UMassAmherst

Remote Collaboration using VR

Project Report

Course ECE 697SD Networked Embedded System Design

Project Remote Collaboration using VR

Teacher Fatima Anwar

Student Xiaohao Xia, Yinxuan Wu, Xin Zhao, Xintao Ding

Contents

ECE697SD

Remote Collaboration using VR

Xiaohao Xia, Yinxuan Wu, Xin Zhao, Xintao Ding

ABSTRACT

Remote collaboration using Virtual Reality (VR) mainly includes three parts: depth camera processing objects, remote transmission, and modeling. The point cloud file is to store the scanned data in three-dimensional coordinate points. We can scan the complete three-dimensional model through point cloud splicing. Unity3D is a multi-platform compatible 3D engine. Its biggest feature is that it has a strong technical exchange community and a plug-in store (Unity Asset Store). In this article, we designed the system framework of ***Python Open 3d+Powershell+Unity*** to realize real-time remoting tracking of the point cloud object by VR. In addition, we analyzed the challenges and other solutions we encountered during the development process. We achieved remote VR headset display at 1 frame per second and local VR headset display at close to 30 frames per second.

Keywords

Realsense, Open3D, point cloud, Unity, Virtual reality (VR), Python, C#, remote tracking, real-time

1 Introduction

Teamwork is essential, it has become a challenge for a global company with departments or subsidiaries all over the world. With the development of remote interaction technology realized by VR, the immersive experience given by VR (virtual reality) will once again change the way people communicate, making face-to-face communication possible. At the same time, the application of VR collaboration and VR tracking can effectively increase the efficiency of remote collaboration.

Based on this, our team used depth cameras (Intel RealSense Depth cameras D435i) and VR equipment (HTC Vive cosmos elite VR headset) to design a VR remote tracking system to realize the function of remote real-time tracking using VR equipment. The codes of our project are publicly available at https://github.com/xinnnnzhao/ECE697SD.

1.1 Motivation

With the development of VR technology and the rise of the animation industry, a new profession has emerged in the animation field, a virtual reality (VR) star. Although VR stars provide real facial expressions and talents, they are displayed to audiences and fans in anime avatars, which also effectively protects the personal privacy of performers. We found that one of the core technologies of VR stars is point cloud face tracking, which captures the faces or actions of actors through depth cameras and generates corresponding point cloud files.

In addition, VR collaboration is now a more popular research direction. For example, Horizon Workrooms of Facebook Oculus can realize remote collaboration through VR and realize three-dimensional visualization, which can greatly improve the efficiency of cooperation. The core technology of this will use VR equipment to remotely track objects.

Based on the above two real-world examples, we became interested in Remote Collaboration using V and decided to implement this project.

1.2 Related Work

A large number of researchers have carried out research and experiments in the field of VR remote collaboration. Theophilus Teo et al. [1] realized MR remote collaboration by using live 360 Video. Andreas Luxenburger et al. [2] designed a MedicaLVR: towards medical remote collaboration using virtual reality. Lei Gao et al. [3] studied the point cloud in remote collaboration, including an oriented point-cloud view for MR remote collaboration. V.D. Lehnerd et al. [4] mainly studied a virtual reality system supporting remote collaboration in vehicle design.

We can find that the use of VR remote collaboration mainly involves the two parts of the camera capturing and processing objects and the VR reading model.

Koestler, L. et al. [5] proposed TANDEM, a real-time monocular tracking and dense mapping framework. For camera tracking, TANDEM outperforms other state-of-the-art conventional and learning-based monocular visual ranging (VO) methods. In addition, TANDEM shows state-of-the-art real-time 3D reconstruction performance.

In 2021, University College London open-sourced the object-oriented SLAM system DSP-SLAM [6], which constructs a rich and accurate federated map of dense 3D models for foreground objects with sparse landmark points to represent the background. It can work at 10 frames per second in 3 different input modes: monocular, binocular or binocular + LiDAR. Object pose and shape reconstruction is improved compared to recent depth prior based reconstruction methods and reduces camera tracking drift on the KITTI dataset.

1.3 Contributions

We have successfully completed the goal of the project and can realize that real-time remoting tracking of point cloud object by VR. The main contributions corresponding to this are as follows:

Control the Intel D435i depth camera to capture targets and export point cloud files via Pyrealsense2. Call classes and functions from the Open3D library to implement point cloud stitching, which allows you to stitch multiple point clouds into a complete model.

A transfer script was designed that use Powershell to start the server to realize remote transmission of point cloud files.

Developing Unity project, use Point Cloud Free Viewer to read point cloud files to create models.

Developing Unity C# script, Unity can customize the path and customize the frame number reread function to refresh the point cloud model, customize the point cloud color, and realize remote real-time reading.

Developing the function of the VR observer, the observer can observe and track the object in the VR world through the VR device, and can control the movement through the handle device.

