COMP 558 Project
3D Reconstruction using Optical Flow

Emil Janulewicz & Haomin (Calvin) Zheng

April 27, 2015

Abstract

In this report we study the problem of 3D city reconstruction from aerial video footage. We use the method of optical flow and study the algorithms by Lucas-Kanade and Gunnar Farnebäck. After finding an estimate of the motion field from the optical flow, we perform reconstruction in openGL to get a realistic rendering of the original scene.

1 Introduction

1.1 Problem Statement

In this report we explore the problem of 3D city reconstruction given an aerial view video. Our methods are based on the theory from optical flow which approximates the motion field of our image plane. If we know the motion field of a particular image, we can accurately calculate the corresponding depth per pixel and thus come up with a good 3D representation of the real world.

1.2 Related Works

We first give a history of the work done in the fields of 3D reconstruction ranging from classical methods to more state of the art. Next we explore the concept of optical flow, and the different algorithms existing in the current literature, which leads directly into our work.

1.2.1 3D Reconstruction

The need for 3D reconstruction of cities is heavily relied upon by services such as navigation systems, location based services, and even augmented reality. Figure 1 gives a sample screen shot of 3D reconstructed Boston, MA from Google Earth.

Figure 1: 3D visualization of Boston, MA as seen in Google Earth

It is important for the extraction of the 3D data to be automatized as much as possible. For example, the method of stereo photogrammetry is a major acquisition effort and any updates of the database can acquire up to 70% of the original acquisition cost [1]. The most popular method used is based on light detection and ranging (LiDAR) [2]. One drawback of airborne laser scanning is poor lateral measurement accuracy. However, there still lacks a fully automated method with low complexity that scales well. In the next section we describe optical flow as it will be our primary method to finding height reconstructions from airborne video captures.

1.2.2 Optical Flow

The optical flow of an image assigns a vector to each pixel which measures the spatial and temporal variations of the image brightness. It is an estimate of the image motion field which is a projection of the 3-D velocity field onto the image plane. Given the motion field, we can infer a lot about the real world such as 3-D motion and structure.

The derivation of optical flow is based on the following assumptions:

- There is only one rigid relative motion between the camera and the observed scene
- The illumination conditions do not change

Given these assumptions, and knowing that the image irradiance is proportional to the scene radiance, we can say that the apparent brightness of moving objects remain constant. We define the image brightness via the function $E(x(t), y(t), t)$. This leads to the derivation of the Image Brightness Constancy Equation:

$$\begin{aligned} \frac{d}{dt}E\left(x(t), y(t), t\right) &= 0 \\ \frac{\partial E}{\partial x}\frac{dx(t)}{dt} + \frac{\partial E}{\partial y}\frac{dy(t)}{dt} + \frac{\partial E}{\partial t} &= 0 \\ E_x v_x + E_y v_y + E_t &= 0 \\ \left(\nabla E\right)^{\mathrm{T}} \mathbf{v} + E_t &= 0 \end{aligned} \tag{1}$$

However, the constraint in equation 1 can only compute the motion field component in the direction of the spatial image gradient. This is known as the Aperture problem which formally states that the direction orthogonal to ∇E is not constrained by the image brightness constancy equation.

Several methods have been proposed in the literature that deal with efficient and accurate calculations of the optical flow. Horn and Schunck [3] presented an iterative implementation which adds another constraint by assuming the apparent velocity and brightness of the image varies smoothly almost everywhere. The advantages are a high density of flow vectors but are prone to noise. Horn later on derived a method for a much simpler case in which all parts of an image move with the same constant velocity [4]. The Lucas-Kanade method [5], [6] assumes the same flow in a particular patch (region) and uses least squared method to approximate for the motion field. If the assumption of constant flow in a patch is close to being correct, we can solve for the aperture problem. However, as with Horn-Schunk, this method is sensitive to noise, especially outliers.

Another disadvantage is that it cannot calculate the flow in areas of uniform image brightness intensity. A more recent algorithm was developed by Farnebäck [7] which approximates the neighborhood of two consecutive frames by a quadratic polynomial.

