " " + leftPupilPos.x + " " + leftPupilPos.y +
     " " +
140             rightPupilPos.x + " " + rightPupilPos.y + " " + leftConfidence +
     " " + rightConfidence);
            sw.Flush();
142     }


144
    private void OnApplicationQuit()
146     {
            PupilTools.UnSubscribeFrom("gaze");
148         sw.Flush();
            sw.Close();
150         swInternalCali.Flush();
            swInternalCali.Close();
152         swModifiedTransform.Flush();
            swModifiedTransform.Close();
154         //sw2.Close();
    }
156
    private void OnDisable()
158     {
            PupilTools.UnSubscribeFrom("gaze");
160         sw.Flush();
            sw.Close();
162         swInternalCali.Flush();
            swInternalCali.Close();
164         swModifiedTransform.Flush();
            swModifiedTransform.Close();
166     }

168   public static void RecordModifiedTransform(int index, Vector3 position, float
      scale)
      {
170     swModifiedTransform.WriteLine(index + " " + position.x + " " + position.y +
      " " + position.z + " " + scale);
            swModifiedTransform.Flush();
172     }
}
```

8.5 DataFilter.cs

```
1 \\Created by Xu Peisen

3 using System.Collections;
  using System.Collections.Generic;
5 using UnityEngine;
  using System.IO;
7
  public class DataFilter
9 {

11   private StreamReader sr;
     private EyeDataProcess eyeProcess;
13
```

```
    public DataFilter(EyeDataProcess eyeProcess){
15    this.eyeProcess = eyeProcess;
    }
17

19  public EyeDataProcess.CalibrationModifier[] Initialise(){
      string recordedDataPath = "/Resources/RecordedData/" + eyeProcess.
      eyeDataFolderName;
21    string path = Application.dataPath + recordedDataPath + "/" + eyeProcess.
      eyeDataFolderName + "_InternalCalibration.txt";

23    sr = new StreamReader(path, true);
          string[] title;
25        if (!sr.EndOfStream)
          {
27            title = sr.ReadLine().Split(null);
          } else
29        {
              return null;
31        }

33    switch (title.Length) {
      case 2:
35    case 3:
        sr.DiscardBufferedData ();
37      sr.BaseStream.Seek (0, SeekOrigin.Begin);
        return ProcessMiddlePointData ();
39    case 7:
        return ProcessNormalData ();
41    default:
        return null;
43    }
    }
45

    private EyeDataProcess.CalibrationModifier[] ProcessMiddlePointData(){
47    int total = 0;
      List<EyeDataProcess.CalibrationModifier> modifiers = new List<EyeDataProcess
      .CalibrationModifier> ();
49    if(!sr.EndOfStream){
        string[] sampleData = sr.ReadLine().Split();
51      total++;
        int pcIndex = int.Parse(sampleData [0]);
53      int caliIndex = -1;
        if (sampleData.Length >= 3) {
55        caliIndex = int.Parse (sampleData [2]);
        }
57      int sampleCount = 0;
        float totalAccuracy = 0f;
59      Dictionary<int, int> preSampleCount = new Dictionary<int, int> ();
        Dictionary<int, float> preTotalAccuracy = new Dictionary<int, float> ();
61      EyeDataProcess.ModifierType type;
        float avgAcc;
63      EyeDataProcess.CalibrationModifier modifier;
        while(!sr.EndOfStream){
65
          float accuracy = float.Parse(sampleData [1]);
67
          if (accuracy <= eyeProcess.outlierThreashold) {
69          sampleCount++;
            totalAccuracy += accuracy;
71        }

73        sampleData = sr.ReadLine().Split();
```

```
        if(sampleData.Length > 1){
          int nextPcIndex = int.Parse(sampleData [0]);
          int nextCaliIndex = -1;
          if (sampleData.Length >= 3) {
            nextCaliIndex = int.Parse (sampleData [2]);
          }

          if (pcIndex != nextPcIndex || caliIndex != nextCaliIndex) {

            int count = sampleCount;
            float acc = totalAccuracy;
            int threshold = eyeProcess.validGazeDirSampleCountThreashold;

            if (preSampleCount.ContainsKey (caliIndex)) {
              count += preSampleCount [caliIndex];
              acc += preTotalAccuracy [caliIndex];
              threshold *= 2;
            }

            if (sampleData.Length >= 3) {
              switch (caliIndex) {
              case 0:
                type = EyeDataProcess.ModifierType.topLeft;
                break;
              case 1:
                type = EyeDataProcess.ModifierType.topRight;
                break;
              case 2:
                type = EyeDataProcess.ModifierType.middle;
                break;
              case 3:
                type = EyeDataProcess.ModifierType.bottomLeft;
                break;
              case 4:
                type = EyeDataProcess.ModifierType.bottomRight;
                break;
              default:
                Debug.LogError ("Unknown Calibration Cube Index!");
                type = EyeDataProcess.ModifierType.fullScreen;
                break;
              }

              if (count >= threshold) {
                avgAcc = acc / count;
              } else {
                avgAcc = -1f;
              }
            } else {
              type = EyeDataProcess.ModifierType.fullScreen;
              if (count >= threshold) {
                avgAcc = acc / count;
              } else {
                avgAcc = -1f;
              }
            }
                        eyeProcess.swLog.WriteLine("Folder: " + eyeProcess.
    eyeDataFolderName + " PCIndex:" + pcIndex + " CaliIndex:" + caliIndex + "
    Total:" + total + " Valid:" + sampleCount);
                        total = 0;
            modifier = new EyeDataProcess.CalibrationModifier (type,
    EyeDataProcess.EyeType.Both, avgAcc, pcIndex);
            modifiers.Add (modifier);
            if (preSampleCount.ContainsKey (caliIndex)) {
```

