Semester project report: Multi-view 360 Stereo

Zhantao Deng
Electrical Engineering

September 9, 2019

1 Abstraction

In traditional multi-view stereo, images at arbitrary poses are used to estimate camera parameters and depth maps. Then, all depth maps are fused into a point cloud to represent scenes. This project addresses the same problem with 360 camera. But in this project, poses for 360 cameras are known, as they are manually deployed.

To deal with distortions of 360 images and estimate depth maps, cubic projection is implemented. It decomposes 360 images into 6 views which can be treated as regular images. Then patchmatching stereo[2] of the COLMAP[7, 8] is used to estimate depth. To improve the performance of depth estimation and leverage the rich textures captured by 360 cameras, a view selection method based on similarity and triangulation angle is implemented. Finally, to generate new views at arbitrary poses, a view synthesis algorithm is accomplished, where indices volumes and costs volumes are defined for pixel-wise selection and texture synthesis.

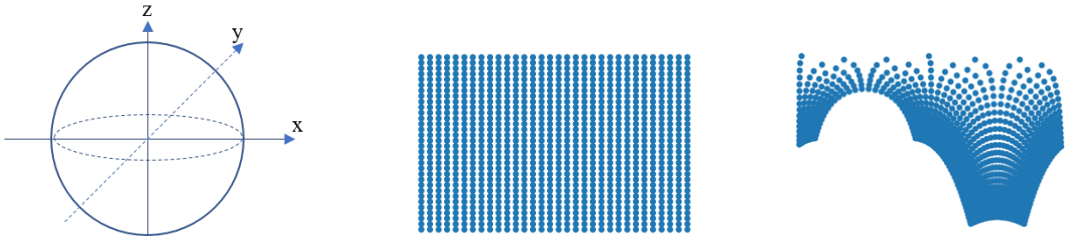
2 Introduction

2.1 Related works

In recent years, many dense reconstruction algorithms have been proposed, such as matching window[1], optical flow[10] and RGB-D mapping[3]. However, since these algorithms have special requirements on camera models, they do not perform well when applied to 360 images directly. The main reason is the strong distortions caused by the 360 camera model, as shown in Figure 1. The distortion is caused by the unevenly distributed pixels on 360 camera sphere as well as the projection to store and visualize 360 images. For example, the density of pixels around polar points is greater than that around equator so that when an equirectangular image is used to show a 360 image, the equirectangular projection introduces distortions. Besides, distortions become stronger when latitude increases.

There have been some works to solve this problem. To obtain robust keypoint matching

performance on cameras having strong radial distortion, researchers develop SIFT-like algorithms[5, 11] to deal with strong distortion while preserving the original invariance to scale and rotation. J. Masci et al.[6] use a similarity-preserving hash framework to map descriptors to Hamming space. In dense reconstruction, Sunghoon Im et al.[4] proposed a sphere sweeping algorithm for practical spherical panorama camera. Moreover, there are also some end-to-end learning based algorithms[12, 9].



(a) 360 camera model

(b) pixel distribution of regular image

(c) pixel distribution after projection

Figure 1: (a) camera model for 360 images. All pixels distribute on the sphere and can be represented by (θ, ϕ) where θ means polar angle from the north polar and ϕ means azimuthal angle from the negative y axis. (b) the blue dots represent pixels. As shown here, pixels distribute uniformly on regular images. (c) The result of projecting the regular image in (b) (assume its field of view is 90°) to the sphere in (a) at $\theta = 30°$ and visualizing the result in equirectangular form. We can find that the distribution of pixels is significantly changed and the distortion is serious.