2 Methods and Algorithms

2.1 Problem Setup

In order to tackle our problem, the first task was to obtain aerial footage of a city. However, retrieving good baseline footage is not so trivial. It is ideal to set certain conditions which will help with depth extraction as we will see later on. This includes the camera to be traveling at a constant velocity with constant height and fixed orientation (heading). We also want to focus on basic shapes and polyhedrons such as rectangles, triangles, and spheres so as to simplify the geometry of the problem. This is a good approach since most objects in a city are made up of polyhedrons.

2.1.1 3D Scene Generation

Achieving our conditions stated above can be easily satisfied using the Unity 3D game engine. Generating aerial footage using Unity is a good baseline approach as it recreates our desired controlled environment where parameters are known (height, speed, lighting conditions). Figure 2 gives a sample snapshot of the aerial footage which corresponds to the 150th frame.

In the scene, we modeled a tall building, a two-floor building, a building with a sphere on top, and a pyramid.

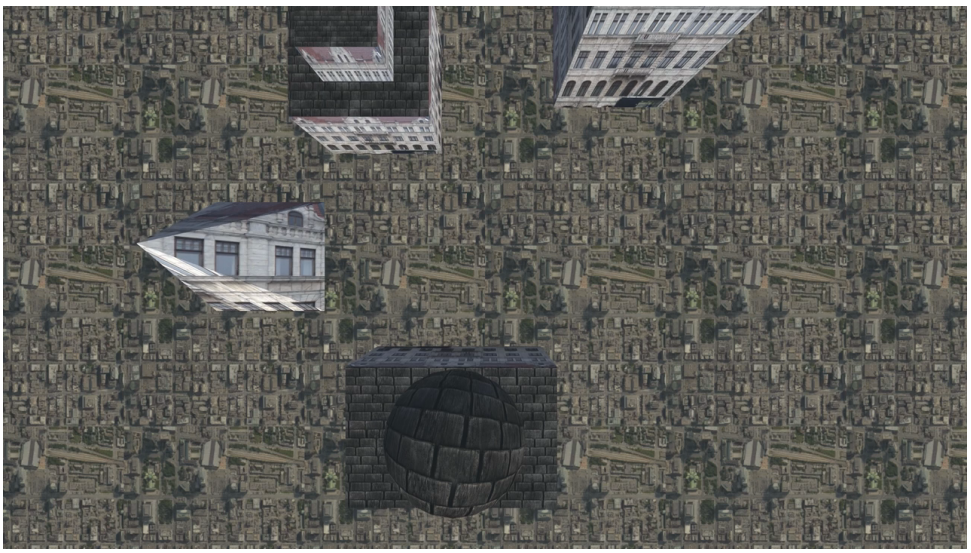


Figure 2: Snapshot from our Unity 3D created overhead video. Video can be downloaded from `https://dl.dropboxusercontent.com/u/8450216/overheadScan.mov`

The generated footage has the property of constant speed traveling in the y-axis and the optical axis perpendicular to the ground. Table 1 gives all our parameters.

Parameter	Value
Camera Height	13m
Camera Speed	5m/s
Resolution	1280×720
Field of View	60°Vertical
Frame per second	60.17

Table 1: Unity 3D Video footage parameters

2.2 Computing Optical Flow Field

2.2.1 Sparse Lucas Kanade

Sparse Lucas Kanade finds the optical flow of particular points of interest known as features. The algorithm implementation is based on OpenCV Python libraries. For the features, the method **cv2.goodFeaturesToTrack()** obtains the Shi-Tomasi [8] corner points which are then tracked via the Lucas Kanade algorithm using the method **cv2.calcOpticalFlowPyrLK()**. This algorithm is run iteratively for each consecutive set of frames.

This implementation assumes the same flow for each 3×3 image patch. Thus, taking our original image brightness constancy constraint, we end up with an over-determined system with 9 equations and 2 unknowns (x and y velocity).

$$\begin{pmatrix} E_x^1 & E_y^1 \\ E_x^2 & E_y^2 \\ \vdots & \vdots \\ E_x^9 & E_y^9 \end{pmatrix} \begin{pmatrix} v_x \\ v_y \end{pmatrix} = \begin{pmatrix} -E_t^1 \\ -E_t^2 \\ \vdots \\ -E_t^9 \end{pmatrix}$$

To solve for the unknowns, v_x and v_y, least squares approach is used which computes the flow as to minimize the error.

