Augmenting Videos with 3D Objects

Andrei Bajenov
abajenov@stanford.edu

Darshan Kapashi
darshank@stanford.edu

Sagar Chordia
sagarc14@stanford.edu

Abstract

We propose a way to automatically augment a video of a static scene with a 3D object. We use SFM algorithms to estimate the position of the camera. In order to properly support occlusions, we use a novel approach to generate dense depth maps for each frame of the video. We use a combination of semi-global block matching, image segmentation using the watershed algorithm, and planar interpolation to remove noise and sharpen edges. The final result is a video augmented with a 3D object that is properly occluded by objects in front of it.

1. Introduction

The idea of augmenting videos has been around for a while, and we see it everywhere today. One prominent example is CGI in movies, which augments reality with computer generated objects and makes the viewer believe that these objects are part of the environment.

The process to do this is generally difficult and requires a lot of specialized software and equipment. In this paper, we describe a system that, given an object mesh and a video, allows anyone to place this object seamlessly into the video without any other external inputs.

There are a number of interesting applications to this. For example, it could be used for seeing how a piece of furniture would look in a room or how a new house would look in a particular location. If integrated with a smartphone's camera, it could also be used when interacting with an environment. For example it could provide navigational pointers, highlight parts of an environment, or even project another person into an environment.

There are a growing number of technologies that are being built to support this. Project Tango [1], by Google, is building a phone that has a built-in depth sensor to make 3D reconstruction and camera tracking easier. Wikitude [2] is an example of a piece of software that is designed for the purpose of augmenting smartphone videos. It allows for camera tracking and 3D reconstruction. There is also quite a bit of research in 3D reconstruction from a set of 2D images.

We have not been able to find a product that specifically takes a video and augments it while supporting occlusions (although a number of technologies like Wikitude support projecting objects based on camera positions). This was our main motivation for building this system.

2. Problem Statement

We propose a system that puts a 3D object in a video of a static scene while supporting occlusions. We break up the problem into two components:

1. Estimate camera matrices for each frame of the video, so that we can project an object back into the scene.

2. Obtain sharp and accurate depth maps for each frame, to deal with occlusions.

Below is an example of a scene that we used for testing our system.

Figure 1: An example of a static scene

3. Previous Work

3.1. Estimating Camera Positions

There has been a lot of research done in the area of Structure from Motion (SFM), and there are a number of existing libraries that implement SFM algorithms, including:

- Theia SFM [3]
- Bundler [11]
- Visual SFM [10]
- OpenCV SFM [9]

We were looking for a few things from the libraries:

- Camera position estimation
- Camera parameter estimation
- Reliable and accurate sparse 3D reconstruction

For this project, it was not our goal to try to improve or optimize any of these libraries. We tried a few of them and picked the one that was easiest to use. In our case it was the Theia SFM library.

3.2. Estimating Depth Map

A critical part to solving our problem was obtaining accurate and dense depth maps for each frame of the video. There were a number of techniques that we considered:

- Reconstructing 3D objects using volumetric stereo and using these reconstructions to obtain depth maps [12]
- Using a combination of the original images and the sparse 3D points obtained from SFM to approximate the 3D surface positions (using segmentation and planar reconstruction).
- Using a combination of SFM and stereo matching algorithms to obtain a dense 3D reconstruction of the scene. [16]

While researching volumetric stereo, we found that it was genereally used to get a 3D reconstruction of a single object within a scene. For our purposes, we needed information about the full scene. To extend this algorithm to work on a full scene, we would have needed very reliable image segmentation algorithms that were determenistic between frames. We were not able to find anything that looked promising in this space, so we abandoned this idea.

We also considered using the sparse 3D points obtained from SFM to approximate a dense 3D reconstruction. We though about partitioning the original image into uniform segments. We would then approximate each segment as a plane in 3D and use the sparse 3D points to estimate these planes. Unfortunately what we found was that we did not have enough points in each image segment to do a planar reconstruction, so we could not use this approach by itself.

Lastly we found a lot of research about using stereo-matching algorithms to aid in dense 3D reconstruction [17]. Specifically, the Middlebury website [13] contains a lot of submissions and evaluations of many stereo-matching algorithms. We ended up pursuing this approach the most because the research here showed the most promising results.

To solve our problem, however, we do not need a full 3D reconstruction of the scene. We just need an approximate depth map that has good accuracy around the object boundaries. What we found while using just stereo-matching algorithms was that they were prone to noise. To overcome this problem and achieve the desired results, we propose a novel approach that uses a combination of image segmentation techniques, stereo-matching, and planar interpolation.

4. Technical Details

Below we describe our solution. We talk about how we calibrate our camera and run SFM. We then describe our method for getting accurate depth maps. Lastly, we describe how we project 3D objects back into the scene.