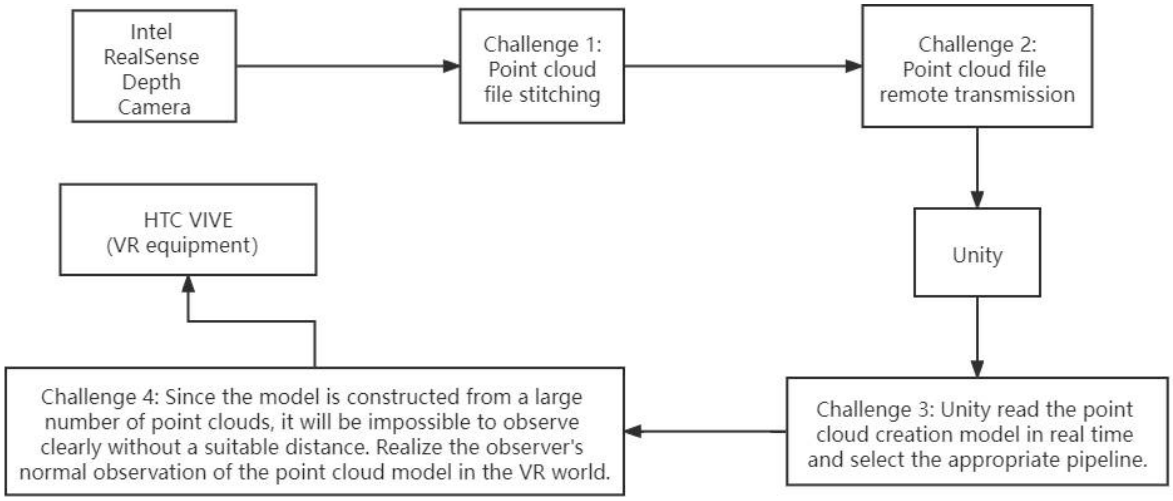

Figure 2-2 Main challenges of the project

2 Framework

For the system architecture of the project, we considered that the working logic of the entire system is mainly composed of the depth camera processing and capturing objects, remote transmission, modeling, and VR observation. As shown in Figure 2-1, for the point cloud generated by the depth camera to capture the object, we use python to achieve point cloud export and point cloud stitching. For remote real-time transmission, we decided to use powershell to complete this function. For the modeling part, we decided to use Unity to achieve real-time modeling. And because of Unity's multi-platform support, we can directly realize data interaction with VR devices through Unity. Finally, for VR observation, we install HTC Viveport to interact with Unity according to the provided equipment. Therefore, the system framework of our project is depth camera (Python)+powershell+Unity+VR equipment.

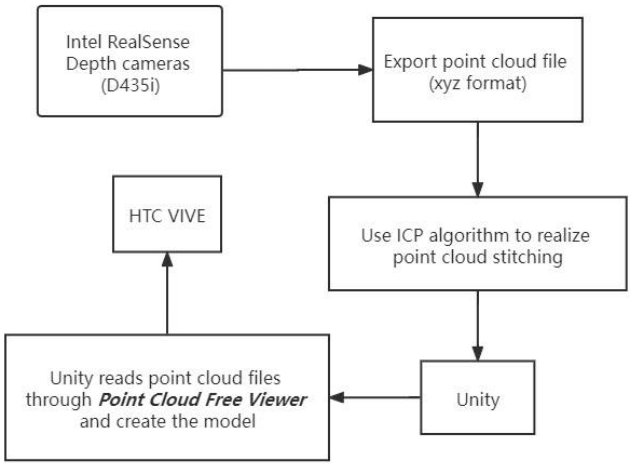

Figure 2-1 Development logic

2.1 Problem Statement

As shown in Figure 2-2, for the framework we designed, we encountered some challenges during the design and development, mainly including:

Challenge 1: Point cloud export and stitching.

Challenge 2: Point cloud remote transmission in real-time.

Challenge 3: Unity reads the point cloud in real-time.

Challenge 4: Set the appropriate number of Unity reading frames and VR viewing angle.

2.2 Architecture

We will discuss the functional parts of the system to solve the challenges of 2.1 in this section, and discuss the technical details of each part of the system in Chapter 3 Implementation.

2.2.1 Point cloud. For the depth camera capture objects, point cloud export and point cloud stitching, the platform we use is Visual Studio 2019, and the computer language used is Python. Compared with other languages, Python has simpler programming logic and has the advantage of calling various libraries to implement functions. Challenge 1 can be completed by using python to call Open3d libraries and design algorithms.

2.2.2 Remote transmission. For remote transmission, based on our computers are all Windows systems, we found that we can use Powershell to run shell scripts to achieve stable and fast file upload and download functions. We will introduce it in detail in Section 3.4 Remote Transmission.

2.2.3 Modeling. For the modeling part (Challenge 3, Challenge 4), we choose Unity as the modeling tool. Unity is a highly compatible open 3D environment open software, using the C# language. We found that Unity can solve Challenge 3 and Challenge 4 through C# scripts and calling plugins. Specifically, we will discuss more details in Section 3.5 and Section 3.6.

2.3 Design Alternatives

2.3.1 Using C++ to process point clouds instead of Python. The official Intel Realsense development documentation provides developers with multiple languages, of which Python and C++ are the two dominant programming languages and are the main languages used by Realsense camera developers. We started out using C++ to control the D435i camera for point cloud output, but we ran into a huge snag when installing the C++ library called PCL. Although the installation was successful and we were able to stitch the point clouds by calling the PCL library, we switched to Python + Open3D midway through the project, considering that the portability of the solution was not ideal.

2.3.2 Unity3D reads OBJ files instead of pointclouds. Unity3D can read OBJ files directly, so as long as the PLY point cloud files exported by the camera are converted to OBJ files, we can make Unity3D visualize directly without using third-party plug-ins, which is another option. We implemented the conversion of the point cloud continuously exported by the camera to OBJ file, but the process takes 0.7~1 second, which is difficult to achieve the real-time requirement, so we abandoned this solution. The details will be described in the first section of Chapter 5 Discussion.

3 Implementation

This chapter will elaborate on the implementation details of each part of the project, focusing on four parts: depth camera, point cloud stitching, teleportation and Unity3D. Each section will include the programming language, the third-party libraries used and their classes and functions.