```
            preSampleCount [caliIndex] = sampleCount;
            preTotalAccuracy [caliIndex] = totalAccuracy;
          } else {
            preSampleCount.Add (caliIndex, sampleCount);
            preTotalAccuracy.Add (caliIndex, totalAccuracy);
          }
          sampleCount = 0;
          totalAccuracy = 0f;
        }

        pcIndex = nextPcIndex;
        caliIndex = nextCaliIndex;
        total++;
      } else {
        break;
      }
    }



    if (sampleData.Length >= 3) {
      int count = sampleCount;
      float acc = totalAccuracy;
      int threshold = eyeProcess.validGazeDirSampleCountThreashold;

      if (preSampleCount.ContainsKey (caliIndex)) {
        count += preSampleCount [caliIndex];
        acc += preTotalAccuracy [caliIndex];
        threshold *= 2;
      }
      switch (caliIndex) {
      case 0:
        type = EyeDataProcess.ModifierType.topLeft;
        break;
      case 1:
        type = EyeDataProcess.ModifierType.topRight;
        break;
      case 2:
        type = EyeDataProcess.ModifierType.middle;
        break;
      case 3:
        type = EyeDataProcess.ModifierType.bottomLeft;
        break;
      case 4:
        type = EyeDataProcess.ModifierType.bottomRight;
        break;
      default:
        Debug.LogError ("Unknown Calibration Cube Index!");
        type = EyeDataProcess.ModifierType.fullScreen;
        break;
      }
      if (count >= threshold) {
        avgAcc = acc / count;
      } else {
        avgAcc = -1f;
      }
    } else {
      int count = sampleCount;
      float acc = totalAccuracy;
      int threshold = eyeProcess.validGazeDirSampleCountThreashold;

      if (preSampleCount.ContainsKey (caliIndex)) {
        count += preSampleCount [caliIndex];
```

```
197           acc += preTotalAccuracy [caliIndex];
              threshold *= 2;
199         }
            type = EyeDataProcess.ModifierType.fullScreen;
201         if (count >= threshold) {
              avgAcc = acc / count;
203         } else {
              avgAcc = -1f;
205         }
          }
207       eyeProcess.swLog.WriteLine ("Folder: " + eyeProcess.eyeDataFolderName + "
      PCIndex:" + pcIndex + " CaliIndex:" + caliIndex + " Total:" + total + "
      Valid:" + sampleCount);
          modifier = new EyeDataProcess.CalibrationModifier (type, EyeDataProcess.
      EyeType.Both, avgAcc, pcIndex);
209       modifiers.Add (modifier);
          return modifiers.ToArray ();
211     } else {
          return null;
213     }
    }
215
217
    private EyeDataProcess.CalibrationModifier[] ProcessNormalData(){
219   List<EyeDataProcess.CalibrationModifier> modifiers = new List<EyeDataProcess
      .CalibrationModifier> ();
      if (!sr.EndOfStream) {
221     int total = 0;
        string[] sampleData = sr.ReadLine().Split();
223     total++;
        int pcIndex = int.Parse(sampleData [0]);
225     int caliIndex = int.Parse (sampleData [6]);
        int leftSampleCount = 0, rightSampleCount = 0, middleSampleCount = 0;
227     float leftTotalAccuracy = 0f, rightTotalAccuracy = 0f, middleTotalAccuracy
       = 0f;
        Dictionary<int, int> preLeftSampleCount = new Dictionary<int, int>(),
      preRightSampleCount = new Dictionary<int, int>(), preMiddleSampleCount = new
       Dictionary<int, int>();
229     Dictionary<int, float> preLeftTotalAccuracy = new Dictionary<int, float>
      (), preRightTotalAccuracy = new Dictionary<int, float> (),
      preMiddleTotalAccuracy = new Dictionary<int, float> ();
        EyeDataProcess.ModifierType type;
231     EyeDataProcess.EyeType eyeType;
        float avgAcc;
233     EyeDataProcess.CalibrationModifier modifier;
        while (!sr.EndOfStream) {
235       float middleAccuracy = float.Parse(sampleData [1]);
          float leftAccuracy = float.Parse(sampleData [2]);
237       float leftConfidence = float.Parse(sampleData [3]);
          float rightAccuracy = float.Parse(sampleData [4]);
239       float rightConfidence = float.Parse(sampleData [5]);

241       if (leftConfidence >= eyeProcess.calibrationConfidenceThreashold &&
      leftAccuracy <= eyeProcess.outlierThreashold) {
            leftSampleCount++;
243         leftTotalAccuracy += leftAccuracy;
          }
245
          if (rightConfidence >= eyeProcess.calibrationConfidenceThreashold &&
      rightAccuracy <= eyeProcess.outlierThreashold) {
247         rightSampleCount++;
            rightTotalAccuracy += rightAccuracy;
```

```
249         }
251         if (leftConfidence >= eyeProcess.calibrationConfidenceThreashold &&
               rightConfidence >= eyeProcess.calibrationConfidenceThreashold &&
    middleAccuracy <= eyeProcess.outlierThreashold) {
253           middleSampleCount++;
              middleTotalAccuracy += middleAccuracy;
255         }
257         sampleData = sr.ReadLine().Split();
            if (sampleData.Length > 1) {
259           int nextPcIndex = int.Parse (sampleData [0]);
              int nextCaliIndex = int.Parse (sampleData [6]);
261           if (pcIndex != nextPcIndex || caliIndex != nextCaliIndex) {
                switch (caliIndex) {
263               case 0:
                    type = EyeDataProcess.ModifierType.topLeft;
265                 break;
                  case 1:
267                 type = EyeDataProcess.ModifierType.topRight;
                    break;
269               case 2:
                    type = EyeDataProcess.ModifierType.middle;
271                 break;
                  case 3:
273                 type = EyeDataProcess.ModifierType.bottomLeft;
                    break;
275               case 4:
                    type = EyeDataProcess.ModifierType.bottomRight;
277                 break;
                  default:
279                 Debug.LogError ("Unknown Calibration Cube Index!");
                    type = EyeDataProcess.ModifierType.fullScreen;
281                 break;
                }
283
                int leftCount = leftSampleCount, rightCount = rightSampleCount,
    middleCount = middleSampleCount;
285             float leftAcc = leftTotalAccuracy, rightAcc = rightTotalAccuracy,
    middleAcc = middleTotalAccuracy;
                int threshold = eyeProcess.validGazeDirSampleCountThreashold;
287
                if (preLeftSampleCount.ContainsKey (caliIndex)) {
289               leftCount += preLeftSampleCount [caliIndex];
                  rightCount += preRightSampleCount [caliIndex];
291               middleCount += preMiddleSampleCount [caliIndex];
                  leftAcc += preLeftTotalAccuracy [caliIndex];
293               rightAcc += preRightTotalAccuracy [caliIndex];
                  middleAcc += preMiddleTotalAccuracy [caliIndex];
295               threshold *= 2;
                }
297
                if (middleCount >= threshold) {
299               avgAcc = middleAcc / middleCount;
                  eyeType = EyeDataProcess.EyeType.Both;
301             } else if (leftCount >= threshold || rightCount >= threshold) {
                  if (leftCount > rightCount * 1.2f) {
303                 avgAcc = leftAcc / leftCount;
                    eyeType = EyeDataProcess.EyeType.Left;
305               } else if (leftCount < rightCount * 1.2f) {
                    avgAcc = rightAcc / rightCount;
307                 eyeType = EyeDataProcess.EyeType.Right;
                  } else {
```