4.1. Sparse 3D Reconstruction and Camera Matrix Estimation

As per standard practice in the camera model used in computer vision, there are 2 parameters:

- Intrinsic matrix K: A 3x3 matrix which incorporates the focal length and camera center coordinates.
- Extrinsic matrix [R T]: A 3x4 matrix which maps world coordinates to camera coordinates. R denotes rotation and T denotes translation.

The camera transformation is given by a matrix,

$$M = K[R\,T]$$

It transforms a point in homogeneous world coordinates to homogeneous image coordinates.

The *point correspondence* problem is defined as follows: Given n images, find points in the images which correspond to the same 3D point. There are several well known algorithms which work reasonably well in practice, for example, SIFT, SURF and DAISY.

The *Structure from motion (SFM)* problem is defined as follows: Given m images and n point correspondences, find m camera matrices (M) and n 3D points. Solving the SFM problem for a set of images will give us a sparse 3D reconstruction of the scene.

To get the intrinsic parameters for our camera we tried a few different approaches:

- Computing K using single view metrology with 3 vanishing points derived from 3 pairs of mutually orthogonal lines in 3D.
- Using a checkerboard image to calibrate using OpenCV routines.
- Allowing *Structure From Motion (SFM)* algorithms to self-calibrate (which is possible given enough viewpoints of a static scene)

Each of these approaches gave us a similar K, so we decided to go with the self-calibration method since it is automatic and we have plenty of views.

We tried a couple different SFM libraries. The SFM library bundled with OpenCV didn't give good results. The Theia [3] library was easier to work with and was able to give us fairly accurate sparse reconstructions of a 3D scene, figures 2, 3.

Figure 2: Theia's camera positions and a sparse 3D reconstruction

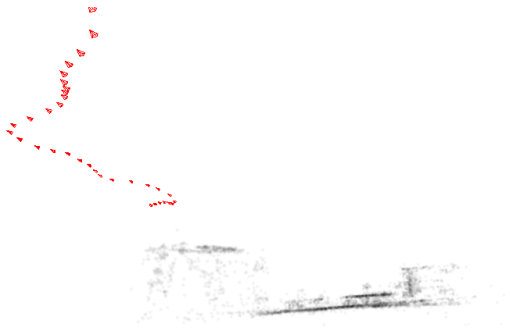


Figure 3: Side view of the 3D reconstruction

At this point, we have the camera intrinsic and extrinsic matrices as well as a small set of 3D points which can be used to get a sparse reconstruction of the scene.

4.2. Depth Map Estimation

A sparse reconstruction is not enough to get a full depth map for each frame.

Below we propose a novel approach of using a combination of stereo-matching, image segmentation (using the watershed algorithm), and planar interpolation to get dense 3D depth maps for each frame.

4.2.1 Terminology

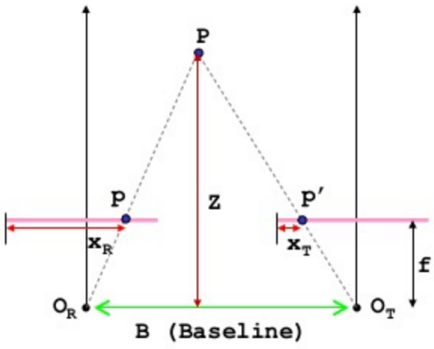


Figure 4: Stereo setup

1. *Disparity map*. Disparity refers to the difference in image location of an object seen by the left and right cameras, resulting from different positions of two cameras as seen in figure 4. A disparity map is a mapping for each pixel in the image to the disparity value for that pixel. The value represents the distance between the locations of a point in the left and right rectified stereo images. It indicates the relative distance to the camera. A higher value means it is closer to the camera.

2. *Depth map*. A mapping for each pixel in the image to the depth value for that pixel. It indicates how far a point is from the camera. A higher value means it is further from the camera.

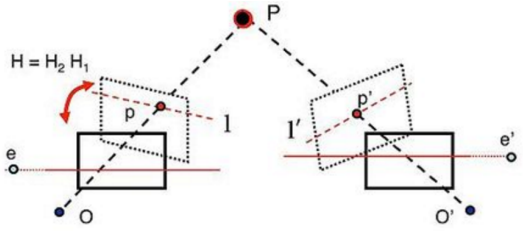


Figure 5: Image rectification

3. *Image rectification*. A transformation to project two images onto a single image plane as seen in figure 5. After rectification, all epipolar lines are parallel in the horizontal axis. All corresponding points have identical vertical coordinates.

4.2.2 Semi-global block matching

After obtaining camera matrices for every frame of the image, we use that information to perform stereo matching between pairs of frames. We picked pairs of frames with a good baseline distance between them (in our case around 2.0), and performed stereo matching on rectified versions of these frames.