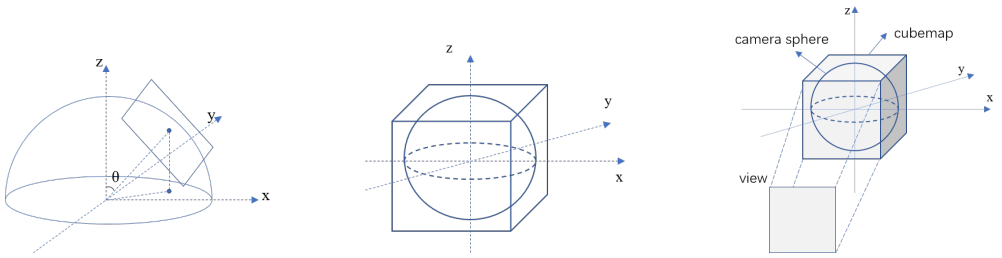
2.2 Our works

In this semester project, we implement a cubic maps based depth estimation method. As shown in Figure 2b, a 360 image is projected to a cube containing the camera sphere so that the 360 image can be decomposed to 6 views that can be treated as regular images. Thus the 360 image depth estimation problem is converted to depth estimation for the 6 views.

To estimate a dense depth map for each views, the patchmatching stereo algorithm integrated in the COLMAP[7, 8] is used, since it provides robust and efficient depth estimation. As 360 images contain rich information about the scene, in order to improve the quality of depth maps, we realized a view selection method. Given a reference view from the reference 360 image, it selects a source view from each source 360 images to make the similarity as well as the triangulation angle between the reference view and source views large enough. Experiments show that the view selection method improves the smoothness and accuracy of depth maps.

After depth estimation, we have depth maps for all views. Then, depth maps of views being projected from the same 360 images are grouped and are projected to 360 camera sphere to obtain 360 depth maps. Therefore, every 360 image has a 360 depth map. The next step is to synthesize new 360 images given these 360 images and 360 depth maps.

As long as there are at least one 360 image having depth map, we can synthesize new 360 images at arbitrary poses. However, with only one source 360 image and depth map, the synthesized 360 images have many invalid textures and holes. To improve the quality of synthesized 360 images, more 360 images are needed. In order to fuse textures from different 360 images, a multi-view synthesis cost volume is defined, which considers the costs from depth estimation, the distance between a 3D point and the ray pointing from camera center to the pixel corresponding to this 3D point, as well as the distance between a 3D point to the camera center. Each pixels in the synthesized view is chosen according to the synthesis costs volume. This part will be introduced in detail in section 7.



(a) tangent projection (b) cubic maps (c) camera sphere, cubic maps and view

Figure 2: (a) project a patch on the sphere to its tangent plane thus the distortions can be removed; (b)cubic maps/cubemaps are the 6 surfaces of a cube which contains the camera sphere as an inscribed sphere; (c) an example to show the relations between camera sphere, cubemap and views.

3 Definitions

3.1 Terminologies

In this project, *360 images* include 360 images, their local coordinates as well as corresponding camera parameters (extrinsic and intrinsic parameters). A *reference image* is the 360 image whose local coordinate is used as the world coordinate and its depth map will be estimated. All the rest 360 images are *source images*.

For a 360 image, its *cubemaps/cubic maps* are the 6 surfaces of a cube which contains the camera sphere as an inscribed sphere. Each of the 6 surfaces is named a *view*, as shown in Figure 2b and 2c. After projecting a 360 image to a cube, the 6 *views* can be treated as regular images taken by virtual regular cameras. Besides, a reference view is one of the 6 *views* of the *reference image* and a source view is one of the 6 *views* of a *source image*.

3.2 Coordinates

In this project, every 360 image has a local coordinate. As shown in Figure 1a, local coordinates are defined in the way that the X axis points to the right, the Y axis to the front (the center part of 360 images) and the Z axis to the top. This definition is also used for the world coordinate. If a 360 image is chosen as the reference image, its local coordinate becomes the world one. Differently, for a given *view*, the coordinate is defined as that X axis points to the right, the Y axis to the bottom, and thus the Z axis to the front, which can be seen in Figure 7.

