Xi'an Jiaotong-Liverpool University
西交利物浦大學

SCHOOL OF ADVANCED TECHNOLOGY
SAT301 FINAL YEAR PROJECT

Synthesis of Multi-Camera Video Datasets via Computer Graphics

Final Thesis

In Partial Fulfillment
of the Requirements for the Degree of
Bachelor of Engineering

Student Name	:	Qinxin Ren
Student ID	:	1824098
Supervisor	:	Ming Xu
Assessor	:	Xuchen Wang

Abstract

Deep-learning based pedestrian detection algorithms has made great progress. However, difficulties arise when the occlusion appears. Occlusion is very common in single-view scenes, while multi-view scenes can efficiently solve the problem. For the training of deep-learning based pedestrian detection algorithms, large scale of annotated datasets are crucial. The real-world scenes are difficult and time-consuming to annotate, while the synthesized datasets can annotate the datasets automatically. Therefore, this project aims synthesize multi-camera pedestrian datasets for the training of deep-learning based pedestrian detection algorithms using the Computer Graphics technique. The synthesized datasets are then verified using the Computer Vision technique. Human character models are built to simulate pedestrians in the real world. These models are programmed to generate in the predetermined area of the scene randomly. The associated building procedures are described in detail. In order to provide the annotated ground truth, cameras at different orientations are placed to shoot the videos. Localization and geometric relationship of the pedestrian are then validated with the multi-view frames. This project shows that the simulated datasets are very close to the real-world datasets and suitable for the training of deep learning pedestrian detection algorithms.

Keywords: Computer Graphics, Multi-view, Synthesized dataset, People detection

Acknowledgements

Words cannot express my gratitude to my professor Ming Xu for his invaluable patient guidance and feedback. There were tough times for me when completing the project. It was his encouragement to get me back. This endeavour would not have been possible without his generous help. I would like to extend my sincere thanks to my family. Although I'm thousands of miles away from home, they still care and support me without any reason. When I was exhausted, their encouragement gave me the power to go through all the dark times and kept my spirits high during the process. Lastly, I would like to thank all of my friends for the entertainment and emotional support. It was an incredible, wonderful experience with them during these four years.

List of Acronyms

JOL	Joint Occupancy Likelihood
RSS	Repulsive Spatial Sparsity
FPN	Feature Pyramid Network
HDRP	High Definition Render Pipeline

List of Figures

Contents

Chapter 1

Introduction

1.1 Motivation, aims and objectives

Pedestrian detection is an active research area in the computer vision community. It has various application scenarios in persistent video surveillance, autonomous driving, traffic monitoring, etc. The training of the deep learning pedestrian detection algorithm requires a large scale of annotated datasets. Although monocular pedestrian detection with deep learning techniques [1] [2] has made good progress, there are still some limitations. The single-camera datasets have the problem of ambiguity caused by occlusion between people, and that makes the algorithm unable to detect the occluded pedestrians efficiently. The localization of occluded pedestrians is also affected in single-view scenes.

In order to improve the robustness and accuracy of the detection, multi-camera datasets are proposed to provide complementary information. The occluded pedestrians can be accurately detected when multi-view scenes are available. The training of deep learning algorithms needs a tremendous number of annotated datasets. However, the annotation of multi-view video datasets is a complicated and slow procedure. Producing a physical nature multi-view dataset is also a complicated procedure since it needs to consider the environment and a variety of social behaviours.

Since the need for large numbers of correctly labelled data is essential for training pedestrian detection algorithms. Various applications have been proposed to provide reliable datasets by using the Computer Graphics technique. The synthesized virtual scenes with real-world backgrounds are presented. They show advantages in the generation of vast automatically annotated multi-view datasets. The modelling procedure is simple compared with the real-world scenes.

In this spirit, the first task of this project aims to synthesize large amounts of multi-view datasets based on the Computer Graphics technique. It uses Unity3D [3] as the tool. These datasets can be automatically annotated and used to train deep learning pedestrian detection algorithms. Various scenes are constructed in this project based on demand. Multiple virtual cameras with different orientations will be placed to shoot the scene. They have overlapping fields of vision and are used to synchronize the multi-view datasets. The generated datasets are automatically annotated. They indicate the positions, trajectories, and bounding boxes of pedestrians. Because of the necessity for large-scale datasets, human character models are needed in addition to scenes. In order to ensure the diversity of the datasets, there are more than 100 human character models have been created. Pedestrians' trajectories are pre-programmed to make the human models walk in a predetermined area in the scene. Because each pedestrian's path is pre-programmed, its location is known.

The second task of this project is to verify the geometric relationship of the synthesized datasets. The localization and geometric relationship of the pedestrians need to be calculated. Multiple Computer Vision algorithms are used in the verification. Since the synthesized dataset is multi-view, and the frames are shot at the same time, the location of each pedestrian in different views should be the same in synchronized frames. The Joint Occupancy Likelihood (JOL) of each pedestrian is calculated, and then the Repulsive Spatial Sparsity (RSS) algorithm is used. The results show that the properties of the synthesized datasets are the same as the real-world datasets.

The organization of this report is shown as follows: In Section 1.2, a historical review and comparison of synthesized pedestrian datasets are proposed; In Section 1.3, the rationale of this project and the industrial relevance are discussed; In Section 2, the methodology used in this project; In Section 3, and results are presented; Conclusions and future work are proposed in Section 4. Finally, the related work and code are attached in the appendix.

1.2 Literature Review

1.2.1 Computer graphics in synthesized datasets

(1) Advantages of synthesized datasets

Datasets for deep learning pedestrian detection algorithms often used to be videos of real-world crowds collected in public places [4, 5] or downloaded from the website. However, it suffers from legal difficulties. Not every country agrees that the video with unrecognizable faces can be used without an individual agreement; the video with recognizable faces is even more problematic. Some datasets are captured in controlled conditions with actors [5], while the scenarios are relatively expensive and time-consuming to build. The simulated scene over a real-world scenario is then widely used. The advantages are manifold. It is a very cost-effective way to generate many data. Privacy is not considered since the scene and human models are all synthesized. Impractical scenarios in real life can also be built into the simulation. Moreover, the crucial advantage is that the simulated scenes can generate large amounts of automatically annotated data instead of manual labelling in real-world scenes.

(2) Limitations of the existing synthesized datasets

Research has used computer graphics simulation in the field of pattern recognition. [6] used 2D silhouettes for gait analysis applications. [7, 8] also used the same idea in human action recognition. [9] used 2.5D data to train the Microsoft Kinect body pose recognition system. [10] was the first to provide a publicly available dataset of the synthetic 3D scene. However, according to [11, 12], the existing datasets do not provide synchronized multi-camera data. They also have a shortage of either low revolution [13, 14] or low duration [15, 16, 17, 18]. [19] then proposed a dataset which is generated from the photo-realistic game GTA V. This dataset was considered as a solution for multi-view while still causing subproblems, e.g., person detection and feature collection.

(3) Human character modelling

Therefore, this project proposes a synthesized dataset using Unity3D. Compared with the previous dataset [10], it shows advantages in generating long duration with the intrinsic cameras. Since the human models are manually built, the appearances are adjustable in different conditions. Since the scene is simulated rather than captured in the game [19], it is more flexible and more scenarios can be generated. Moreover, for the human character modelling, [20] proposed The PersonX engine to create the models. The visual variables, e.g., illumination, scenery and background, are editable in this engine. However, it is a tedious, time-consuming procedure to adjust all these parameters when building models manually. These details are also considered not so important in pedestrian detection. Therefore, this project uses the engine Fuse to create the models. The procedure of building a model is quite simple since there have been plenty of prefabs on the body, face, and clothes.

1.2.2 Computer vision in datasets verification

(1) Homography in datasets calibration

In verification of the datasets, the spatial distribution of the pedestrians and the occlusion among them are crucial. The object coordinates in a common reference are estimated to achieve robustness in terms of appearance variability between views. In most cases of calibration, the homography is the first to be examined [23] . In their technique [24], Stauffer and Tieu use tracking data to predict homography from one camera to the next. [23, 24, 25] also compute planar homographies between multi-camera views instead of projecting the views on a reference ground plane. Those techniques, however, fail to alleviate the occlusion problem.

(2) Localization of foreground people

After projecting the images into the reference ground plane, [26] use the estimated trajectories collected by each camera to match objects. The results verify that the shadow extracted with a person will affect the matching procedure. Both the feet and head region of the foreground people are extracted in the location detection. In a multi-view arrangement, Reddy et al. in [27] employ compressed sensing to recognize and track individuals. They make use of the cameras to extract foreground silhouettes. Then in [28], Fleuret et al. use the multi-view infrastructure to accurately follow persons across multiple cameras when the foreground silhouettes are degraded. A mathematical framework is developed in their work. However, the framework has high potential false positive rate because the sparsity is not carefully considered. Therefore, in this project, the framework proposed by [29], which cope with the limitations of previous work is used. It works well to any number of cameras, even single cameras. The sparsity in the desired solution is explicitly considered in this framework.

1.3 Industrial Relevance

Various data suggest that using deep learning pedestrian detection algorithms in multiple industries could be beneficial. Such technology is required for safety, public transit regulation, and autonomous driving. At the same time, there was a critical issue with the absence of correctly labelled datasets in the training of the algorithms. In this project, various scenes are built through Unity3D. Human character models are programmed to walk in the predetermined area. With multi-orientation cameras placed in each scene, it is possible to synthesize massive training datasets. The verification part of the project further proves that these datasets can be widely used in the training of deep learning pedestrian detection algorithms. Compared with traditional real-world scenes, these synthesized datasets show advantages in: (1). It is possible to use a wide range of pedestrian appearances (e.g., skin, gender, height, weight, clothes) to generate many synthesized data. (2) Since the pedestrians in the simulated scene are programmed to generate, the models can be generated for any location in the scene. It can be programmed to be created in the predetermined area. **(3)** The location of the surrounding building and other static objects in the scene can be included in the training of detection algorithms. It can improve the training for occlusion detection. (4) Since the ground truth is known, manual data labelling is not required. It can generate large amounts of automatically annotated data. Computer-generated images have been successfully applied in pedestrian detection. A more precise pedestrian detection algorithm may be achieved due to the datasets. Furthermore, the methodologies created in this research will be generic and, in theory, relevant to the numerous sectors in which 3D data is now utilized.

Chapter 2

Methodology

The whole structure of the framework is shown in Figure 1. I consider the following condition is satisfied: (1) the geometrical management of the scene, e.g., the "pedestrian region" is the area where pedestrians possibly appear, the obstacles like the building are rendered as not walkable area, and the pedestrians could either be occluded or physically unable to present; (2) intrinsic and external camera parameters are available. In this project, the works have been done: Scene modelling; Human character modelling; Camera modelling; Tsai's calibration; extraction of the masks by using MaskRCNN; Calculating the JOL to draw rectangles; filtering the rectangles by using RSS. The pictorial illustration of the framework is shown in the figures below.

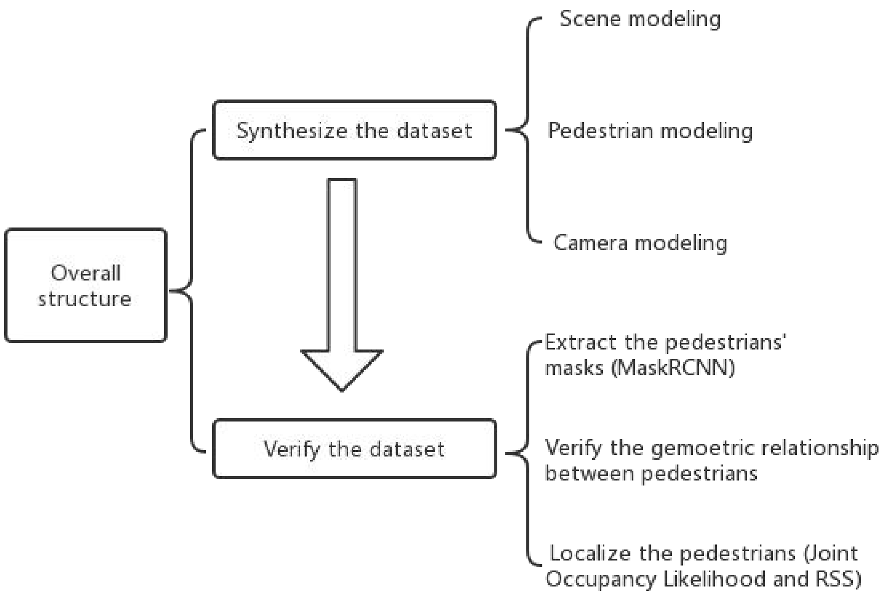

Overall structure
Synthesize the dataset
Scene modeling
Pedestrian modeling
Camera modeling
Verify the dataset
Extract the pedestrians' masks (MaskRCNN)
Verify the gemoetric relationship between pedestrians
Localize the pedestrians (Joint Occupancy Likelihood and RSS)

Figure 2.1: The overall structure of this project, it contains two main parts: synthesize of the datasets and verification of the datasets.

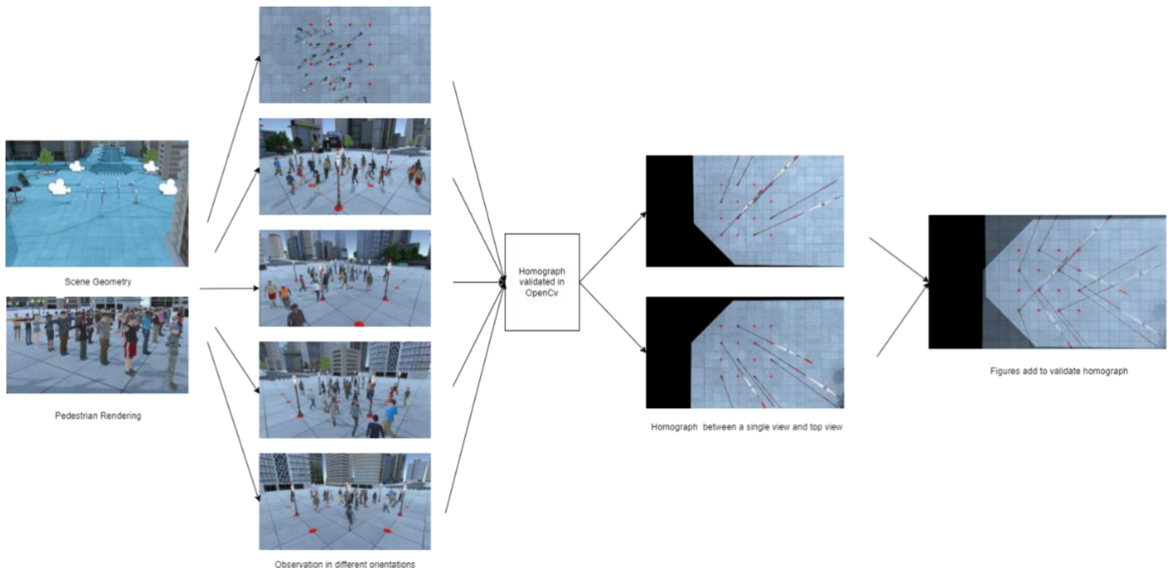

Scene Geometry
Pedestrian Rendering
Observation in different orientations
Homograph validated in OpenCv
Homograph between a single view and top view
Figures add to validate homograph

Figure 2.2: Overview: The geometrically walkable areas is automatically rendered in the navigation, while the surrounding buildings are not rendered. Pedestrian models are programmed to walk in the area. Photos from different orientations are generated. Different orientation view is validated in homography transformation.

2.1 Synthesis of the datasets

In this part, the details of synthesizing the datasets using Unity3D will be introduced step by step. It includes three main parts: human modeling; scene modeling and camera modeling.

2.1.1 Human character modeling

The detailed steps of human character modelling are shown as below:

(1) Software introduction

Adobe Fuse CC [21] is a data-driven application for 3D characters modeling. It allows building human character models using internal library assets. The assets are high-quality 3D, which contain textures, bodies, clothes, and faces. Each content contains various attributes, e.g., clothing fabric, torso shapes, hats. These attributes can be customized together to achieve the final look. Moreover, the attributes like the torso are adjusted automatically when the body of the character's size and proportion is changed. This application can also export the human character model to the website "Mixamo" [22] for further settings.

(2)Assemble the initial body of the character

In this step, the initial body of the character is created. Body parts head, torso, arms, and legs in the library can be chosen in the right Edit panel. These parts can be added to the left panel by clicking. Besides, the full matching body parts can be automatically fused by right-clicking the part and choosing "Add Matching Parts" in the options menu. The details are shown in the figures below.

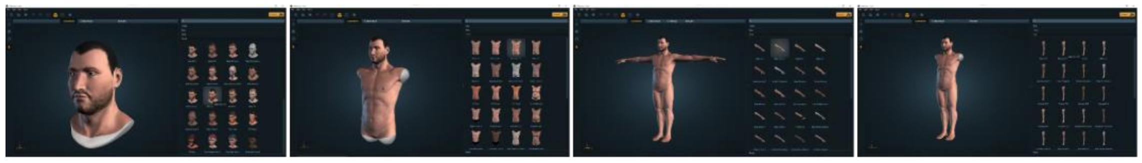


Figure 2.3: The procedure of creating the initial body for a male character.

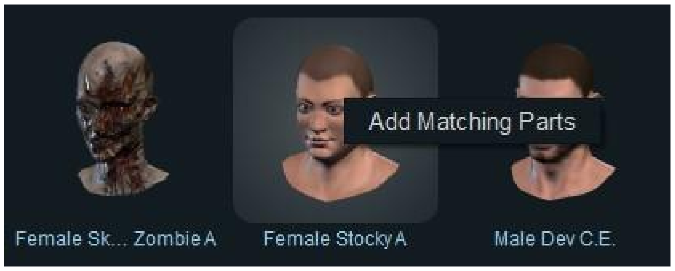


Figure 2.4: Automatically matching parts.