We tried different stereo-matching algorithms and picked *Semi-global block matching (SGBM)* since it was readily available in OpenCV and had good performance on the Middlebury dataset.

SGBM aims to minimize a global energy function E for the disparity image D, based on the idea of pixel-wise matching of mutual information and approximating a global 2D smoothness constraint by combining many 1D constraints.

It takes as input 2 rectified images, taken from the camera on the left and from the camera on the right. It also takes the camera matrix K. It produces disparity maps for the left and right images. Figure 6 shows an example disparity map.

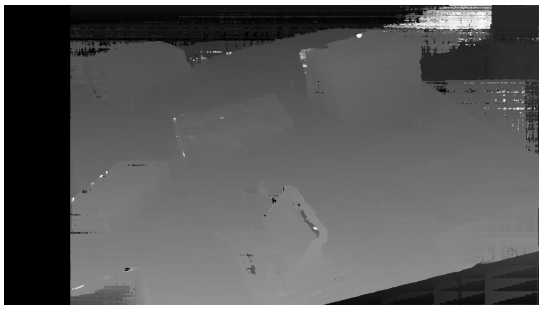


Figure 6: SGBM Disparity Map

After obtaining a disparity map, we do a first pass to remove noise. We found a technique called *Weighted Least Squares filter* [9]. The result is in figure 7.

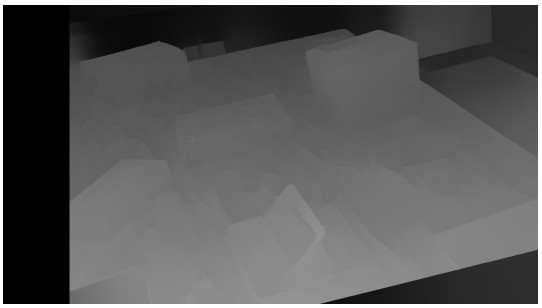


Figure 7: Weighted Least Squares filtering on an SGBM disparity map

4.2.3 From disparity maps to 3D points

Disparity maps alone don't help, since they don't tell us the exact depth of objects in each frame. To convert between disparity maps and depth maps, we first need to reproject the disparity values to 3D.

$p = (x, y)$ is a point in the disparity map. The matrix Q incorporates the transform between the left and right cameras which is obtained during image rectification. We can get the homogeneous coordinate in 3D using this equation.

$$[X\ Y\ Z\ W]^T = Q * [x\ y\ disparity(x, y)\ 1]^T$$

And finally get a mapping from 2D to 3D.

$$3d\_image(x, y) = (X/W, Y/W, Z/W)$$

Figure 8 is the dense 3D reconstruction obtained from the above equation.

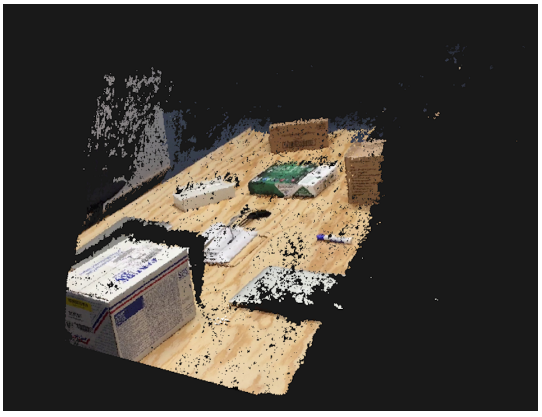


Figure 8: 3D reconstruction from a disparity map

4.2.4 From 3D points to a depth map

So far, we are able to obtain 3D points from a disparity map. These points are not in world coordinates though, so we need to transform them to world coordinates before generating depth maps.

Let x be the point in world coordinates. Let p be the point in the original image. We rectify the image for stereo matching. Rectification is a homographic transform. Let H be the inverse of this transform. K, R and T are camera parameters.

The point x maps to p using the camera transform.

$$p = KRx + KT$$

The point p_r is the point in the rectified image, which corresponds to the point p in the original image. x_r is the 3D point in rectified coordinates.

$$p_r = K_r R_r x_r + K_r T_r$$

We get p by applying the rectification transform H on p_r,

$$p = Hp_r$$

Using these equations, we can derive the equation for point x in original world coordinates.

$$x = R^{-1}K^{-1}HK_rR_rx_r + R^{-1}K^{-1}HK_rT_r - R^{-1}T$$

We reproject these points back into the original frame and compute the depth for each pixel. Figure 9 is a depth map for the frame from which we generated the disparity map.

