3D Reconstruction

Yoonyoung Cho, Anne Ku

May 4 2016

1 Introduction

3D reconstruction is process of capturing the physical features of an object. For this project, we decided to obtain 2D images of an object by forward-projecting its 3D model, and inverse-projecting the output images to reverse-engineer the object's physical features.

1.1 Context

We use 2D images to reconstruct 3D models every day. For example, our eyes view an object with overlapping fields of view to get a 3D interpretation of the objects we see. This process is known as *Binocular Vision.* Each eye takes in a 2D image from the object. When the 2D perceptions from each eye are combine, the brain sees a 3D point.

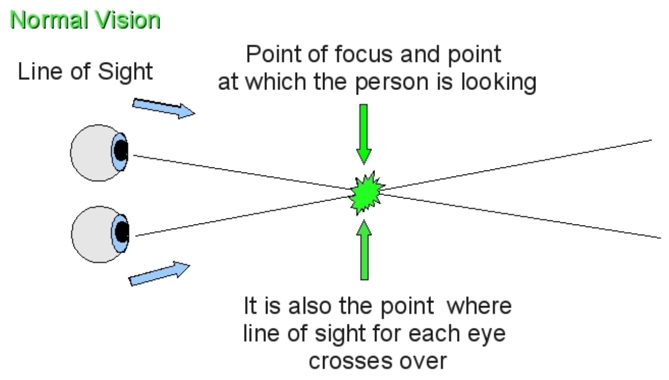

Normal Vision
Line of Sight
Point of focus and point at which the person is looking
It is also the point where line of sight for each eye crosses over

Figure 1: How our eyes use binocular vision

From a corporate point of view, 3D reconstruction from 2D images help companies make realistic visualizations. For example, Disney Research, in Zurich, develops 3D scenes based on high-resolution 2D images. Google Earth also uses 2D images to generate 3D terrain for its software. These examples show that 3D reconstruction from 2D images is a compelling topic.

1.2 Topics

General 3D reconstruction relies upon various principles covered in Linearity, including projection, vector properties, matrix-based transformations. In addition, we have explored in the realm of linearity in geometric context, which wasn't covered quite as exhaustively during the course.

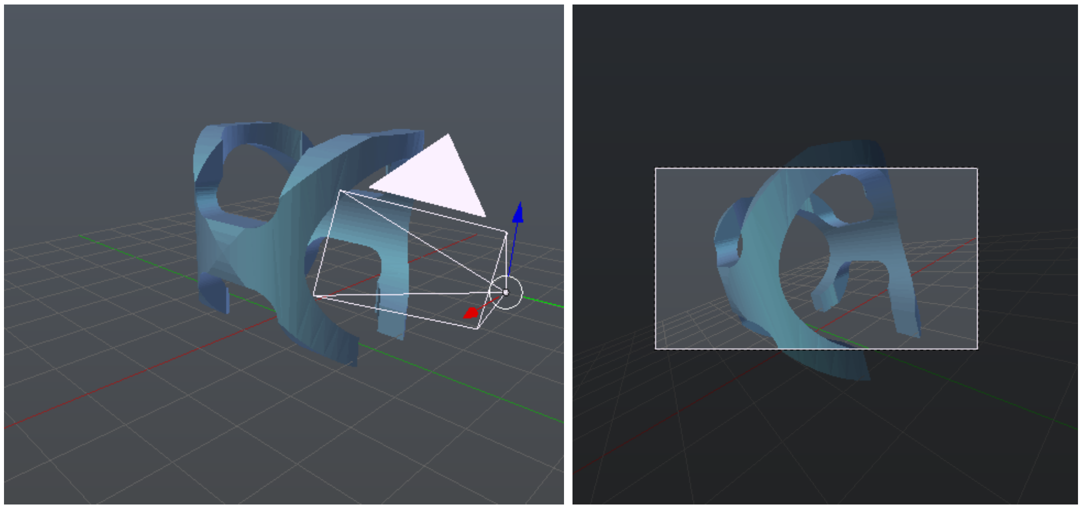
1.3 Procedure

It is difficult to measure the coordinates and the direction of the camera in the real world; in order to accurately evaluate where the camera is pointing at, we would have to create, or buy, a measuring device composed of three gimbals, each providing readings of pitch, yaw, and roll. Moreover, technical information such as the field of view and the focal length of the lens is necessary to fully characterize the system. Unfortunately, this is beyond the scope of this project.