3.1 Realsense camera

Intel offers a variety of language packages for Realsense camera developers. At first we used C++, but later changed to Python to call the official development documentation package Pyrealsense2.

3.1.1 Camera synopsis. The pyrealsense2 package is the official Python wrapper for Intel Realsense, which supports Realsense SDK 2.0. packages are available at https://pypi.python.org/pypi/pyrealsense2 . We can install the package via pip install pyrealsense2. Windows users can install the RealSense SDK 2.0 from the release tab to get the pre-compiled binaries for the wrapper, for both x86 and x64 architectures. (Both Python 2.7 and Python 3 (3.6, 3.7, 3.8, 3.9) are supported).

3.1.2 Downsampling. Before the camera can export the point cloud, we need to downsample the point cloud information captured by the camera. There are two main reasons for doing this, one is that the size of the point cloud exported directly from the camera is over 10Mb, which is unacceptable for remote transmission. The second reason is that due to the accuracy limitation of D435i, the point cloud captured by the camera contains a lot of error information for objects beyond 2m, and subjectively at least half of the point cloud information is wrong, which causes great interference to the point cloud stitching, so some of the error point clouds must be removed by downsampling.

Librealsense implementation provides post-processing filters to improve the quality of depth data and reduce the noise level. All filters are implemented as separate blocks in the core of the library and can be used in the client's code.

There are three filter classes for effective downsampling of the point clouds acquired by the camera, the names and roles of these three classes are shown below.

```
# Decimation - reduces depth frame density
pyrealsense2.decimation_filter ()
# Spatial    - edge-preserving spatial smoothing
pyrealsense2.spatial_filter()
# Temporal   - reduces temporal noise
pyrealsense2.temporal_filter()
```

Decimation filter can effectively reduce the depth scene complexity. The filter runs on kernel sizes [2x2] to [8x8] pixels. For patches of size 2 and 3, intermediate depth values are chosen. For larger kernels (e.g., 4-8 pixels), the average depth is used for performance reasons. The image size is scaled down in both dimensions to maintain the aspect ratio. Internally, the filter imposes a 4-pixel block alignment on the width and height of the output frame size. For example, for an input size (1280X720) and a scale factor of 3, the output size is calculated as [1280,720]/3 -> [426.6666667 , 240] -> [428 ,240]. The filled rows/columns are zero filled. After generating the resulting frame, the intrinsic parameters of the frame are recalculated to compensate for the resolution changes. The filter also provides some hole-filling capabilities, as the filter uses only valid (non-zero) pixels.

The main feature of the Spatial Edge-Preserving filter is that it is a 1D edge-preserving spatial filter using a higher-order domain transformation. Calculated in linear time, independent of parameter selection, the Spatial filter performs a series of 1D horizontal and vertical passes or iterations to enhance the smoothness of the reconstructed data.

The purpose of the Temporal filter is to improve the persistence of depth data by manipulating per-pixel values based on previous frames. The filter processes the data in a single pass, adjusting the depth value while updating the trace history. In the case of missing or invalid pixel data, the filter uses a user-defined persistence pattern to determine whether the missing values should be corrected with the stored data. Note that due to its reliance on historical data, the filter may introduce visible blurring/smearing artifacts and is therefore best suited for static scenes.

3.1.3 Depth thresholding filter. The Intel Reality D400 Depth Camera provides the most accurate depth ranging data for nearby objects. The depth error is proportional to the square of the distance. Based on our real-world testing, the depth (spatial) noise that occurs when the D435i photographs an object 2 meters away is unacceptable. The image below shows two different views of the point cloud on the bedroom side, and you can see that there is noise everywhere on the otherwise flat wall, which makes stitching the point cloud in post impossible.

Pylibrealsense2's class, pyrealsense2.threshold_filter, allows depth cropping of the camera's depth frames to retain points within a specified distance (e.g., removing points that are not in the range of 0.15m to 2m).

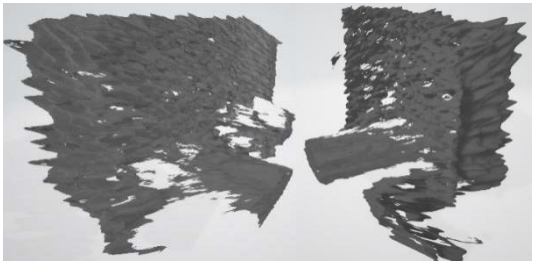

Figure 3-1 Depth noise collected by D435i

Pylibrealsense2's class, threshold_filter, can depth crop the depth frames of the camera and keep the points within the specified distance (e.g. remove the points that are not within the range of 0.15m to 2m). With this filter, we can keep only the point cloud information of items or portraits within the specified distance, thus improving the success rate of stitching.

3.1.4 Pointclouds Output. The camera exports the ply point cloud by reading the point cloud information from frames with pyrealsense2.save_to_ply or pyrealsense2.points.export_to_ply() to generate the ply.

The frames class wraps data with properties, so frames point cloud information is read-only and cannot be modified. It is not possible to modify the point cloud captured by the camera in advance (e.g. remove the background) provided that the point cloud is exported using save_to_ply or export_to_ply.

As stated in Section 3.1.3, depth error is proportional to the square of the distance, and the D435i camera will contain devastating spatial noise when photographing objects or people beyond 2 meters, rendering point cloud information virtually unusable.

3.2 PointClouds Registration

We export the point cloud of the depth camera in Python and implement the stitching (registration) of multiple point clouds through the third-party library Open3D.

