Report Project's Computer Vision

Master degree in Computer Engineering for Robotics and Smart Industry

Valentini Michel VR472456

September 2022

Contents

1 Project goal

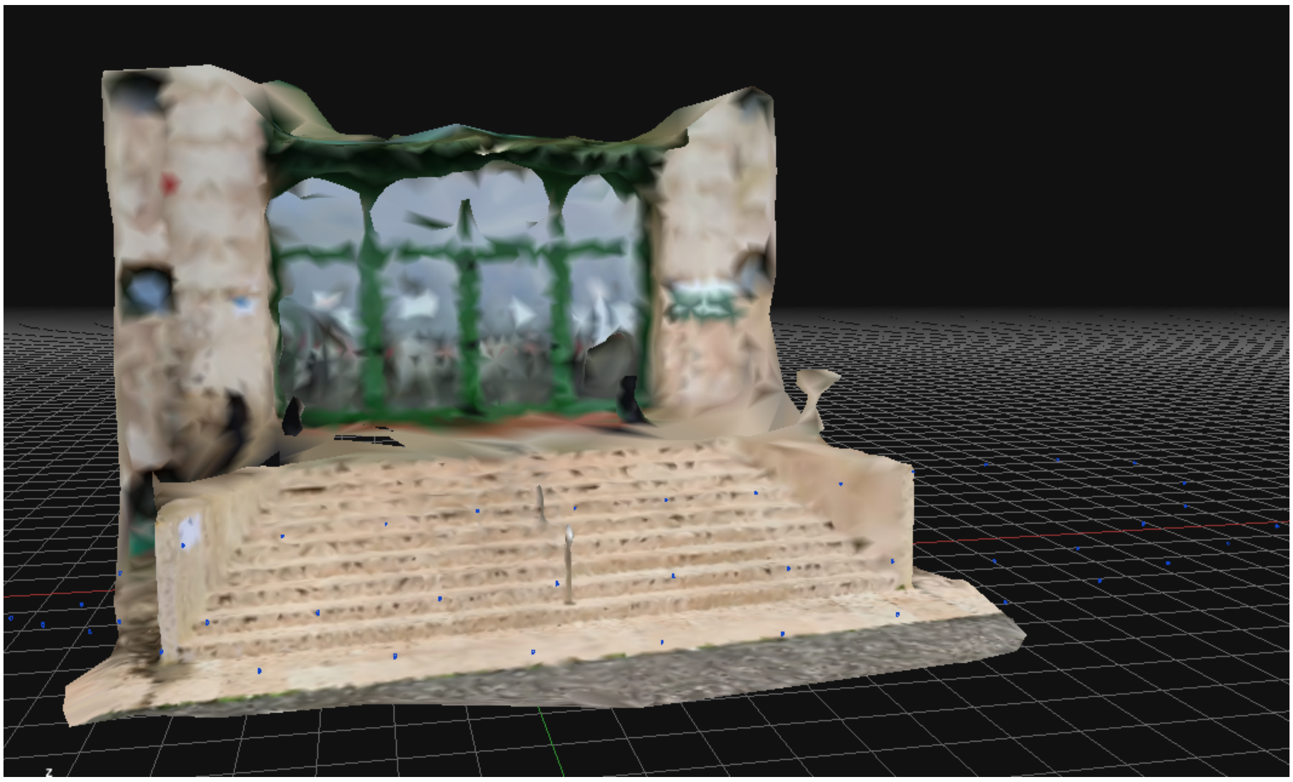
The computer vision project revolves around reconstructing a real-world scene within a virtual environment. Subsequently, we aim to incorporate a new object into this scene, one that appears realistic and seamlessly fits into the recreated environment as perceived by the observer. This endeavor serves as an illustrative example of Augmented Reality.

2 Reconstract the environment

The initial step involved selecting a scene to serve as our virtual environment and capturing a series of images using a standard camera, such as a mobile phone. It's important to note that the more images we capture, the more precise our reconstruction will be. Alternatively, we can obtain images from a video by extracting individual frames. Remember to acquire from more angle possible (Top, Bottom, Left, Right)

A valuable tip is to capture these images under conditions that minimize shadows, such as on a cloudy day, to ensure consistent lighting across the scene. Shadows can pose challenges during the 3D reconstruction process, so avoiding them as much as possible is advisable.
Now that we have build the dataset is possible to upload the images in a software to reconstract 3d models from images, I had used Zephyr

Here, it's sufficient to begin a new project, and step by step, you will be guided through the process of uploading images and constructing the 3D model.
Once the model is successfully constructed, you'll notice a point cloud that you can inspect to ensure the model has been loaded correctly. On the left panel, you can access various settings, including the option to view the mesh with textures and individual images used for the reconstruction. Additionally, the software can autonomously identify the camera's intrinsic parameters, eliminating the need for manual input.