When recording the pose of a 360 image, both rotation R and translation t are from world to local, i.e. $R = R_{w2l}$ and $t = t_{w2l}$, where the $w2l$ means from world to local. The world to local means that assuming $[R|t]$ to be the T_{w2l} matrix, given a point P_w in world coordinate, the corresponding local position P_l can be calculated by $P_l = T_{w2l} * P_w$. Similarly, the pose of a *view* is expressed as a rotation matrix R_{l2c} from local to the view as given in Figure 7.

4 Cubic maps

To decompose a 360 image into 6 cubic maps, we define some variables as follow:

- (θ, ϕ): the sphere coordinate of the tangent point
- $(\Delta\theta, \Delta\phi)$: the field of view along θ and ϕ, respectively.
- d the radius of the camera sphere.
- (r_w, r_h): the expected resolution of the cubic map.
- $(\theta_{i,j}, \phi_{i,j})$: for pixel (i, j) of the cubic map, the corresponding projection on the sphere.

Figure 3a and 3b shows the relations between these variables. Then the transformation can be written as:

$$\theta_{i,j} = \arccos(\frac{z_{i,j}}{\sqrt{d^2 + h_{i,j}^2 + w_{i,j}^2}})$$

$$\phi_{i,j} = \phi + \arctan(\frac{w_{i,j}}{d' + h_{i,j}})$$

where

$$h = 2 \times d \times \tan(\Delta\theta/2)$$

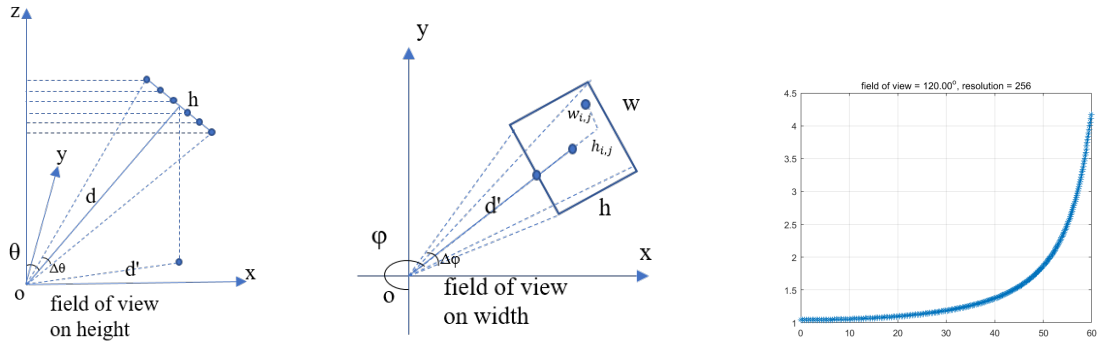
$$d' = d \times \sin(\theta)$$

$$h_{i,j} = \frac{h}{r_h} \times i$$

$$w_{i,j} = 2 \times d \times \tan(\frac{\Delta\phi}{2})$$

$$z_{i,j} = d \times \cos(\theta) + h_{i,j} \times (\theta)$$

Where $i = -\frac{r_h}{2} + k,\ k = 0, 1, 2, ..., r_h$ and $l = -\frac{r_w}{2} + k,\ k = 0, 1, 2, ..., r_w$



(a) viewing along with the tangent plane

(b) viewing from positive z-axis

(c) number of generated pixels at different angle

Figure 3: (a)(b)In both figures, there are several right triangles and thus the correspondence between pixels on the tangent plane and pixels on the sphere can be computed accurately. (c) The x labels are the angle between the optical axis and the ray pointing from the camera center to pixels. The y labels are the number of points generated from the same pixel on the 360 camera sphere. If y is greater than 1, the projection leads to upsampling.

During the projection, we project all pixels of cubemaps to the 360 camera sphere and use interpolation to obtain textures for these pixels. Then we copy these textures to corresponding pixels on cubemaps and get the 6 views. However, there are several problems.