(3) Customize the body's shape and facial features

In this step, the character's specific body parts can be customized. Firstly, the mouse pointer hovers over the body to select the region. The selected part will have a blue boundary. Then the corresponding attributes in the right Edit panel can be customized to achieve the desired shape by dragging. The details are shown in the figure below.

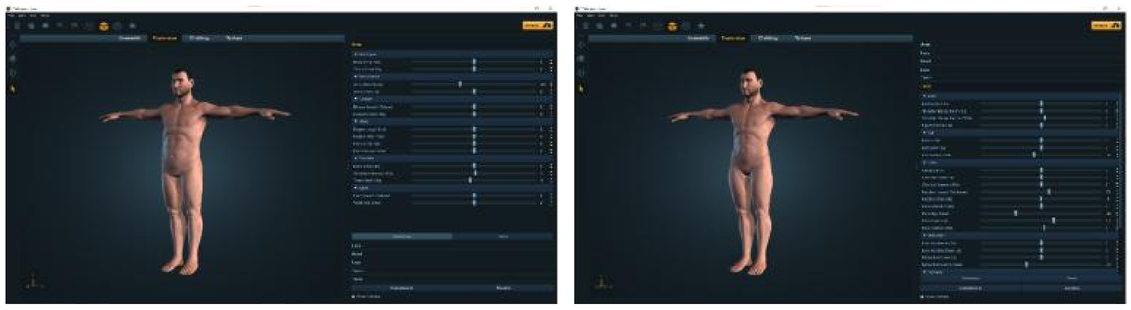


Figure 2.5: Customize the body's shape.

(4) Dress the character using cloth library assets

In this step, the character can be dressed. The application provides build-in clothing assets, including tops, bottoms, shoes, hair, hats, eyewear etc.al. These assets can be chosen in the right Edit panel, then the chosen clothing will be automatically wrapped around the character's body. The details are shown in the figure below.

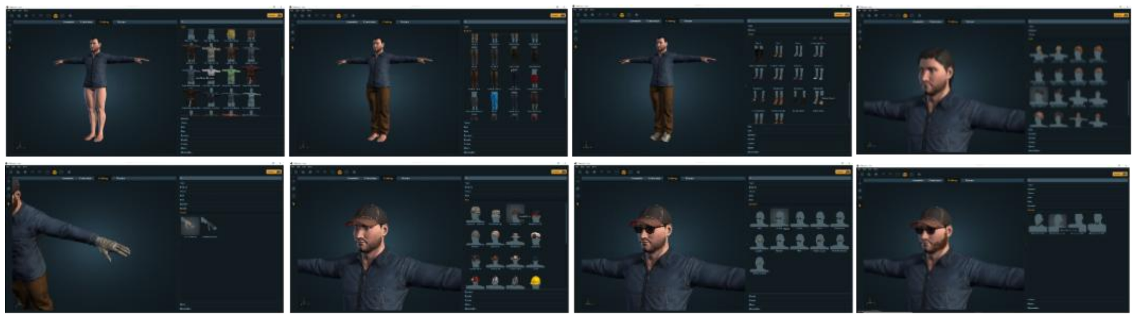


Figure 2.6: The details of dressing the character.

(5) Customize the texture and material of the body, hair, and clothing

In this step, the texture of the clothing item can be customized in Texture mode. The right Edit panel provides texture parameters, which can be adjusted by moving the button.

For the body part, the skin color, age, skin details can be adjusted. For the hair part, the hair color, saturation, and extra categories can be adjusted. For the clothing, the main fabric, collar, pockets et.al can be adjusted. The details are shown in the figure below.

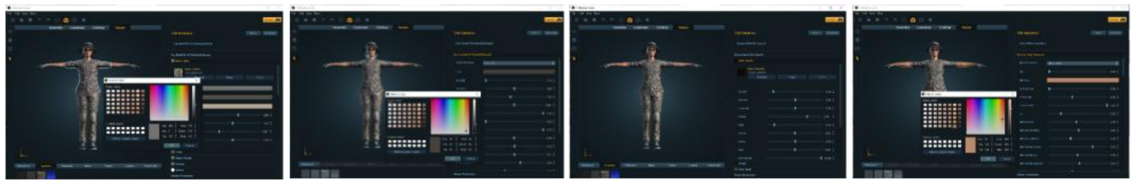
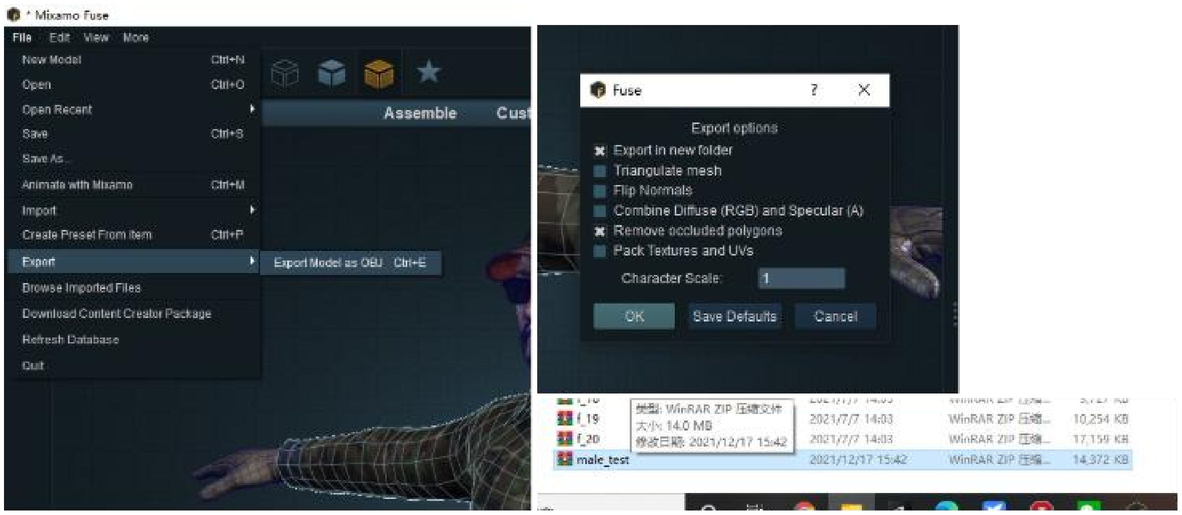


Figure 2.7: Customize the texture and material of these three parts.

(6) File saving

In this step, the character model can be exported. To further bind the skeletons, the character needs to export as ".obj" format. The the file needs to be compressed as a ".zip" file. The details are shown in the figure below.



Export options
- [✓] Export in new folder
- [ ] Triangulate mesh
- [ ] Flip Normals
- [ ] Combine Diffuse (RGB) and Specular (A)
- [✓] Remove occluded polygons
- [ ] Pack Textures and UVs

Character Scale: 1

类型: WinRAR ZIP 压缩文件
大小: 14.0 MB
修改日期: 2021/12/17 15:42

f_19	2021/7/7 14:03	WinRAR ZIP 压缩...	10,254 KB
f_20	2021/7/7 14:03	WinRAR ZIP 压缩...	17,159 KB
male_test	2021/12/17 15:42	WinRAR ZIP 压缩...	14,372 KB

Figure 2.8: The procedure of exporting the character model.

(7) Skeleton binding of the characters

When the different human characters have been built and exported as zip files, these characters should be uploaded to the "Mixamo" to bind the skeleton and extract the texture and material. Each of the characters is tied with 25 skeletons, considering the running speed of the final program. If more vivid humanoid behavior is needed, the bind skeleton can add up to standard 65. Texture and material are extracted as files, respectively, to make the detail of these characters better. The binding procedure is shown in the figure below.

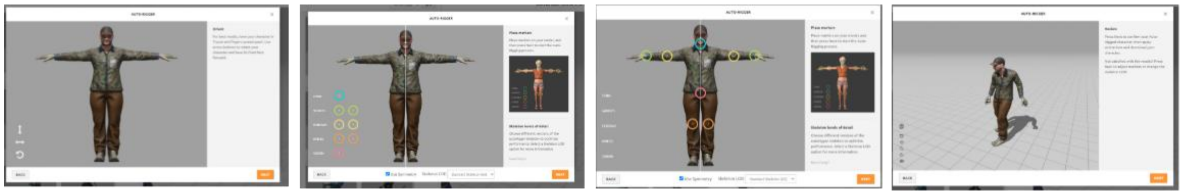
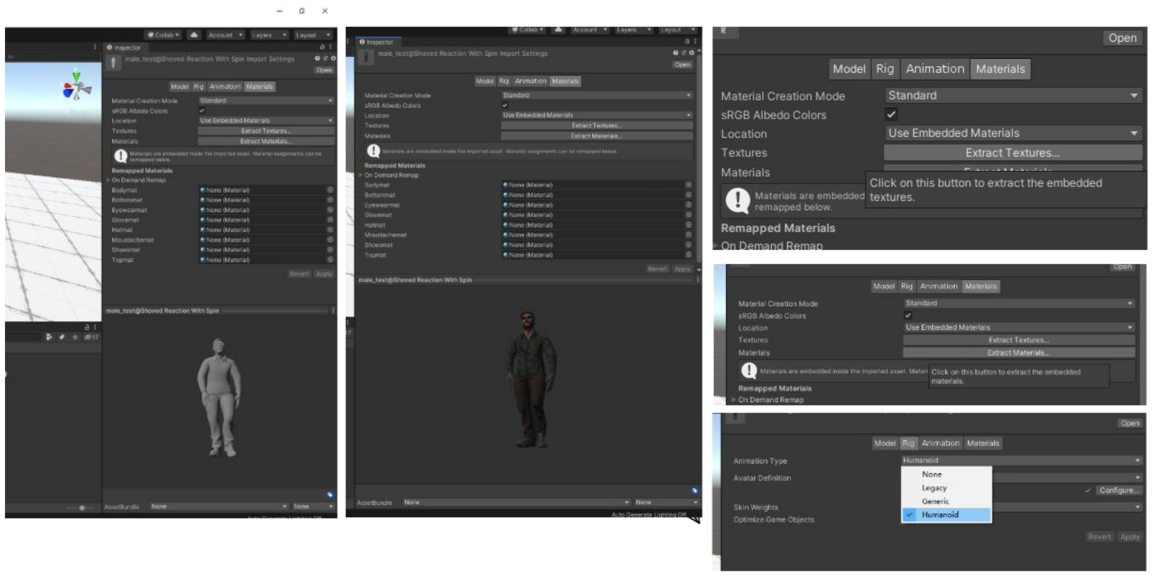


Figure 2.9: The procedure of binding the human character model. It includes the skeleton binding and extraction of texture and material.

(8) Unity import of the results

Among these two procedures, 100 characters have been built and bound skeleton. Then these characters are exported to Unity. In Unity, "NewCharacter" is created to store the character models. All these models' Animation types are set as "humanoid" in the inspector window. Textures and Materials of these models are exported to the project as well. Set the type in ”Material Creation Mc” as ”Standard (Legacy)”. The previously extracted files set the inspector’s characters’ texture and material. After that, drag these character models to the scene to make prefabs, adjusting thescale simultaneously. The procedure is shown in the figure below.



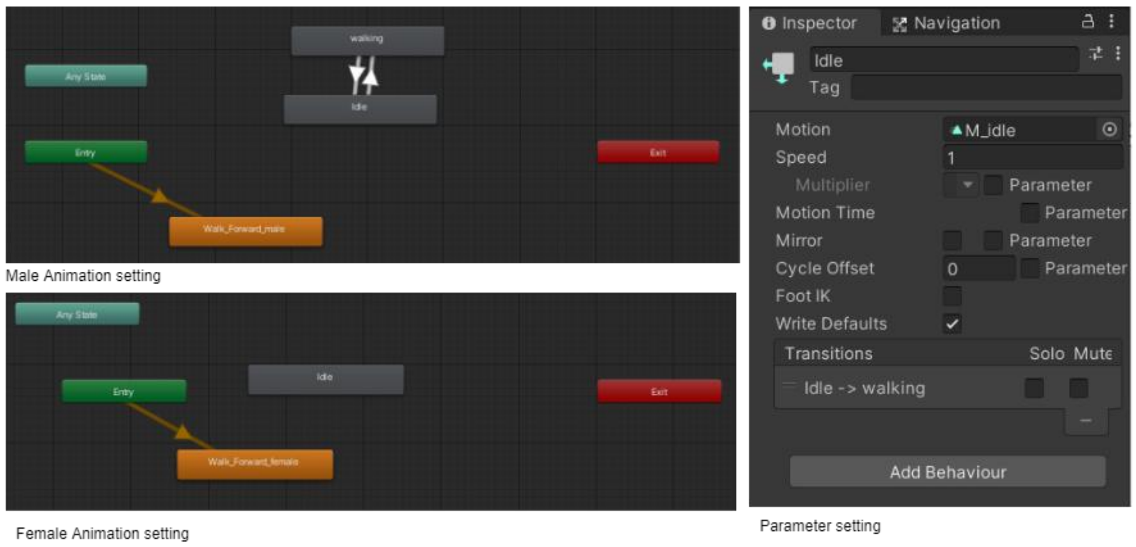
Material Creation Mode: Standard
sRGB Albedo Colors: ✓
Location: Use Embedded Materials
Textures: Extract Textures...
Materials: Extract Materials...
Click on this button to extract the embedded textures.
Materials are embedded inside the imported asset. Material assignments can be remapped below.
Click on this button to extract the embedded materials.
Remapped Materials
On Demand Remap

Animation Type: Humanoid
Avatar Definition
Skin Weights
Optimize Game Objects

Figure 2.10: The procedure of importing the human models in Unity. Includes the settings of the parameters.

(9) Animation binding of the characters

For the prefabs, different animation is bound with different gender characters. There are two statuses of each character. For male characters, they are bound with ”m animation”, which contains ”walk” and ”idle” status. Female characters are bound with ”f animation”, with ”catwalk”, and ”idle” two status. The trigger of different statuses is ”walking,” a Boolean parameter. This parameter is programmed to control the movement of characters. When the project starts, the parameter is set to be ”true”; when the characters move to the destination, the parameter is ”false”. The procedure of binding the animation is shown in the figure below.

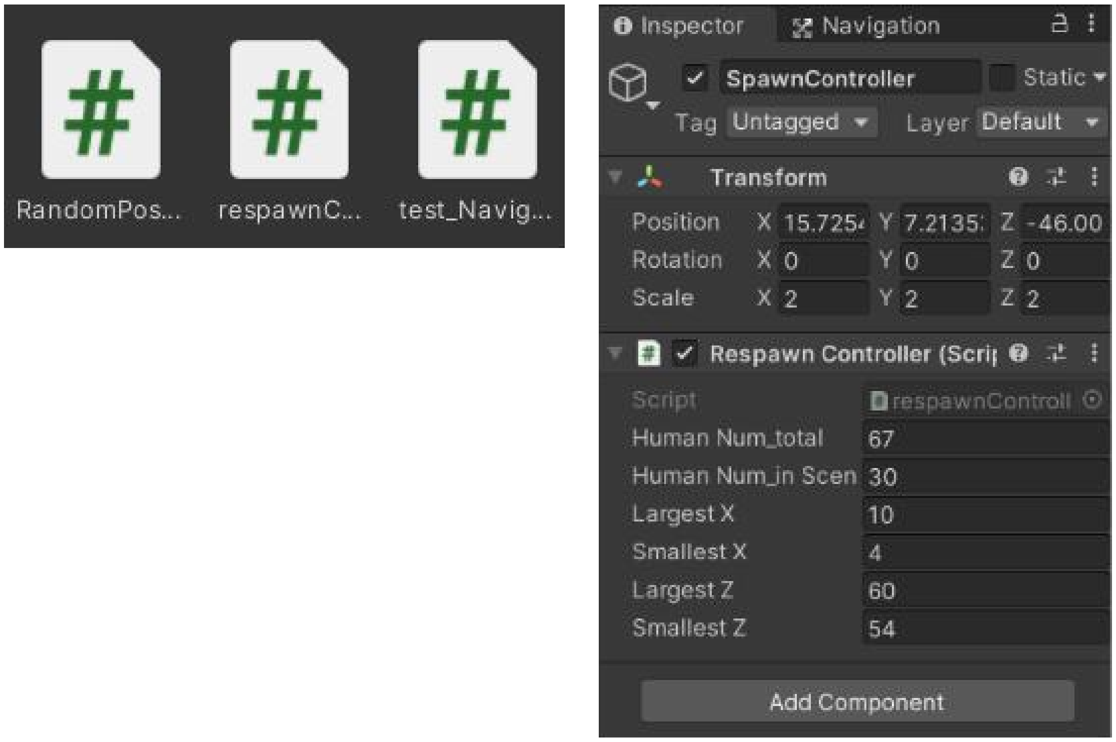


Idle
Tag
Motion M_idle
Speed 1
Multiplier Parameter
Motion Time Parameter
Mirror Parameter
Cycle Offset 0 Parameter
Foot IK
Write Defaults ✓
Transitions Solo Mute
Idle -> walking

Figure 2.11: The animations of male and female models are different. A Boolean parameter ”walking” is used to switch the status between ”idle” and ”walking”.

(10) Human character Programming

For human character programming, a script named ”respawnController” is used. When the keyboard code Space is down, human characters will generate in a random position and walk. First of all, to let the human characters walk in the predetermined area, the navigation in the AI window needs to be baked. After baking, the project will recognize the plain area automatically, and the trajectory will generate. In the script, c language is used. A function called “Randoms” generates the randomized human characters list. In the Update function, public parameters ”largestX”, ”smallestX”, ”largestZ”, and ”smallestZ” are used to constrain the range of character generation. These parameters can be modified externally, making the script adaptable for different scenarios. The number of generated characters can be modified externally as well. The external parameters are shown in the figure below.