```
309                 if (leftAcc >= rightAcc) {
                      avgAcc = leftAcc / leftCount;
311                   eyeType = EyeDataProcess.EyeType.Left;
                    } else {
313                   avgAcc = rightAcc / rightCount;
                      eyeType = EyeDataProcess.EyeType.Right;
315                 }
                  }
317             } else {
                  avgAcc = -1f;
319               eyeType = EyeDataProcess.EyeType.Both;
                }
321
                eyeProcess.swLog.WriteLine ("Folder: " + eyeProcess.
    eyeDataFolderName + " PCIndex:" + pcIndex + " CaliIndex:" + caliIndex + "
    Total:" + total + " ValidLeft:" + leftSampleCount
323             + " ValidRight:" + rightSampleCount + " ValidMiddle:" +
    middleSampleCount);
                total = 0;
325             modifier = new EyeDataProcess.CalibrationModifier (type, eyeType,
    avgAcc, pcIndex);
                modifiers.Add (modifier);
327             if (!preLeftSampleCount.ContainsKey (caliIndex)) {
                  preLeftSampleCount.Add (caliIndex, leftSampleCount);
329               preRightSampleCount.Add (caliIndex, rightSampleCount);
                  preMiddleSampleCount.Add (caliIndex, middleSampleCount);
331               preLeftTotalAccuracy.Add (caliIndex, leftTotalAccuracy);
                  preRightTotalAccuracy.Add (caliIndex, rightTotalAccuracy);
333               preMiddleTotalAccuracy.Add (caliIndex, middleTotalAccuracy);
                } else {
335               preLeftSampleCount [caliIndex] = leftSampleCount;
                  preRightSampleCount [caliIndex] = rightSampleCount;
337               preMiddleSampleCount [caliIndex] = middleSampleCount;
                  preLeftTotalAccuracy [caliIndex] = leftTotalAccuracy;
339               preRightTotalAccuracy [caliIndex] = rightTotalAccuracy;
                  preMiddleTotalAccuracy [caliIndex] = middleTotalAccuracy;
341             }
                leftSampleCount = 0;
343             rightSampleCount = 0;
                middleSampleCount = 0;
345             leftTotalAccuracy = 0f;
                rightTotalAccuracy = 0f;
347             middleTotalAccuracy = 0f;
              }
349           pcIndex = nextPcIndex;
              caliIndex = nextCaliIndex;
351           total++;
            } else {
353           break;
            }
355       }
          switch (caliIndex) {
357       case 0:
            type = EyeDataProcess.ModifierType.topLeft;
359         break;
          case 1:
361         type = EyeDataProcess.ModifierType.topRight;
            break;
363       case 2:
            type = EyeDataProcess.ModifierType.middle;
365         break;
          case 3:
367         type = EyeDataProcess.ModifierType.bottomLeft;
```

```
        break;
      case 4:
        type = EyeDataProcess.ModifierType.bottomRight;
        break;
      default:
        Debug.LogError ("Unknown Calibration Cube Index!");
        type = EyeDataProcess.ModifierType.fullScreen;
        break;
      }
      int lCount = leftSampleCount, rCount = rightSampleCount, mCount =
    middleSampleCount;
      float lAcc = leftTotalAccuracy, rAcc = rightTotalAccuracy, mAcc =
    middleTotalAccuracy;
      int t = eyeProcess.validGazeDirSampleCountThreashold;

      if (preLeftSampleCount.ContainsKey (caliIndex)) {
        lCount += preLeftSampleCount [caliIndex];
        rCount += preRightSampleCount [caliIndex];
        mCount += preMiddleSampleCount [caliIndex];
        lAcc += preLeftTotalAccuracy [caliIndex];
        rAcc += preRightTotalAccuracy [caliIndex];
        mAcc += preMiddleTotalAccuracy [caliIndex];
        t *= 2;
      }

      if (mCount >= t) {
        avgAcc = mAcc / mCount;
        eyeType = EyeDataProcess.EyeType.Both;
      } else if (lCount >= t || rCount >= t) {
        if (lCount > rCount * 1.2f) {
          avgAcc = lAcc / lCount;
          eyeType = EyeDataProcess.EyeType.Left;
        } else if (lCount < rCount * 1.2f) {
          avgAcc = rAcc / rCount;
          eyeType = EyeDataProcess.EyeType.Right;
        } else {
          if (lAcc >= rAcc) {
            avgAcc = lAcc / lCount;
            eyeType = EyeDataProcess.EyeType.Left;
          } else {
            avgAcc = rAcc / rCount;
            eyeType = EyeDataProcess.EyeType.Right;
          }
        }
      } else {
        avgAcc = -1f;
        eyeType = EyeDataProcess.EyeType.Both;
      }
            eyeProcess.swLog.WriteLine("Folder: " + eyeProcess.eyeDataFolderName
     + " PCIndex:" + pcIndex + " CaliIndex:" + caliIndex + " Total:" + total + "
     ValidLeft:" + leftSampleCount
                        + " ValidRight:" + rightSampleCount + " ValidMiddle:" +
    middleSampleCount);
            modifier = new EyeDataProcess.CalibrationModifier (type, eyeType,
    avgAcc, pcIndex);
      modifiers.Add (modifier);
      return modifiers.ToArray ();
    } else {
      return null;
    }
  }
}
```

8.6 EyeDataProcess.cs