Firstly, as interpolation is used to generate textures, what we obtain is not the real cubemaps but an interpolated one. Besides, as the Figure 3c demonstrates, when the angle between the tangent point and a pixel on the sphere is large, the pixel can be projected to more than one pixels on cubic maps. This is also shown by Figure 5c. For pixels close to image boundaries, their textures are generated by interpolating the same pixels, which leads to distortions on boundaries.

Besides, to estimate depth maps for 360 images, we have to project all 360 images to cubemaps first and then after estimating their depth, all depth maps have to be projected back to 360 camera sphere. The two projections are computationally expensive, especially on high resolution 360 images. Moreover, our algorithm performs poorly on scenes with few textures, since the patchmatching stereo relies on rich textures for accurate depth estimation.

5 Experiments for depth estimation with cubemaps

5.1 Algorithm

5.1.1 Patchmatching stereo of COLMAP

COLMAP[7, 8] is one of the state of the art large scale structure from motion system. In its dense reconstruction part, patchmatching stereo[2] is implemented on GPU which provides efficient and accurate depth estimation. During the dense reconstruction, COLMAP reads

images, corresponding poses and camera parameters from its workspace and then computes the photometric depth map for all images. Then, to remove outliers and make depth maps smooth, COLMAP computes a geometric depth map as well as a cost volume for each of the images. The geometric depth maps are what we expected to get and the cost volumes will be used in section 7, view synthesis part.

5.1.2 Depth estimation with cubemaps

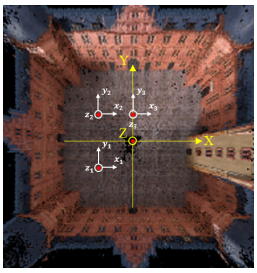
Given N 360 images $I_i, i = 1, 2, ..., N$ and their poses $T_i, i = 1, 2, ..., N$, we want to estimate the depth map $d_i, i = 1, 2, ..., N$ for each of them. To this end, each 360 image I_i is decomposed by the cubic projection to 6 views $v_i^j, j = 1, 2, ..., 6$. These views can be treated as regular images and thus patch-matching stereo in the COLMAP can be used to estimate their geometric depth maps $d_i^j, j = 1, 2, ..., 6$. After that, all views are projected back to the 360 camera sphere to obtain 360 depth maps d_i. This back projection is an inverse operation of the cubic projection. The only difference is that when projecting textures from 360 camera sphere to cubic maps, interpolation is used while the nearest neighbour is used for projecting depth maps from cubic maps to camera sphere. The pipeline is explained in the Algorithm 1.

Data: N 360 images I_i and their poses $T_i, i = 1, 2, ..., N$;
Result: the depth map of each of the 360 images $d_i, i = 1, 2, ..., N$;
1. Set the world coordinate by choosing a 360 image as the reference image;
2. decompose I_i to cubic maps $v_i^j, i = 1, 2, ..., N, j = 1, 2, ..., 6$;
3. group all views v_i^j according to their orientations as $G_k = \{v_i^k\}, i = 1, 2, ..., N$, e.g. $k = 1$ and $k = 3$ represents views looking at back and front, respectively;
4. organize workspaces for COLMAP;
5. execute the patchmatching stereo of COLMAP on $G_k, k = 1, 2, ..., 6$
6. obtain geometric depth maps $d_i^j, i = 1, 2, ..., N, j = 1, 2, ..., 6$ together with cost volumes (for view synthesis);
7. project depth maps d_i^j back to 360 camera sphere to obtain 360 depth maps d_i.