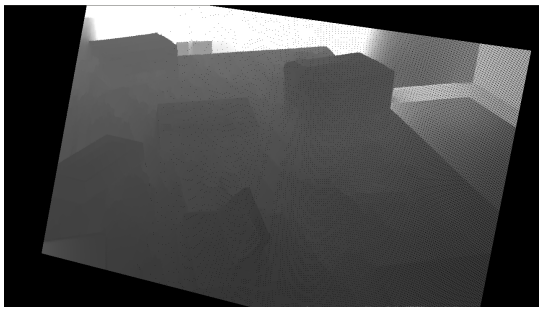


Figure 9: Example depth map obtained from a disparity map

4.2.5 Missing depth maps

We don't have a disparity map for every frame. Frames where the camera is moving forward, for example, don't have a good corresponding stereo frame. Rectification between such views introduces too much distortion.

As such, we need to be able to reconstruct a depth map for any frame, using a depth map generated from some other frame. Fig 10 is an example of a depth map viewed from a different camera:

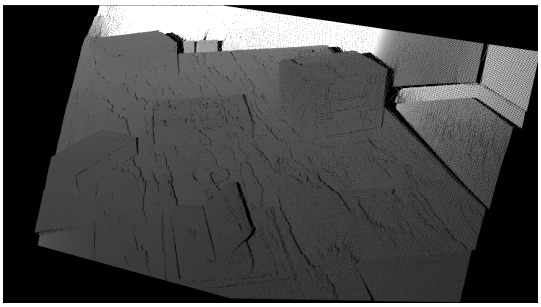


Figure 10: Depth map from another camera location - more occlusions

There are far more missing depths, which are mostly there due to occlusions. To help fill in the rest of this depth map, we use a novel technique which combines image segmentation and planar reconstruction, as described in subsections below.

4.2.6 Image segmentation

We use the marker controlled watershed algorithm for image segmentation. The watershed algorithm is based on the concept of flooding the image from its minima and preventing the merging of water coming from different sources. This partitions the image into 2 parts: the catchment basins and the watershed lines. This approach results in over-segmentation, so we use a variant which is based on starting to flood from a set of markers.

To find the set of markers, we apply the following set of transformations to the image. In the process of *thresholding* an image, we set the value for each pixel to either 0 or 1 based on a threshold. We use adaptive thresholding to the image. It considers local variations in intensity and makes pixels white and black. This is significantly better than using a global threshold because the lighting in the scene is not uniform. Then, the transformed image has several small holes. We use morphological opening to fill in these holes and have a much smaller set of bigger segments. Next, we apply a distance transform, followed by a thresholding

transform which gives us a candidate set of markers. Using this, we can apply the watershed algorithm to segment the images.

This method has several parameters that can be tuned to get a segmentation of desired quality. This includes

- Window size of adaptive threshold
- Kernel for the morphological opening
- Threshold for the distance transform

Figure 11 is an example of a segmented image.

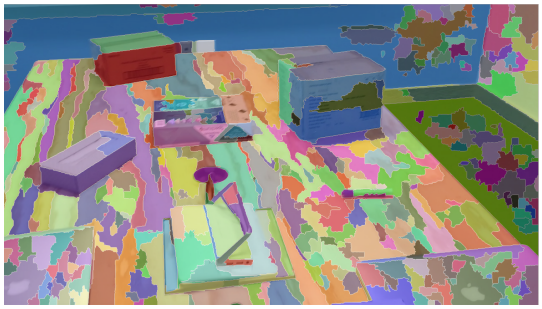


Figure 11: Image segmented using the watershed algorithm

Note that when tuning parameters, our goal is to make sure that segments don't spill between objects. We achieve this by tuning the parameters to generate small segments.

4.2.7 Planar interpolation

We have now divided the image into several segments. We assume that each segment is part of a plane. We have the depth map for each 2D image point, which means we have a 3D point corresponding to each 2D point. The camera matrix K has the following structure:

$$\begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

For each 2D point, we can compute the corresponding 3D point. z is the depth of this point from the depth map.

$$p = (z * (p_y - c_y)/f_y,\ z * (p_x - c_x)/f_x,\ z)$$

For each image segment, we collect all the 2D points for which we know a depth (z is not equal to 0). We get corresponding 3D points. For n points in a segment, we construct a nx3 matrix A where each row is a 3D point $p = (x, y, z)$. The matrix t is a nx1 matrix of -1. We can use this set of points to estimate a plane using SVD decomposition for the linear system $Ax = t$. This gives us a plane.

$$ax + by + cz + d = 0$$

This plane gives us the depth for every point on it, irrespective of whether we had a depth for it previously from the stereo matching algorithms. This is how we fill up holes in the depth map. We can now trace a ray which starts from the camera and hits the approximate plane. We can compute the length of this line segment and this is the depth of this image point.