3.2.1 Open3d. Open3D is an open source library that supports the rapid development of software for processing 3D data. open3D back-end is implemented in C++ and exposed through a front-end interface in Python. open3D provides three data structures: point cloud (point cloud), mesh (mesh) and RGB-D images. For each representation, open3D implements a set of basic processing algorithms, such as I/O, sampling, visualization and data conversion. In addition, some common algorithms are included, such as ICP registration, which is the algorithm that implements point cloud stitching.

RealSense (librealsenseSDK v2) has been integrated into Open3D (v0.12+) and we can use it via C++ and Python APIs without the need for librealsense to install separate SDKs on Linux, macOS and Windows.

3.2.2 Reduce the resolution of pointclouds(reduce the number of points). We can use Open3D's Voxel downsampling to downsample the exported PointCloud. This is a voxelized grid approach that downsamples the point cloud by reducing the number of points.

This algorithm creates a 3D voxel grid (think of a voxel grid as a set of tiny 3D boxes in space) on the input point cloud data. Then, in each voxel (i.e., 3D box), all points present will be approximated (i.e., downsampled) by their centers of mass. This method is slower than approximating them with the centers of the voxels, but it represents the underlying surface more accurately.

Voxel downsampling is commonly used as a pre-processing step for many point cloud processing tasks. The algorithm proceeds in two steps.

1. the points are divided into voxels.

2. Each occupied voxel generates exactly one point by averaging all points inside.

3.2.3 Vertex normal estimation. Another basic operation for point clouds is point normal estimation. In Open3D, we can use estimate_normals to calculate the normals of each point. This function finds the neighboring points and calculates the principal axes of the neighboring points using covariance analysis. The function takes an instance of the KDTreeSearchParamHybrid class as an argument. The two key parameters radius = 0.1 and max_nn = 30 specify the search radius and the maximum nearest neighbor. It has a search radius of 10 cm and only considers a maximum of 30 neighbors to save computation time.

The covariance analysis algorithm produces two opposite directions as normal candidates. Without knowing the global structure of the geometry, both can be correct. This is known as the normal orientation problem. Open3D tries to orient the normal to align with the original normal if it exists. Otherwise, Open3D does a random guess. Further orientation functions such as orient_normals_to_align_with_direction and orient_normals_towards_camera_location need to be called if the orientation is a concern.

3.2.4 Registration Algorithm. The point cloud stitching algorithms available for the case of Double cameras are ICP registration, Robust ICP, Colored pointcloud registration, and Global registration.

The point cloud stitching algorithms available for the case of three depth cameras and more are Multiway registration.

The ICP (Iterative Closest Point) registration algorithm has been a mainstay of geometric registration in both research and industry for many years. The input are two point clouds and an initial transformation that roughly aligns the source point cloud to the target point cloud. The output is a refined transformation that tightly aligns the two point clouds. We can use a helper function draw_registration_result visualizes the alignment during the registration process. On top of ICP, we can add robust kernels to weed out outliers.

A variant of ICP allows for registration using both geometry and color. it implements the algorithm of [Park2017]. The color information locks the alignment along the tangent plane. As a result, this algorithm is more accurate and robust than the previous point cloud registration algorithm, while running at a speed comparable to the ICP registration algorithm.

Both ICP registration and color point cloud registration are called local registration methods because they rely on rough alignment as initialization. In the project we use another class of registration methods, called global registration. This family of algorithms does not require alignment as initialization. They usually produce less strict alignment results and are used as initialization for local methods.

3.3 PointClouds Compression

3.3.1 Powershell Compression. We use the Compress-Archive command under PowerShell of Windows OS to compress the cloud file after generated from the camera side. So the compressed file could be as small as possible which could reduce the latency for remote transmission. Besides, the compressed algorithm happened in the local machine, so it is far faster than the remote transmission. On the VR headsets side, after the compressed file is downloaded, we use the Expand-Archive command to expand the cloud file.

3.3.2 Alternative Solutions. In addition to using conventional compression methods, there are also many mature dedicated algorithms for point cloud compression, but since this project has already compressed a single point cloud file to less than 1MB by downsampling and other means, additional point cloud compression algorithms are no longer needed. The following is only a supplement to what we know about the current status of point cloud compression algorithm development.

Draco is a library for compressing and decompressing 3D geometric meshes and point clouds. It is designed to improve the storage and transfer of 3D graphics. draco is designed and built for compression efficiency and speed. The code supports compression of points, connection information, texture coordinates, color information, normals, and any other general properties related to geometry. Using Draco, applications that use 3D graphics can be significantly reduced in size without compromising visual fidelity. For users, this means that applications can now be downloaded faster, 3D graphics can now be loaded faster in the browser, and VR and AR scenes can now be transferred and rendered quickly with a fraction of the bandwidth.Draco is distributed as C++ source code and can be used to compress 3D graphics as well as C++ and Javascript decoders that encode data.

3.4 Remote Transmission

First of all, we have an EC2 instance running in the AWS cloud as an FTP cloud server. To set up the FTP services, we use a python command ‘python -m SimpleHTTPServer’, which could set up an FTP service to accept file uploads and downloads. Besides, we also have SSH services running on the server as another option for file uploads and downloads.

On the camera side, we have a PowerShell script as one of the remote transmission clients. In this script, we could compress the cloud file generated from the camera by the Compress-Archive command and then use the SCP command to upload the compressed file to the cloud server.

And all those processes are running in a dead loop so that we can do the transmission constantly. For sure we can insert a sleep command to control the upload's speed rate.

On the VR-headset side, we have another Powershell script client. In this script, we could download the data from the cloud server by the wget command or also SCP command. After that, we expand the file by the Expand-Archive command. All those processes are also running in a dead loop which will transmit the file in real-time.