```
\\Created by Xu Peisen
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO;
using System;
using System.Linq;



public class EyeDataProcess : MonoBehaviour {

  //file reader
    public string configFile;
    public string eyeDataFolderName;
    private string RecordedDataPath;
    private PointCloudConfig pcConfig;
    private StreamReader sr;

    private StreamWriter sw;
  private StreamReader srTransformData;
    [HideInInspector]
    public StreamWriter swLog;

  //current PC
    private string currentObject;
  private string currentDistortedObject;
    private List<Vector3> currentPointCloud = new List<Vector3>();
  private List<Vector3> currentDistortedPointCloud = new List<Vector3> ();
    private List<float> currentPointGazeImportance = new List<float>();
  private List<float> currentDistortedPointGazeImportance = new List<float> ();
    private int currentIndex;
  private List<Vector3> pointWithinRange = new List<Vector3>();
  private List<int> pointIndices = new List<int>();

  //sample info
    int sampleIndex;
    int pcIndex;
    float timeStamp;
    Vector3 camPos;
    Vector3 camRot;
    Vector2 leftGazePoint;
    Vector2 rightGazePoint;
    float leftConfidence;
    float rightConfidence;


    private ExperimentManager.ExperimentMode expMode;

  //line Counter
    private bool next;
  private int lineCounter;
  private int sampleCounter;

  //cam config
    private float fov;
  public Camera cam;

  //processing param
    public float globalAngleThreshold = 0.5f;
```

```
62      public float acceptingDepthRange;

64    //fixation param
      public float confidenceThreashold = 0.6f;
66    public float fixationMaxDispersion = 3f;
      public float fixationMinInterval = 0.003f;
68
      //fixation list
70      private List<GazePoint> gazePoints = new List<GazePoint>();
        private List<GazePoint> registeredGazePoint = new List<GazePoint>();
72
      //Data Filter Param
74    public float calibrationConfidenceThreashold = 0.6f;
      public float outlierThreashold = 5f;
76    public int validGazeDirSampleCountThreashold = 100;

78    //internal cali modifier
      public enum ModifierType{
80      topLeft,
        topRight,
82      middle,
        bottomLeft,
84      bottomRight,
        fullScreen
86    };

88    public enum EyeType {
        Left,
90      Right,
        Both
92    };

94    private CalibrationModifier[] modifiers;

96    public class CalibrationModifier{

98      public float accuracy;
        public EyeType eyeType;
100     public int pcIndex;
        public ModifierType type;
102
        public CalibrationModifier(ModifierType type, EyeType eyeType, float
       accuracy, int pcIndex){
104       this.type = type;
          this.eyeType = eyeType;
106       this.accuracy = accuracy;
          this.pcIndex = pcIndex;
108     }
      }
110
      private static class CaliGazePos
112   {
        public static Vector2 topLeft = new Vector2 (0.4188f, 0.6756f);
114     public static Vector2 topRight = new Vector2 (0.5812f, 0.6756f);
        public static Vector2 middle = new Vector2 (0.5000f, 0.4506f);
116     public static Vector2 bottomLeft = new Vector2 (0.4188f, 0.2256f);
        public static Vector2 bottomRight = new Vector2 (0.5812f, 0.2256f);
118     public const float middleRadius = 0.15f;
      }
120
      private List<ModifiedTransform> modifiedTransforms = new List<
       ModifiedTransform>();
122
```

```
class ModifiedTransform{
  public int index;
  public Vector3 position;
  public float scale;

  public ModifiedTransform(int index, Vector3 position, float scale){
    this.index = index;
    this.position = position;
    this.scale = scale;
  }
}

  class GazePoint
  {
      public Vector3 gazePointWorld;
      public Vector3 camPos;
      public float timeStamp;
  public Vector2 gazePointViewport;

  public GazePoint(Vector3 gazePointWorld, Vector3 camPos, float timeStamp,
   Vector2 gazePointViewport)
      {
          this.gazePointWorld = gazePointWorld;
          this.camPos = camPos;
          this.timeStamp = timeStamp;
    this.gazePointViewport = gazePointViewport;
      }
  }

  [System.Serializable]
  public class PointCloudConfig
  {
      public PointCloud[] pointClouds;
  }
  [System.Serializable]
  public class PointCloud
  {
      public string original;
      public string distorted;
  }

  private bool firstStart;

  private void Awake()
  {
      string oPath = Application.dataPath + RecordedDataPath + "/Result";
      string path = oPath;
      int duplicated = 1;
      while (File.Exists(path + ".txt"))
      {
          path = oPath + "(" + duplicated + ")";
          duplicated++;
      }
      path = path + ".txt";
      File.WriteAllText(path, string.Empty);
      swLog = new StreamWriter(path, true);
      swLog.AutoFlush = true;
  }

  // Use this for initialization
  void OnEnable()
  {
      next = true;
```

```
        firstStart = true;
186     registeredGazePoint.Clear();
        gazePoints.Clear();
188     RecordedDataPath = "/Resources/RecordedData/" + eyeDataFolderName;
        string configPath = "/StreamingAssets/" + configFile + ".json";
190     configPath = Application.dataPath + configPath;
        string eyeDataPath = Application.dataPath + RecordedDataPath + "/" +
    eyeDataFolderName + "_EyeDataOutput.txt";
192 string transformDataPath = Application.dataPath + RecordedDataPath + "/" +
    eyeDataFolderName + "_ModifiedTransform.txt";

194

        if (File.Exists(configPath))
196     {
            string dataAsJson = File.ReadAllText(configPath);
198         pcConfig = JsonUtility.FromJson<PointCloudConfig>(dataAsJson);
        }
200     else
        {
202         Debug.LogError("Config not found!");
            Application.Quit();
204     }