For a point $p = (x, y)$ in the image plane, we can compute the point of intersection between the plane and the ray and compute the depth as

$$depth = -d/(a * (p_y - c_y)/f_y + b * (p_x - c_x)/f_x + c)$$

For certain segments, because of measurement noise, poor segmentation, or non-flat surfaces, it is possible to end up with a bad estimate of the plane. We found a simple heuristic to prune these bad planes, $||Ax - t|| > threshold$. This helps to reduce noise in this approach for reconstruction.

With the above, we get an depth map that looks something like:

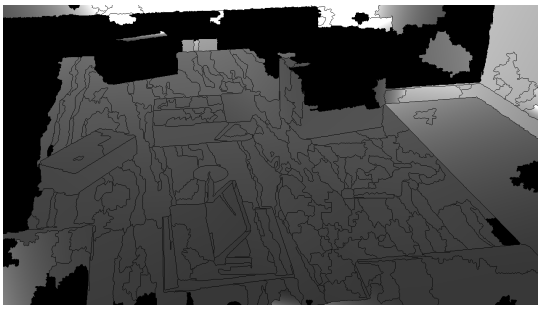


Figure 12: Planar interpolation of image segments

For areas where the planes couldn't be reconstructed, we fill those with the original depths, to get:

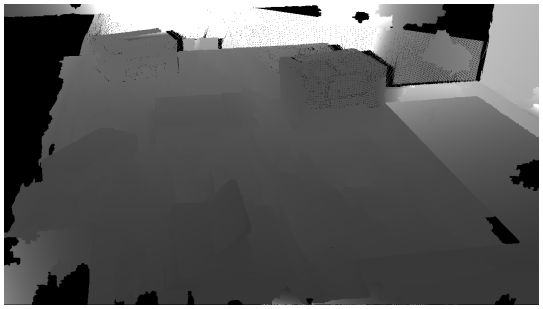


Figure 13: Filling in missing segments with original depths

4.2.8 Combining depth maps from multiple views

So far we've dealt with depth maps obtained from a single pair of rectified images. We observed that using a single pair of images doesn't give a full depth map when viewed from different camera angles. Fortunately, in a video sequence, we have many pairs of such images that can help improve results and fill in missing depths.

To find a depth map for the current frame, we take a number of nearby frames for which we have obtained high-quality depth maps. We deem a depth map to be high quality if it was obtained from a disparity map with a good baseline distance which is not too small and not too large. We then re-project the depth map from those frames into the current view. What we end up with is multiple depths per pixel.

To pick the desired depth, we sort the depths and pick the first value for which the value between it and the next value is no higher than 15 percent. This is a rudimentary way to pick the smallest z-value cluster. We pick the smallest z-value cluster to eliminate noise and ignore occluded objects.

With the above approach we obtain a depth map as seen in Fig 14. Notice that there is less noise than in previous depth maps, and more pixels have a depth value.

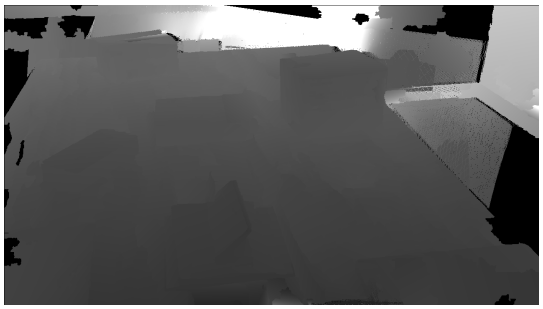


Figure 14: Depth map generated by merging depth maps from multiple viewpoints

4.3. Augmenting Video with 3D Objects

The last step of our system is augmenting the video with a 3D object. At this point we have estimated camera matrices and depth maps. The location of the object within the scene is determined manually, i.e. we take 3D coordinates in the object and translate them such that it is placed behind one of the boxes in the scene. In a real application, you can imagine a user interface which lets you drag and drop the object in the scene. We do not solve this problem here and focus on the mathematical aspects. The object (a bird) is a 3D object. It looks like a 2D blob because we did not add shading to it.

The 3D object is given as a mesh of triangles. An easy way to augment the frames of the video with this object is to apply the camera transform on each of the vertices and fill in triangles with the object texture. It gets slightly more complex when we want to handle occlusions.

Since we want to be as accurate as possible, we make sure that the triangles which make up the object are small enough. If they aren't, we can split each triangle into 3 smaller triangles using the centroid and the current vertices. A triangle is visible if all 3 vertices of the triangle are visible. With small enough triangles, this is a good approximation.