Therefore, in order to simulate a real-world camera, we first created a 3D model of a cube and calculated the forward projection into a 2D image based on 3D graphics theory, which provides a reasonable approximation of a physical camera. By creating and placing a custom camera on the scene, we would essentially have all the information that is necessary to invert the projection in the later process.

2 Forward Projection

2.1 View Matrix



(a) Global Coordinates (b) Camera Coordinates

Figure 2: The View Matrix transforms the object's coordinates in the perspective of the camera.

The view matrix is the coordinate transformation from global coordinates to the local coordinates(of the camera). This matrix is composed of two operations: rotation and translation. In analogy, this matrix translates the camera to the origin and rotates it to point towards the positive Z axis. When $\vec{p}$ is the 3D point in the world coordinate system, and $\vec{p'}$ is the point in the camera's coordinate system, **V** represents the official View Matrix that is based off rotations, **R**, and translations, **T**.

$$\vec{p'} = V \cdot \vec{p}$$
$$V = R \cdot T$$
$$\vec{p'} = R \cdot T \cdot \vec{p}$$

Note that translation is performed first, i.e.

$$\vec{p'} = R \cdot (T \cdot \vec{p})$$

In practice, this matrix is constructed by finding the local coordinate system of the camera given two points: the position of the camera, and the position of the object it is pointing at. The subtraction of the former from the latter yields vector $\vec{z}$, the direction of the camera. The cross product of $\vec{z}$ with the global y axis yields $\vec{x}$, the horizontal axis of the camera.

$$
\begin{aligned}
\vec{z'} &= \vec{o} - \vec{c} \\
\vec{x'} &= \vec{z'} \times \vec{y} \\
\vec{y'} &= \vec{x'} \times \vec{z'} \\
R &= \begin{pmatrix} \begin{pmatrix} \vec{x'} & \vec{y'} & \vec{z'} \end{pmatrix} & 0 \\ 0 & 1 \end{pmatrix}^{\intercal}
\end{aligned}
$$

Here, $R^{\intercal}$ is equivalent to R^{-1}. Before the inversion, this matrix is essentially the coordinate transformation from the camera coordinates to the global coordinates. By inverting, we obtain the reverse operation[1].

The key assumption to creating the view matrix is that the camera does not have a roll: that the x axis of the camera is aligned with the horizontal plane. This assumption is necessary due to the problem known as the *gimbal lock*, in which two defining vectors of the coordinate become parallel, and the system becomes underdefined.

2.2 Projection Matrix

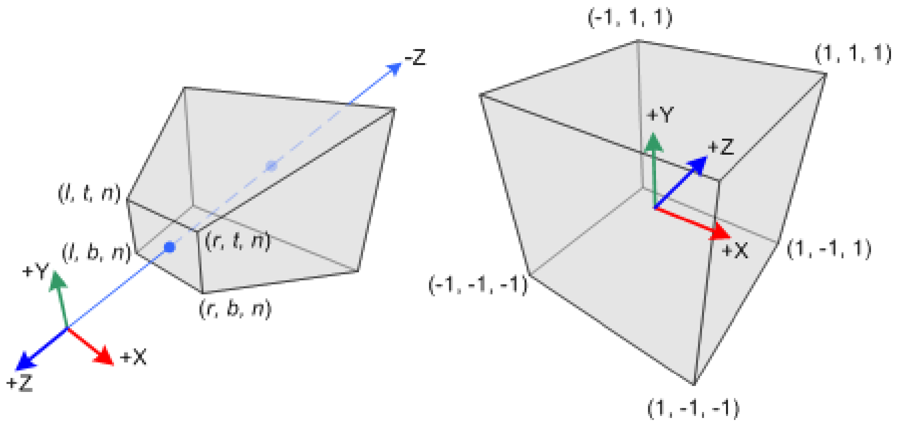


Figure 3: The canonical perspective projection operation, which transforms the frustum into a cube in the range of [-1,1].

$$
\begin{pmatrix}
\frac{\cot\left(\frac{\text{fov}}{2}\right)}{\text{ar}} & 0 & 0 & 0 \\
0 & \cot\left(\frac{\text{fov}}{2}\right) & 0 & 0 \\
0 & 0 & \frac{f+n}{f-n} & -\frac{2fn}{f-n} \\
0 & 0 & 1 & 0
\end{pmatrix}
$$

The perspective projection matrix, as opposed to orthographic projection matrix, assumes a radial line of sight, as would be the case in the real world. This procedure is equivalent to a shear, or transforming a frustum into a cube, in that each object along the same line of sight would correspond to the same coordinates in the projected image.