206 //Initialise Modifiers
    DataFilter filter = new DataFilter (this);
208 modifiers = filter.Initialise ();
    if (modifiers == null) {
210   Debug.LogWarning ("No Internal CalibrationData!");
            modifiers = new CalibrationModifier[0];
212 }
    foreach(CalibrationModifier modifier in modifiers){
214   Debug.LogWarning ("Folder: " + eyeDataFolderName + " modifier details: " +
     modifier.accuracy + " " + modifier.eyeType + " " + modifier.type + " " +
    modifier.pcIndex);
            swLog.WriteLine("Folder: " + eyeDataFolderName + " modifier details:
     " + modifier.accuracy + " " + modifier.eyeType + " " + modifier.type + " "
    + modifier.pcIndex);
216 }

218


220     if (File.Exists(eyeDataPath))
        {
222         sr = new StreamReader(eyeDataPath, true);

224     } else {
            Debug.LogError("Eye Data Not Found!");
226         Application.Quit();
        }
228
        lineCounter = 0;
230 sampleCounter = 0;
        string[] str = sr.ReadLine().Split(null);
232     lineCounter++;
        currentIndex = 1;
234     if (str.Length >= 2)
        {
236         fov = float.Parse(str[1]);
            cam.fieldOfView = fov;
238     } else
        {
240         Debug.LogWarning("FOV Not Found! Using Default.");
            cam.fieldOfView = 109.9064f;
```

```
        }
        if (str.Length >= 6)
        {
            if (str[5].StartsWith("single")) {
                expMode = ExperimentManager.ExperimentMode.single;
            } else
            {
                expMode = ExperimentManager.ExperimentMode.dsis;
            }
        } else
        {
            expMode = ExperimentManager.ExperimentMode.single;
        }
        sr.ReadLine();
        lineCounter++;
        modifiedTransforms.Clear();
    if (File.Exists (transformDataPath)) {
      srTransformData = new StreamReader (transformDataPath, true);
      srTransformData.ReadLine ();
      while (!srTransformData.EndOfStream) {
        string[] st = srTransformData.ReadLine ().Split(null);
        if (st.Length >= 5) {
          bool addNew = true;
          foreach (ModifiedTransform m in modifiedTransforms) {
            if(m.index == int.Parse(st[0])){
              addNew = false;
            }
          }
          if (addNew) {
            ModifiedTransform modifiedTransform = new ModifiedTransform (int.
    Parse (st [0]), new Vector3 (float.Parse (st [1]), float.Parse (st [2]),
    float.Parse (st [3])), float.Parse (st [4]));
            modifiedTransforms.Add (modifiedTransform);
          }
        }
      }
    } else {
      Debug.LogError ("Transform Data Not Found!");
      Application.Quit ();
    }

    }

    void Update()
    {
        if (next)
        {
            next = false;
            LoadNextLine();
        }
    }

    private void WritePointCloud()
    {
    currentIndex = (currentIndex % 2 == 0) ? (currentIndex - 1) : currentIndex;
    string oPath = Application.dataPath + RecordedDataPath + "/" +
    eyeDataFolderName + "_pointCloud_"+currentIndex + "_" + currentObject;
        string path = oPath;
        int duplicated = 1;
        while (File.Exists(path+".txt"))
        {
            path = oPath + "(" + duplicated + ")";
            duplicated++;
```

```
302         }
            path = path + ".txt";
304         File.WriteAllText(path, string.Empty);
            sw = new StreamWriter(path, true);
306         sw.WriteLine("PosX PosY PosZ GazeCount");
            sw.Flush();
308         for(int i = 0; i < currentPointCloud.Count; i++)
            {
310             sw.WriteLine(currentPointCloud[i].x + " " + currentPointCloud[i].y +
     " " + currentPointCloud[i].z + " " + currentPointGazeImportance[i]);
                sw.Flush();
312         }
            sw.Dispose();
314     if (expMode == ExperimentManager.ExperimentMode.dsis) {
       string oDistortedPath = Application.dataPath + RecordedDataPath + "/" +
    eyeDataFolderName + "_pointCloud_"+(currentIndex+1) + "_" +
    currentDistortedObject;
316    string distortedPath = oDistortedPath;
       int du = 1;
318    while (File.Exists(distortedPath+".txt"))
       {
320      distortedPath = oDistortedPath + "(" + du + ")";
         du++;
322    }
       distortedPath = distortedPath + ".txt";
324    File.WriteAllText(distortedPath, string.Empty);
       sw = new StreamWriter(distortedPath, true);
326    sw.WriteLine("PosX PosY PosZ GazeCount");
       sw.Flush();
328    for(int i = 0; i < currentDistortedPointCloud.Count; i++)
       {
330      sw.WriteLine(currentDistortedPointCloud[i].x + " " +
    currentDistortedPointCloud[i].y + " " + currentDistortedPointCloud[i].z + "
    " + currentDistortedPointGazeImportance[i]);
         sw.Flush();
332    }
            sw.Dispose();
334         }
        }
336
        private void LoadPointCloud(int index)
338     {
            currentPointCloud.Clear();
340         currentPointGazeImportance.Clear();
            currentDistortedPointCloud.Clear();
342         currentDistortedPointGazeImportance.Clear();
            //bool isDistorted = (currentIndex % 2 == 0);
344         currentObject = pcConfig.pointClouds[index].original;
     string path = Application.dataPath + "/Resources/" + currentObject + ".ply";
346         if (File.Exists(path))
            {
348             var stream = File.Open(path, FileMode.Open, FileAccess.Read,
    FileShare.Read);
                var header = ReadDataHeader(new StreamReader(stream));
350             var body = ReadDataBody(header, new BinaryReader(stream));
                //Transform modifiedTransform = Resources.Load<GameObject>(
    isDistorted ? pcConfig.pointClouds[index].distorted : pcConfig.pointClouds[
    index].original).transform;
352
                for (int i = 0; i < body.vertices.Count; i++)
354             {
                    Vector3 v = body.vertices[i];
356         foreach (ModifiedTransform modifiedTransform in modifiedTransforms) {
```