SpawnController
Static
Tag Untagged
Layer Default

Transform

Position	X	15.725	Y	7.2135	Z	-46.00
Rotation	X	0	Y	0	Z	0
Scale	X	2	Y	2	Z	2

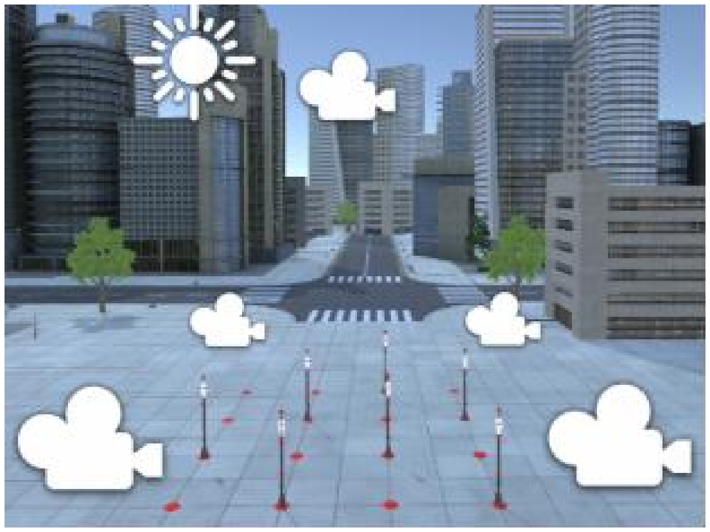
Respawn Controller (Scri

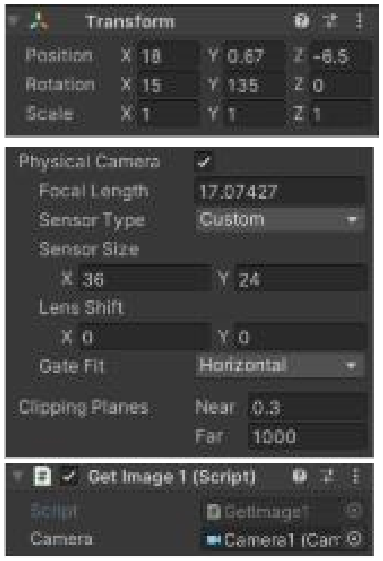
Script	respawnControll
Human Num_total	67
Human Num_in Scen	30
Largest X	10
Smallest X	4
Largest Z	60
Smallest Z	54

Figure 2.12: Every human character model bonds with the ”respawnController” script. The parameters are public and can be modified externally.

2.1.2 Camera modeling

In the city scenario, twenty-five cameras are put in different orientations. These are the top view and eight views at 3 different heights. Every camera has either 15, 30 and 45 degrees of inclination and is placed in the corner and middle of side in the scene. To make the screenshot, scripts are added to these cameras. In the script “GetImage”, this project is programmed to take a photo when the keyboard code “S” is pressed. The photos will automatically save in the file “Screenshot”. Besides, little red dots and poles are put on the floor as the benchmark to convenient the calibration. These make the calibration procedure more convenient and clearer. The details are shown in the figure below.





```
Transform
Position   X 18    Y 0.67   Z -6.5
Rotation   X 15    Y 135    Z 0
Scale      X 1     Y 1      Z 1

Physical Camera     ✓
  Focal Length      17.07427
  Sensor Type       Custom
  Sensor Size
    X 36            Y 24
  Lens Shift
    X 0             Y 0
  Gate Fit          Horizontal
Clipping Planes     Near 0.3
                    Far  1000

Get Image 1 (Script)
Script              GetImage1
Camera              Camera1 (Car
```

Figure 2.13: Cameras are placed at the corners of the scene. The inclination angle is 15 degrees on the x-axis and adding 90 degrees on the y-axis. The script ”GetImage” is added to each camera to take the photos. Dots and poles are placed on the ground as the benchmark.

2.1.3 Scene Modeling

There are two scenes built in this project. The first one is the city scene. Construction of this scene can be divided into these steps: 1. set the terrain; 2. put the prefabs in the scene; 3. set the lighting system. To build this scene, the prefabs are needed. All the prefabs are shown in the figure below.

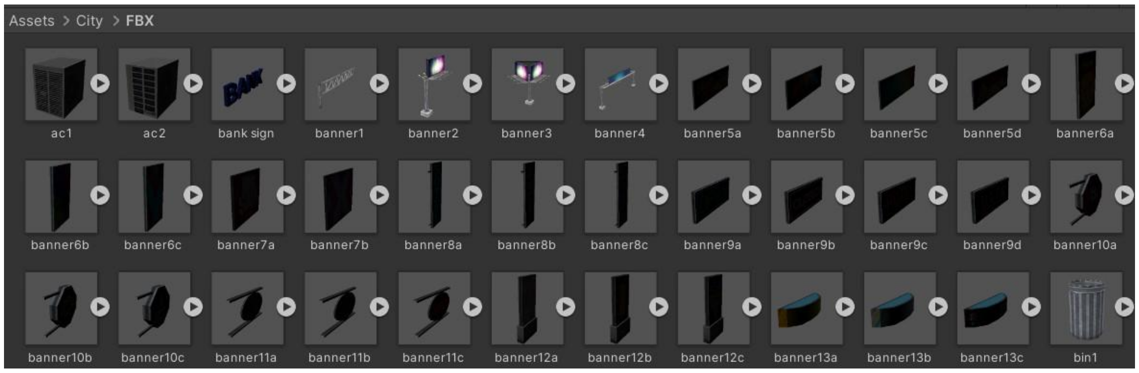


Figure 2.14: The prefabs of the city scene. There are banners, trees, buildings, streetlights et al.

These prefabs need to be dragged into the scene. The shortcut key “W” is used to transform the prefabs; “E” is used to rotate the prefabs; “R” is used to resize the prefabs. A square is built in the center of this scene, and it is used as the walkable area for the pedestrians. Various buildings and surrounding structures like bench, crossroad, and trees are added to the scene. These details are shown in the figures below.

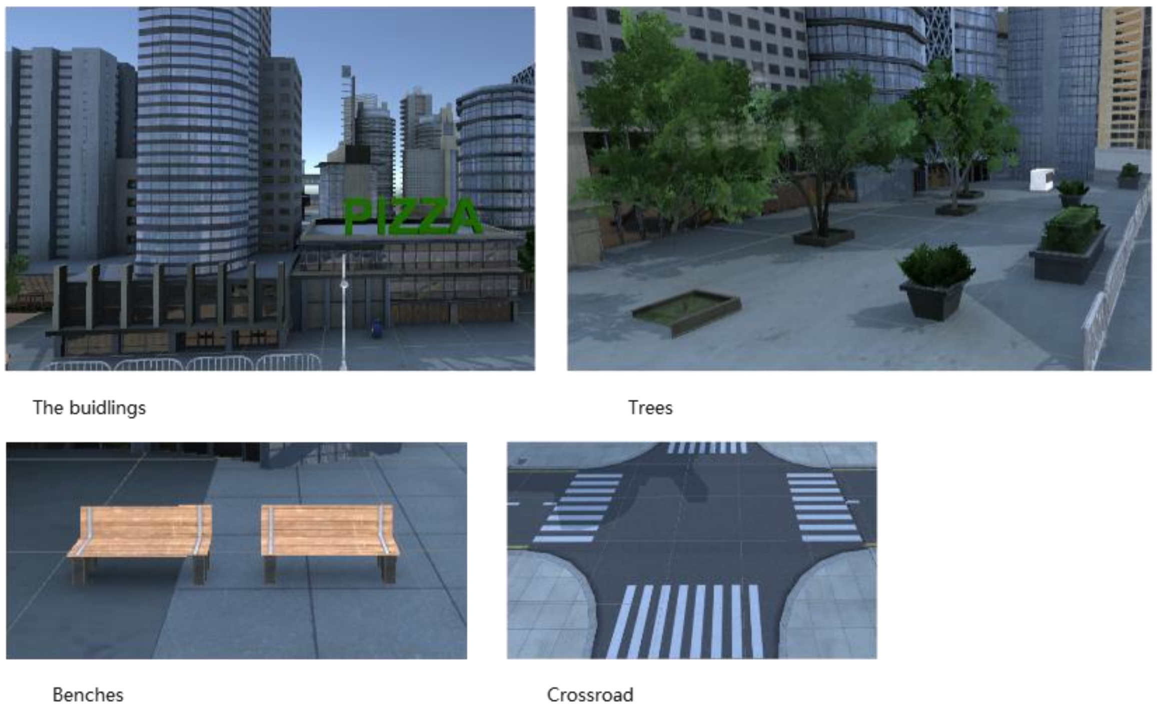
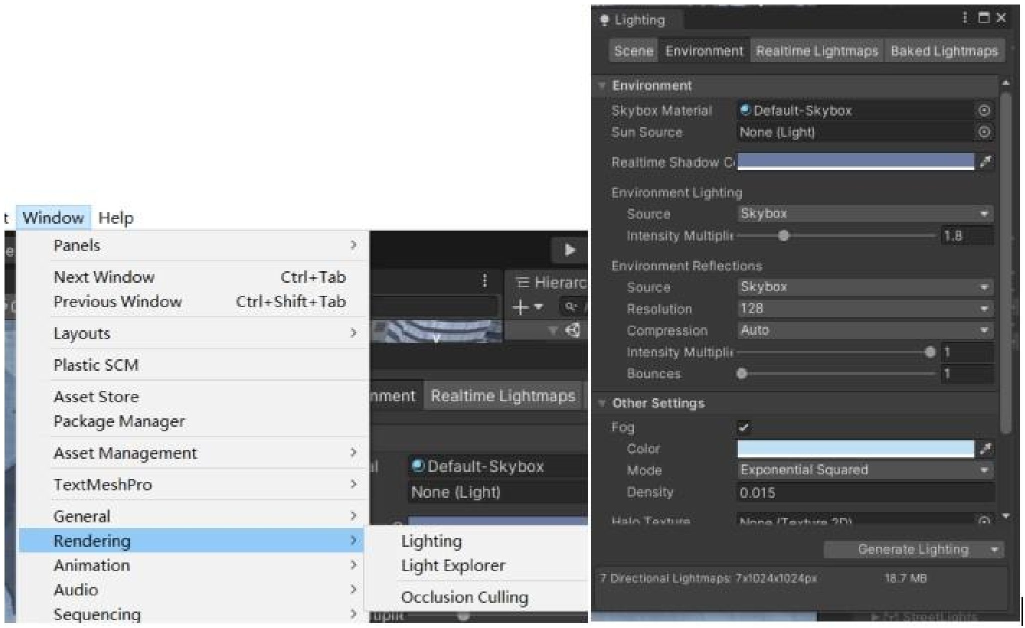


Figure 2.15: The prefabs put in the scene and combined together to construct the city scene The lighting system is also set with the rendering light and sky box.



```
Environment
  Skybox Material          Default-Skybox
  Sun Source               None (Light)
  Realtime Shadow C
  Environment Lighting
    Source                 Skybox
    Intensity Multipli     1.8
  Environment Reflections
    Source                 Skybox
    Resolution             128
    Compression            Auto
    Intensity Multipli     1
    Bounces                1
Other Settings
  Fog                      ✓
    Color
    Mode                   Exponential Squared
    Density                0.015
7 Directional Lightmaps: 7x1024x1024px     18.7 MB
```

Figure 2.16: The setup of skybox in Unity.

The whole structure of the scene is shown in the figure below. There are central square, surrounding buildings, crossroad and trees.

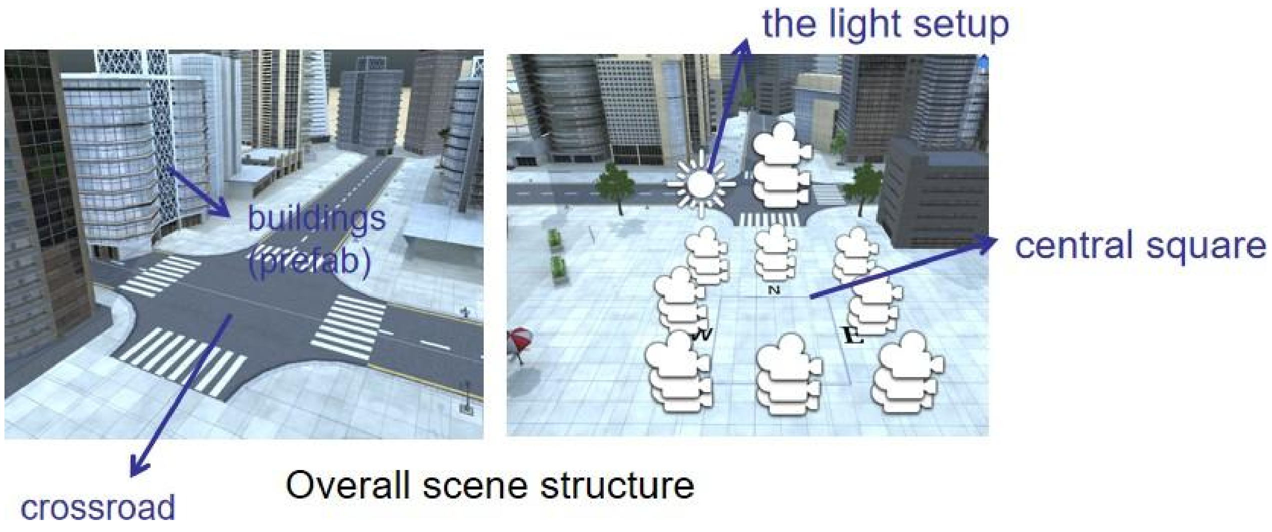
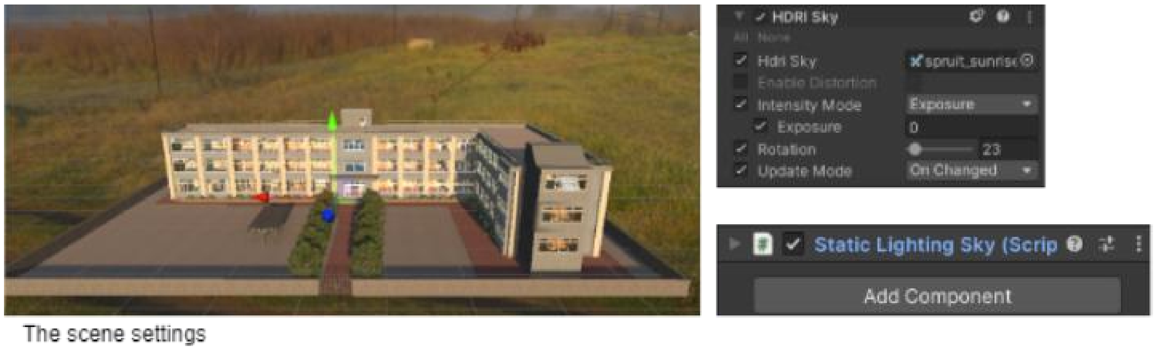


Figure 2.17: The overall output of the city scene.

Additionally, a High-Definition Render Pipeline (HDRP) project is also created. Compared with the traditional Lightweight Pipeline (LWRP), HDRP has a massive advance in graphical realism. It can also achieve realistic graphics in demanding scenarios. Referring to the global datasets, an indoor scene is made by the Unity HDRP project. It is a scene of campus.

The construction of this scene is the same as the previous scene. Firstly, the terrain of the campus is set. The skybox is added to the scene, and the material “spruit_sunrise” is selected in the skybox. Then the prefabs are dragged to the scene to construct the campus scenario. The scene has the following structures: stairs, rooftop, floor, ceiling, doors, wall, pillar, and roof. These structures are combined to form these properties: external, classroom, restroom and corridors. For the lighting system, directional light is added and lights up the whole scene. A ”spotlight” and ”AreaLightNeon” are combined to light up the room and located in the right position in each classroom. The details are shown in the figures below.



HDRI Sky
Hdri Sky: spruit_sunrise
Enable Distortion
Intensity Mode: Exposure
Exposure: 0
Rotation: 23
Update Mode: On Changed
Static Lighting Sky (Scrip

Figure 2.18: ”spruit_sunrise” is chosen as the sky box; the script is bonded to the scene to generate the environment light.

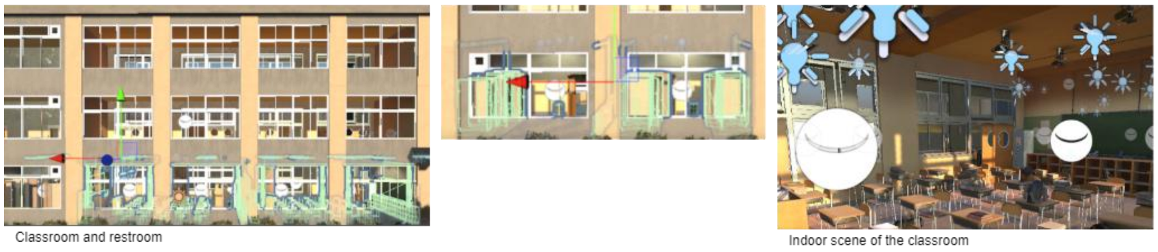


Figure 2.19: Restrooms and classrooms built in the scene.

This scene is used for further investigation of the indoor scene. The grid and human character models can be put in it if necessary.

2.2 Verification of the datasets

Since the cameras have generated images of each direction, these images should then be verified. It is a pedestrian datasets used for pattern recognition, thus the localization of pedestrian models needs to be outlined. The datasets is a 3d scene, so each model's location can not be detected directly. To achieve this goal, maskRCNN, Tsai's calibration algorithms, RSS algorithms are used. The steps are shown in detail below.