Overall, this is the fast way to make the whole transmission system is set up and run. But the overheads are kind of high maybe 0.5S. The reason is that the file is written to the file system many times for every single remote transmission, which is not necessary. Besides, the SCP command may also involve extra overheads from ssh authentication, and we can not optimize the performance based on our needs. For future optimization, we can use an in-memory file server in the cloud which could avoid not unnecessary IO, or even we could let the two clients communicate with each other directly as a P2P architecture, which could reduce overheads a lot for sure.

3.5 Unity3D reading point cloud and modeling

We use the Unity3d engine to visualize the model from the point cloud file obtained from the depth camera. The Unity version we use is 2020.3.7f1. It uses C# as the development language and has the advantage of supporting multiple pipelines for rendering and multi-platform development.

3.5.1 Modeling by Point Cloud Free Viewer. One of Unity's strengths is that it has a wide range of community platforms to realize technical exchanges, and at the same time we can find the plug-ins and models we need in the Unity asset store.

We found the professional point cloud reading tool Point Cloud Viewer and Tools and its free version Point Cloud Free Viewer in the Unity asset store. We decided to use Point Cloud Free Viewer as a tool for Unity to read point clouds and model them.

Point Cloud Free Viewer is a solution for viewing laser scanned point cloud data inside Unity for free.

Due to free restrictions, Point Cloud Free Viewer only supports reading point cloud files in xyz format, unlike Point Cloud Viewer and Tools that supports CGO, ASC, ply and other color point clouds in multiple formats.

Based on this, through functions such as private List<Color>, we add the point cloud color setting function to the script, which allows us to customize the single color of the model for observation during modeling.

At the same time, we use the PCX tool plug-in to read the ply format file to compare different ways of reading the point cloud. We will have a discussion in the 5 Discussions chapter.

3.5.2 Real-time reading. Based on the idea of real-time reading based on the number of refresh frames, we plan to design a script to read the point cloud file with a custom path and a custom number of frames.

Declare datapath and publicly, and set void loadPointCloud() function to implement custom path reading point cloud.

First, we use the if function to implement single and multiple read options. At the same time, in the void Start () function, we set ReadInterval, and use m_Timer = ReadInterval to realize Unity to re-read the point cloud with a custom number of frames, thus realizing the real-time refresh function we want.

3.6 Interaction between VR equipment and Unity

The VR equipment needs to configure the corresponding environment on the computer to realize the data interaction between the VR equipment and Unity.

According to the installation instructions of the VR equipment, we downloaded the HTC Viveport from the HTC VIVE official website, which is a software for configuring the computer environment. Through Viveport, we will continue to install VR Stream and set the environment path with the VR device.

Unity (version 2020.3.7f1) has helped us set up the basic settings for VR work. Our main job is to set the initial position and vision module (Camera component) of the VR observer in Unity.

First, we set up the camera to support the visual function of the VR observer. At the same time, we set up prefabs named Lefthand, Righthand and XRRig to realize the movement of the observer in the VR world. Lefthand and Righthand realize the data output of the left and right handles respectively. XRRig is responsible for processing the data of the handle function and controlling the movement of the Camera, and finally realizes the movement function of the observer in the VR world.

Figure 3-2 Use the joystick to manipulate the VR viewing model.

4 Evaluation & Results

4.1 metrics

The evaluation metric for the experiment is the video frame rate of Unity3D. This can be obtained by calculating how many times per second Unity3D reads the point cloud file on average.

4.2 Stitching Pointclouds

Since the accuracy of the depth camera D435i is too low to verify the performance of the stitching algorithm, we first used the Lidar lens of the iPhone 12Pro and the 3d Scanner App to scan the left and right halves of the same bedroom to verify the feasibility of global registration. As shown in the figure below, the two point clouds were stitched together to form a complete bedroom point cloud.

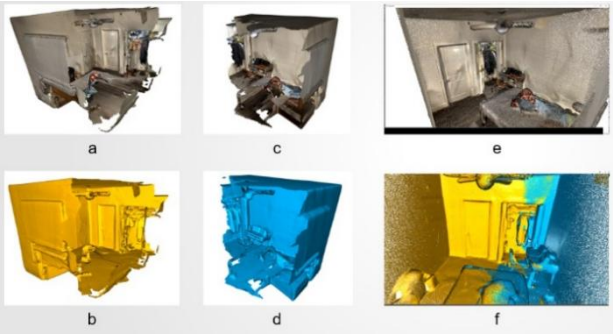

Figure 4-1 Combine the left half and the right half of the bedroom pointclouds to form a complete bedroom point cloud

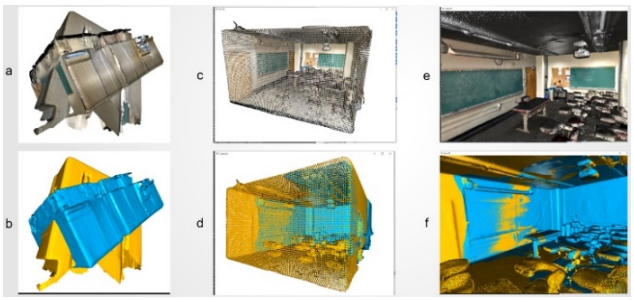

Figure 4-2 Combine the left half and the right half of the classroom pointclouds to form a complete bedroom point cloud

Similarly, to further verify the feasibility of the stitching algorithm, we continued the same operation for the classroom using iPhone12Pro and successfully stitched the left half of the classroom point cloud and the right half of the classroom point cloud into a complete classroom point cloud using the global registration algorithm.