```
          if (modifiedTransform.index == index) {
358         v *= modifiedTransform.scale;
            v += modifiedTransform.position;
360       }
        }
362             currentPointCloud.Add(v);
                currentPointGazeImportance.Add(0f);
364       }
        }
366     else
        {
368         Debug.LogWarning("currentObject Not Found: " + currentObject);
        }
370
    if (expMode == ExperimentManager.ExperimentMode.dsis) {
372   currentDistortedObject = pcConfig.pointClouds [index].distorted;
      string distortedPath = Application.dataPath + "/Resources/" +
    currentDistortedObject + ".ply";
374   if (File.Exists(distortedPath))
      {
376     var stream = File.Open(distortedPath, FileMode.Open, FileAccess.Read,
    FileShare.Read);
        var header = ReadDataHeader(new StreamReader(stream));
378     var body = ReadDataBody(header, new BinaryReader(stream));
        //Transform modifiedTransform = Resources.Load<GameObject>(isDistorted ?
     pcConfig.pointClouds[index].distorted : pcConfig.pointClouds[index].
    original).transform;
380
        for (int i = 0; i < body.vertices.Count; i++)
382     {
          Vector3 v = body.vertices[i];
384       foreach (ModifiedTransform modifiedTransform in modifiedTransforms) {
            if (modifiedTransform.index == index) {
386           v *= modifiedTransform.scale;
              v += modifiedTransform.position;
388         }
          }
390       currentDistortedPointCloud.Add(v);
          currentDistortedPointGazeImportance.Add(0f);
392     }
      }
394   else
      {
396     Debug.LogWarning("currentDistortedObject Not Found: " + currentObject);
      }
398 }
    }
400
    private void LoadNextLine()
402 {
    if (!sr.EndOfStream) {
404   try {
        lineCounter++;
406     string[] str = sr.ReadLine ().Split (null);
        sampleIndex = int.Parse (str [0]);
408     pcIndex = int.Parse (str [1]);
        if ((int)Mathf.Ceil(currentIndex/2f) == (int)Mathf.Ceil(pcIndex/2f)) {
410       if(firstStart){
                    firstStart = false;
412       int index = (int)Mathf.Ceil((float)currentIndex / 2.0f) - 1;
          if (index < pcConfig.pointClouds.Length)
414       {
            LoadPointCloud(index);
```

```
416             }
            }
418             timeStamp = float.Parse (str [2]);
            camPos = new Vector3 (float.Parse (str [3]), float.Parse (str [4]),
    float.Parse (str [5]));
420             camRot = new Vector3 (float.Parse (str [6]), float.Parse (str [7]),
    float.Parse (str [8]));
            leftGazePoint = new Vector2 (float.Parse (str [9]), float.Parse (str
    [10]));
422             rightGazePoint = new Vector2 (float.Parse (str [11]), float.Parse (str
     [12]));
            leftConfidence = float.Parse (str [17]);
424             rightConfidence = float.Parse (str [18]);
          } else {
426             if(currentPointCloud.Count > 1){
              WritePointCloud ();
428             }
            if(pcIndex > currentIndex-2){
430             currentIndex = pcIndex;
              int index = (int)Mathf.Ceil ((float)currentIndex / 2.0f) - 1;
432             LoadPointCloud (index);
            } else {
434             sr.Close ();
              swLog.WriteLine("Folder: " + eyeDataFolderName + " Total:" + (
    lineCounter-2) + " Valid: " + sampleCounter);
436             gameObject.SetActive (false);
            }
438             next = true;
            return;
440         }
        } catch (Exception e) {
442         Debug.LogWarning ("Line " + lineCounter + " corrupted!");
          next = true;
444       }

446       if (!next) {
            Debug.Log (sampleIndex);
448         if (leftConfidence > confidenceThreashold || rightConfidence >
    confidenceThreashold) {

450           Vector2 gazeMiddle = Vector2.zero;
            if (leftConfidence > confidenceThreashold && rightConfidence >
    confidenceThreashold) {
452             gazeMiddle = (leftGazePoint + rightGazePoint) / 2f;
            } else if (leftConfidence > confidenceThreashold) {
454             gazeMiddle = leftGazePoint;
            } else if (rightConfidence > confidenceThreashold) {
456             gazeMiddle = rightGazePoint;
            }
458           CalibrationModifier currentModifier = FindCurrentModifier(gazeMiddle);
            if (currentModifier != null && currentModifier.accuracy > 0f) {
460             if (currentModifier.eyeType == EyeType.Left) {
                if (leftConfidence > confidenceThreashold) {
462               gazeMiddle = leftGazePoint;
                } else {
464               next = true;
                  return;
466             }
              } else if (currentModifier.eyeType == EyeType.Right) {
468             if (rightConfidence > confidenceThreashold) {
                  gazeMiddle = rightGazePoint;
470             } else {
                  next = true;
```

```
472                 return;
                  }
474             }
                CalculateFixation (gazeMiddle);
476             sampleCounter++;
              }
478         }
        }
480
    } else {
482     WritePointCloud ();
        sr.Close ();
484             swLog.WriteLine("Folder: " + eyeDataFolderName + " Total:" + (
  lineCounter - 2) + " Valid: " + sampleCounter);
            gameObject.SetActive (false);
486 }
        next = true;
488 }

490 CalibrationModifier FindCurrentModifier(Vector2 gazePos){
        float[] dists =
492     {
            (gazePos - CaliGazePos.middle).magnitude,
494         (gazePos - CaliGazePos.topLeft).magnitude,
            (gazePos - CaliGazePos.topRight).magnitude,
496         (gazePos - CaliGazePos.bottomLeft).magnitude,
            (gazePos - CaliGazePos.bottomRight).magnitude,
498     };

500     int minIndex = Array.IndexOf(dists, dists.Min());

502     switch (minIndex)
        {
504         case 0:
                foreach (CalibrationModifier modifier in modifiers)
506             {
                    if (modifier.pcIndex == currentIndex && (modifier.type ==
  ModifierType.middle || modifier.type == ModifierType.fullScreen))
508                 {
                        return modifier;
510                 }
                }
512             break;
            case 1:
514             foreach (CalibrationModifier modifier in modifiers)
                {
516                 if (modifier.pcIndex == currentIndex && (modifier.type ==
  ModifierType.topLeft || modifier.type == ModifierType.fullScreen))
                    {
518                     return modifier;
                    }
520             }
                break;
522         case 2:
                foreach (CalibrationModifier modifier in modifiers)
524             {
                    if (modifier.pcIndex == currentIndex && (modifier.type ==
  ModifierType.topRight || modifier.type == ModifierType.fullScreen))
526                 {
                        return modifier;
528                 }
                }
530             break;
```