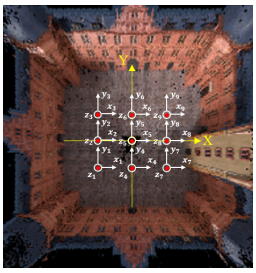
Algorithm 1: Depth estimation

5.2 Settings

As shown in Figure 4a, the scene is a castle where 3 360 cameras I_1,I_2,I_3 are deployed at (-4,-4,1), (-4,4,1) and (0,4,1) respectively. The 3 cameras have the same orientation as the world coordinate, so their rotation matrices are the identity matrix. Besides, all 360 cameras share the same intrinsic parameters including camera centres and focal lengths. The resolution is 512×1024 for 360 images and 512×512 for cubic maps. In addition, we only consider the 4 horizontal views for depth estimation as the top and bottom views are not as informative as horizontal ones. The second scene is the same as the first one but it has 9 360 cameras arranged to a 3 by 3 camera array, as given in Figure 4b. Figure 4c shows the ground truth depth map for the camera at (0, 4 ,1) whose estimated depth map is discussed in Figure 6.



(a) simulation scene 1 and camera poses



(b) simulation scene 2 and camera poses

(c) ground truth depth map

Figure 4: (a) shows a bird-eye view of the scene and the deployment of 360 cameras. The yellow coordinate represents the world coordinate while the small white coordinates represent local coordinates of the 3 360 images. The red dot means Z-axis is perpendicular to the paper and points to the reader. (b) has the same settings as (a) but there are 9 views in total and they are deployed as a 3×3 camera array. (c) is the ground truth depth map for the 360 image at (0,4,1).

5.3 Results

With the above settings, according to formulations in section 4, we can decompose every 360 image into 6 views without overlaps. One example is shown in Figure 5a and 5b. For depth estimation, the estimated depth maps for the 360 image at (0,4,1) in scene 1 and 2, are provided in Figure6a. Figure 6b includes the corresponding error maps. As can be found in the two figures, both the two depth maps are smooth at the center, while there are a lot of invalid depth and holes on the ground and near the boundaries of cubic maps. Besides, when we use 9 360 images, the depth map is more complete than that of using 3 360 images.

(a) an example of 360 images

(b) cubic maps for the 360 image

(c) example of interpolation

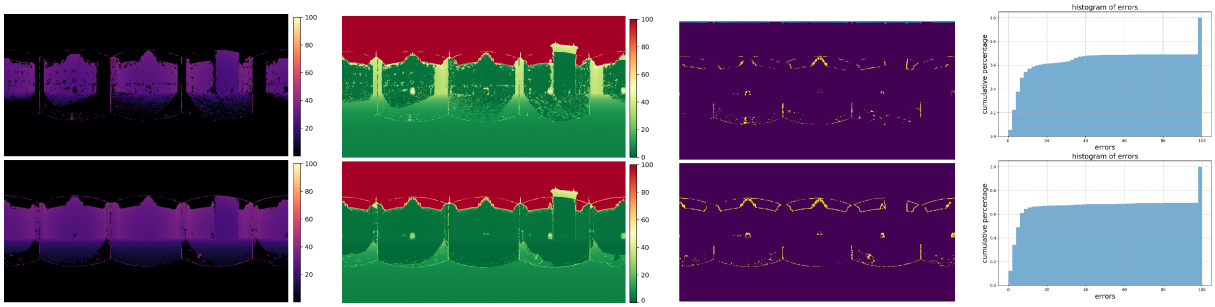
Figure 5: (a) An example of 360 images. (b) The cubemaps of the 360 image in (a). From left to right the horizontal 4 views are: the back, the left, the front and the right while the rest two views are the top and the bottom.(c) The problem caused by interpolation near image boundary.

The RMSE for depth map is 9.69 for scene 1 and 10.91 for scene 2. The result is weird as it indicates that using more images increases errors. This weird result is caused by the way of calculating RMSE. Since RMSE in this project only considers pixels with valid depth, a depth map can have very small RMSE even if it has only one tiny patch of depth as long

as these estimated depth values are accurate. In contrast, if a depth map is complete but having some outliers, its RMSE could be larger than the one with only a patch of depth.