After we solve the optical flow using Lucas Kanade method, we acquire a set of points in the image where we know the flow magnitude and direction. However, in order to reconstruct the 3D model, we need to know the exact depth of each pixel of the image. To calculate the missing depths, we linearly interpolate the depths of the known points to estimate unknown points. This way we can get an approximation of a dense flow field of the image.

2.2.2 Dense Lucas Kanade

Dense Lucas Kanade essentially tracks each pixel in the image. In this case, there is no need to tractk feature points and we directly use **cv2.calcOpticalFlowPyrLK()** on all points. However, the algorithm complexity is substantially higher when compared to the sparse method due to the number of points we need to process.

2.2.3 Gunnar Farnebäck

In order to process the dense optical flow computations more efficiently, we turn to the Gunnar Farnebäck algorithm [7]. Again here we use the Python OpenCV library utilizing the method **cv2.calcOpticalFlowFarneback**.

The main highlights of the algorithm are as follows: each pixel and its neighborhood is approximated by a quadratic polynomial which is calculated via a polynomial expansion transform. Given the properties of a polynomial transform under a translation, you can estimate the displacement of each neighborhood. It is a robust algorithm allowing efficient calculation of dense optical flow.

2.2.4 Special Case: Fixed Orientation

Consider an aerial drone with a fixed camera pointing towards the ground (optical axis is perpendicular to the ground) traveling with constant velocity V_y. Therefore, we only have translation in the y-axis and thus $T_y = -V_y$.

For a particular pixel on the image, the motion flow vector can be expressed as $v_y = \frac{-T_y f}{Z}$, where f is the focal length of the camera and Z is the depth of the pixel in the real world with respect to the camera. This is due to the fact that the aerial drone is traveling exclusively along the y-axis. Thus, we can recover the depth as $Z = \frac{-T_y f}{v_y}$. Given the height of the drone, Z_d, the reconstructed height is given as $Z_d - Z$.

What's left is to recover the image velocity v_y. For this we return back to the image brightness equation and constant brightness assumption.

$$\begin{aligned} \frac{d}{dt} E\left(x(t), y(t), t\right) &= 0 \\ \frac{\partial E}{\partial x}\frac{dx(t)}{dt} + \frac{\partial E}{\partial y}\frac{dy(t)}{dt} + \frac{\partial E}{\partial t} &= 0 \\ E_x v_x + E_y v_y + E_t &= 0 \\ v_y &= \frac{-E_t}{E_y} \end{aligned} \tag{2}$$

The last simplification is made in light of the fact that we know $v_x = 0$. Thus for each point in the image, we have a linear equation with one unknown and thus we can successfully recover the velocity and ultimately the true depth/height. For completeness, the pixel height (relative to the ground) is as follows:

$$Z_d - \frac{V_y f E_y}{E_t} \tag{3}$$

2.3 3D Reconstruction from Motion Field

2.3.1 Depth Map from Motion Field

Using our optical flow as the approximated motion field and following the general motion field equation[9] we have:

$$v = f\frac{Z\mathbf{V} - V_z\mathbf{P}}{Z^2} \tag{4}$$

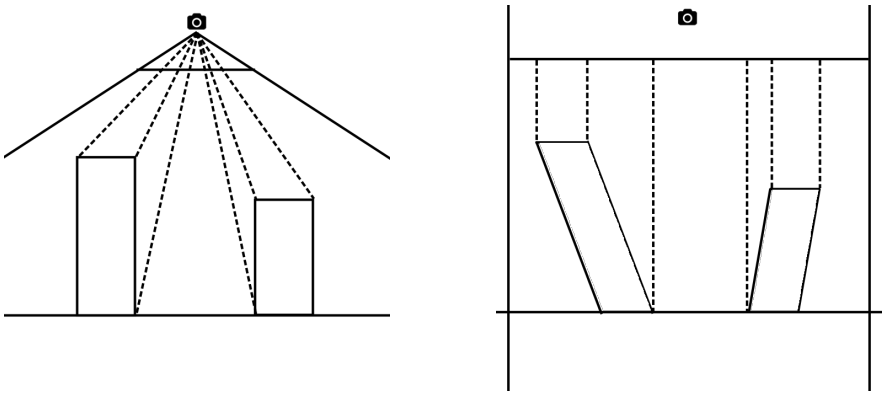
where $\mathbf{V} = \{V_x, V_y, V_z\}$ is the 3-D velocity and $\mathbf{P} = \{X, Y, Z\}$ is the real world coordinate of the pixel. In our case, we assume no rotation and no movement on the z and x axis. Thus the equation simplifies to:

$$v_y = \frac{fV_y}{Z} \tag{5}$$

Using the flow field v_y as computed using the previous algorithms, we can easily produce the depth Z of individual pixels.

2.3.2 Perspective Restore

The depth map recovered in the previous section is not a true representation in the real world coordinate system since cameras are in the perspective coordinate system.



(a) Perspective camera (b) Depth recovered

As we can see in figure 3b, a vertical building will appear to be skewed outwards in the depth map. To restore the actual shape, we need to calculate the x, y in the world coordinate, relative to the look-at point of the camera. With the depth information, we can easily do this by:

$$X = \frac{xZ}{f} \tag{6}$$

$$Y = \frac{yZ}{f} \tag{7}$$

2.3.3 Applying Texture

For more accurate and aesthetically pleasing results, we can recover texture information from the video image. We extract the color intensity from the image plane and superimpose them onto the 3D points generated for every corresponding pixel. After all the data is aggregated, we render the points with colors using OpenGL. The results are shown in the next section.

3 Results

3.1 Optical Flow Result

The following results are computed using the 150th and 151th frame in the video, we only preserved the flow in the positive y direction, with white indicating maximum velocity and black indicating minimum velocity. For the original frame image, you can refer to figure 2 which includes the url for the video.

3.1.1 Sparse Lucas Kanade

Using Shi-Tomasi corner points, around 10000 feature points are used to calculate the optical flow. The results for all other points are interpolated.

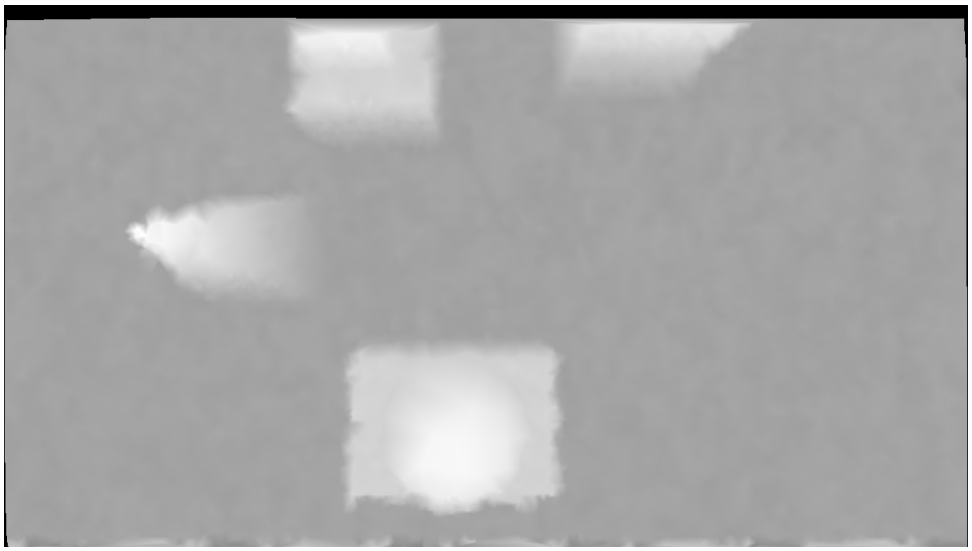


Figure 4: Flow field Using Sparse Lucas Kanade

The running time for this algorithm is fairly fast at 0.61 seconds.

In figure 4 we can clearly see the overall structure of the scene, but the edges, e.g. the top of the pyramid is not very clean, due to interpolation.

