Multiple View Stereo Reconstruction

Andreas Molzer, Yu Wang, Yue Zhu, Sricharan Chiruvolu
Technical University of Munich
Department of Informatics

***Abstract*—In this work, we realize a stereo reconstruction on a multi-view stereo dataset and build up its 3D model.**

I. Introduction

Multiple View Stereo Reconstruction re-constructs 3D model from multiple RGB images with no depth information. These images can be either be captured simultaneously by different cameras or can be sequentially acquired via a moving camera. The reconstruction is done by generating a rectified image over the epipolar lines of two aligned close-posed images. For this, a block matching algorithm is performed onto these two images to find the corresponded pixel matches. A triangulation algorithm is later applied onto the corresponding pairs to compute the distance values along-with their correspondent 3D coordinates to generate the point clouds. Finally, the pointclouds computed from multiple corresponding stereo-image pairs are merged and aligned to build the final model.

II. Dataset

The dataset used is a set of 120 stereo images of a bunny model, resting on a table. The intrinsic the extrinsic matrices of all images are provided together with the images. The average variation of the rotation angles between two successive images are about 1 degree.

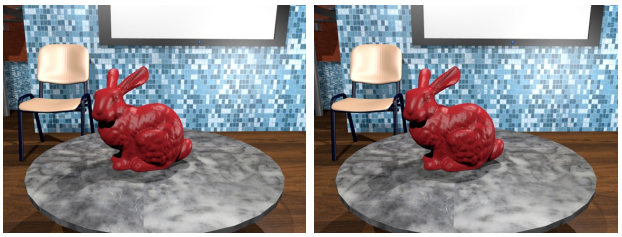

Fig. 1. two examples in the bunny dataset, which have a close pose and so that be able to do the stereo reconstruction

III. Our algorithm

The main algorithm is show in the Algorithm 1.

Algorithm 1 Main steps

```
Read images and intrinsic and extrinsic matrices;
for every close stereo-image pair do
    Rectify image pair;
    Computing matching pixel pairs using block matching with sum-squared-error loss;
    Triangulate pixel pairs to compute depth and compute 3D coordinates in real world system;
    Form a pointcloud;
end for
Merge and align pointclouds;
Write output mesh;
return Final mesh
```

A. Read images and intrinsic and extrinsic matrices

Each image is read into matrices of size 640*480 and 3 layers for RGB color. We also store a correspondent gray-scale image. Both images are associated with a 3*3 intrinsic matrix and a 4*4 extrinsic matrix.

B. Rectification

The rectification step aimed to project a pair of stereo images onto a common image plane. Normally for a horizontal stereo image pair, their horizontal epipolar lines would be on the same height. We applied the rectification process using the gray-scale images, then applied the same transformations onto the RGB images.

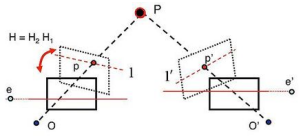

Fig. 2. two examples in the bunny dataset

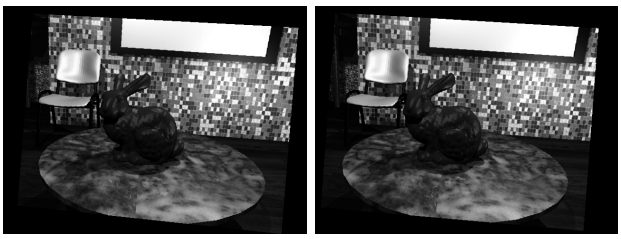

Fig. 3. two rectified images computed from those in Fig 1

The rectification step simplifies the problem of finding matching points between images, where we only need to compare the blocks in the same line instead of comparing every blocks in the image, so that largely reduce the possibility of having wrong matchings.

C. Matching

We tried four different method for matching blocks between two images. Starting with an own-implemented block-matching, we used a sum-square distances as the cost function. It realize a block matching of blocks within a square of 7*7-pixels in the same vertical height. To test its accuracy, an opencv class StereoBM is borrowed, which also does the block matching operation using its inner cost function, and we compared them. Both two matching gives noisy disparity maps. So that another opencv class StereoSGBM (semi-global block matching) is borrowed. Semi-global means the smoothness will also be considered while doing the matching, thus producing a smoother and more continuous distances than the former two matching methods. Finally, a patch matching method is implemented, but seems largely noisy compared to StereoSGBM , so we did not use it.