This is exactly the case for scene 1 and 2. As using 3 360 images can not estimate small structures such as windows and doors in detail, most of the small structures do not have depth. In contrast, when 9 360 images are used, depth can be extracted from small structures. However, those depth values could have large errors, as textures for small structures are not rich enough. Therefore, the RMSE of using 3 images is smaller than that of using 9 images. Figure 6c shows the pixels with large errors and it can be seen that the depth map from 9 images has more details although the errors for these details are large. After removing these pixels, the filtered RMSE is 3.55 for scene 1 and 3.46 for scene 2. Figure6d shows the cumulative distribution of errors and we can find that using 9 360 images reduced the ratio of pixels having large errors by increasing the ratio of pixels having small errors.

In order to further improve the quality of depth maps, more information is required. For example, we can use more 360 images to do the estimation or make views share more overlaps, as the more overlaps two views share, the better the quality. For regular image, it is impossible to increase the overlapping of two given image as their poses are fixed. For 360 images, as they have 360° field of view, they can be projected to cubic maps in any direction with any field of view. Therefore, it makes sense to have a view selection for omnidirectional images.



(a) estimated 360 depth maps

(b) two error maps

(c) pixels with large depth errors

(d) cumulative errors distributions

Figure 6: The four figures on the top are for scene 1 in Figure4a and the bottom 3 figures are for scene 2 in Figure4b. (a) the two depth maps estimated from scene 1 and 2 respectively. As shown in the color bar, the darker of the color the smaller of the depth. Black means no estimation. (b) the corresponding two error maps. (c) the corresponding two maps of pixels with large errors. Yellow dots represent these pixels. (d) cumulative percentages of errors of the two depth maps.

6 View selection

6.1 Problem formalizing

In order to leverage all information captured by 360 images to improve the quality of depth maps, a view selection method is implemented. Given a reference view v_{ref} and N source 360 images $I_i, i = 1, 2, ..., N$, it selects a source view $v_{src,i}$ from I_i and calculate a view selection score s_i for each $v_{ref} - v_{src,i}$ view pair. Then all selected source views $v_{src,i}$ will be sorted by their scores s_i. If users want to use $M(M \leq N)$ views for reconstruction, the best M views will be chosen. Algorithm 2 demonstrates pseudo codes for view selection.

The view selection score considers to two criteria: overlaps between the reference view and source view should be as many as possible; the triangulation angle between the reference view and the source view should be greater than a certain threshold. The more overlaps and the larger angle a view pair has, the higher the score it obtains. The reason for considering triangulation angles is that images sharing small viewpoint changes always have high texture similarity as long as the illumination changes are not very serious. However, those images are not good candidates for reconstruction as they do not have enough baseline for depth estimation.

6.2 Selection procedure

6.2.1 Initialization

Given a reference view v_{ref} and a source 360 image I, to iteratively select a good source view v_{src} from I_i for reconstruction, we have to chose a initial source view $v^0_{src,i}$. Under the assumption that views looking at the same direction are more likely to have enough overlaps, $v^0_{src,i}$ is set to be the view looking at the same direction as v_{ref}, as shown in Figure 8a. Since the poses of cameras are known, the normal vector of $v^0_{src,i}$ can be calculated as

$$n^0_{src} = R_{w2s} * R^{-1}_{w2r} * R^{-1}_{r2c} * n_{ref}$$

where the $n_{ref} = [0 \quad 0 \quad 1]$, the normal vector of v_{ref}. R_{r2c}, R_{w2r} and R_{w2s} are shown in the Figure 7. Then we can convert the normal vector to polar coordinate

$$\theta_0 = \arccos(z/||n^0_{src}||) \quad \phi_0 = \arctan(\frac{y}{x})$$

where x,y and z are the three elements of n^0_{src}. With the $[\theta_0, \phi_0]$, $v^0_{src,i}$ can be projected from the source 360 image I by the projection introduced in section 4. When the view selection is disabled, $v^0_{src,i}$ is used for dense reconstruction. If view selection is enabled, $v^0_{src,i}$ will be optimized iteratively as algorithm 2 shows.