3.1.2 Dense Lucas Kanade

Figure 5: Flow field Using Dense Lucas Kanade

The running time was very high at 635.13 seconds. This is even after reducing the video footage resolution to 480p.

The results have better defined edges as seen in figure 5, but at the expense of complexity when compared to Sparse Lucas Kanade, even at a lower resolution.

3.1.3 Gunnar Farnebäck

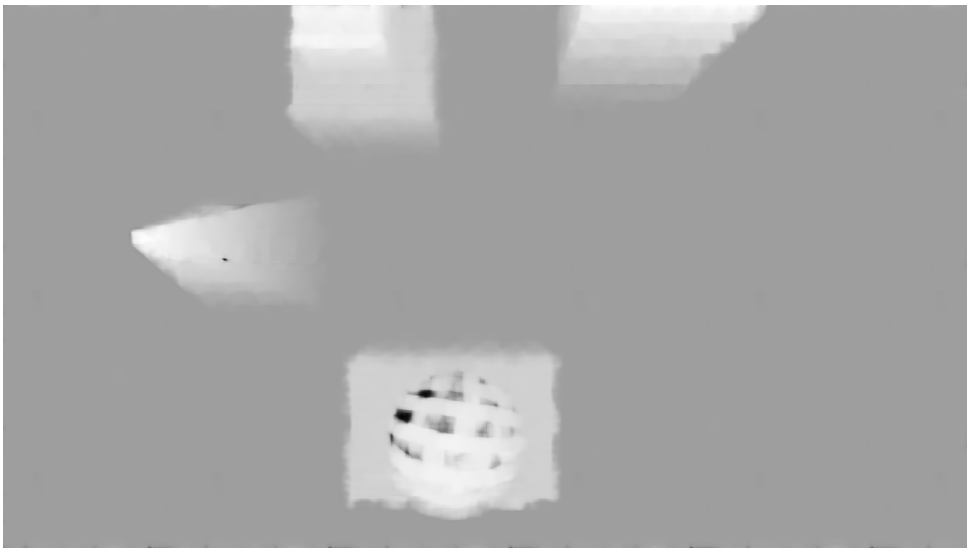


Figure 6: Flow field Using Gunnar Farnebäck

The running time obtained is 0.43 seconds.

It operates extremely fast, the quickest of the three, and we can see clear edges in most cases (figure 6). However, we do notice some dark spots, especially in the sphere which

indicates no flow vector. These spots would produce high error in height reconstruction. This is caused by the near uniform color on the roof top, but this does not have too much negative effect on the overall shape of the model. We will use this result for the remainder of our work dealing with reconstruction.

3.2 Reconstruction

Our reconstruction algorithm resulted in 4 steps:

1. Depth Map Recovery
2. Perspective Restore
3. Texture Application
4. Frame Stacking

3.2.1 Rendering from Depth Map Recovery

Using equation 5, we calculated the relative depth of each point and rendered the resulting points using OpenGL. We chose a different viewpoint when compared to the original image so as to see the effects. The results are visualized in figure 7.