For each vertex $v = [x\ y\ z]$, which is a point in 3d world coordinates, we transform it into camera coordinates point $p = (x, y, z)$ using the camera transform.

$$(x, y, z) = [R\,T]\ v$$

The depth of the point is the z coordinate in camera coordinates (z_p) and the depth in the frame of the video is as computed using stereo correspondence and planar reconstruction (z_i). If the 3d object point is at a greater depth, $z_i < z_p$, it is hidden in the image, otherwise $z_i > z_p$, it is visible.

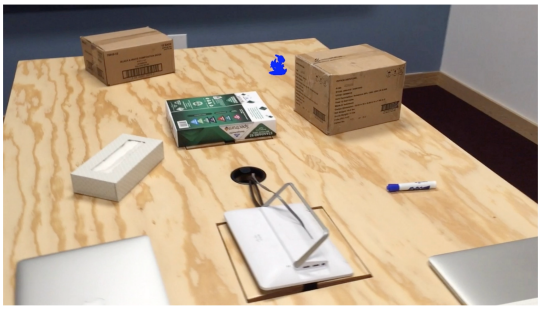


Figure 15: Bird is completely visible in this view

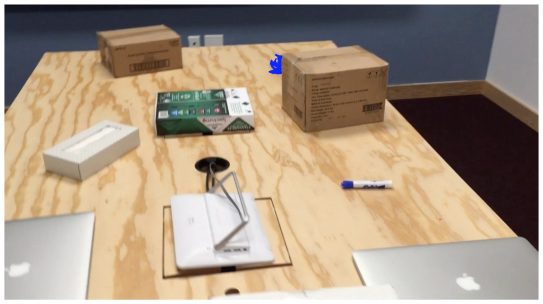


Figure 16: Bird is half hidden behind the box in this view

5. Evaluation

As discussed in section 4.1, we use existing algorithms and libraries to estimate camera parameters in each image of the video. Our main contribution lies in effectively connecting various computer vision algorithms to augment a video. The novelty in this paper is about refining depth maps using image segmentation and approximating a reconstruction using planes. Hence, we will skip evaluation for camera calibration and focus on our method of refining depth maps using planar reconstruction.

For easy evaluation of individual components of stereo matching algorithms, the computer vision group from Middlebury have designed a stand-alone flexible C++ implementation. [13] They also provide a collection of datasets and benchmark it against various state-of-art algorithms. The evaluation framework is flexible and supports easy additions of new algorithms. We integrate our methodology and compare the results with other stereo matching algorithms already implemented in the framework.

We describe the quality metrics we use for evaluating the performance of various stereo correspondence algorithms and the techniques we used for acquiring our image data sets and ground truth estimates. [18]

1. RMS (root-mean-squared) error, measured in disparity units, between the computed disparity map $d_C(x, y)$ and the ground truth map $d_T(x, y)$, i.e.,

$$R = \left(\frac{1}{N} \sum_{(x,y)} (|d_C(x, y) - d_T(x, y)|^2) \right)^{\frac{1}{2}}$$

   where N is the total number of pixels.

2. Percentage of bad matching pixels,

$$B = \frac{1}{N} \sum_{(x,y)} (|d_C(x, y) - d_T(x, y)| > \delta_d)$$

where δ_d is the disparity error tolerance.

There are various parameters which can be tweaked while evaluating stereo matching in the Middlebury framework. We use default values for most parameters except one parameter. *eval_bad_thresh* which controls thresholding to decide whether a pixel is a bad match or not was increased from 1.0 to 5.0. We found 1.0f was too strict and $> 90\%$ pixels were marked as bad pixels in most images. But a value of 5.0 gives good results for most images.

Figure 17a shows an image in the Middlebury evaluation dataset. Figure 17b shows the ground-truth disparity map of figure 17a. Disparity map of figure 17a is computed using Semi-Global Block Matching (SGBM) algorithm and is shown in figure 17c. Figure 17d shows image-segmentation on the original image figure 17a. Figure 17e shows refined disparity maps obtained by combining figure 17c and plane fitting on figure 17d. Computed disparity maps figure 17c and figure 17e are compared to the ground truth disparity map figure 17b and the above two metrics are computed. We also compare other algorithms from the Middlebury evaluation framework to our methodology.

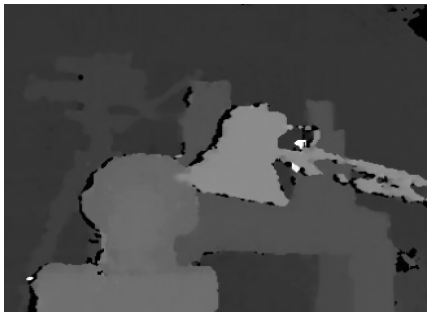
We can control the strictness of plane fitting to refine disparity maps by $norm\_thresh$. When the error of fitting a plane to points of a given image segment is greater than $norm\_thresh$ then we don't refine the disparity map and use the original disparity map. In figure 17e we can see how disparity maps are affected as $norm\_thresh$ value is increased. Figure 17e is bad compared to figure 17d in terms of both metrics and hence it is critical to tune this parameter correctly.