6.2.2 View selection score

Similar to the COLMAP[7, 8], sparse features are used to select views where overlaps are measured by the number of matched features. In order to make the selected view shares

more overlaps with the reference view while keeping the triangulation angle to be large, the score is defined as

$$score = 2 - \frac{(min(\bar{\alpha}, \alpha) - \bar{\alpha})^2}{\bar{\alpha}^2} - \frac{(min(\bar{n}, n) - \bar{n})^2}{\bar{n}^2}$$

where n is the number of matched features and α is the angle between the optical axis of v_{ref} and $v_{src,i}$, as can be seen in the Figure 8b. $\bar{\alpha}$ and $\bar{n}$ are two prior threshold. This function assigns low scores to source views for which the angle and the number of matches are below the two priori thresholds. With view selection scores, source views are sorted and only the top several views are chosen for dense reconstruction.

6.2.3 Sampling new views

During view selection, there are three cases for a source view $v^j_{src,i}$: if $v^j_{src,i}$ shares enough overlaps with v_{ref}, it will be returned and used for reconstruction; if $v^j_{src,i}$ has many (but not enough) overlaps with v_{ref}, it will be turned toward the reference view according to the two centroids c_{ref} and c_{src} of feature points; if there are few overlaps between $v^j_{src,i}$ and v_{ref}, a new source view$v^{j+1}_{src,i}$ will be picked randomly around $v^j_{src,i}$ with Gaussian distribution.

For the second case, given the centroids c_{ref} and c_{src} of feature points and the orientation $[\theta^j, phi^j]$ of $v^j_{src,i}$, we would like to compute the orientation for the new source view. Firstly, we compute the polar coordinates $[\theta_{ref}, \phi_{ref}]$ and $[\theta_{src}, \phi_{src}]$ for c_{ref} and c_{src} by the projection in the section 4, respectively. Then we calculate the change of angles

$$\Delta\theta_{src2ref} = \theta_{ref} - \theta_{src} \quad \Delta\phi_{src2ref} = \phi_{ref} - \phi_{src}$$

Thus, the orientation of the new source view is

$$\theta^{j+1} = \theta^j + \lambda\Delta\theta_{src2ref} \quad \phi^{j+1} = \phi^j + \lambda\Delta\phi_{src2ref}$$

The λ is the changing rate and it is set to be 1 in this project. With the $[\theta^{j+1}, \phi^{j+1}]$, $v^{j+1}_{src,i}$ can be projected from the source 360 image by the projection in section 4.

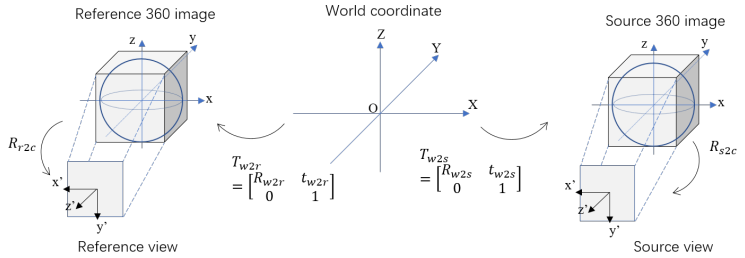
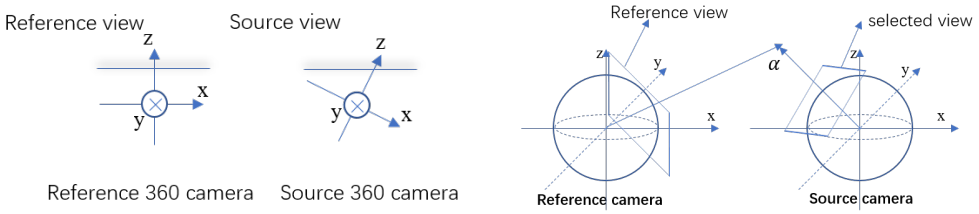