2.2.1 Homography Transformation

Homography describes the position mapping between the world coordinate system and the pixel coordinate system. The corresponding transformation matrix is called the homography matrix (2.1).

$$H = s \times \begin{matrix} f_x & \gamma & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{matrix} \times [r_1 r_2 t] = s \times M[r_1 r_2 t] \quad (2.1)$$

H: Homography matrix, S: Arbitrary scale factor, M: Camera intrinsic parameter. The matrix can be divided into four parts: $\begin{bmatrix} f\_x & \gamma \\ 0 & f\_y \end{bmatrix}$ represents the linear transformation, e.g., scaling, shearing and rotation. [0 0] is used for translation, $[u_0 v_0]^T$ is used to generate the perspective transformation. The transformation matrix can transform one quadrilateral into another quadrilateral.

In this part, OpenCV is used to calculate the homography matrix of two different frames. Initially, PS is used to get the pixel positions of the same location in two photos of different visual angles. Four positions are needed and stored in a vector2 parameter. Then use the function "findHomography" to calculate the corresponding homography matrix. The calculation of the homography matrix needs the corresponding points "srcPoints" and "dstPoints". The points are the matrix in the form of Vector¡Point2f¿. After getting the homography matrix, the function "warpPerspective" can map a picture into another visual angle.

2.2.2 Tsai's calibration

It can be seen that the homography works well for ground point in world coordinate to image coordinate transformation, which is a 2d transformation. However, in this project, it is a 3d scene. The rods are fully distorted through homograph transformation. Therefore, Tsai's algorithm is then used. This algorithm works well for world coordinate (Xw Yw Zw), camera coordinate (x, y, z), and image coordinate (image pixel and physical coordinate) transformation. The transformation relationships among these coordinates are shown in the figure below.

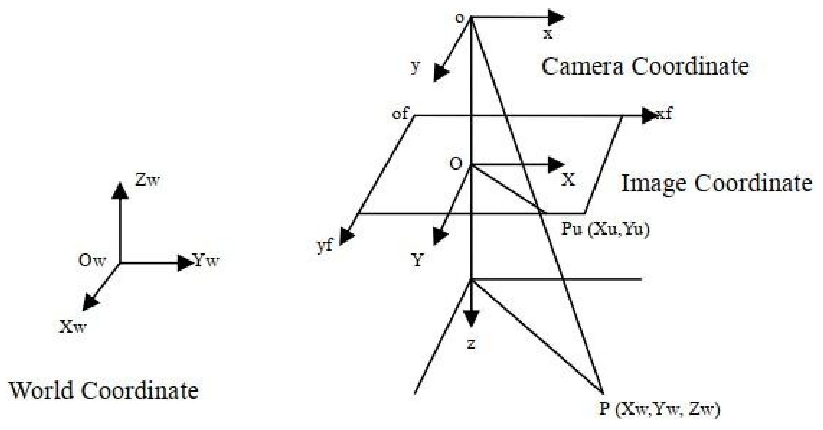


Figure 2.20: The relationships of three basic coordinates. (camera coordinate, image coordinate and world coordinate)

In this project, it requires to transform the image from world coordinate to image coordinate. The formula is shown in the function (2.2) below.

$$\frac{X}{f} = \frac{u - u_0}{f_x} = \frac{r_{11}X_w + r_{12}Y_w + r_{13}Z_w + t_x}{r_{31}X_w + r_{32}Y_w + r_{33}Z_w + t_z}, \qquad \frac{Y}{f} = \frac{u - u_0}{f_y} = \frac{r_{21}X_w + r_{22}Y_w + r_{23}Z_w + t_y}{r_{31}X_w + r_{32}Y_w + r_{33}Z_w + t_z} \tag{2.2}$$

Where R is the orthogonal rotational matrix. Its elements satisfy the condition:

$$\begin{aligned} r_{11}^2 + r_{12}^2 + r_{13}^2 &= 1, \\ r_{21}^2 + r_{22}^2 + r_{23}^2 &= 1, \\ r_{31}^2 + r_{32}^2 + r_{33}^2 &= 1 \end{aligned} \tag{2.3}$$

To achieve this transformation, the Matlab code is written. Besides, a script is also written in Matlab to save all the world to image points in two "txt" files. One of these files is to store the ground point, another is to save the height point.

2.2.3 Division of the ground truth into a grid of position

The next step is to split the walkable areas into small grids. Small dots are placed in the square, each 20cm apart in the top view. These small dots will then be used as the midpoint of the bottom of the rectangle to draw the rectangles. The height of each rectangle is 170cm, and the width is 0.35 times of height. However, these rectangles' absolute world position and pixel position will change when observed in different orientations. Therefore, Tsai's algorithm is introduced to achieve this transformation. Besides, the character of each direction is placed on the ground, makes the observation more convenient. The figure of top view with small dots is shown in figure below.

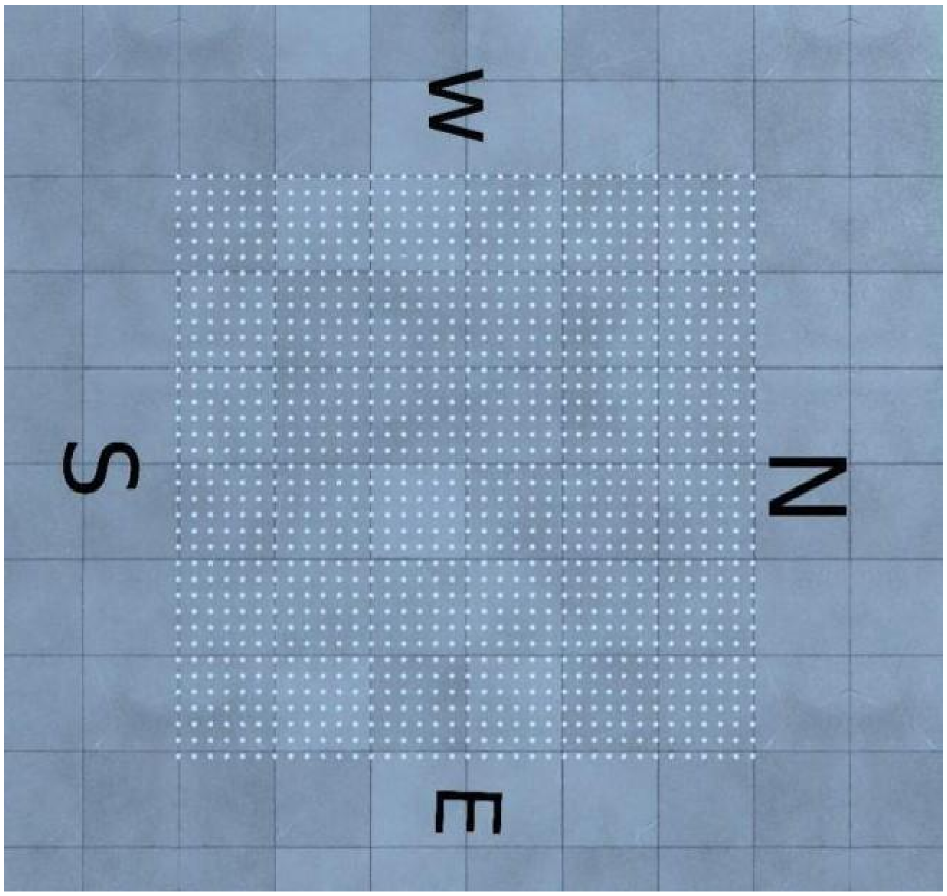


Figure 2.21: The top view of the scene.

2.2.4 Pedestrian detection in single view

In this step, the algorithm of maskRCNN is introduced to extract the mask of every pedestrian. MaskRCNN is widely used for segmentation and object detection.

Bounding boxes and segmentation masks are generated in this model for each instance of an object in the image. It is based on Feature Pyramid Network (FPN) and a ResNet101 backbone. The main construction structure is shown in figure below.

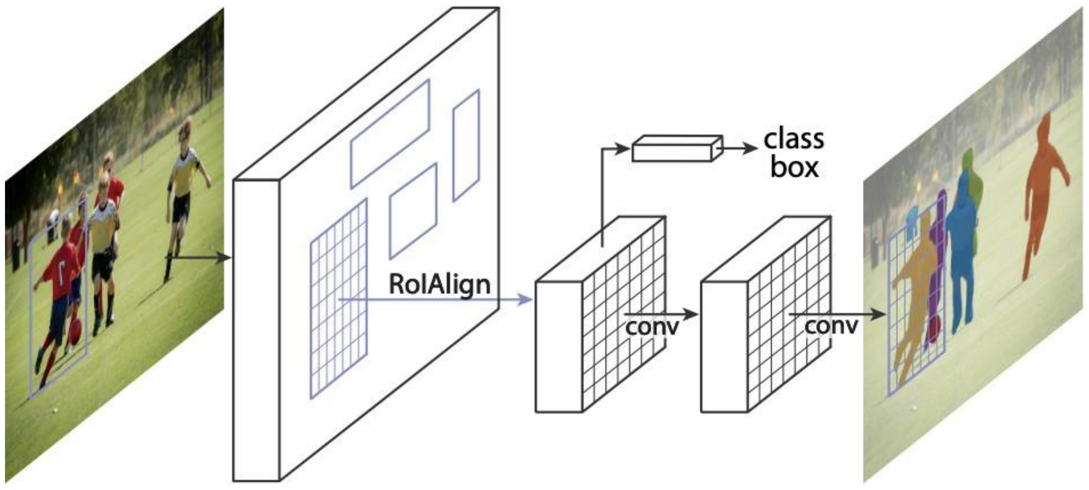


Figure 2.22: The instance segmentation framework of Mask R-CNN cited from [30].

This is a standard convolutional neural network (ResNet50 and ResNet101) that acts as a feature extractor. The bottom level detects low-level features (edges and corners, etc.), and the higher level detects higher-level features (cars, people, etc.). The image is converted from a tensor of 1024x1024x3 (RGB) to a feature map of shape 32x32x2048 forward propagation of the backbone network. The network is further improved by introducing the FPN, which is an extension of the backbone network to better characterize targets at multiple dimensions. The framework is shown in the figure below.

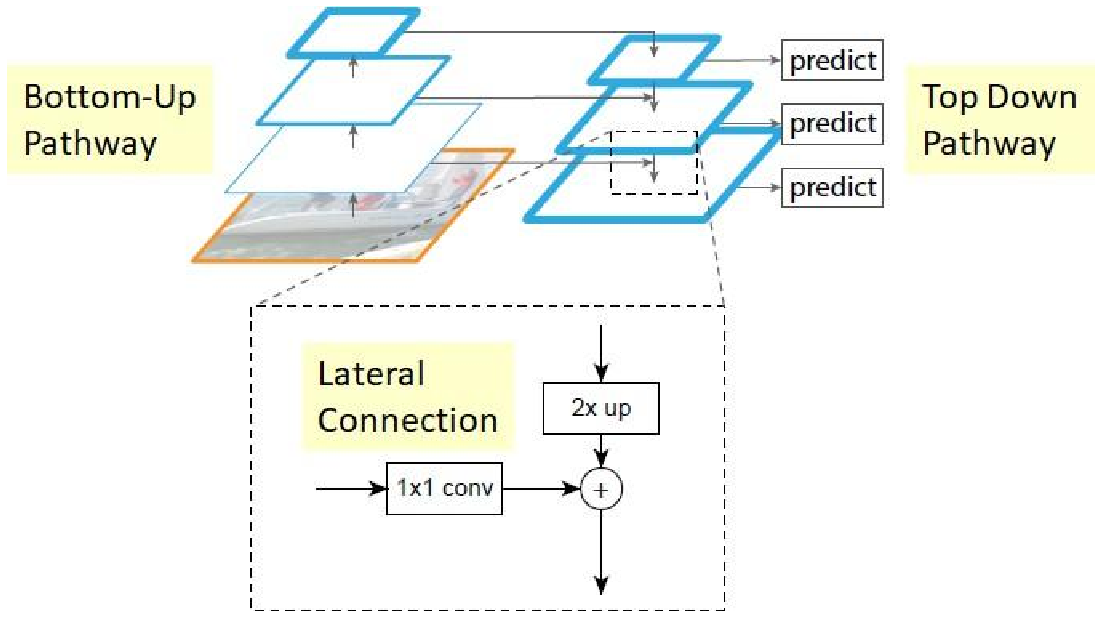


Figure 2.23: The framework of FPN cited from [30].

Since this project needs the localization of each pedestrian, then the mask needs to be detected at first. After using MaskRCNN, the mask of each pedestrian in a different orientation is shown below. It needs to mention that only the pedestrians' masks are needed to be detected. Therefore, the category of detection needs to be narrowed down to only "person".

2.2.5 Joint Occupancy Likelihood

In step2, the middle point of each rectangle's bottom and the height, and width of each rectangle have been calculated. However, there will be too many rectangles if every rectangle is drawn. Therefore, the joint occupancy likelihood will be calculated, then draw the rectangles whose JOL is higher than the threshold. JOL is to calculate the percentage of mask and size of one rectangle in different views.

First of all, it needs to transform the frames of masks after using MaskRCNN to gray images. After this transformation, the pixel value of the mask is 255 and the background part value is 0. Iterate every pixel in this rectangle, and count the ratio of mask and background pixels. The ratio is called Occupancy Likelihood. After calculating the Occupancy Likelihood of one rectangle in different views, It needs to multiply all these likelihoods together and then calculate the square root. The result is the Joint Occupancy Likelihood. After calculating the JOL of every rectangle, set up a threshold and draw rectangles whose JOL is larger than the threshold. The pseudocode is shown in the below.

Algorithm 1 JOL algorithm

```
Input: input parameters ground points, height points, threshold
Output: output JOL_rectangles
1: draw JOL rectangles larger than the threshold
2: for height_point, ground_point in heights_points, ground points do
3:     $H_r \leftarrow height\_point, ground\_point$
4:     $W_r \leftarrow H_r * 0.35$
5:     iterate every pixel in this rectangle
6:     if the pixel value is larger than 1 then
7:         v+1
8:     end if
9:     $OL = v/W_r * H_r$
10:    $JOL = \sqrt[n]{\sum_{i=0}^{n} OL}$
11:    if JOL > threshold then
12:        add to JOL_rectangles
13:    end if
14: end for
15: return result
```

2.2.6 Repulsive Spatial Sparsity (RSS)

It can be seen that there are still plenty of rectangles on each pedestrian after computing the Joint Occupancy Likelihoods and filtering the rectangles. This is because the JOL of each pedestrian is different. If the pedestrian's size is large, the JOL will be large as well. A universal threshold can not filter all the rectangles. Therefore, there are multiple rectangles whose JOL is larger than the threshold on each pedestrian. Therefore, another algorithm RSS is used. It is a greedy algorithm which iterates all the rectangles on each character. First of all, the JOLs need to be sorted from the highest to lowest. Then a radius is chosen to filter the rectangles. Inside of the circle, only the rectangle with the highest JOL will remain. The logic of the pesudocode is shown in below.

Algorithm 2 RSS algorithm

Input: input parameters JOL_rectangles, radius
Output: output RSS_rectangles

```
1: iterate all the rectangles in JOL_rectangles
2: while JOL_rectngales is not empty do
3:     sort the rectangles from highest to lowest
4:     $Hr \leftarrow JOL\_rectangles[0]$
5:     $H_p \leftarrow JOL\_rectangles[0]["position"]$
6:     for JOL_rectngale i in JOL_rectangles do
7:         $R_p \leftarrow JOL\_rectangles[i]["position"]$
8:         if R_p < radius then
9:             delete JOL_rectangle[i] from JOL_rectangles
10:        end if
11:    end for
12: end while
13: return result
```

Chapter 3

Results

During this final year, the results are: (1) Create the human character models. (2) Shoot the pedestrian photos from different orientations. (3) Generate the city scene and HDRI campus scene. (4) Validate the homography of the figures at different orientations in the synthesized scene. (5) Masks extracted by using MaskRCNN. (6) Tsai's calibration result. (7) JOL results. (8) RSS results. These will be discussed in details.

3.1 Pedestrian datasets sybthesis

More than 100 human character models have also been built during this semester. The details are shown in the figure below.

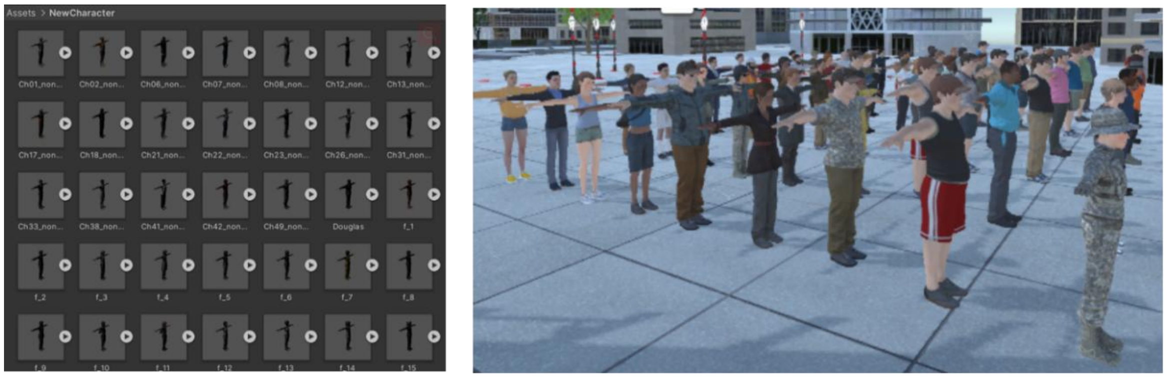


Figure 3.1: The human character models in the file and in the scene.