Sometimes refined disparity maps may not be better than the original disparity map because of bad image segmentation. Figure 18b is a better approximation to figure 18a compared to figure 18c. In figure 18c, plane fitting on bad image segmentation results in a weird disparity map. So it's important to control the quality of image segmentation to ensure refined disparity maps are better.

We compute disparity maps for 5 test images in the Middlebury evaluation dataset using our method as well as predefined algorithms in Middlebury. In the following table we report mean values of RMS error and percentage bad pixels. **normal-SGBM** refers to our SGBM implementation of disparity map. **planar-SGBM** refers to the filtered disparity map generated by fitting planes using image segmentation. [18] describes the other algorithms used for encoding. As seen in the table, planar-SGBM performs better than normal-SGBM in both metrics we defined earlier.

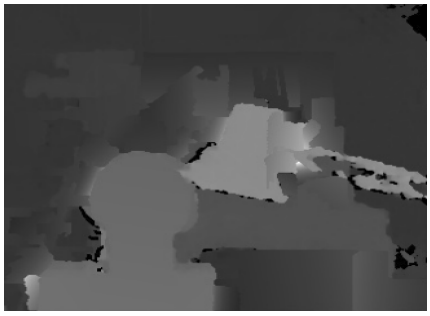
(a) Image to be evaluated from Middlebury dataset

(b) Ground truth disparity map

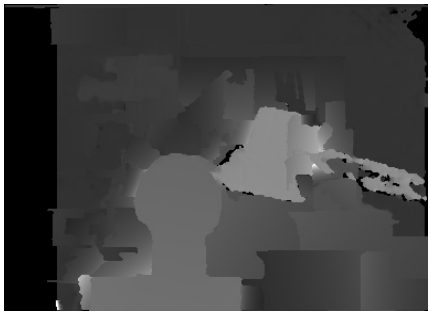


(c) Disparity map with Semi-Global Block Matching (SGBM)

(d) Image segmentation using watershed transform

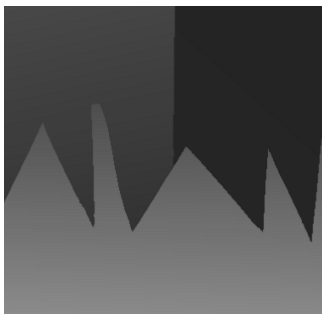


(e) Filtered disparity map combining SGBM and plane fitting on image segmentation

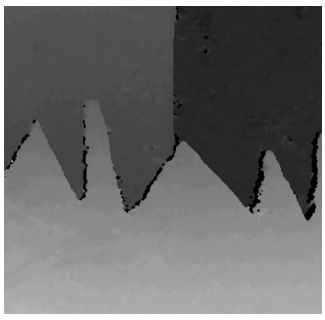


(f) Filtered disparity map combining SGBM and aggressive plane fitting on image segmentation

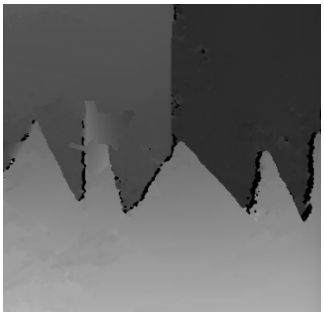
Figure 17: StereoMatch Evaluation using middlebury dataset



(a) True disparity map of sawtooth



(b) Disparity map by SGBM



(c) Refined disparity map using image segmentation

Figure 18: Another test image from Middlebury evaluation dataset

Algorithm	Mean RMS error	Mean Bad pixel ratio
SSD09bt05	1.559039	0.024049
SSD09t20	1.714662	0.030914
SADmf09bt05	1.733064	0.031358
SADmf09t02	2.4118	0.051564
SAD09t02	2.565821	0.058292
SADmf09t01	3.019650	0.084793
planar-SGBM	3.177850	0.071215
normal-SGBM	3.200869	0.072295
SAD09t01	3.717277	0.135821

6. Future Work

There are still a few things that we would have liked to try to improve our results.

