**3D Reconstruction of Teddy and Castle**

**Introduction**

The objective of this work was to generate a 3-D model using multiple 2-D views of the object of interest. The 3-D reconstruction pipeline was used to generate point clouds and eventually to render 2 different 3-D models : Teddy Bear and Castle. For the reconstruction of the teddy bear, 16 2-D views were used where each of the view contained the image of the same teddy bear taken from the same camera where the angular difference between any 2 consecutive views was 22.5 degrees to capture all the surface information which would be utilized to generate the 3D model of the Teddy Bear. In similar fashion, 3-D model of the castle was generated using 19 2-D views where each of the view was captured using the same camera by rotating the castle model by approximately 19 degrees. The 2-D images were used as an input to the 3-D reconstruction pipeline to generate the final model. Figure placeholder reflects the reconstruction pipeline which was used to generate the model. In the first stage of the 3-D pipeline feature extraction was performed. The point correspondence between 2 images was performed next once we had the feature points. The point correspondences were refined to mitigate the effect of outlier and mismatched points. The best point correspondences were used to chain the multiple views and generate frames containing 3 consecutive views to generate the point view matrix or the measurement matrix. Using the point view matrix generated in the chaining segment of the pipeline, 3-D point cloud was generated. This point cloud served as an input to the renderer which was used to render the 3-D model of the object of interest.

**Feature Extraction**

The first module of the 3-D reconstruction pipeline is feature extraction. Feature extraction helps to keep track of points which are visible in different 2-D views which is essential for generating the final 3-D point cloud. From the 2-D views, Harris and Hessian features were extracted separately using vl\_sift and then Hessian and Harris features were combined together to create the consolidated feature set. Combining the feature set helps to increase the interest points for which point correspondences could be found and this would in turn help to create smoother and more robust 3-D point cloud. Harris feature detector could be theoretically used to extract both edges and corners but for our case the response of the Harris detector corresponding to corner is more useful as edges contains uninformative details which is not of any use in our 3-D reconstruction pipeline. In order to detect corners in the 2-D images, the response of the Harris corner detect should be positive and large which signifies that the both the eigen values of the structure tensor matrix or the second moment matrix is positive and large. Hessian feature are computed in more or less similar fashion, where the Hessian features are detected in different scale spaces and like Harris features, Hessian features are affine invariant. The difference between the Harris features and Hessian features is that the Harris features are computed with the help of structure tensor matrix while the Hessian features are computed using the Hessian matrix which is consists of second partial derivatives and the way the response of the detector is computed for the Hessian is the same as the way response is computed for Harris feature detector. Narrative about our implementation of Harris Detector

Figures Needed ( 1 castle data 1 teddy data )

Harris Features overlaid on an arbitrary image, Hessian Feature on the same image with probably different colour or in a different figure window.

Custom Harris points overlaid on an arbitrary image, Our implementation of Harris points overlaid on same image with different colour or in a different figure window.

**Point Correspondence and Outlier Mitigation**

The Harris and Hessian features are used to build the consolidated feature set which contains the descriptors and the coordinates of the features. With this consolidated feature set for each image of the 2D view, corresponding feature points are computed for 2 consecutive image sequence using the ubcmatch function. The most crude implementation of this point correspondence computation can be done by comparing the each feature descriptor of first image of the sequence to all the descriptors of the second image and then finding if any of the descriptor is a close match to the first image descriptor on the basis of a threshold. This has a time complexity of Nimage1xNimage2. The threshold of ubcmatch is set to 1.5 which is the default threshold of the function. This generates point correspondences between 2 consecutive images. The correspondences generated in this fashion might be susceptible to outliers and mismatches. The matches are passed on to the RANSAC algorithm which tries to estimate the best model the best inliers to refine the correspondences which would be used further to build the point view matrix. Outliers in the point correspondences can distort the 3D point cloud generation. The threshold for RANSAC is set as 10. From the point correspondences, 8 points are picked randomly and the fundamental matrix is computed and the Sampson distance between the original point and the transformed points are computed. If the distance is less than 10, the points are treated as inliers. This process is run iteratively to come up with the best model and the best inliers. The best inliers contains the refined matches which are relatively free from mismatches and other outliers. Using the best model which was estimated, we transform the images to generate corresponding points in one image and the resulting epipolar line in the other image.

Figure Needed (1 for teddy and 1 for castle)

2 arbitrary images with points and corresponding epipolar lines

Data Required : Number of correspondences using ubcmatch and the Number of Best inliers.

**Chaining and Structure from Motion**

With the best point correspondences we chain the images to generate the frames which contains 3 views. The column of the point view matrix contains the points and the rows comprise of views. A full column indicates that a point is present in all the views which isn’t very likely because of occlusions and camera movement. The last 2 chaining sequences are treated with special care as they overlap with the first 2 chaining sequence. We observe that for the teddy dataset the point-view matrix is quite dense while there is some sparsity in the castle point view matrix which induces artefacts in the 3-D point cloud estimation using affine structure from motion. The chaining process provides us with the point view matrix on which affine structure from motion is performed. The point view matrix which is the measurement matrix first normalised to centre the data about the origin. Then using the normalized datapoints we perform single value decomposition to generate the structure and the motion matrix. The transformation is not unique and there is a need to eliminate affine ambiguity. To eliminate affine, Cholesky decomposition is performed and the matrix C is obtained. The M and the S matrices are updated by multiplying C inverse and C matrix respectively. Now stitching of the point cloud is performed where the point cloud from all the views are generated iteratively and all unique points are added to the merged cloud. Once the stitching process is over, we get the 3-D point cloud of the model generated from the 2-D views using the affine structure from motion

Figure Needed (1 for teddy and 1 for castle)

3-D point cloud of teddy and 3-D point cloud of castle

**Rendering**

Narrative Pending