```
            case 3:
                foreach (CalibrationModifier modifier in modifiers)
                {
                    if (modifier.pcIndex == currentIndex && (modifier.type ==
  ModifierType.bottomLeft || modifier.type == ModifierType.fullScreen))
                    {
                        return modifier;
                    }
                }
                break;
            case 4:
                foreach (CalibrationModifier modifier in modifiers)
                {
                    if (modifier.pcIndex == currentIndex && (modifier.type ==
  ModifierType.bottomRight || modifier.type == ModifierType.fullScreen))
                    {
                        return modifier;
                    }
                }
                break;
            default:
                return null;

        }

        return null;
    }

    void CalculateFixation(Vector2 gazeMiddle){
      cam.transform.position = camPos;
      cam.transform.eulerAngles = camRot;
      Vector3 gazePointWorldPos = cam.ViewportToWorldPoint(new Vector3(gazeMiddle.
    x, gazeMiddle.y, 1f));
      gazePoints.Add(new GazePoint(gazePointWorldPos, camPos, timeStamp,
    gazeMiddle));
      if(gazePoints[gazePoints.Count-1].timeStamp - gazePoints[0].timeStamp >
    fixationMinInterval)
      {
        GazePoint gpToRegister = gazePoints[0];
        gazePoints.RemoveAt(0);
        registeredGazePoint.Add(gpToRegister);
        if(registeredGazePoint[registeredGazePoint.Count - 1].timeStamp -
    registeredGazePoint[0].timeStamp > fixationMinInterval)
        {
          Vector3 avgCamPos = Vector3.zero;
          Vector3 avgGazePos = Vector3.zero;
          Vector2 avgGazePosViewport = Vector2.zero;
          for(int i = 0; i < registeredGazePoint.Count-1; i++)
          {
            avgCamPos += registeredGazePoint[i].camPos / (registeredGazePoint.
    Count - 1);
            avgGazePos += registeredGazePoint[i].gazePointWorld / (
    registeredGazePoint.Count - 1);
            avgGazePosViewport += registeredGazePoint[i].gazePointViewport/(
    registeredGazePoint.Count - 1);
          }
          registeredGazePoint.RemoveRange(0, registeredGazePoint.Count - 1);
          Vector3 gazeRay = avgGazePos - avgCamPos;

          CalibrationModifier currentModifier = FindCurrentModifier(
    avgGazePosViewport);
                if (currentModifier != null)
                {
```

```
584                     RegisterPoints(gazeRay, avgCamPos, currentModifier.accuracy)
    ;
                } else
                {
                    Debug.LogWarning("Current Modifier Not Found!");
                }
      }
    }

    if (gazePoints.Count > 1)
    {
      Vector3 avgCamPos = Vector3.zero;
      foreach (GazePoint gp in gazePoints)
      {
        avgCamPos += gp.camPos / gazePoints.Count;
      }

      Vector3 ray1 = gazePoints[gazePoints.Count - 1].gazePointWorld - avgCamPos
    ;
      for (int i = 0; i < (gazePoints.Count-1); i++)
      {
        Vector3 ray2 = gazePoints[i].gazePointWorld - avgCamPos;
        float angle = Mathf.Abs(Vector3.Angle(ray1, ray2));
        if(angle > fixationMaxDispersion)
        {
          gazePoints.Clear();
          break;
        }
      }
    }
  }

  void RegisterPoints(Vector3 gazeRay, Vector3 camPos, float
    currentAngleThreshold)
    {
        pointWithinRange.Clear();
        pointIndices.Clear();
    float angleThreshold = Mathf.Max (globalAngleThreshold,
    currentAngleThreshold);
        float minDistance = float.MaxValue;
        Vector3 closestPoint = new Vector3(0f, 0f, 0f);

    if (expMode == ExperimentManager.ExperimentMode.dsis && currentIndex % 2 ==
    0) {

      for (int i = 0; i < currentDistortedPointCloud.Count; i++) {
        Vector3 point = currentDistortedPointCloud [i];
        Vector3 dir = point - camPos;
        float angleInDegree = Mathf.Abs (Vector3.Angle (gazeRay, dir));

        if (angleInDegree < angleThreshold) {
          pointWithinRange.Add (point);
          pointIndices.Add (i);
          float distance = Mathf.Abs (Vector3.Dot (dir, gazeRay) / gazeRay.
    magnitude);
          if (distance < minDistance) {
            minDistance = distance;
            closestPoint = point;
          }
        }
      }

      List<int> recordedPointIndices = new List<int> ();
```

```
642       if (pointWithinRange.Count > 0) {
            for (int i = 0; i < pointWithinRange.Count; i++) {
644           Vector3 point = pointWithinRange [i];
              Vector3 diffvec = point - closestPoint;
646
              float depth = Mathf.Abs (Vector3.Dot (gazeRay, diffvec) / gazeRay.
    magnitude);
648           if (depth < acceptingDepthRange) {
                recordedPointIndices.Add (pointIndices [i]);
650             //currentDistortedPointGazeImportance [pointIndices [i]] += 1f / (
    angleThreshold * angleThreshold);
              }
652         }
          }
654
          foreach (int i in recordedPointIndices) {
656               //Vector3 point = currentDistortedPointCloud[i];
                  //Vector3 dir = point - camPos;
658               //float angleInDegree = Mathf.Abs(Vector3.Angle(gazeRay, dir));
                  //currentDistortedPointGazeImportance[i] += 100f * Mathf.Exp(-
    angleInDegree * angleInDegree / (2f * angleThreshold * angleThreshold)) /
    Mathf.Sqrt(2f * Mathf.PI * angleThreshold * angleThreshold);
660               currentDistortedPointGazeImportance [i] += 100f /
    recordedPointIndices.Count;
          }
662