2.2.1 Analysis

We decided to analyze the eigenvalues and eigenvectors of our 4x4 projection matrix. We obtained interesting results that imply rotation. Each variable represents a property of the camera: **ar** is a camera's aspect ratio, **f** is the distance from the camera to the far plane, **n** is the distance from the camera to the near plane, and

[1]Note that this is a 4x4 matrix in homogeneous coordinates.

t is the camera's field of view.
The projection matrix's eigenvalues:

$$\begin{pmatrix} \frac{\mathrm{ar}(f+n)-\sqrt{\mathrm{ar}^2(f^2(1-8n)+2fn(4n+1)+n^2)}}{2\mathrm{ar}(f-n)} \\ \frac{\sqrt{\mathrm{ar}^2(f^2(1-8n)+2fn(4n+1)+n^2)}+\mathrm{ar}(f+n)}{2\mathrm{ar}(f-n)} \\ \frac{\cot\left(\frac{\mathrm{fov}}{2}\right)}{\mathrm{ar}} \\ \cot\left(\frac{\mathrm{fov}}{2}\right) \end{pmatrix}$$

The projection matrix's eigenvectors:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{\mathrm{ar}(f+n)-\sqrt{\mathrm{ar}^2((1-8n)f^2+2n(4n+1)f+n^2)}}{2\mathrm{ar}(f-n)} & \frac{\mathrm{ar}(f+n)+\sqrt{\mathrm{ar}^2((1-8n)f^2+2n(4n+1)f+n^2)}}{2\mathrm{ar}(f-n)} \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

The eigenvectors and eigenvalues depend solely on the distance from the camera to near plane. These results are interesting because, when **n** is greater than 1/8, there are two complementary imaginary eigenvectors and eigenvalues. In that situation, the expression inside the square root is negative.

3 Inverse Projection

Because the depth of the image was lost in the process of projection, finding the inverse projection is not as simple as finding the inverse matrix of the projection matrix; in order to compensate for this loss, at least two images obtained from two different cameras are needed to reconstruct the 3D model. In our case, these images were obtained from the prior forward projection.

3.1 Initial Approach

At first, we attempted triangulation, where we had the x and y coordinates of the image, both which are within the range of -1 and 1. These points span the width and height of the camera's image plane, defined by the focal length and the field of view; from this, it is possible to restore the global directional vector towards the object from the camera. We find the intersection of those vectors. That intersection should represent the original 3D point. Binocular vision relies upon this principle.

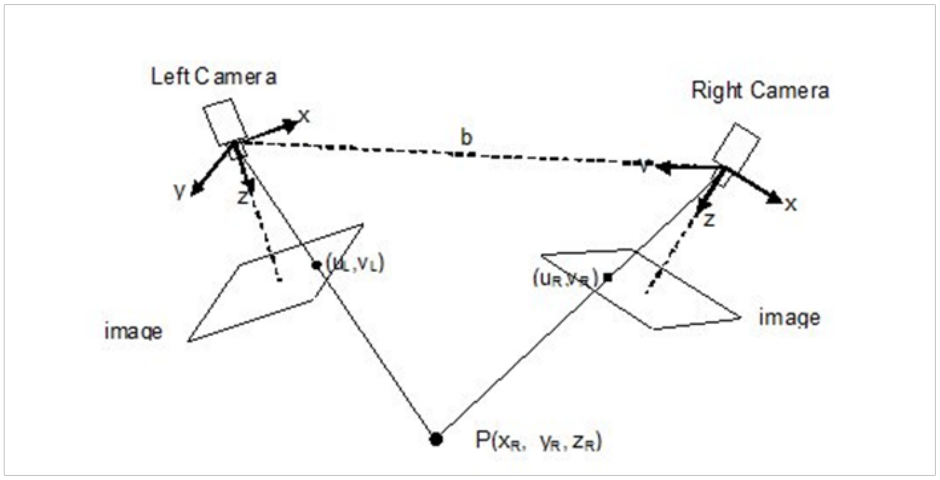


Figure 4: Triangulation, in context of binocular vision.