After obtaining the reconstructed model, it's essential to export it in either .MTL or .OBJ file formats along with the textures. Additionally, make sure to export the camera parameters, which will be saved in XMP format.

3 Upload Model and Cameras

Now was used a software for modelling, animation, rendering and texturing of 3d images like the free software Blender

3.1 Model

To import the model is easy, will just open (file → import → model .OBJ). The 3d model with the mesh and texture will be upload, but an important tip is rotate the model in the 0,0,0 axes rotation. In way to have the model right configure with the camera parameters that we will import.

3.1.1 Cameras

To import all the cameras parameters is convenient create a python script in Blender. The code for this project is upload on this repository

However, it's important to note that the coordinate systems between Zephyr and Blender differ. Therefore, from the parameters exported by Zephyr, we can confidently map them to the correct ones used in Blender.

After creating a "camera" object in Blender's script and loading the parameters from the **.xmp** files, it's necessary to set the parameter values for the sensor dimensions (width, height) and the sensor shift on the x and y axes relative to the camera.

First and foremost, let's generate the intrinsic matrix from the data acquired through Zephyr. The intrinsic matrix **K**, should have the following form:

$$K = \begin{bmatrix} f_u & skew & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix}$$

Where:

- f_u, f_V rappresent the focal distance in pixel (vertical and horizontal).
- c_u and c_v are the coordinate in pixel about the optical center of the camera on the image plane
- *skew* represents the skew factor, which is close to zero for most cameras.

Then, the basic formula on how to calculate the focal length is as follows:

$$f = (Sensorsize * W)/FOV$$

On our case we apply the formula for width and height using the intrinsic's matrix like:

$$sensor_{width} = \frac{f \cdot xmp_{data}[w]}{K[0][0]}; sensor_{height} = \frac{f \cdot xmp_{data}[h]}{K[1][1]}$$

where, f is the focal lenght in mm choose in primis, $xmp_{data}[w], xmp_{data}[h]$ are the dimension in pixel and K[0][0] and K[1][1] are respectively the parameters of intrinsic matrix.

After that, we have to calculate the Cam Shift and here we need to translate the Perspective Projection Matrix of Standard Computer vision to OpenGl PPM:

This is the cv ppm, that is matric that use Zephyr for the cameras:

$$K[I|0] = \begin{bmatrix} -fk_u & 0 & u_0 & 0 \\ 0 & -fk_v & v0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

But here if we moltiply it for a point P :

$$P = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

we get :

$$p^| = \begin{bmatrix} x^| \\ y^| \\ z^| \end{bmatrix}$$

prospective division $\frac{1}{z^|}$ we optain the point on the image plane in homogeneous coordinate.

$$p = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

But we lost the value on z axes. So Blender used a OpenGL's Perspective Projection Matrix adding a row in the standard matrix to optain so the value on Z. This model have different advantage for the rendering of the image.

Before we proceed, it's important to specify some differences between the standard projection matrix in Computer Vision and the one used by Blender (OpenGL):
Standard Perspective Projection Matrix:

- In computer vision, the standard perspective projection matrix is used to project three-dimensional (3D) points onto a two-dimensional (2D) image plane. This is commonly used in applications such as 3D reconstruction from images or computer vision tasks.
- The standard perspective projection matrix does not preserve the depth value Z (the distance of a point from the camera) in the projection. This means that once projected onto a 2D plane, the point loses its depth information.

Perspective Projection Matrix in Blender or OpenGL:

- In Blender and OpenGL, the perspective projection matrix maps 3D points to a 3D space called Normalized Device Coordinates (NDC). It is a space in which all points are within the range of -1 to 1 along all axes.
- The main difference from the standard perspective projection matrix is that the Blender or OpenGL perspective projection matrix preserves the depth value Z during projection. This means that you can still determine the distance of a point from the camera after projection, which is crucial for many graphics applications, such as 3D rendering.