664     } else {
          for (int i = 0; i < currentPointCloud.Count; i++)
666       {
            Vector3 point = currentPointCloud[i];
668         Vector3 dir = point - camPos;
            float angleInDegree = Mathf.Abs(Vector3.Angle(gazeRay, dir));
670
            if (angleInDegree < angleThreshold)
672         {
              pointWithinRange.Add(point);
674           pointIndices.Add(i);
              float distance = Mathf.Abs(Vector3.Dot(dir, gazeRay) / gazeRay.
    magnitude);
676           if (distance < minDistance)
              {
678             minDistance = distance;
                closestPoint = point;
680           }
          }
682     }

684     List<int> recordedPointIndices = new List<int> ();

686     if (pointWithinRange.Count > 0)
        {
688       for (int i = 0; i < pointWithinRange.Count; i++)
          {
690         Vector3 point = pointWithinRange[i];
            Vector3 diffvec = point - closestPoint;
692
            float depth = Mathf.Abs(Vector3.Dot(gazeRay, diffvec) / gazeRay.
    magnitude);
694         if (depth < acceptingDepthRange)
            {
696           recordedPointIndices.Add (pointIndices [i]);
```

```
            //currentPointGazeImportance[pointIndices[i]] += 1f/(angleThreshold
    * angleThreshold);
698         }
        }
700   }
      foreach (int i in recordedPointIndices) {
702             //Vector3 point = currentPointCloud[i];
                //Vector3 dir = point - camPos;
704             //float angleInDegree = Mathf.Abs(Vector3.Angle(gazeRay, dir));
                //currentPointGazeImportance [i] += 100f * Mathf.Exp(-
    angleInDegree * angleInDegree / (2f * angleThreshold * angleThreshold)) /
    Mathf.Sqrt(2f * Mathf.PI * angleThreshold * angleThreshold);
706             currentPointGazeImportance [i] += 100f / recordedPointIndices.
    Count;
        }
708 }
    }
710
    enum DataProperty
712 {
        Invalid,
714     X, Y, Z,
        R, G, B, A,
716     Data8, Data16, Data32
    }
718
    static int GetPropertySize(DataProperty p)
720 {
        switch (p)
722     {
            case DataProperty.X: return 4;
724         case DataProperty.Y: return 4;
            case DataProperty.Z: return 4;
726     }
        return 0;
728 }

730 class DataHeader
    {
732     public List<DataProperty> properties = new List<DataProperty>();
        public int vertexCount = -1;
734 }

736 class DataBody
    {
738     public List<Vector3> vertices;

740     public DataBody(int vertexCount)
        {
742         vertices = new List<Vector3>(vertexCount);
        }
744
        public void AddPoint(
746         float x, float y, float z
        )
748     {
            vertices.Add(new Vector3(x, y, z));
750     }
    }
752
    DataHeader ReadDataHeader(StreamReader reader)
754 {
        var data = new DataHeader();
```

```
        var readCount = 0;

        // Magic number line ("ply")
        var line = reader.ReadLine();
        readCount += line.Length + 1;
        if (line != "ply")
            throw new ArgumentException("Magic number ('ply') mismatch.");

        // Data format: check if it's binary/little endian.
        line = reader.ReadLine();
        readCount += line.Length + 1;
        if (line != "format binary_little_endian 1.0")
            throw new ArgumentException(
                "Invalid data format ('" + line + "'). " +
                "Should be binary/little endian.");

        // Read header contents.
        for (var skip = false; ;)
        {
            // Read a line and split it with white space.
            line = reader.ReadLine();
            readCount += line.Length + 1;
            if (line == "end_header") break;
            var col = line.Split();

            // Element declaration (unskippable)
            if (col[0] == "element")
            {
                if (col[1] == "vertex")
                {
                    data.vertexCount = Convert.ToInt32(col[2]);
                    skip = false;
                }
                else
                {
                    // Don't read elements other than vertices.
                    skip = true;
                }
            }

            if (skip) continue;

            // Property declaration line
            if (col[0] == "property")
            {
                var prop = DataProperty.Invalid;

                // Parse the property name entry.
                switch (col[2])
                {
                    case "x": prop = DataProperty.X; break;
                    case "y": prop = DataProperty.Y; break;
                    case "z": prop = DataProperty.Z; break;
                }

                if (col[1] == "char" || col[1] == "uchar")
                {
                    if (prop == DataProperty.Invalid)
                        prop = DataProperty.Data8;
                    else if (GetPropertySize(prop) != 1)
                        throw new ArgumentException("Invalid property type ('" +
line + "').");
                }
```

```
818                 else if (col[1] == "short" || col[1] == "ushort")
                    {
820                     if (prop == DataProperty.Invalid)
                            prop = DataProperty.Data16;
822                     else if (GetPropertySize(prop) != 2)
                            throw new ArgumentException("Invalid property type ('" +
     line + "').");
824                 }
                    else if (col[1] == "int" || col[1] == "uint" || col[1] == "float
    ")
826                 {
                        if (prop == DataProperty.Invalid)
828                         prop = DataProperty.Data32;
                        else if (GetPropertySize(prop) != 4)
830                         throw new ArgumentException("Invalid property type ('" +
     line + "').");
                    }
832                 else
                    {
834                     throw new ArgumentException("Unsupported property type ('" +
     line + "').");
                    }
836

838                 data.properties.Add(prop);
            }
840         }

842         // Rewind the stream back to the exact position of the reader.
            reader.BaseStream.Position = readCount;
844
            return data;
846     }

848     DataBody ReadDataBody(DataHeader header, BinaryReader reader)
        {
850         var data = new DataBody(header.vertexCount);

852         float x = 0, y = 0, z = 0;

854         for (var i = 0; i < header.vertexCount; i++)
            {
856             foreach (var prop in header.properties)
                {
858                 switch (prop)
                    {
860                     case DataProperty.X: x = reader.ReadSingle(); break;
                        case DataProperty.Y: y = reader.ReadSingle(); break;
862                     case DataProperty.Z: z = reader.ReadSingle(); break;
                        case DataProperty.Data8: reader.ReadByte(); break;
864                     case DataProperty.Data16: reader.BaseStream.Position += 2;
    break;
                        case DataProperty.Data32: reader.BaseStream.Position += 4;
    break;
866                 }
            }
868
                data.AddPoint(x, y, z);
870         }
            return data;
872     }
}
```