When the lines don't exactly intersect, we calculate the Closest Point of Approach, a point which represents the point on each lines that is closest to the other line. We averaged those two points to get an intermediate point that approximates the intersection.

Despite the fact that this is possible for real cameras, we weren't able to identify the projection plane for the canonical OpenGL projection matrix. This caused inaccuracies in the directional vectors we obtained upon the assumption that the projection plane would be equivalent to the near-plane of the frustum. Thus, we had to devise a workaround.

3.2 Workaround

We were more successful when we used 3 cameras and constructed various planes. Each plane is determined by two different vectors: one from the camera to the object and one from the known 1st and 2nd coordinates of the vector and unknown third. By finding plane defined by those vectors, we get one plane for each camera.

There is one caveat to the workaround: the cameras cannot face at the exact same point, otherwise the 3 planes intersect to get a line instead of the intended 3D point. Although it is hard to tell, the figure below demonstrates the result of having all cameras face the same point–the intersection between the planes becomes a line, as the planes must simultaneously contain the point and the object.

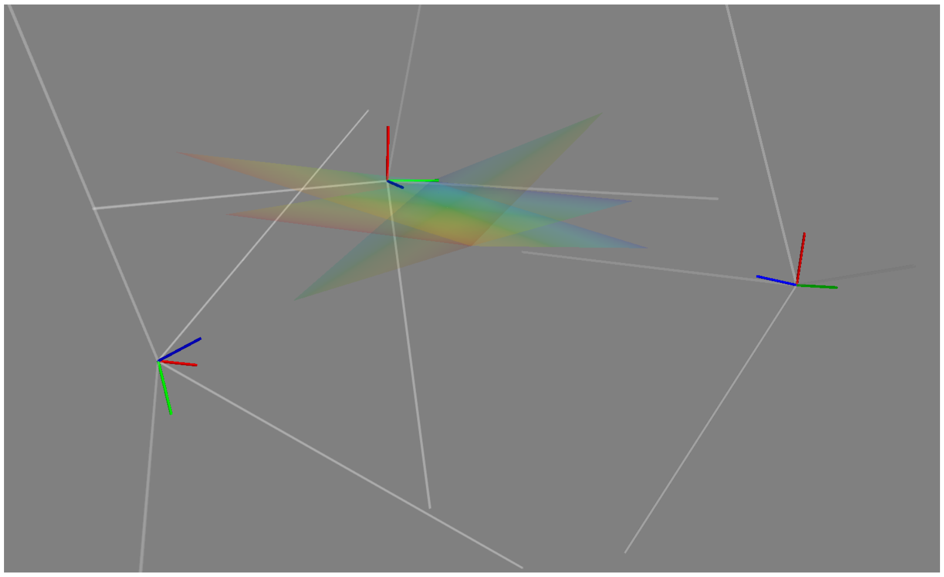


Figure 5: When all the cameras point toward the same point, the planes intersect in a line.