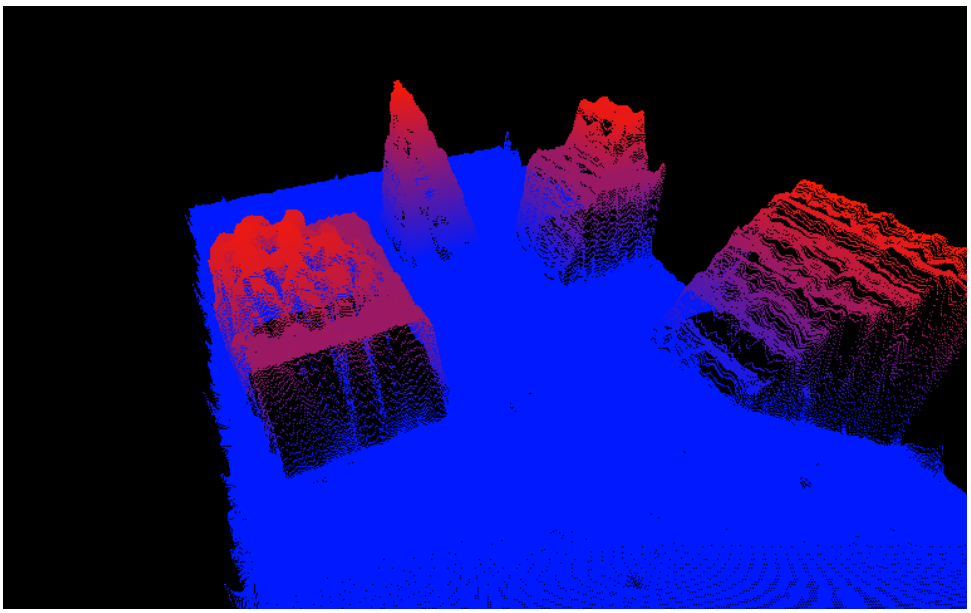


Figure 7: 3D Render before perspective restore

3.2.2 Perspective Restore

Before performing our perspective restore, we remove the noise caused by outliers (extreme values). After correcting for perspective, the resulting rendering is seen in figure 8. We can see that the building are now correctly restored to their vertical positions.

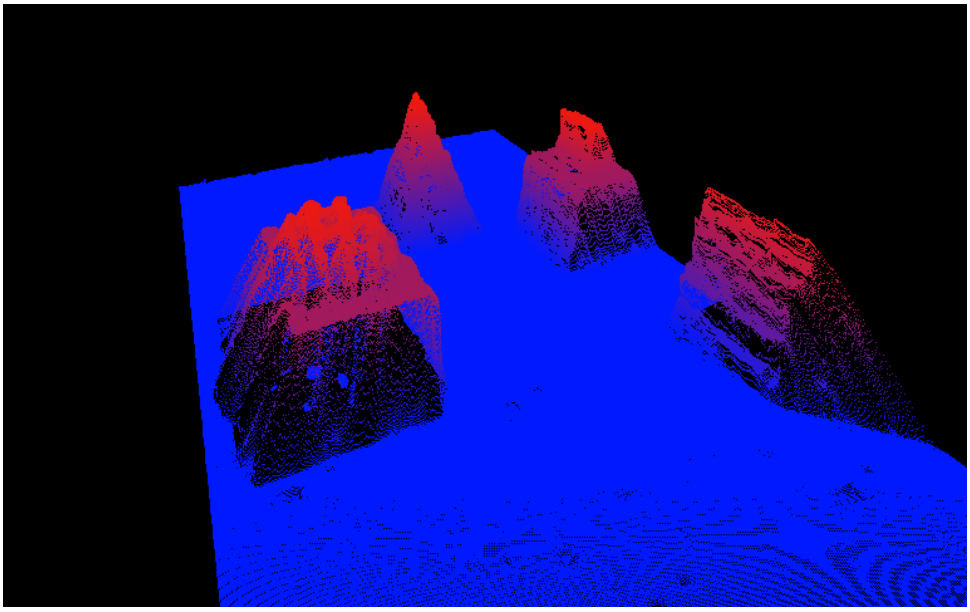


Figure 8: 3D Render after perspective restore

3.2.3 Texture Application

Using the color intensity information from our video frame, we can apply the same color pallete to our rendered scene. This gives a nice representation of the true scene as seen in figure 9.