4.3 Unity3D reads the stitched pointclouds

After achieving the stitching of the point clouds of the bedroom and the classroom, we found a way to downsample and de-contextualize the camera point cloud, which laid the prerequisite for stitching the two camera point clouds. As shown in Figure 4-3 and Figure 4-4, we continued to use the Global Registration algorithm and successfully stitched the left and right halves of the person together to form a complete frontal point cloud image of the person.

Figure 4-3 Pointclouds exported from two cameras.

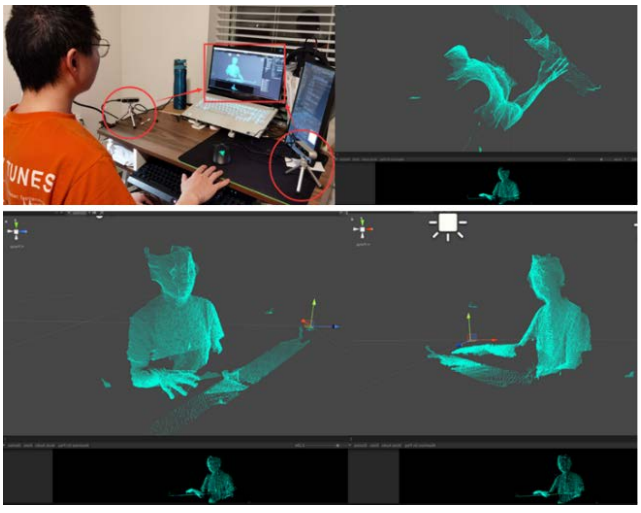

Figure 4-4 Stitching of pointclouds exported from two cameras.

After a real test, the success rate of Global Registration algorithm is close to 80% when two D435i cameras are about 10 cm apart. However, if they are separated by 70 cm as in Figure 4-3, and there is only one person in front of the camera, the success rate is much lower. If there are two people in front of the two lenses and keep one left front and one right back (or one right front and one left back), the stitching success rate may be 20%.

4.4 Unity3D reads point clouds in real time

4.4.1 Unity3D continuously reads point clouds from a single local camera. We first implemented local Unity to continuously read the point cloud exported from a single local camera in real time, at which point Unity displayed the model at a frame rate of up to 28 frames per second (every 0.035 seconds the point cloud file was read in xyz format).

4.4.2 Unity3D continuously reads the stitched pointclouds from two local cameras. Based on the previous results, we integrated the point cloud stitching code and the camera continuous output code to achieve the local Unity real-time continuous reading of the stitched point clouds exported by the two local cameras, at which time Unity displays the model at a frame rate of up to 10 frames per second (every 0.1 seconds the xyz format point cloud file is read).

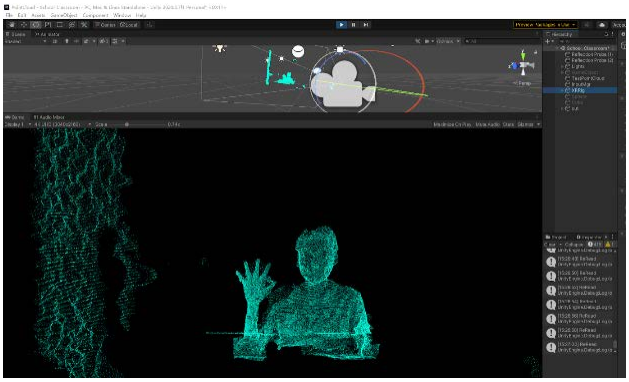

Figure 4-5 Real-time stitching of point clouds exported in real time from two cameras.

4.4.3 Unity3D continuously reads pointclouds from a single remote camera. We further implemented local Unity to continuously read the point cloud exported from a remote single camera in real time, where Unity displays the model at a frame rate of up to 1 frame/second (reading the point cloud file in xyz format once every 1 second).

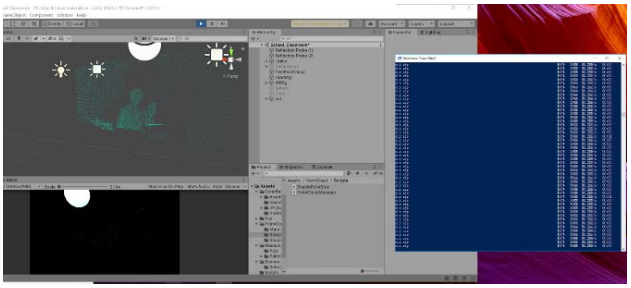

Figure 4-6 Unity3D continuously reads pointclouds from a single remote camera.

4.4.4 Unity3D continuously reads the stitched pointclouds from two remote cameras. In the end, we integrated all the features we had previously implemented to enable local Unity to continuously read the point clouds exported from the two cameras that were stitched together remotely in real time, at a maximum Unity display model frame rate of 0.6

frames per second (every 1.5 seconds for xyz formatted point cloud files).

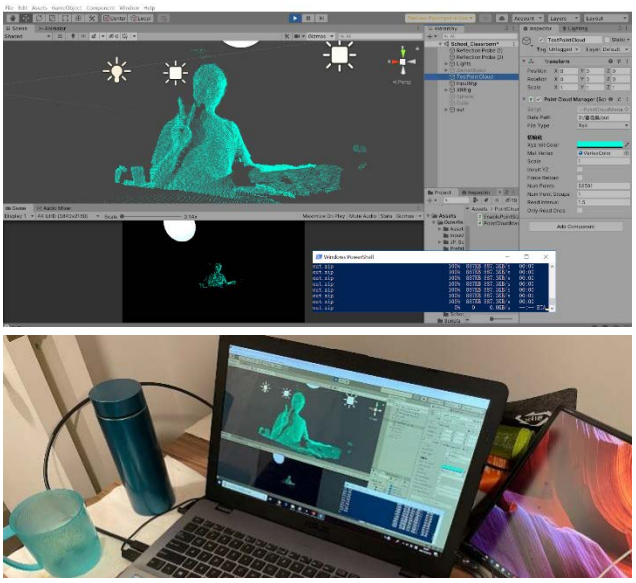

Figure 4-7 Unity3D continuously reads the stitched pointclouds from two remote cameras.