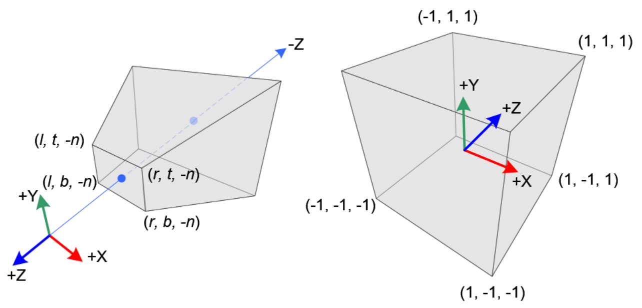
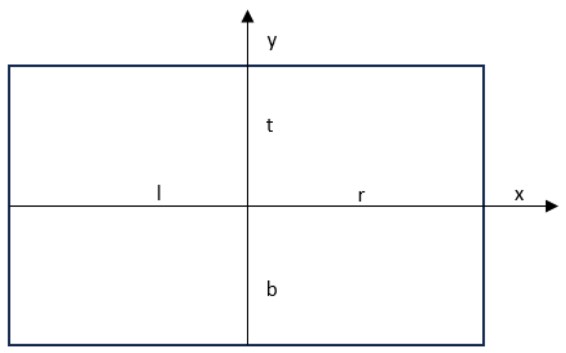


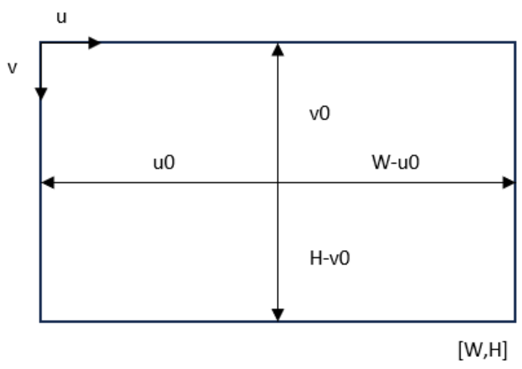
Figure 1: Camera Space , NDC

$$M = \begin{bmatrix} \frac{2n}{r-i} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

where we are interested about the shift of x and y, so the fraction $\frac{r+l}{r-l}$ for the movement on x and $\frac{t+b}{t-b}$ for y.

However, this refers to the OpenGL coordinate system, which is in millimeters. Our values, on the other hand, are in pixels and involve the Perspective Projection Matrix [3x4] commonly used in Computer Vision.





- In OpenGL (first image), the image plane shift is expressed through the positions of the sides of the image plane in the camera reference system.
- In Computer Vision (second image), the shift of the image plane is expressed through the image coordinates in pixels of the optical center projected onto the plane.

Therefore, we need to change the frame of reference, and this can be achieved by applying the following formulas:

$$r = \frac{(W - U_0)}{K_u}; l = -\frac{(U_0)}{K_u}$$
$$t = \frac{V_0}{K_v}; b = -\frac{(H - V_0)}{K_v}$$

This brings our coordinates into the correct format for OpenGL. Therefore, the final form of the shift will be:

$$\mathrm{shift\_x} = \frac{\mathrm{W} - 2\mathrm{U}_0}{2\mathrm{W}}$$
$$\mathrm{shift\_y} = \frac{-\mathrm{H} + 2\mathrm{V}_0}{2\mathrm{H}}$$

Important is divided by 2 because the shift calculated is referred to on both sides, which means that our value represents double distance from the center. So, we divide by 2

Now, we only need to adjust the translation and rotation of the camera, and for this choice, Euler angles have been used (it's also possible to do this with quaternions). To achieve this, we must apply a transformation to the matrices loaded from the XMP file.

We instantiate two rotation matrices along the y and z axes, the first one is used to transition from the computer vision world reference system to the Blender world reference system, while the second one is used to switch from the computer vision camera reference system to the OpenGL camera reference system.

$$V = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$
$$U = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

We calculate the new camera rotation by multiplying the rotation matrix V by the rotation matrix R, then transposing the result because initially we have the orientation from the camera to the world and trasponding the result we optain the orientation from the world to the camera and at the end converting it into Euler angles.

$$R^{|} = V(U \cdot R)^T$$

Do the same for the new camera position by multiplying the rotation matrix U by the translation matrix T, then transposing the result and multiplying it by the rotation matrix U again.

$$T^{|} = V(-R^{|}(U \cdot T)) = V(-(U \cdot R)^T(U \cdot T))$$

4 Animation

To create a realistic animation of a ball bouncing on the stairs in our environment, Blender's physics simulation capabilities were employed. In this process, gravity was applied to both objects—the passive object representing the stairs and the active object representing the ball.
Additionally, by adjusting attributes such as elasticity, it became possible to recreate a lifelike effect of the ball bouncing on the stairs.
In order to achieve a convincing augmented reality effect, we incorporate the original image (the one used to construct the 3D object) at the conclusion. This image is placed as the background in the camera view. Subsequently, the 3D object is removed, leaving only the animation superimposed on the original image. The outcome is a virtual animation seamlessly integrated into a real image, delivering the anticipated result akin to virtual reality.