After finishing the matching, we compute the disparity map to see the matching results, which shows the distances between the corresponding pixel pairs.

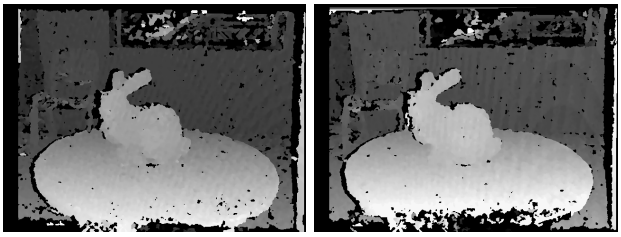

Fig. 4. two disparity maps corresponding to the former two rectified images

D. triangulation and 3D coordinates computation

We realize the triangulation using $distance = \frac{baseline*f}{disparity-baseline}$

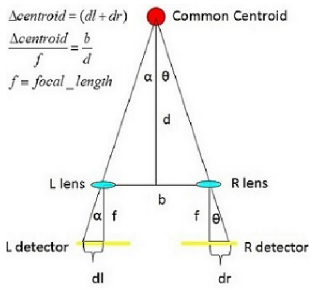

Fig. 5. triangulation of stereo image pairs

After the distance has been computed, the (x, y, distance) can be transformed into 3D coordinates by using the projection matrix $K(R|T)$

We have now a pointcloud for each stereo image pairs. We just put them together as the final pointcloud.

IV. Result

We finally get the meshes and pointcolouds in Fig 6:

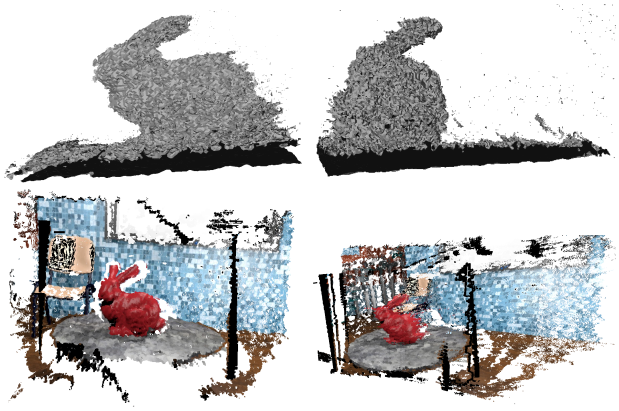

Fig. 6. Our result 3D Mesh and colorful pointclouds, Thus the boundaries are still not smooth

V. Conclusion

We were successful in reconstructing a 3D model from multiple stereo images. Although the process was clear and direct, we faced a few intermediate difficulties during the implementation as depicted below.

A. finding the right dataset

We started our experiments with "TempleSparseRing", a 16 images stereo dataset from Middlebury. The bottleneck was the large angles between each pair of poses, together with large black backgrounds that made our feature matching difficult.

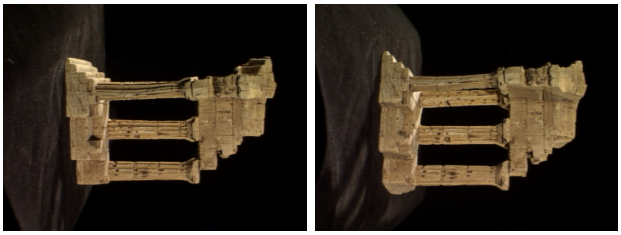

Fig. 7. Samples of two neighbor image pair from "TempleSparseRing" dataset, we can see a large angle differences between them

B. baseline size

Another weakness of the large angle between the pair of poses is its larger baseline, which we can't find the good correspondence and lead to a very noisy disparity map. However, when we use the small angle between the pair of poses on the bunny dataset (about 1 degree), The denominator in the formula in the triangulation step becomes very small, which makes the output distances value quite inaccurate. We finally use a 3 degrees rotation difference between each two stereo image pairs.

C. Opencv application

We have borrowed a few opencv functions from this famous library, but in many cases the documentations are not precise. This lead to quite a few confusion of coordinate systems and pixel indices which had to be resolved by trial and error.