5 Discussions

This chapter discusses the current work completion, the challenges solved, and the bottlenecks encountered. To achieve character tracking, we need to separate the character from the point cloud of the scene. We can only achieve observation of the character as an environment element by constantly refreshing the read xyz format point cloud file to achieve observation of the character's position. If we want to achieve real-time tracking of the character point cloud, the environment point cloud and the character point cloud need to be separated first, and the character point cloud needs to be processed algorithmically at the same time, which is a huge challenge.

5.1 Alternative Architecture:

Unity3D reads OBJ files instead of reading point clouds. The OBJ file format is a simple data format for representing 3D geometry, containing data such as the position of each vertex, UV position, normals, and a list of vertices that make up the faces (polygons). Because vertices in this format are stored in the counterclockwise direction by default, there is no need to save the face normal data. Although the coordinates in the OBJ file format do not have specific units, scaling information can be annotated in the file in the form of comments.

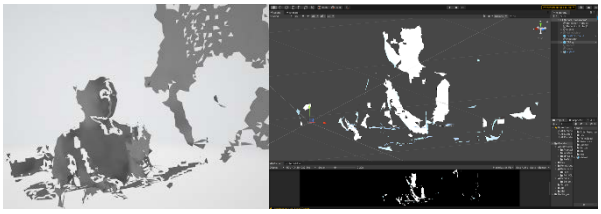

Figure 5-1 Unity3D reads OBJ files converted from PLY

The Inter Realsense camera can only export ply point clouds with the location of each point and its color information, while obj files store point-face information. Converting a ply point cloud to a mesh file like an obj requires calculating the relative distance between each point in the point cloud, which is very computationally intensive and therefore takes a long time to convert and requires high computer performance. Besides, the obj file obtained from the point cloud file does not work well, and there is no construction surface between many points, which should be solved by tuning the parameters. As shown in the figure below, we also found that if the stitched point cloud file is converted into an obj file, there may be a problem that the model faces are reversed, which may be related to the calculation of face normals.

5.2 Point cloud format selection

Point cloud files have many formats, such as xyz, xyzrgb, cgo, asc, ply, etc. The original export format of the depth camera is ply, so we first try to use the PCX plug-in to read the point cloud ply format.

However, since the working logic of reading the ply format is to read all the point cloud data at one time and then process it, and there is a header file limitation in the ply format, we found that it is impossible to directly add a self-contained path function in the script of the PCX plug-in And frame number function to achieve repeated reading. In addition, we find that choosing the ply format for reading is also prone to cross-border problems. The xyz format point cloud is to store each point cloud data in the format of xyz three-dimensional coordinates. Unity reads one line of point cloud coordinates at a time, which will help us to use the script to refresh the point cloud.

Therefore, we consider changing the ply format to xyz or xyzrgb format to read, so as to avoid the constraints of the ply format header file.

5.3 Engine choice

Choosing the suitable engine to implement modeling is one of the challenges we encountered when developing this project.

When choosing the modeling tool at the beginning, we considered three engines, and finally we chose the Unity engine as the modeling tool.

We will discuss our reasons for choosing Unity as the modeling engine in this section. We will compare and analyze the three 3D game engines, Egret Engine, Unreal Engine 4 and Unity.

5.3.1 Egret Engine. Egret Engine is a game engine developed based on TypeScript language. Two modes of CPU rendering and WebGL rendering are provided. At the same time, it supports WebAssembly (also known as wasm) technology, which mainly supports Egret Engine to load high-level languages for the browser faster. The biggest advantage of Egret Engine is the compatibility of the modeling function with the browser, and the 3D model built by its 3d version is suitable for browser observation.

At first we considered using a browser to visualize the point cloud modeling and then using Windows 3D drawing to import the VR device, but because the point cloud of the point cloud model is too large, there will be a reading error in the browser visualization, we decided Give up using Egret Engine.

5.3.2 Unreal Engine. Unreal Engine is a game engine developed by Epic Games. Its pipeline integration feature makes the model rendering function better than other engines, which can make the model rendering compatible with multiple situations and reach a high-level standard.

Connect media production pipelines through support for industry standards such as FBX, USD, and Alembic. First-class USD support enables users to better collaborate and work in parallel with team members. Unreal Engine can read the USD file from any location on the disk without the time-consuming complete import process, and write the changes back to the file to overwrite the original content; reload the USD payload to immediately update the changes made by other upstream users .

In addition, the support for python scripts makes it easy to integrate Unreal Engine into the project pipeline, and the full support for Python

scripts automates the workflow.

5.3.3 Unity3D. Although Unreal Engine has the advantage of pipeline integration, Unity has one of the biggest advantages over Unreal Engine, that is, it has a strong development community and a wealth of plug-ins. We can find the plug-ins we need in the Unity Asset Store to use and modify, which will save us a lot of development time.

In addition, for pipeline rendering, we found that Unity's built-in rendering pipeline is sufficient for xyz point cloud reading and modeling. Since the xyz file is a model directly composed of a large number of point clouds, using Scriptable Render Pipeline (SRP) will not improve the rendering effect.