- Implement state-of-the-art algorithms for stereo-matching and see how they perform with and without planar interpolation. We only had time to add planar interpolation on top of SGBM, but there are better algorithms out there.
- Look into using the DAISY descriptor instead of stereo-matching for dense 3D reconstruction. See: "A Fast Local Descriptor for Dense Matching" by Tola et. al. [5]
- Try different image segmentation approaches to remove spilling out of objects. We started looking into using the canny edge detector to seed the watershed algorithm. [14]
- Exploit the fact that we have a video instead of a set of photos to run SFM in real-time.
- Exploit the fact that we need a depth map of only a small region where the 3D object is projected into the scene (to make our algorithm real-time).
- Use a better rectification algorithm, as described in "A simple and efficient rectification method for general motion" [19]
- [17] describes how to add better depth map merging techniques - "Metric 3D Surface Reconstruction from Uncalibrated Image Sequences".
- Project more interesting geometry into the scene. Either by integrating with a ray-tracing library or using OpenGL and taking advantage of its built-in shaders.

7. Conclusion

In this paper, we proposed a system that takes a 3D object mesh and a video, and augments that video with the object. The system is able to estimate camera positions and generate depth maps for each frame (to support occlusions).

We used the Theia SFM library to estimate camera positions, and proposed a novel method to estimate depths in each frame. To estimate depths, we used a combination of image segmentation techniques (watershed algorithm), stereo matching (SGBM), and planar interpolation. When compared to stereo matching alone, the combination of these technique allowed us to improve depth map accuracy while at the same time significantly reducing noise and improving sharpness around object boundaries.

We were able to successfully project a 3D object back into a video. Our algorithm was fully autonomous, and did not require anything other than specifying the object position.

You can find our code by going to: `https://bitbucket.org/bajenov1/cs231a/`

You can find our final augmented video here: `https://www.youtube.com/watch?v=X37SP4Dihhg`

References

[1] Lee, J. C., and R. Dugan. "Google project tango."

[2] Perry, Simon. "Wikitude: Android app with augmented reality: Mind blowing." digital-lifestyles. info 23.10 (2008).

[3] Sweeney, Christopher, Tobias Hollerer, and Matthew Turk. "Theia: A Fast and Scalable Structure-from-Motion Library." Proceedings of the 23rd Annual ACM Conference on Multimedia Conference. ACM, 2015.

[4] Ravimal Bandara, Image Segmentation using Unsupervised Watershed Algorithm with an Over-segmentation Reduction Technique.

[5] Tola, Engin, Vincent Lepetit, and Pascal Fua. "A fast local descriptor for dense matching." Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on. IEEE, 2008.

[6] Mur-Artal, Raul, J. M. M. Montiel, and Juan D. Tardos. "ORB-SLAM: a versatile and accurate monocular SLAM system." Robotics, IEEE Transactions on 31.5 (2015): 1147-1163.

[7] Furukawa, Yasutaka, and Jean Ponce. "Accurate, dense, and robust multiview stereopsis." Pattern Analysis and Machine Intelligence, IEEE Transactions on 32.8 (2010): 1362-1376.

[8] Kundu, Abhijit, et al. "Joint semantic segmentation and 3d reconstruction from monocular video." Computer VisionECCV 2014. Springer International Publishing, 2014. 703-718.

[9] Bradski, Gary, and Adrian Kaehler. Learning OpenCV: Computer vision with the OpenCV library. " O'Reilly Media, Inc.", 2008.

[10] Wu, Changchang. "VisualSFM: A visual structure from motion system." (2011).

[11] Snavely, Noah. "Bundler: Structure from motion (SFM) for unordered image collections." Available online: phototour. cs. washington. edu/bundler/(accessed on 12 July 2013) (2010).

[12] Eisert, Peter. "Reconstruction of Volumetric 3D Models." 3D Videocommunication: Algorithms, Concepts and Real-Time Systems in Human Centred Communication (2005): 133-150.

[13] Scharstein, Damiel, and R. Szeliski. "Middlebury stereo datasets." 2014-04-06]. http://vision, middlebury, edu/stereo/data (2006).

[14] Canny, John. "A computational approach to edge detection." Pattern Analysis and Machine Intelligence, IEEE Transactions on 6 (1986): 679-698.

[15] Haris, Kostas, et al. "Hybrid image segmentation using watersheds and fast region merging." Image Processing, IEEE Transactions on 7.12 (1998): 1684-1699.

[16] Pollefeys, Marc, Reinhard Koch, and Luc Van Gool. "A simple and efficient rectification method for general motion." Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on. Vol. 1. IEEE, 1999.

[17] Pollefeys, Marc, et al. "Metric 3D surface reconstruction from uncalibrated image sequences." 3D Structure from Multiple Images of Large-Scale Environments. Springer Berlin Heidelberg, 1998. 139-154.

[18] Scharstein, Daniel, and Richard Szeliski. "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms." International journal of computer vision 47.1-3 (2002): 7-42.

[19] Pollefeys, Marc, Reinhard Koch, and Luc Van Gool. "A simple and efficient rectification method for general motion." Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on. Vol. 1. IEEE, 1999.