Human character models are programmed to walk in the predetermined area. The number of randomly generated models is 40 in each scene. Whenever the keyboard code space is down, 40 random human characters of 100 models will randomly generate in the area. The area is constrained by the parameter “LargestX”, “LargestZ”, “SmallestX”, and “SmallestZ”. All these parameters can be modified externally according to the different scenes.

3.2 Pedestrian photos in different orientations

Moreover, the script for photo shooting is also completed. When the program is running, pressing the keyboard button “s” will let the cameras generate a set of photos at different orientations. These photos are stored in the relative path “Assets/Screenshot”. The shooting photos at different orientations with randomly generated pedestrians are shown in the figures below.

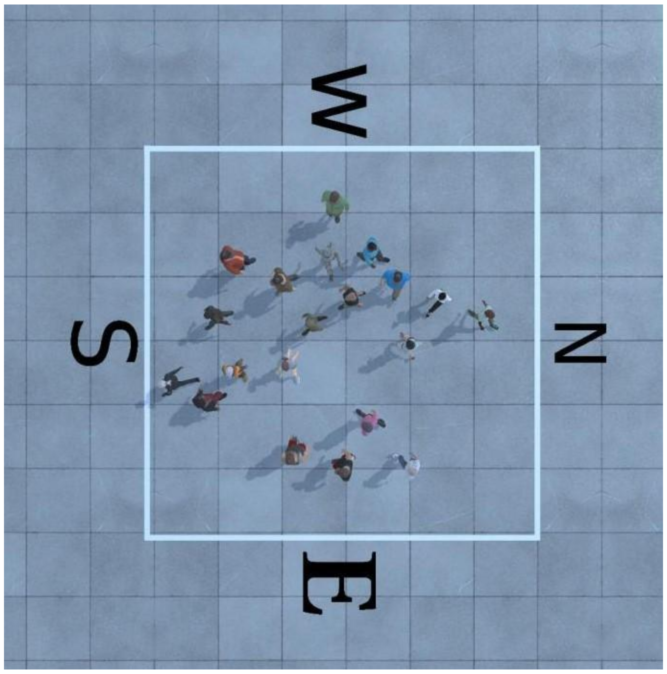
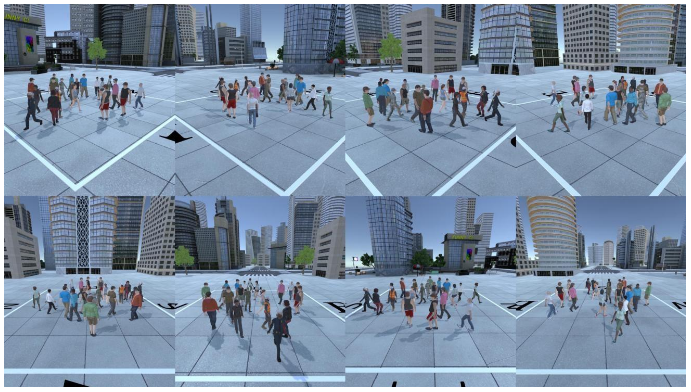
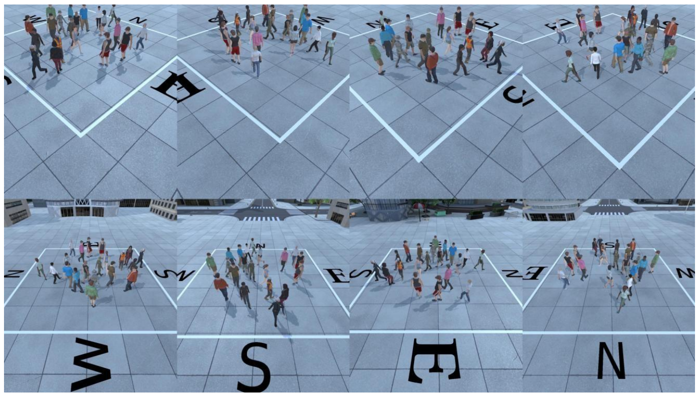


Figure 3.2: Photo taken in the top view.

(a) Photos taken in the low altitude.


(b) Photos taken in the middle altitude.



(c) Photos taken in the high altitude.

Figure 3.3: Randomly generated pedestrians walk in the predetermined area. Photos are shot from twenty five different orientations.

3.3 The synthesized scenes

The synthesized city scene is shown in the figure below. There are a central square, buildings, and trees. The central square is used as the walkable area for the pedestrians.

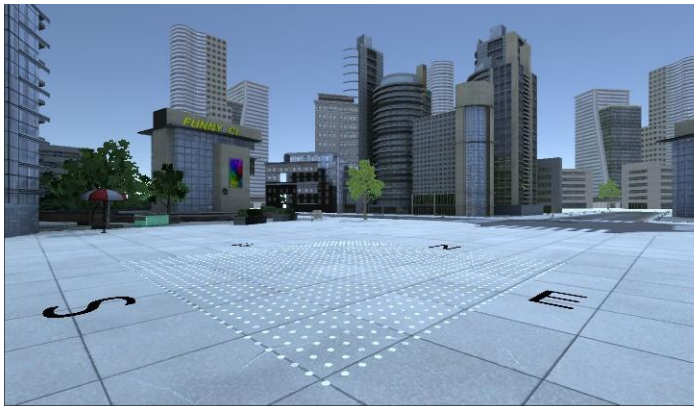


Figure 3.4: The overall structure of the city scene.

The HDRI campus scene is shown in the Figure 29. It is a campus scene, rendered in HDRI pipeline. It has everything needed in a real-world classroom scene. Compared with the previous one, it is much more realistic. The render result is shown as the figure below.

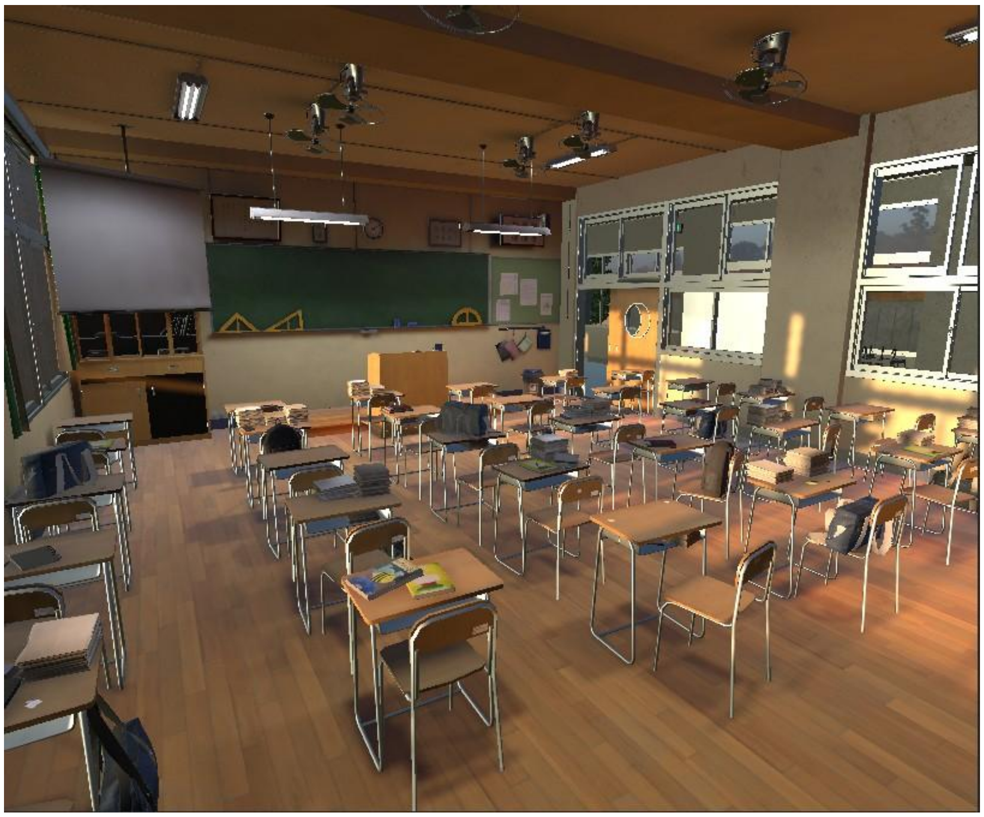


Figure 3.5: The overall structure of the campus scene.

3.4 Homography validation of the figures

After generating these scenes, the homography is validated through OpenCV. For homography mapping, the photos at each orientation are mapped with the top view photos. The results are shown in figure below.

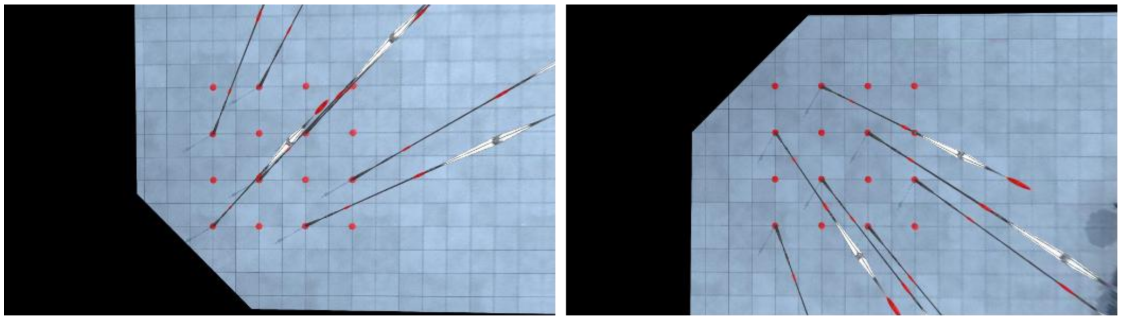


Figure 3.6: Homography transformation of views of Camera0 with Camera1 and Camera 2.

It can be seen that the photos are transformed successfully. The length of the poles is different, and it can be considered the shadow of different angles of light. The figures are added together to figure out whether the transformation is accurate to have further validation. The adding weight is set as 0.5 to have a clearer result. The result is shown in the figure below.

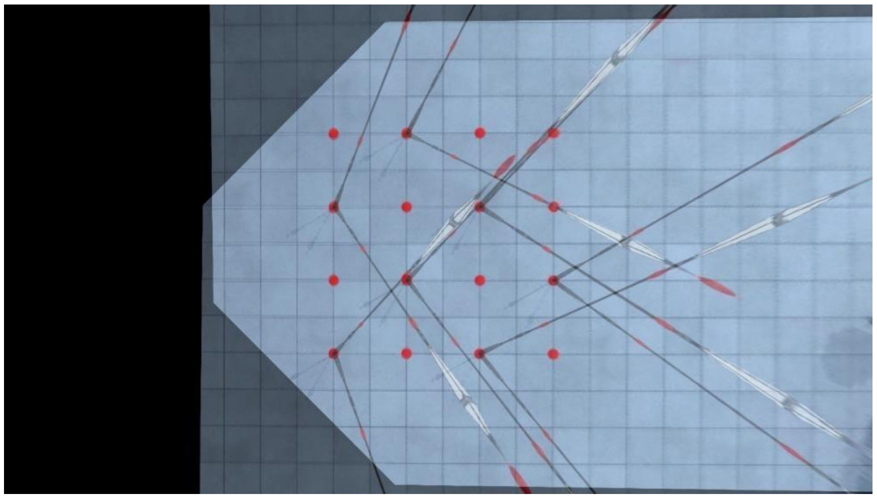


Figure 3.7: Add the figures of each homography results together, with the weight of 0.5 each.

It can be seen that the dots coincide perfectly. However, the lines outside the predetermined region somehow incline due to the transformation points being selected only in the predetermined area.

3.5 Masks extracted by using MaskRCNN

After validating the homography of the synthesized scene, the localization and geometric relationship of the pedestrians are verified. The result of MaskRCNN is shown in the figure below.

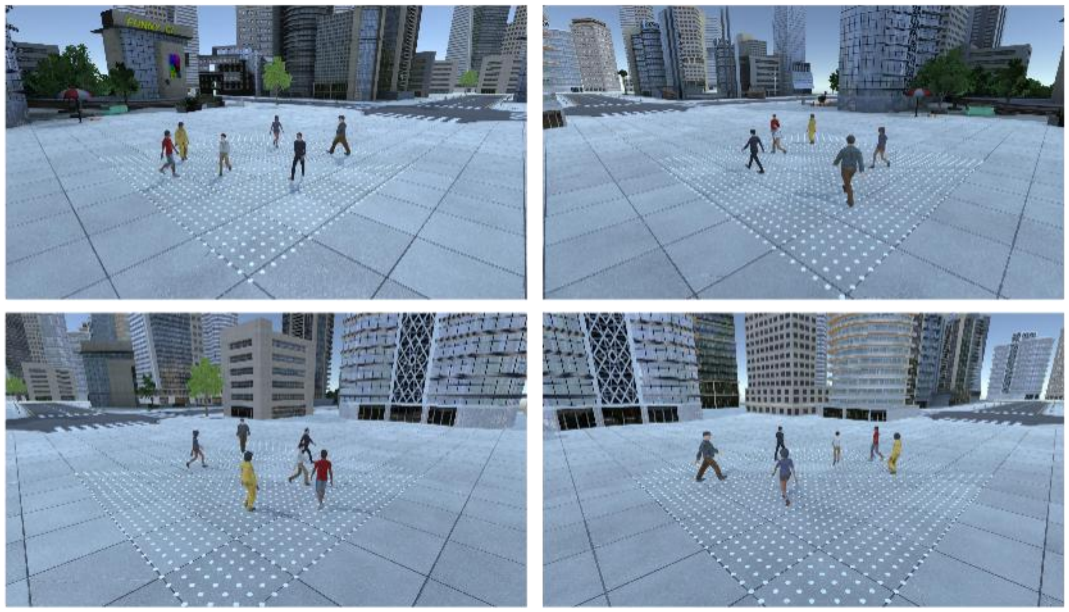


Figure 3.8: The images taken from four different views (southwest, northwest, southeast, and northeast).

The images are then transformed into the gray level images. These gray images are shown in the figure 33. It can be seen that the the persons mask are extracted and transformed into white color.

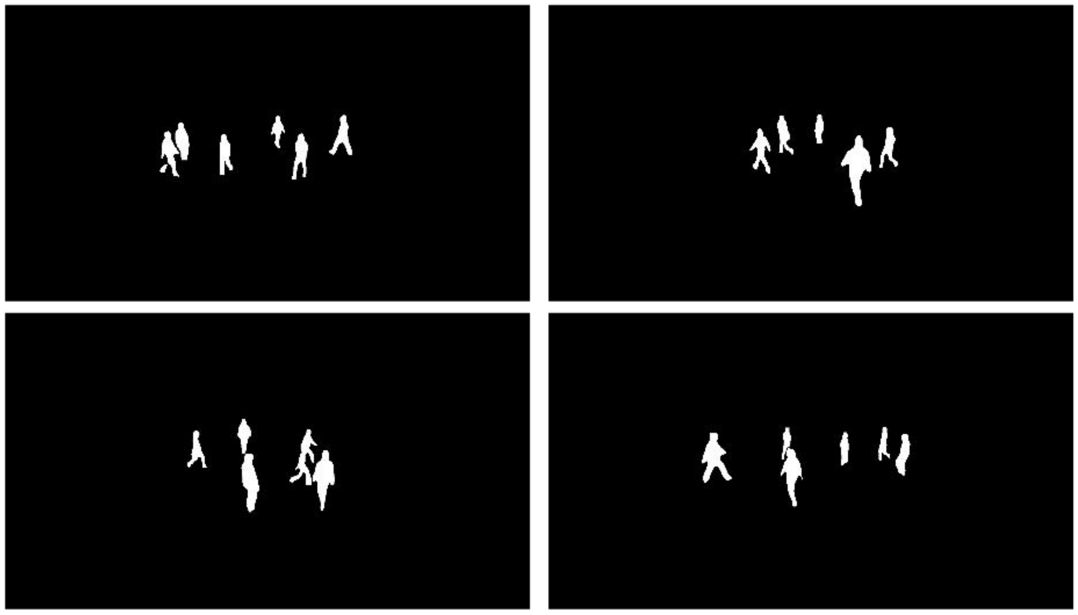


Figure 3.9: The gray image of the masks images.

3.6 Tsai's calibration result

The world coordinate has been transformed to image coordinate successfully. In the program, if a point is clicked on the left side, the corresponding calculated point will be generated on the right side. There are two corresponding points in the image. One is the ground point and the another one is the height point. The height is set as 170cm in the program. The output of the Tsai's algorithm is shown in the figure below.

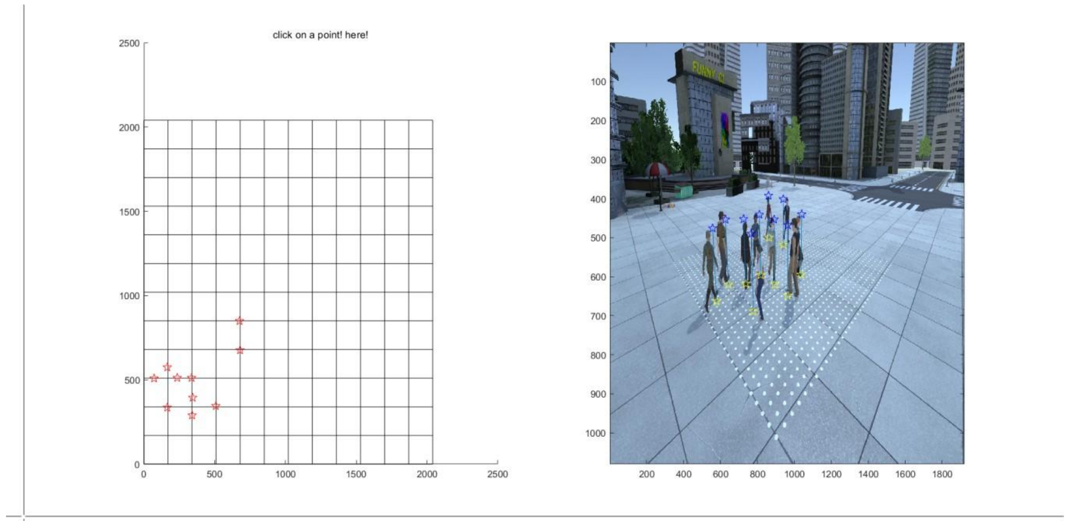


Figure 3.10: The output of the Tsai's algorithm. The left one is world coordinate and the right one is the image coordinate. The points on each side are corresponding to each other.