In summary, we can find that Unity is the most used platform for 3D applications, and it has extensive support available and most of the APIs are open source, and finally we chose Unity as the modeling tool.

6 Future Works

While Intel provides a Python language package for developers of the Realsense camera, the underlying layer is still C++ code, which probably means that calling the camera via Python is not the most efficient way to do it. Currently we can get the camera to export up to 30 this point cloud files per second using Python, which means the VR headset can receive up to 30 frames per second as well. We have also installed a C++ environment to try to control the camera, but we have encountered a lot of obstacles in the environment configuration of the PCL library, which is far less convenient than installing the Open3D library in Python. In the future, we need to verify whether using C++ to control the camera can further improve the rate of point cloud output.

Although this project implements basic point cloud modeling operations, it does not introduce machine learning methods, so it cannot implement functions such as dynamic tracking of people and movements in the camera, which is the main direction of future work.

In implementing point cloud segmentation operations such as background removal, the methods we use are based on Open3D's linear operations. This meets the basic requirements but is not smart enough. There are now many mature deep learning network architectures that can perform segmentation operations on point clouds more intelligently and better meet the potential requirements.

The most important part of this project is point cloud stitching. We use the mainstream Global Registration algorithm to achieve the stitching of two point clouds. This algorithm is more suitable for stitching point clouds with high accuracy. The success rate of this algorithm is close to 100% in stitching the high precision bedroom point cloud and classroom point cloud taken by iPhone 12pro. However, the success rate of this algorithm in stitching the point clouds exported by the D435i camera in real time is very low, on the one hand, because the point clouds exported by the D435i camera have a lot of depth noise causing them to contain a lot of error information. On the other hand, it is due to the lack of nonlinear intervention of the algorithm and the linear fit is not robust enough to cope with outliers. Future work needs to incorporate deep learning methods such as machine learning to further improve the success rate and robustness of point cloud stitching.

Regarding remote transfer, we used AWS server as a staging point to upload and download via Windows Powershell command. We found that the Powershell operation has a delay close to 1s, which leads to a maximum display frame rate of 1 frame per second for our remote VR, so future work can implement socket communication in Python and socket acceptance in C# on the VR headset side, thus reducing the transmission delay to about 0.05s, which is expected to increase the VR display frame rate to 15 frames per second.

7 Conclusions

We designed a system framework based on Pyrealsense + Open3D+Powershell+Unity3D, and implemented a remote VR headset to display 3D models at a maximum actual frame rate of 1 frame per second in this system. We can use Python to control a single Intel D435i camera to output a point cloud up to 30 times per second, which is the upper limit of the theoretical VR headset display frame rate that our proposed framework can achieve (i.e., 30 frames per second). In the case of real-time stitching of point clouds from two cameras, up to 10 stitched point clouds can be output per second. Future work will focus on introducing deep learning to improve the success rate of point cloud stitching in real time and to achieve intelligent operations such as tracking of specified characters/objects and recognition of gestures and facial expressions.

8 Acknowledgement

We would like to express our gratitude to our instructor, Professor Fatima M. Anwar, who provided us with guidance and help throughout the project. Besides, we would also like to show our appreciation to Adeel Nasrullah and Yasra Chandio for their help in our project.

References

[1] G. A. Lee, T. Teo, S. Kim and M. Billinghurst, "A User Study on MR Remote Collaboration Using Live 360 Video," 2018 IEEE International Symposium on Mixed and Augmented Reality (ISMAR), 2018, pp. 153-164, doi: 10.1109/ISMAR.2018.00051.G. A. Lee, T. Teo, S. Kim and M. Billinghurst, "A User Study on MR Remote Collaboration Using Live 360 Video," 2018 IEEE International Symposium on Mixed and Augmented Reality (ISMAR), 2018, pp. 153-164, doi: 10.1109/ISMAR.2018.00051.

[2] Andreas Luxenburger, Alexander Prange, Mohammad Mehdi Moniri, and Daniel Sonntag. 2016. MedicaLVR: towards medical remote collaboration using virtual reality. In Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct (UbiComp '16). Association for Computing Machinery, New York, NY, USA, 321–324. DOI:https://doi.org/10.1145/2968219.2971392

[3] Lei Gao, Huidong Bai, Gun Lee, and Mark Billinghurst. 2016. An oriented point-cloud view for MR remote collaboration. In SIGGRAPH ASIA 2016 Mobile Graphics and Interactive Applications (SA '16). Association for Computing Machinery, New York, NY, USA, Article 8, 1–4. DOI:https://doi.org/10.1145/2999508.2999531

[4] V. D. Lehner and T. A. DeFanti, "Distributed virtual reality: supporting remote collaboration in vehicle design," in IEEE Computer Graphics and Applications, vol. 17, no. 2, pp. 13-17, March-April 1997, doi: 10.1109/38.574654.

[5] Koestler L, Yang N, Zeller N, Cremers D. TANDEM: Tracking and Dense Mapping in Real-time using Deep Multi-view Stereo. 2021. Accessed December 16, 2021. https://search.ebscohost.com/login.aspx?direct=true&db=edsarx&AN=edsarx.2111.07418&site=eds-live&scope=site

[6] Anders Grunnet-Jepsen, Paul Winer, Aki Takagi, John Sweetser, Kevin Zhao, Tri Khuong, Dan Nie, John Woodfill. Multi-Camera configurations - D400 Series Stereo Cameras. 7 July, 2021. https://dev.intelrealsense.com/docs/multiple-depth-cameras-configuration