3.3 Results

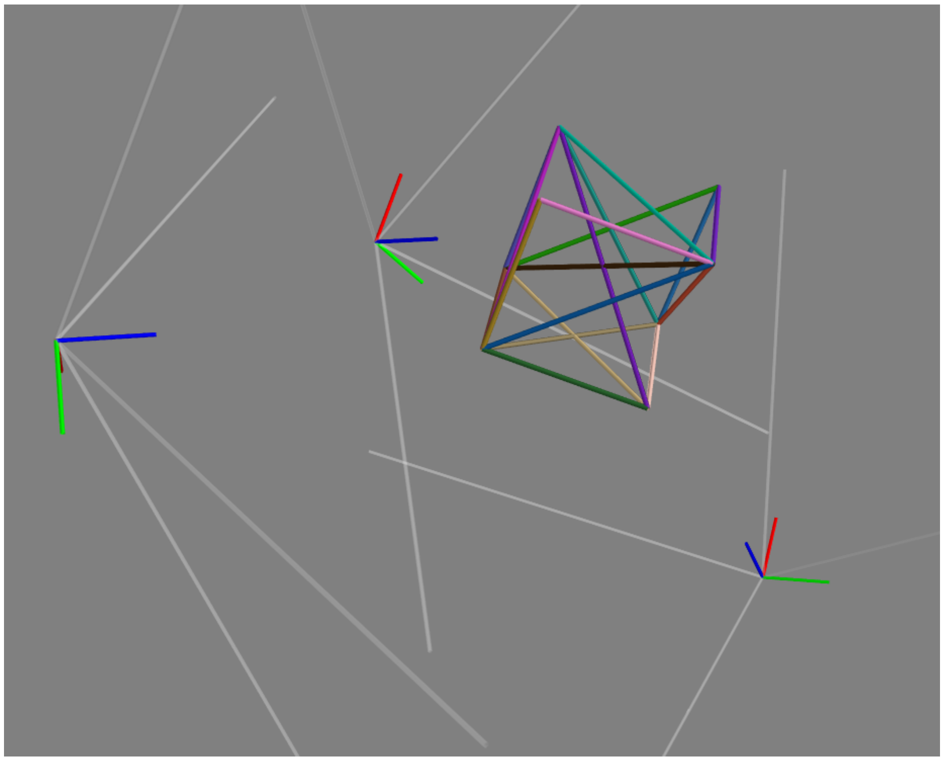
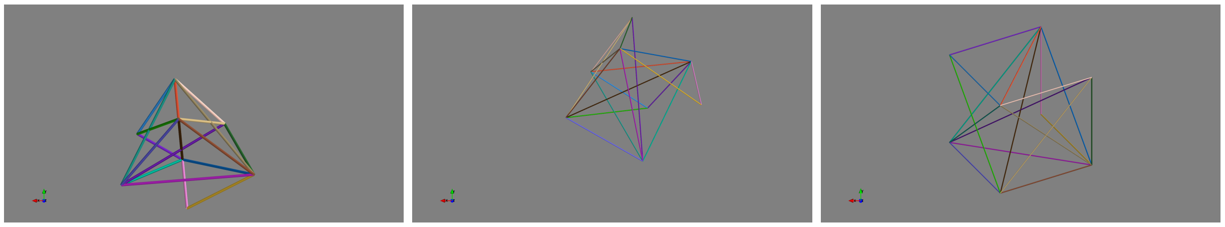


Figure 6: The Original Image, with the 3 Cameras. The red,green,and blue lines represents the x,y,and z axis of the cameras, respectively.



(a) Cam1

(b) Cam2

(c) Cam3

Figure 7: Projected image, or the "photographs" of the 3D model in the perspective of the three cameras

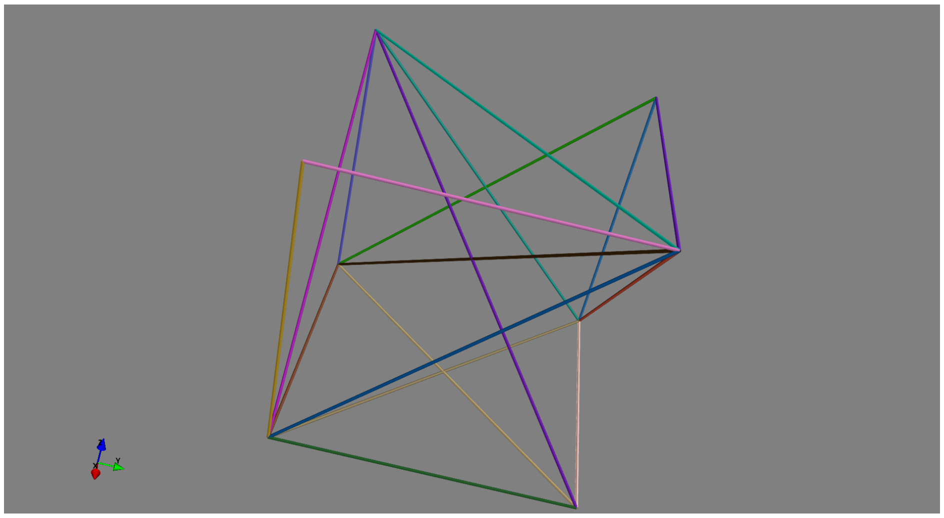


Figure 8: The Reconstructed 3D model.

4 Next Steps

If we were to continue this project, we would implement ways to identify the points of interest, such as vertexes and midpoints, automatically. Machine learning has a few tools to help identify similar features among 2D images, so we should be able to use it for this task. We could also apply our code to real cameras and account for the Fish Eye distortion and find the image plane accurately. If you would like to check out the work we have done, we documented our code in our git repository here: `https://github.com/LinearityI/Lin1_Project`