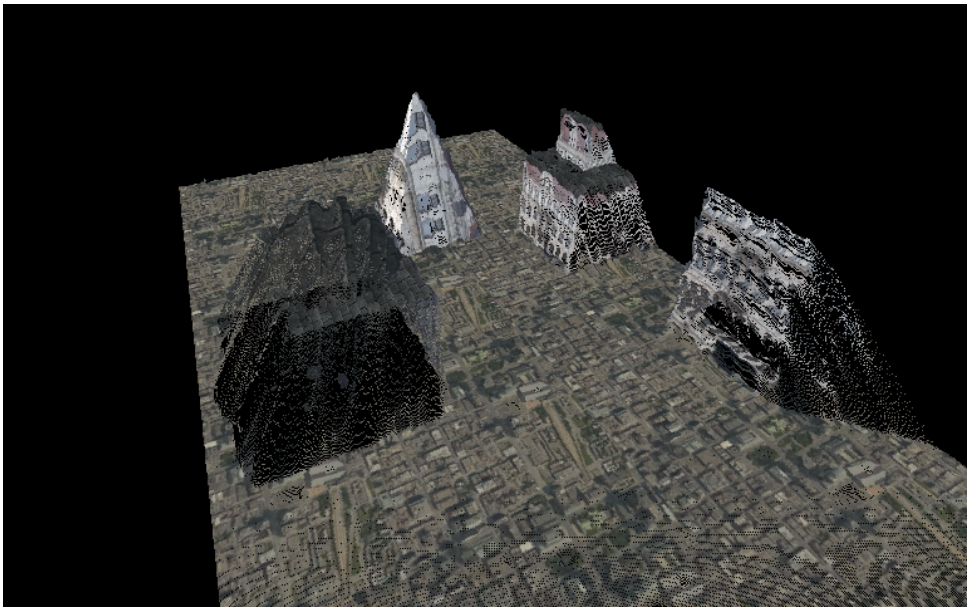


Figure 9: After texture application

3.2.4 Frame Stacking

Since the sides of building lack points from a single frame, we stack results from 40 consecutive frames to give a more dense representation. The results are shown in figure

10.

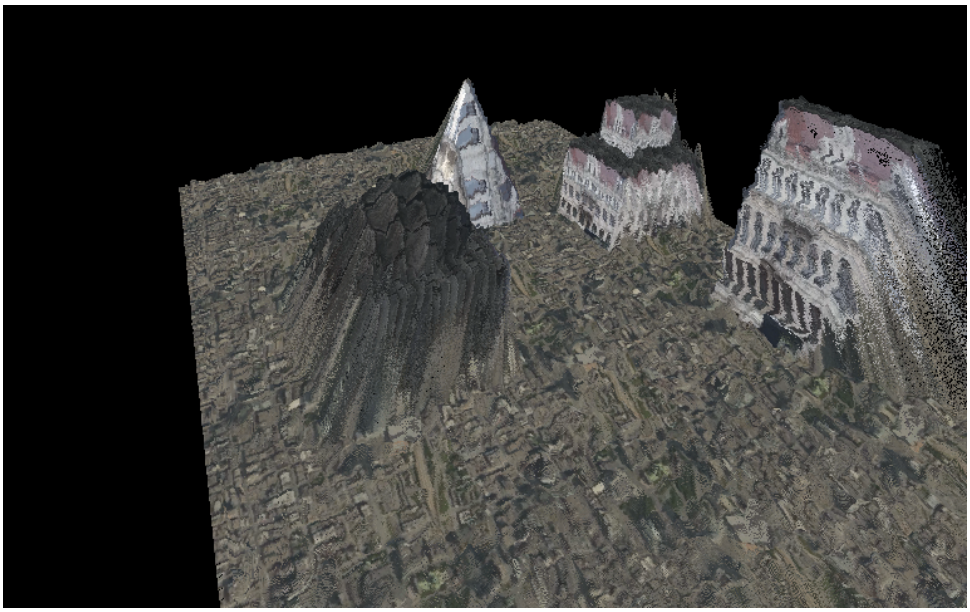
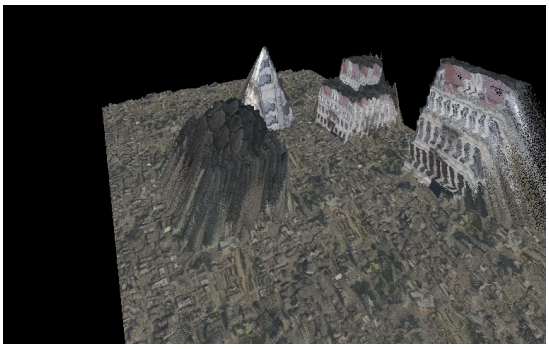


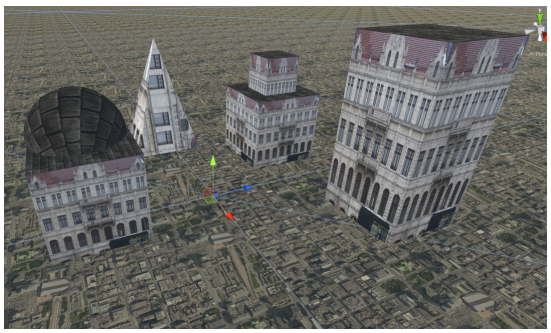
Figure 10: After frame stacking

Stacking more points gives a much more complete view of the scene. Due to hardware memory limitations, we can only render results with a maximum of 40 frames.