3.7 JOL results

After calculating the JOL, rectangles are drawn if the JOL is larger than the threshold. The rectangles in different views are corresponding to each other. However, it can be seen that there are multiple rectangles on each character, and the number of rectangles on each character is different. This is because the threshold is a fixed value, while the size of each character is different. Therefore, the number of rectangles whose JOL is larger than the threshold on each character is different. The result is shown in figure below.

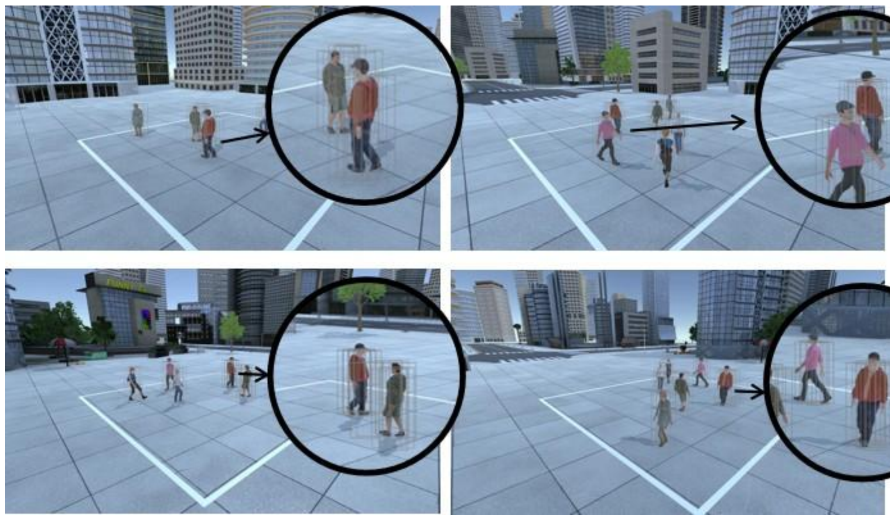


Figure 3.11: The result of using JOL algorithm to draw the rectangles. The rectangles in each view is corresponding to each other.

3.8 RSS results

After applying the RSS algorithm, it can be seen that there is only one rectangle remains on each character in different views. These corresponding rectangles are drawn in the same color. The result is shown in the figure below.

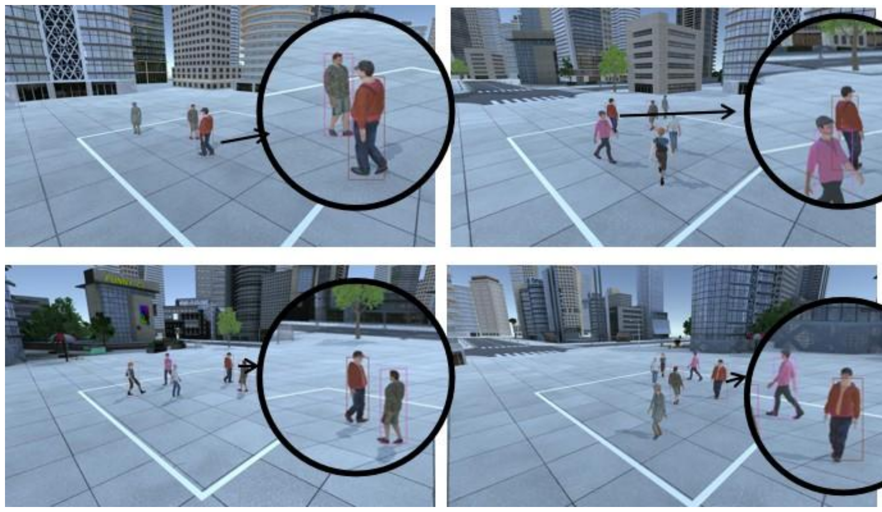


Figure 3.12: The result of using RSS algorithm to draw the rectangles. The rectangles in each view is corresponding to each other. Only one rectangle remains on each character.

Chapter 4

Conclusions and Future Work

4.1 Conclusions

During this year, I have realized modelling and verification part. Besides, human character modelling fits well in the synthesized scene. The pedestrian models are programmed to walk in the predetermined area and are generated randomly. The photos are shot through the cameras placed at different orientations. Then the dots and poles are put in the scene as the benchmark to validate the homography. The results show that the homography transformation has succeeded. Moreover, the localization and geometric relationship of these characters are verified correctly as well.

In the procedure, difficulties have been met. First of all, human modelling is a long procedure. Since the existing public models are inadequate, while the project requires many pedestrian models, "fuse" is used to create the models. Moreover, these models are set as "rigidBody" to avoid occlusion. The lighting system in the scene has been adjusted to make clear photos. The size and position of the benchmark points are also adjusted for calibration and validation. The radius of RSS is also adjust for achieving the best result. The results show that the synthesized scene can be used as the pedestrian detection training dataset. The verification of the synthesized dataset also means that it has the same properties as the real-world dataset.

4.2 Progress Analysis

As mentioned in the Project Specification Report, I should have done these works in semester 1: (1) Literature review for the related work. (2) Create the human character models and add them to the scene. (3) Adjust the models to fit the scene and program the models to walk in the predetermined area randomly. According to my preliminary results, these works have been done successfully. More than 100 human character models have been built using the “fuse”. They are then added to the synthesized scene in Unity3D. These models are programmed to generate and walk in the predetermined area randomly. C programming language is used in the scripts. Besides, dots and poles are put in the scene to validate the homography. The validation is successful, according to the previous analysis. I also need to finish these work in semester 2: (1)Verify the synthesized datasets. (2) Adjust the datasets. (3) Finish the final Report. In the final result, algorithms like Tsai’s calibration, Joint Occupancy Likelihod, MaskRCNN et al are used to verified the datasets. The results work as the expectation. Matlab, and python scripts are written to achieve these algorithms in semester 2.

4.3 Future Work

There are still some limitations in the present stage. Firstly, the scene is limited. Only the city scene has pedestrian models. Moreover, the synthetic dataset should be adjusted according to the demand. The algorithms I used to verify the dataset work no so well when too many pedestrians are in the scene and they are occluded. Therefore, a more robust and high performance algorithm may be used to verify the datasets in the future.

The advantages of synthetic datasets are obvious. The future work of these datasets will focus on the following aspects: (1) Build a more interactive, better data quality synthetic data set engine with richer (and more variable) conditions, giving users more authority to obtain data that meets as much further research needs as possible. (2) Research on more efficient unsupervised domain adaptation methods to make more efficient use of synthetic data and existing real data sets. (3) The problem of ”what-how-why” in visual tasks is deeply studied and explored. The controllability of synthetic data is used to design more scientific quantitative experiments and rating indexes so that visual research can develop in a balanced way between engineering and science.

Reference

[1] Wojek, C., Dollar, P., Schiele, B., Perona, P.: Pedestrian Detection: An Evaluation of the Stateof the Art. IEEE Transactions on Pattern Analysis and Machine Intelligence 34(4), pp. 743–761, 2012.

[2] Benenson, R., Omran, M., Hosang, J., Schiele, B.: Ten Years of Pedestrian Detection, What Have We Learned?. In: Proceedings of European Conference on Computer Vision, pp. 613–627, 2014

[3] ”Unity user manual”, https://docs.unity3d.com/Manual/index.html

[4] J. Ferryman, D. Tweed, An overview of the pets 2007 dataset, in: Workshop on Performance Evaluation of Tracking and Surveillance, 2007

[5] J. Ferryman, A. Shahrokni, An overview of the pets 2009 dataset, in: Workshop on Performance Evaluation of Tracking and Surveillance, 2009.

[6] Y. Liu, R. Collins, Y. Tsin, Gait sequence analysis using frieze patterns, in: Computer Vision –ECCV 2002, vol. 2351, 2002, pp. 657–671.

[7] H. Ragheb, S. Velastin, P. Remagnino, T. Ellis, Vihasi: virtual human action silhouette data for the performance evaluation of silhouette-based action recognition methods, in: 2008 Second ACM/IEEE International Conference on Distributed Smart Cameras, ICDSC 2008, 2008, pp. 1–10.

[8] S. Singh, S. Velastin, H. Ragheb, Muhavi: A multicamera human action video dataset for the evaluation of action recognition methods, in 2010 Seventh IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), 2010, pp. 48–55.

[9] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, A. Blake, Real-time human pose recognition in parts from single depth images, Computer Vision and Pattern Recognition.

[10] Courty, Nicolas, et al. ”Using the AGORASET dataset: Assessing for the quality of crowd video analysis methods.”Pattern Recognition Letters 44 (2014): 161-170.

[11] Anton Milan, Laura Leal-Taix´e, Ian Reid, Stefan Roth, and Konrad Schindler. Mot16: A benchmark for multi-object tracking. arXiv preprint arXiv:1603.00831, 2016.

[12] Patrick Dendorfer, Hamid Rezatofighi, Anton Milan, Javen Shi, Daniel Cremers, Ian Reid, Stefan Roth, Konrad Schindler, and Laura Leal-Taixe. Cvpr19 tracking and detection challenge: How crowded can it get? arXiv preprint arXiv:1906.04567, 2019.

[13] Jerome Berclaz, Francois Fleuret, Engin Turetken, and Pascal Fua. Multiple object tracking using k-shortest paths optimization. IEEE transactions on pattern analysis and machine intelligence, 33(9):1806–1819, 2011.

[14] Lijun Cao, Weihua Chen, Xiaotang Chen, Shuai Zheng, and Kaiqi Huang. An equalised global graphical model-based approach for multi-camera object tracking. arXiv preprintarXiv:1502.03532, 2015.

[15] Francois Fleuret, Jerome Berclaz, Richard Lengagne, and Pascal Fua. Multicamera people tracking with a probabilistic occupancy map. IEEE transactions on pattern analysis and machine intelligence, 30(2):267–282, 2007.

[16] Tiziana D'Orazio, Marco Leo, Nicola Mosca, Paolo Spagnolo, and Pier Luigi Mazzeo. A semi-automatic system for ground truth generation of soccer video sequences. In 2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance, pages 559–564. IEEE, 2009.

[17] Christophe De Vleeschouwer, Fan Chen, Damien Delannay, Christophe Parisot, Christophe Chaudy, Eric Martrou, Andrea Cavallaro, et al. Distributed video acquisition and annotation for sport-event summarization. NEM summit, 8, 2008.

[18] J Ferryman and A Shahrokni. An overview of the pets 2009 challenge. 2009.

[19] Kohl, Philipp, et al. ”The MTA Dataset for Multi-Target Multi-Camera Pedestrian Tracking by Weighted Distance Aggregation.” Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops. 2020.

[20] Sun, Xiaoxiao, and Liang Zheng. ”Dissecting person re-identification from the viewpoint of viewpoint.” Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019.

[21] ”Adobe Fuse CC”, https://helpx.adobe.com/beta/fuse/help/create-first-3D-model.html

[22] Mueller, K., Smolic, A., Droese, M., Voigt, P., Wienand, T.: Multitexture modeling of 3d traffic scenes. In: Proc. of the 2003 Int'l Conference on Multimedia, vol. 2, pp. 657–660, 2003

[23] Stauffer, C., Tieu, K.: Automated multi-camera planar tracking correspondence modeling. In: Proc. IEEE Int'l Conference on Computer Vision and Pattern Recognition, vol. 1, pp. 259–266, 2003

[24] Black, J., Ellis, T., Rosin, P.: Multi view image surveillance and tracking. In: Proc. IEEE Workshop on Motion and Video Computing, pp. 169–174, 2002

[25] Kim, K., Davis, L.: Multi-camera tracking and segmentation of occluded people on ground plane using search-guided particle filtering. In: Proc. European Conference on Computer Vision, vol. 3, pp. 98–109, 2006

[26] Reddy, D., Sankaranarayanan, A., Cevher, V., Chellappa, R.: Compressed sensing for multi-view tracking and 3-D voxel reconstruction. In: Proc. IEEE Int'l Conference on Image Processing, pp. 221–224, 2008

[27] Fleuret, F., Berclaz, J., Lengagne, R., Fua, P.: Multicamera people tracking with a probabilistic occupancy map. IEEE Trans. Pattern Anal. Mach. Intell.30(2), 267–282, 2008

[28] Alahi, A., Jacques, L., Boursier, Y. et al. Sparsity Driven People Localization with a Heterogeneous Network of Cameras. J Math Imaging Vis 41, 39–58 (2011). https://doi.org/10.1007/s10851-010-0258-7

[29] He, Kaiming, et al. ”Mask r-cnn.” Proceedings of the IEEE international conference on computer vision. 2017.

Appendix A

C# code

A.1 RespawnController

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class respawnController : MonoBehaviour
{
    public int humanNum_total;
    public int humanNum_inScene;
    // Change the area size
    public float largestX;
    public float smallestX;
    public float largestZ;
    public float smallestZ;

    private ArrayList humanList=new ArrayList();   //list of ID of human,
        starts from zero
    private ArrayList objectList=new ArrayList();
    private ArrayList r_humanList; //list of the random IDs of human
    //private ArrayList r_SpawnList;   list of the random spawn point
    //private ArrayList r_x;
    //private ArrayList r_z;
    private int SpawnCounter = 0;
    void Start()
    {

        for(int i = 0; i < humanNum_total; i++)
```

```
            {
                humanList.Add(i);
            }
        }

        // Update is called once per frame
        void Update()
        {
            if(Input.GetKeyDown(KeyCode.Space))
            {
                if(SpawnCounter == 0)
                {
                    SpawnCounter++;
                }
                else
                {
                    foreach(GameObject Object in objectList)
                    {
                        Destroy(Object);
                    }
                }

                r_humanList = Randoms(0, humanNum_total, humanNum_inScene);
                //r_x = Randoms(smallestX, largestX, humanNum_inScene);
                //r_z = Randoms(smallestZ, largestZ, humanNum_inScene);
                for(int i=0; i < humanNum_inScene; i++)
                {
                    GameObject humanPrefab = Resources.
                        Load<GameObject>(r_humanList[i].ToString());
                    float X = UnityEngine.Random.Range(smallestX, largestX);
                    float Z = UnityEngine.Random.Range(smallestZ, largestZ);
                    Vector3 SpawnPosition = new Vector3(X, 0, Z);
                    GameObject human = Instantiate(humanPrefab,
                        SpawnPosition, UnityEngine.Random.rotation);
                    objectList.Add(human);
                }
                /*
                r_humanList=RandomNum(humanList);
                r_SpawnList = RandomList();
                for(int i= 0; i <humanNum_inScene; i++)
                {
                    GameObject testPrefab = Resources.Load<GameObject>(r_
                        humanList[i].ToString());
                    GameObject human = Instantiate(testPrefab, (Vector3)
                        r_SpawnList[i], UnityEngine.Random.rotation);
                    objectList.Add(human);
                }
                */
            }
        }

        public ArrayList Randoms(int begin, int end, int num)
        {
            ArrayList random = new ArrayList();
            /*
            System.Random rnd = new System.Random(Guid.NewGuid().GetHashCode
                ());
            for(int i = 0; i < humanNum_inScene; i++)
            {
                random.Add(rnd.Next(begin, end));
            }
            return(random);
            */
            while (random.Count< num)
            {
                int i=UnityEngine.Random.Range(begin,end);
                if(!random.Contains(i))
                {
                    random.Add(i);

                }
            }
            return random;
        }


        /*
        public ArrayList RandomNum(ArrayList list)
        {
            int resultID;
            ArrayList result = new ArrayList();
            ArrayList listClone = new ArrayList();
            foreach(int num in list)
            {
                listClone.Add(num);

        }

        while (listClone.Count > list.Count - humanNum inScene )
        {
            resultID= UnityEngine.Random.Range(0, listClone.Count);
            result.Add(listClone[resultID]);
            listClone.RemoveAt(resultID);
        }

            return(result);
        }

    public Array List RandomList ()
    {
        int x;
        int z;
        Array List SpawnList = new Array List (); while
        ( SpawnList . Count < humanNum inScene ) _
        {
            x = UnityEngine.Random.Range(smallestX, largestX);
            z = UnityEngine.Random.Range(smallestZ, largestZ);
            Vector 3 Spawn Position = new Vector 3 ( x , 0 , z ) ;
            if  ( ! SpawnList . Contains ( Spawn Position ) )
            {
                SpawnList . Add( Spawn Position ) ;
            }
        }
        return SpawnList ;
    }*/
    }
```

RespawnController

A.2 Test_navigator

```
using   System.Collections;
using   System.Collections.Generic;
using   UnityEngine;
using   UnityEngine.AI;
using   System.IO;
using   System.Text;

```

```
public class testNavigator : MonoBehaviour
{
    // Start is called before the first frame update
    private NavMeshAgent nav;
    private Animator animator;
    float smallestX;
    float largestX;
    float smallestZ;
    float largestZ;



    void Start()
    {
        nav = GetComponent<NavMeshAgent>();
        animator = GetComponent<Animator>();
        InvokeRepeating("randomPosition", 2, 2 + Random.Range(0f, 2f));
        respawnController SpawnController = GameObject.Find("
            SpawnController").GetComponent<respawnController>();
        smallestX = SpawnController.smallestX;
        largestX = SpawnController.largestX;
        smallestZ = SpawnController.smallestZ;
        largestZ = SpawnController.largestZ;