We can now compare our final reconstruction result rendered in OpenGL, with the original 3D models in Unity.



(a) Reconstructed



(b) Original

4 Discussion

- Comparison of Optical Flow Methods

  The Gunnar Farnebäck algorithm excels both in efficiency and quality, but it is not robust enough to process flow vectors on small near uniform color patches. This is due to the fundamental aperture problem of optical flow, as one cannot derive any flow on a uniform colored surface. This will, however, rarely happen in an actual city scan footage, as the building structures and roads are clustered in most

downtown areas. But if applied to a scan footage of a countryside or farm field, we would need to improve our algorithm to get a reasonable result.

- Difficulty of Optical Flow Calculation via Equation 2

  From equation 2 we have $v_y = \frac{-E_t}{E_y}$. We fall into two issues here. First, the time gradient may be overestimated for far away objects with richer texture then near objects. Secondly, the aperture problem still persists in areas of uniform color. Smoothing in the temporal domain may help overcome the first problem.

- Assumptions and Parameters

  In this simulated environment, we assume a perfectly aligned flying route, with all the camera parameters known and defined. If the flying route and the camera orientation is not defined, we can still obtain a relative depth map using two consecutive frame. However we can not stack more results together as we have no information of the relative translation between frames.

- Difficulty in Result Visualization

  After the perspective restore, the result is no longer a height map on a well-aligned grid, and we can not use the *Mesh* command in Matlab to present the results. We also tried to visualize the result using scatter points, but the performance suffers when we try to display millions of pixels at once. In the end, we then switched to direct rendering in openGL, with buffers provided in GPU memory. This is able to significantly speed up the rendering process. The final result shown has around 36 million entries of 3D points.

- Side of Building Appear Blurred Out

  As we can see in the reconstructed scene in figure 11a, the sides of the buildings that are facing away from the center are blurred out. This is due to lack of information in the original video. As the camera pans through the y-axis, the outside of the building structure is not visible throughout the video clip. To recover a complete reconstruction of the scene, we should record multiple scans of the scene, each parallel to each other and then apply the result together.

References

[1] "Building reconstruction from images and laser scanning", *International Journal of Applied Earth Observation and Geoinformation*, vol. 6, no. 3–4, pp. 187–198, 2005.

[2] N. Haala and C. Brenner, "Generation of 3d city models from airborne laser scanning data", in *Proceedings EARSEL workshop on LIDAR remote sensing on land and sea*, 1997, pp. 105–112.

[3] B. K. P. Horn and B. G. Schunck, "Determining optical flow", *ARTIFICAL INTELLIGENCE*, vol. 17, pp. 185–203, 1981.

[4] B. K. P. Horn, *Determining constant optical flow*, 2003.

[5] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision", in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'81, Vancouver, BC, Canada: Morgan Kaufmann Publishers Inc., 1981, pp. 674–679. [Online]. Available: `http://dl.acm.org/citation.cfm?id=1623264.1623280`.

[6] B. D. Lucas, "Generalized image matching by the method of differences", PhD thesis, Robotics Institute, Carnegie Mellon University, Jul. 1984.

[7] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion", in *Proceedings of the 13th Scandinavian Conference on Image Analysis*, ser. SCIA'03, Halmstad, Sweden: Springer-Verlag, 2003, pp. 363–370, ISBN: 3-540-40601-8. [Online]. Available: `http://dl.acm.org/citation.cfm?id=1763974.1764031`.

[8] J. Shi and C. Tomasi, "Good features to track", in *IEEE CVPR*, 1994, pp. 593–600, ISBN: 0-8186-5825-8. DOI: `10.1109/CVPR.1994.323794`. [Online]. Available: `http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=323794`.

[9] E. Trucco and A. Verri, *Introductory Techniques for 3-D Computer Vision*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998, ISBN: 0132611082.