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

        int velocity = Animator.StringToHash("Velocity");
        if(transform.position.x==nav.destination.x&& transform.position
            .z== nav.destination.z)
        {
            // Debug.Log("Stooooop!");
            nav.isStopped = true;
            // animator.SetBool("isWalking", false);
        }
        animator.SetFloat(velocity, nav.velocity.magnitude);
        if (!nav.isStopped)
        {
            /* Unable to rotate to the correct angle
```

```
                 * probably because of misunderstanding of quaternion
                     functions
                 *
                int rotation = Animator.StringToHash("Rotation");
                float rotateAngle = Quaternion.LookRotation(nav.destination
                    - transform.position).eulerAngles.y - transform.rotation
                   .eulerAngles.y;
                  if(rotateAngle > 180)
                {
                     rotateAngle = 360 - rotateAngle;
                }
                elseif(rotateAngle < -180)
                {
                     rotateAngle = -360 - rotateAngle;
                }
                float r = rotateAngle * (1.0f/180.0f);
                */



               // Debug.Log(r);
               // animator.SetFloat(rotation,r);
                transform.rotation = Quaternion.RotateTowards(transform.
                    rotation, Quaternion.LookRotation(nav.destination -
                    transform.position), 2);
                // animator.SetFloat(rotation,0);

           }
           if (Input.GetKeyDown(KeyCode.A))
           {
                Debug.Log("(" + transform.position.x +","+ transform.
                    position.y + "," + transform.position.z + ")");
                groundTruth();
           }

      }

      void randomPosition()
      {

           float y = transform.position.y;
           float x = Random.Range(smallestX, largestX);
           float z = Random.Range(smallestZ, largestZ);
```

```
        /*
        int rotation = Animator.StringToHash("Rotation");

        Vector3 forwardDir = nav.destination - transform.position;
        Quaternion lookAtRot = Quaternion.LookRotation(forwardDir);
        Vector3 resultEuler = lookAtRot.eulerAngles;
        float rotateAngle = resultEuler.y - transform.rotation.
            eulerAngles.y;
        if(rotateAngle > 180)
        {
            rotateAngle =360 - rotateAngle;
        }
        elseif(rotateAngle < -180)
        {
            rotateAngle = -360 - rotateAngle;
        }

        float r = rotateAngle * (1.0f/180.0f);
        */
        //while(transform.rotation != Quaternion.LookRotation(nav.
            destination - transform.position))
        //{
        //    animator.SetFloat(rotation, r);
         //}
       //animator.SetFloat(rotation,0);
        nav.destination = new Vector3(x,y,z);
        nav.isStopped = false;
        //Debug.Log(rotateAngle);
        //animator.SetBool("isWalking", true);

        //transform.LookAt(nav.destination);
        //transform.rotation=Quaternion.RotateTowards(transform.rotation
            , Quaternion.LookRotation(nav.destination- transform.
            position),3);
        //Debug.Log("walking!!!");
    }

    public void groundTruth()
    {
        File.AppendAllText("F:\\ACADEMIC FILES\\SURF\\GroundTruth\\Test.
            txt", "(" + transform.position.x + "," + transform.position
            .y + "," + transform.position.z + ")" + "\r\n", Encoding.
            Default);
        }
}
```

Test_navigator

Appendix B

Matlab code

B.1 Tsai's calibration

```
%Verify transition from world coordinates to image coordinates
%------------%
%camera parameters
Ncx=576;
Nfx=576;
dx=0.023;
dy=0.023;
sx =1;
%left world coordinate
subplot(1,2,1);
CreateWorld;
%right image coordinate
subplot(1,2,2);
I=imread('View/cam1_1.jpg');imagesc(I);
hold on;
Cx=size(I,2)/2;
Cy=size(I,1)/2;
hold on;
%calculate Tsai's model parameter
L=load('Tsai Input/Pic Point 1.txt');
LW=load('Tsai Input/WorldPoint1.txt');
Xf=L(:,1);
Yf=L(:,2);
xw=LW(:,1);
yw=LW(:,2);
[M,N]=size(LW);
zw=zeros(M,1);
```

```

[R, T, f, k1] = Tsai(Xf, Yf, xw, yw, zw, Ncx, Nfx, dx, dy, Cx, Cy, sx);

csvwrite('TsaiResult/R_1.txt',R);
csvwrite('TsaiResult/T_1.txt',T);
csvwrite('TsaiResult/f_1.txt',f);
csvwrite('Tsai Result/k1_1.txt',k1);
%------------------------%
%verify correctness
n=2500;
figer = 1;
for i=1:n
subplot(1,2,1);
title('click on a point! here!');
%Click on a point in the world frame and mark it in red
%------------------%
%interval_X = 20.8;
%interval_Y = 20.8;
%Xw=intervalX*(mod(i-1,50));
%Yw=intervalY*floor((i-1)/50);
q=ginput(1);
hold on;
plot(q(1),q(2),'rp','markersize',10);
Xw=q(1);
Yw=q(2);
Zw = 170; %This is the height of the point in the clicked world
    coordinate system, -100 is 100 units on the ground
%-----------------------------------%
%The image coordinates corresponding to points in the world coordinate system
[xf,yf]=pic(Xw,Yw,Zw,f,dx,dy,Cx,Cy,R,T);
%The image coordinates of points on the earth plane corresponding to
    points in the world coordinate system
[xff,yff]=pic(Xw,Yw,0,f,dx,dy,Cx,Cy,R,T);
%Mark verification points in the image coordinate system and the lines
    of corresponding points and two points on the ground plane
subplot(1,2,2);
hold all;
plot(xf,yf,'bp','markersize',10);
plot(xff,yff,'yp','markersize',10);
line([xff,xf],[yff,yf]);
%Save the world coordinates of the verification point, the image
    coordinates of the corresponding point, and the image coordinates of
```

```
     the corresponding point on the ground plane
if(figer==1)
 fil1 = fopen('CalibrationResult/CalibrationWorld.txt','w'); %Verify the
    world coordinates of the point
fprintf(fil1,'%5.6f',Xw);
 fprintf(fil1,'');
fprintf(fil1,'%5.6f',Yw);
 fprintf(fil1,'\n');
fclose(fil1);

 fil2 = fopen('CalibrationResult/CalibrationPic.txt','w'); %The graph
    coordinates of the corresponding points
 fprintf(fil2,'%5.6f',xf);
fprintf(fil2,'');
 fprintf(fil2,'%5.6f',yf);
fprintf(fil2,'\n');
 fclose(fil2);

 fil3 = fopen('CalibrationResult/CalibrationPicGround.txt','w'); %The
    image coordinates of the corresponding ground plane points
fprintf(fil3,'%5.6f',xff);
 fprintf(fil3,'');
fprintf(fil3,'%5.6f',yff);
 fprintf(fil3,'\n');
fclose(fil3);
 figer=figer+1;
else

fil1 = fopen('CalibrationResult/CalibrationWorld.txt','a');
 fprintf(fil1,'%5.6f',Xw);
fprintf(fil1,'');
 fprintf(fil1,'%5.6f',Yw);
fprintf(fil1,'\n');
 fclose(fil1);

 fil2 = fopen('CalibrationResult/CalibrationPic.txt','a');
 fprintf(fil2,'%5.6f',xf);
 fprintf(fil2,'');
 fprintf(fil2,'%5.6f',yf);
 fprintf(fil2,'\n');
 fclose(fil2);

fil3 = fopen('CalibrationResult/CalibrationPicGround.txt','a');
```

```
fprintf(fil3 ,'%5.6f',xff);
fprintf(fil3 ,'');
fprintf(fil3 ,'%5.6f',yff);
fprintf(fil3 ,'\n');
fclose(fil3);
end
end
```

Tsai's calibration

Appendix C

Python code

C.1 JOL algorithm

```
import os
from PIL import Image
import shutil
import matplotlib.patches as patches
import matplotlib.pyplot as plt
import numpy as np
import cv2 as cv

ground1_file_path = './savedpoints/Ground1.txt'
ground2_file_path = './savedpoints/Ground2.txt'
ground3_file_path = './savedpoints/Ground3.txt'
ground4_file_path = './savedpoints/Ground4.txt'

height1_file_path = './savedpoints/Height1.txt'
height2_file_path = './savedpoints/Height2.txt'
height3_file_path = './savedpoints/Height3.txt'
height4_file_path = './savedpoints/Height4.txt'

image1_file_path = './img_out2/cam1.png'
image2_file_path = './img_out2/cam2.png'
image3_file_path = './img_out2/cam3.png'
image4_file_path = './img_out2/cam4.png'

image1_rec_file_path = './images_rec/cam1.jpg'
image2_rec_file_path = './images_rec/cam2.jpg'
image3_rec_file_path = './images_rec/cam3.jpg'
image4_rec_file_path = './images_rec/cam4.jpg'
```

```
image_1_save_path = './img_out3/cam1.png'
image_2_save_path = './img_out3/cam2.png'
image_3_save_path = './img_out3/cam3.png'
image_4_save_path = './img_out3/cam4.png'

ground1 = []
ground2 = []
ground3 = []
ground4 = []

height_1 = []
height_2 = []
height_3 = []
height_4 = []

with open(ground1_file_path, 'r') as file_to_read:
    while True:
        lines = file_to_read.readline()
        if not lines:
            break
        x_tmp, y_tmp = [float(i) for i in lines.split()] #If the
            separator is a space, no arguments are passed in parentheses
            . If it is a comma, the ',' character is passed.
        ground1.append([int(x_tmp), int(y_tmp)]) # Adds newly read data
        pass
    ground1 = np.array(ground1) # Adds newly read data
    ground1 = ground1.reshape((50,50,2))
    pass

with open(ground2_file_path, 'r') as file_to_read:
    while True:
        lines = file_to_read.readline()
        if not lines:
            break
        x_tmp, y_tmp = [float(i) for i in lines.split()] #If the
            separator is a space, no arguments are passed in parentheses
            . If it is a comma, the ',' character is passed.
        ground2.append([int(x_tmp), int(y_tmp)]) # Adds newly read data
        pass
    ground2 = np.array(ground2) #
                                    listarray
    ground2 = ground2.reshape((50,50,2))
```

```
        pass

with open(ground3_filepath,'r') as file_to_read:
        while True:
                lines = file_to_read.readline() #
                if not lines:
                        break
                x_tmp, y_tmp = [float(i) for i in lines.split()] #If the
                        separator is a space, no arguments are passed in parentheses
                        . If it is a comma, the ',' character is passed.
                ground3.append([int(x_tmp), int(y_tmp)]) # Adds newly read data
                pass
            ground3 = np.array(ground3) #
                                        listarray
        ground3 = ground3.reshape((50,50,2))
        pass

with open(ground4_filepath,'r') as file_to_read:
        while True:
                        lines = file_to_read.readline() #
                        if not lines:
                                break
                        x_tmp, y_tmp = [float(i) for i in lines.split()] #

                                ,

                        ground4.append([int(x_tmp), int(y_tmp)])      #


                        pass
            ground4 = np.array(ground4) #
                                        listarray
            ground4 = ground4.reshape((50,50,2))
            pass

with open(height1_filepath,'r') as file_to_read:
            while True:
                        lines = file_to_read.readline() #
                        if not lines:
                                break
                                pass
                        x_tmp, y_tmp = [float(i) for i in lines.split()] #

                                ,
            height1.append([int(x_tmp), int(y_tmp)])    #

            pass
        height1 = np.array(height1) #
                        listarray
        height1 = height1.reshape((50,50,2))
        pass

with open(height2filepath,'r') as file_to_read:
        while True:
            lines = file_to_read.readline() #
            if not lines:
                break
                pass
            x_tmp, y_tmp = [float(i) for i in lines.split()] #
                ,
            height2.append([int(x_tmp), int(y_tmp)])    #

            pass
        height2 = np.array(height2) #
                        listarray
        height2 = height2.reshape((50,50,2))
        pass

with open(height3filepath,'r') as file_to_read:
        while True:
            lines = file_to_read.readline() #
            if not lines:
                break
                pass
            x_tmp, y_tmp = [float(i) for i in lines.split()] #
                ,
            height3.append([int(x_tmp), int(y_tmp)])    #

            pass
        height3 = np.array(height3) #
                        listarray
        height3 = height3.reshape((50,50,2))
        pass

with open(height4filepath,'r') as file_to_read:
        while True:
            lines = file_to_read.readline() #
            if not lines:
                break
                pass
            x_tmp, y_tmp = [float(i) for i in lines.split()] #
                ,
            height4.append([int(x_tmp), int(y_tmp)])    #

            pass
        height4 = np.array(height4) #
                            listarray
        height4 = height4.reshape((50,50,2))
        pass

img1 = cv.imread(image1_filepath,0)
img1 = np.copy(img1)
img_rec1 = cv.imread(image1_rec_filepath)
img_rec1 = np.copy(img_rec1)

img2 = cv.imread(image2_filepath,0)
img2 = np.copy(img2)
img_rec2 = cv.imread(image2_rec_filepath)
#img_rec2 = np.copy(img_rec2)

img3 = cv.imread(image3_filepath,0)
img3 = np.copy(img3)
img_rec3 = cv.imread(image3_rec_filepath)
#img_rec3 = np.copy(img_rec3)

img4 = cv.imread(image4_filepath,0)
img4 = np.copy(img4)
img_rec4 = cv.imread(image4_rec_filepath)
#img_rec4 = np.copy(img_rec4)
posi = []

'''
#calculate the rectangular in width
# for i in range(ground1.shape[0]):
#     for j in range(ground1.shape[1]-3):
#         left_down1 = ground1[i,j]
#         left_down2 = ground2[i,j]
#               left_down3 = ground3[i,j]
#               left_down4 = ground4[i,j]

#               right_up1 = height1[i, j+3]
#               right_up2 = height2[i,j+3]
#               right_up3 = height3[i,j+3]
#               right_up4 = height4[i,j+3]

#               cnt1 = 0;
#               cnt2= 0;
#               cnt3 =0;
#               cnt4= 0;

#               size1 = abs((right_up1[0] - left_down1[0]) * (right_up1[1]
    - left_down1[1]))
#               size2 = abs((right_up2[0] - left_down2[0]) * (right_up2[1]
    - left_down2[1]))
#               size3 = abs((right_up3[0] - left_down3[0]) * (right_up3[1]
    - left_down3[1]))
#               size4 = abs((right_up4[0] - left_down4[0]) * (right_up4[1]
    - left_down4[1]))


#               for x in range(min(left_down1[0], right_up1[0]),
   max(left_down1[0], right_up1[0])):
#                       for y in range(min(left_down1[1], right_up1[1]), max
   (left_down1[1], right_up1[1])):
#                               if img1[y,x] > 0:
#                                       cnt1+=1
#               portion1 = cnt1/size1

#               for x in range(min(left_down2[0], right_up2[0]),
   max(left_down2[0], right_up2[0])):
#                       for y in range(min(left_down2[1], right_up2[1]), max
   (left_down2[1], right_up2[1])):
#                               if img2[y,x] > 2:
#                                       cnt2+=1
#               if size2 != 0:
#                       portion2 = cnt2/size2


#               for x in range(min(left_down3[0], right_up3[0]),
   max(left_down3[0], right_up3[0])):
```

```
#                       for y in range(min(left_down3[1], right_up3[1]), max
    (left_down3[1], right_up3[1])):
#                               if y<1080:
#                                       if img3[y,x] > 2:
#                                               cnt3+=1
#               if size3 != 0:
#                       portion3 = cnt3/size3

#               for x in range(min(left_down4[0], right_up4[0]),
    max(left_down4[0], right_up4[0])):
#                       for y in range(min(left_down4[1], right_up4[1]), max
    (left_down4[1], right_up4[1])):
#                               if img4[y,x] > 2:
#                                       cnt4+=1
#               if size4 != 0:
#                       portion4 = cnt4/size4
#               if portion1 > 0.6:
#                       cv.rectangle(img_rec1, tuple(left_down1), tuple
    (right_up1),(128,128,128),1)

                # if pow(portion1*portion2*portion3*portion4, 1/4) > 0.3:
                #       posi.append([i,j])
                #       cv.rectangle(img_rec1, tuple(left_down1), tuple
    (right_up1),(128,128,128),1)
                #       cv.rectangle(img_rec2, tuple(left_down2), tuple
    (right_up2),(128,128,128),1)
                #       cv.rectangle(img_rec3, tuple(left_down3), tuple
    (right_up3),(128,128,128),1)
                #       cv.rectangle(img_rec4, tuple(left_down4), tuple
    (right_up4),(128,128,128),1)
                '''
cor_path = './cor/cor.txt'

pos1 = []
pos2 = []
pos3 = []
pos4 = []

def checkposi(posi1, posi2, posi3, posi4):
        posi=0
        if(posi1 < 0.1):
                posi=pow(posi2*posi3*posi4,1/3)
        if(posi2 < 0.1):
            posi = pow(posi1 * posi3 * posi4, 1/3)
        if(posi3 < 0.1):
            posi = pow(posi2 * posi1 * posi4, 1/3)
        if(posi4 < 0.1):
            posi = pow(posi2 * posi3 * posi1, 1/3)
        return posi

for i in range(50):
        for j in range(50):
            point_g1 = ground1[i,j]
            point_h1 = height1[i,j]
            point_height1 = abs(point_g1[1] - point_h1[1])
            point_width1 = 0.35 * point_height1
            point_g1_l = int(ground1[i,j][0] - point_width1/2)
            point_g1_r = int(ground1[i,j][0] + point_width1/2)

            point_g2 = ground2[i,j]
            point_h2 = height2[i,j]
            point_height2 = abs(point_g2[1] - point_h2[1])
            point_width2 = 0.35 * point_height2
            point_g2_l = int(ground2[i,j][0] - point_width2/2)
            point_g2_r = int(ground2[i,j][0] + point_width2/2)

            point_g3 = ground3[i,j]
            point_h3 = height3[i,j]
            point_height3 = abs(point_g3[1] - point_h3[1])
            point_width3 = 0.35 * point_height3
            point_g3_l = int(ground3[i,j][0] - point_width3/2)
            point_g3_r = int(ground3[i,j][0] + point_width3/2)

            point_g4 = ground4[i,j]
            point_h4 = height4[i,j]
            point_height4 = abs(point_g4[1] - point_h4[1])
            point_width4 = 0.35 * point_height4
            point_g4_l = int(ground4[i,j][0] - point_width4/2)
            point_g4_r = int(ground4[i,j][0] + point_width4/2)


            cnt1 = 0
            cnt2 = 0
            cnt3 = 0
            cnt4 = 0
            start_p1 = [point_g1_l , min(point_g1[1], point_h1[1])]
            end_p1 = [point_g1_r, max(point_g1[1], point_h1[1])]

            start_p2 = [point_g2_l , min(point_g2[1], point_h2[1])]
            end_p2 = [point_g2_r , max(point_g2[1], point_h2[1])]

            start_p3 = [point_g3_l , min(point_g3[1], point_h3[1])]
            end_p3 = [point_g3_r , max(point_g3[1], point_h3[1])]

            start_p4 = [point_g4_l , min(point_g4[1], point_h4[1])]
            end_p4 = [point_g4_r , max(point_g4[1], point_h4[1])]

            size1 = abs((point_g1_l-point_g1_r)*(point_g1[1] - point_h1
      [1]))
            size2 = abs((point_g2_l-point_g2_r)*(point_g2[1] - point_h2
      [1]))
            size3 = abs((point_g3_l-point_g3_r)*(point_g3[1] - point_h3
      [1]))
            size4 = abs((point_g4_l-point_g4_r)*(point_g4[1] - point_h4
      [1]))



            for x in range(int(start_p1[0]),int(end_p1[0])):
                for y in range(int(start_p1[1]), int(end_p1[1])):
                    if y <1080 and x <1920:
                        if img1[y,x] > 0:
                            cnt1+=1
            portion1 = cnt1/size1
            for x in range(int(start_p2[0]),int(end_p2[0])):
                for y in range(int(start_p2[1]), int(end_p2[1])):
                    if y <1080 and x <1920:
                        if img2[y,x]> 0:
                            cnt2+=1
            portion2 = cnt2/size2
            for x in range(int(start_p3[0]),int(end_p3[0])):
                for y in range(int(start_p3[1]), int(end_p3[1])):
                    if y <1080 and x <1920:
                        if img3[y,x]> 0:
                            cnt3+=1
            portion3 = cnt3/size3

            for x in range(int(start_p4[0]),int(end_p4[0])):
                for y in range (int(start_p4[1]),int(end_p4[1])):
                    if y <1080 and x <1920:
                        if img4[y,x] > 0:
                            cnt4+=1
            portion4 = cnt4/size4
            #print(portion
            4)'''
            if portion4 > 0.4:
                cv.rectangle(img_rec4, tuple(start_p4), tuple(end_p4),
 (128,128,128),1)

            if(portion1< 0.1):
                portion = pow(portion2*portion3*portion4, 1/3)
                if portion > 0.4:
                    cv.rectangle(img_rec2,tuple(start_p2),tuple
(end_p2),(128,128,128),1)
                    cv.rectangle(img_rec3,tuple(start_p3),tuple
(end_p3),(128,128,128),1)
                    cv.rectangle(img_rec4,tuple(start_p4),tuple
(end_p4),(128,128,128),1)
                    with open(cor_path,"a") as f:
                        f.writelines(str(i)+" " + str(j) + " " +
 str(portion) +"\n")

            elif(portion2 < 0.1):
                portion=pow(portion1*portion3*portion4,1/3)
                if portion > 0.3:
                    cv.rectangle(img_rec1,tuple(start_p1),tuple
(end_p1),(128,128,128),1)
                    cv.rectangle(img_rec3,tuple(start_p3),tuple
(end_p3),(128,128,128),1)
                    cv.rectangle(img_rec4,tuple(start_p4),tuple
(end_p4),(128,128,128),1)
                    with open(cor_path,"a") as f:
                        f.writelines(str(i)+""+str(j)+""+
 str(portion)+"\n")

            elif(portion3< 0.1):
                portion = pow(portion1*portion2*portion4, 1/3)
                if portion > 0.4:
                    cv.rectangle(img_rec2,tuple(start_p2),tuple
(end_p2),(128,128,128),1)
```

```
                        cv.rectangle(img_rec1, tuple(start_p1), tuple
    (end_p1),(128,128,128), 1)
                        cv.rectangle(img_rec4, tuple(start_p4), tuple
    (end_p4),(128,128,128), 1)
                        with open(cor_path, "a") as f:
                              f.writelines(str(i)+" " + str(j) + " " +
     str(portion) +"\n")

            elif(portion4 < 0.1):
                 portion=pow(portion1*portion3*portion2,1/3)
                  if portion > 0.2:
                        cv.rectangle(img_rec1, tuple(start_p1), tuple
    (end_p1),(128,128,128), 1)
                        cv.rectangle(img_rec2, tuple(start_p2), tuple
    (end_p2),(128,128,128), 1)
                        cv.rectangle(img_rec3, tuple(start_p3), tuple
    (end_p3),(128,128,128), 1)
                        with open(cor_path, "a") as f:
                              f.writelines(str(i)+""+str(j)+""+
     str(portion)+"\n")

            else:
                  portion = pow(portion1*portion3*portion2*portion4,
    1/4)
                  ifportion > 0.4:
                        cv.rectangle(img_rec1, tuple(start_p1), tuple
    (end_p1),(128,128,128), 1)
                        cv.rectangle(img_rec2, tuple(start_p2), tuple
    (end_p2),(128,128,128), 1)
                        cv.rectangle(img_rec3, tuple(start_p3), tuple
    (end_p3),(128,128,128), 1)
                        cv.rectangle(img_rec4, tuple(start_p4), tuple
    (end_p4),(128,128,128), 1)
                        with open(cor_path, "a") as f:
                              f.writelines(str(i)+""+str(j)+""+
     str(portion)+"\n")
'''
            portion = pow(portion1*portion3*portion2*portion4, 1/4)
            if portion > 0.5:
                  cv.rectangle(img_rec1, tuple(start_p1), tuple(end_p1),
     (128,128,128),1)
                  cv.rectangle(img_rec2, tuple(start_p2), tuple(end_p2),
     (128,128,128),1)
```

```
                    cv.rectangle(img_rec3, tuple(start_p3), tuple(end_p3),
 (128, 128, 128), 1)
                    cv.rectangle(img_rec4, tuple(start_p4), tuple(end_p4),
 (128,128,128), 1)
                    with open(cor_path, "a") as f:
                        f.writelines(str(i) + " " + str(j) + " " + str(
portion) + "\n")

            # if portion > 0.35:
            #       with open(cor_path, "a") as f:
            #           f.writelines(str(i) + " " + str(j) + " " + str
(portion) + "\n")

            #       posi.append([i, j, portion])

            #       pos1.append([start_p1, end_p1, portion])
            #       pos2.append([start_p2, end_p2, portion])
            #       pos3.append([start_p3, end_p3, portion])
            #       pos4.append([start_p4, end_p4, portion])

            #      cv.rectangle(img_rec1, tuple(start_p1), tuple(end_p1
), (128,128, 128),1)
            #      cv.rectangle(img_rec2, tuple(start_p2), tuple(end_p2
), (128,128, 128),1)
            #      cv.rectangle(img_rec3, tuple(start_p3), tuple(end_p3
), (128,128, 128),1)
            #      cv.rectangle(img_rec4, tuple(start_p4), tuple(end_p4
), (128,128, 128),1)

cv.imwrite(image1_save_path, img_rec1)
cv.imwrite(image_2_save_path, img_rec2)
cv.imwrite(image_3_save_path, img_rec3)
cv.imwrite(image4_save_path, img_rec4)
```

JOL algorithm

C.2 RSS algorithm

```
import os
from PIL import Image
import shutil
from cv2 import sort
```

```
import matplotlib.patches as patches
import matplotlib.pyplot as plt
import numpy as np
import cv2 as cv
import random

def randomcolor():
    colorArr = ['1','2','3','4','5','6','7','8','
9','A','B','C','D','E','F']
    color=""
    for i in range(6):
        color += colorArr[random.randint(0,14)]
    return "#"+color

def random_colour():
    return random.randint(0,255),random.randint(0,255),random.randint
(0,255)

image1_file_path = './images_rec/cam1.jpg'
image2_file_path = './images_rec/cam2.jpg'
image3_file_path = './images_rec/cam3.jpg'
image4_file_path = './images_rec/cam4.jpg'

image_1_save_path = './img_out4/cam1.png'
image_2_save_path = './img_out4/cam2.png'
image_3_save_path = './img_out4/cam3.png'
image_4_save_path = './img_out4/cam4.png'

posipath='./cor/cor2.txt'

posi=[]

ground1 = []
ground2 = []
ground3 = []
ground4 = []

height1 = []
height2 = []
height3 = []
height4 = []

ground_1_file_path='./savedpoints/Ground1.txt'
```

```
ground2_file_path = './savedpoints/Ground2.txt'
ground3_file_path = './savedpoints/Ground3.txt'
ground4_file_path = './savedpoints/Ground4.txt'

height1_file_path = './savedpoints/Height1.txt'
height2_file_path = './savedpoints/Height2.txt'
height3_file_path = './savedpoints/Height3.txt'
height4_file_path = './savedpoints/Height4.txt'

radius=4

with open(ground1_file_path,'r') as file_to_read:
    while True:
        lines=file_to_read.readline()#
        if not lines:
            break
        x_tmp,y_tmp=[float(i) for i in lines.split()]#
            ,
        ground1.append([int(x_tmp), int(y_tmp)])    #

        pass
    ground1 = np.array(ground1)#
                        listarray
    ground1 =ground1.reshape((50,50,2))
    pass

with open(ground2_file_path,'r') as file_to_read:
    while True:
        lines=file_to_read.readline()#
        if not lines:
            break
        x_tmp,y_tmp=[float(i) for i in lines.split()]#
            ,
        ground2.append([int(x_tmp), int(y_tmp)])    #

        pass
    ground2 = np.array(ground2)#
                        listarray
    ground2 = ground2.reshape((50,50,2))
    pass
```

```
with open(ground3_filepath,'r') as file_to_read:
        while True:
                lines=file_to_read.readline() #
                if not lines:
                        break
                x_tmp, y_tmp = [float(i) for i in lines.split()] #

                          ,
                ground3.append([int(x_tmp), int(y_tmp)])    #

                pass
        ground3 = np.array(ground3) #
                           listarray
        ground3 = ground3.reshape((50,50,2))
        pass

with open(ground4_filepath,'r') as file_to_read:
        while True:
                lines=file_to_read.readline() #
                if not lines:
                        break
                x_tmp, y_tmp = [float(i) for i in lines.split()] #

                          ,
                ground4.append([int(x_tmp), int(y_tmp)])    #

                pass
        ground4 = np.array(ground4) #
                           listarray
        ground4 = ground4.reshape((50,50,2))
        pass

with open(height1_filepath,'r') as file_to_read:
        while True:
                lines=file_to_read.readline() #
                if not lines:
                        break
                        pass
                x_tmp, y_tmp = [float(i) for i in lines.split()] #

                          ,
                height1.append([int(x_tmp), int(y_tmp)])    #
```

```
            pass
        height1 = np.array(height1) #
                          listarray
        height1 = height1.reshape((50,50,2))
        pass

with open(height2filepath,'r') as file_to_read:
        while True:
            lines = file_to_read.readline() #
            if not lines:
                break
                pass
            x_tmp, y_tmp = [float(i) for i in lines.split()] #
                            ,
            height2.append([int(x_tmp), int(y_tmp)])      #

            pass
        height2 = np.array(height2) #
                          listarray
        height2 = height2.reshape((50,50,2))
        pass

 with open(height3filepath,'r') as file_to_read:
        while True:
            lines = file_to_read.readline() #
            if not lines:
                break
                pass
            x_tmp, y_tmp = [float(i) for i in lines.split()] #
                            ,
            height3.append([int(x_tmp), int(y_tmp)])      #

            pass
        height3 = np.array(height3) #
                          listarray
        height3 = height3.reshape((50,50,2))
        pass

with open(height4filepath,'r') as file_to_read:
        while True:
            lines = file_to_read.readline() #
            if not lines:
                break
                pass
            x_tmp, y_tmp = [float(i) for i in lines.split()] #
                                                              ,
            height4.append([int(x_tmp), int(y_tmp)])    #

            pass
        height4 = np.array(height4) #
                                listarray
        height4 = height4.reshape((50,50,2))
        pass

with open(posi_path,'r') as file_to_read:
        while True:
            lines = file_to_read.readline() #
            if not lines:
                break
                pass
            x_tmp, y_tmp, posi_temp = [float(i) for i in lines.split()]
    #
                                                              ,
            posi.append([int(x_tmp), int(y_tmp), posi_temp])    #

            pass
        #posi = np.array(posi) #
                                listarray
        pass

img1 = cv.imread(image1_file_path)
img1 = np.copy(img1)

img2 = cv.imread(image2_file_path)
img2 = np.copy(img2)
img3 = cv.imread(image3_file_path)
img3 = np.copy(img3)
img4 = cv.imread(image4_file_path)
img4 = np.copy(img4)

#posi = sorted(posi, key = lambda posi:posi[2], reverse=True)

```

```
while len(posi)!=0:
    posi_tmp = sorted(posi, key = lambda posi:posi[2], reverse=True)
    #the tmp larget posibility
    x_tmp, y_tmp, pos_tmp = posi_tmp[0][0], posi_tmp[0][1], posi_tmp[0][2]
    i=0
    while i<len(posi):
        x,y,pos=posi[i][0],posi[i][1],posi[i][2]
        if ((x>=x_tmp-radius and x<= x_tmp+radius) and (y>=y_tmp-radius and y<=y_tmp+radius)):
            posi.remove(posi[i])
            i-=1
        i+=1

    point_g1=ground1[posi_tmp[0][0],posi_tmp[0][1]]
    point_h1=height1[posi_tmp[0][0],posi_tmp[0][1]]
    point_height1 = abs(point_g1[1] - point_h1[1])
    point_width1 = 0.35 * point_height1
    point_g1_l=int(ground1[posi_tmp[0][0],posi_tmp[0][1]][0]-point_width1/2)
    point_g1_r=int(ground1[posi_tmp[0][0],posi_tmp[0][1]][0]+point_width1/2)

    point_g2= ground2[posi_tmp[0][0], posi_tmp[0][1]]
    point_h2=height2[posi_tmp[0][0], posi_tmp[0][1]]
    point_height2 = abs(point_g2[1] - point_h2[1])
    point_width2=0.35*point_height2
    point_g2_l=int(ground2[posi_tmp[0][0],posi_tmp[0][1]][0]-point_width2/2)
    point_g2_r=int(ground2[posi_tmp[0][0],posi_tmp[0][1]][0]+point_width2/2)

    point_g3=ground3[posi_tmp[0][0],posi_tmp[0][1]]
    point_h3= height3[posi_tmp[0][0], posi_tmp[0][1]]
    point_height3 = abs(point_g3[1] - point_h3[1])
    point_width3 = 0.35 * point_height3
    point_g3_l=int(ground3[posi_tmp[0][0],posi_tmp[0][1]][0]-point_width3/2)
    point_g3_r=int(ground3[posi_tmp[0][0],posi_tmp[0][1]][0]+point_width3/2)

    point_g4= ground4[posi_tmp[0][0], posi_tmp[0][1]]
    point_h4=height4[posi_tmp[0][0], posi_tmp[0][1]]
```

```
        point_height4 = abs(point_g4[1] - point_h4[1])
        point_width4 = 0.35 * point_height4
        point_g4_l = int(ground4[posi_tmp[0][0], posi_tmp[0][1]][0] -
    point_width4/2)
        point_g4_r = int(ground4[posi_tmp[0][0], posi_tmp[0][1]][0] +
    point_width4/2)

        start_p1 = [point_g1_l, min(point_g1[1], point_h1[1])]
        end_p1 = [point_g1_r, max(point_g1[1], point_h1[1])]

        start_p2 = [point_g2_l, min(point_g2[1], point_h2[1])]
        end_p2 = [point_g2_r, max(point_g2[1], point_h2[1])]

        start_p3 = [point_g3_l, min(point_g3[1], point_h3[1])]
        end_p3 = [point_g3_r, max(point_g3[1], point_h3[1])]

        start_p4 = [point_g4_l, min(point_g4[1], point_h4[1])]
        end_p4 = [point_g4_r, max(point_g4[1], point_h4[1])]

        color = random_colour()

        cv.rectangle(img1, tuple(start_p1), tuple(end_p1), color, 1)
        cv.rectangle(img2, tuple(start_p2), tuple(end_p2), color, 1)
        cv.rectangle(img3, tuple(start_p3), tuple(end_p3), color, 1)
        cv.rectangle(img4, tuple(start_p4), tuple(end_p4), color, 1)
        print(posi_tmp[0][0], posi_tmp[0][1])

cv.imwrite(image_1_save_path, img1) cv.
imwrite(image_2_save_path, img2) cv.
imwrite(image_3_save_path, img3) cv.
imwrite(image_4_save_path, img4)

#print(posi[0])
```

RSS algorithm