Figure 7: The coordinate transformation between reference image and source image. R_{w2r} and R_{w2s} mean the rotation from world coordinate to reference image and source image, respectively, and so as the t_{w2r} and t_{w2s} which mean translation vectors. R_{r2c} and R_{s2c} are the rotation matrix from reference and source coordinates to view coordinates.



(a) initial pose for views election

(b) triangulation angle α between reference view and source view

Figure 8: (a) The two lines with shadows are the reference view and the initial source view, respectively. The two crosses at the center of the two coordinates denote that the y-axis is perpendicular to the paper and points into the paper. (b) The α is the angle for view selection score, which is between the optical axes of the reference view and source view.

Data: a reference view v_{ref}, a source 360 image I_i, $maxiters$, $minfeature$, $maxdist$;
Result: the selected source view $v_{src,i}$ or None if there is no source view selected;

1. set $j = 0$ and initialize $v^0_{src,i}, \theta_0, \phi_0$ as introduced in 6.2.1;
while $j < maxiters$ **do**
  2. features in v_{ref} and $v^j_{src,i}$ are detected and matched;
  3. count the number of matches n;
  if $n < minfeature$ **then**
    4. randomly pick $[\theta_{j+1}, \phi_{j+1}]$ and project a new source view $v^{j+1}_{src,i}$ as the third case in 6.2.3;
    5. $j = j + 1$;
    continue ;
  else
    6. calculate centroids c_{ref} and c_{src} of features in v_{ref} and $v^j_{src,i}$ separately;
    7. calculate the vector $vec = c_{src} - c_{ref}$;
    if $||vec|| < maxdist$ **then**
      8. compute the angle α between the optical axes of v_{ref} and $v^j_{src,i}$;
      9. use n and α to compute the view selection score s_i, as in 6.2.2;
      10.RETURN $v^j_{src,i}$ and s_i;
    else
      11. compute the orientation $[\theta_{j+1}, \phi_{j+1}]$ of the new source view $v^{j+1}_{src,i}$ as the second case in 6.2.3;
      12. project $v^{j+1}_{src,i}$ from the source image at $[\theta_{j+1}, \phi_{j+1}]$;
      13. $j = j + 1$;
    end
  end
end
14. There is no valid source view found so RETURN None;

Algorithm 2: View selection

6.3 Experiments

6.3.1 Settings

In this experiment, we use the same settings as Figure 4a in 5.2. The only difference is that the view selection is implemented and the patch matching stereo is executed on selected views.

6.3.2 Results

Figure 9 demonstrates the differences of views before and after view selection. It can be observed that, comparing to original views, selected views tend to share more overlaps with the reference view by rotating the tangent planes toward the reference view.

Besides, the 360 depth maps estimated with and without view selection are shown in the Figure 10. It can be viewed that, after implementing view selection, the depth maps become more complete and the outliers at boundaries of cubic maps disappear. Error maps are given in Figure 10b. RMSE is reduced from 9.69 (no selection) to 9.12. If pixels with large errors (introduced in Figure 6c) are filtered out, the RMSE becomes 3.55 (no selection) and 3.24. The RMES shows that view selection improve the quality of depth maps. Similarly, it can be observed from the Figure10c that view selection increases the percentage of pixels with small errors, comparing the depth map estimated without view selection.

Although the view selection algorithm improves the quality of depth maps, there is an important drawback. As can be seen in the algorithm 2, sparse features are the core of this algorithm. Since sparse features rely on rich textures, once there are not many textures in a scene, the view selection might diverge or converge to a wrong view.

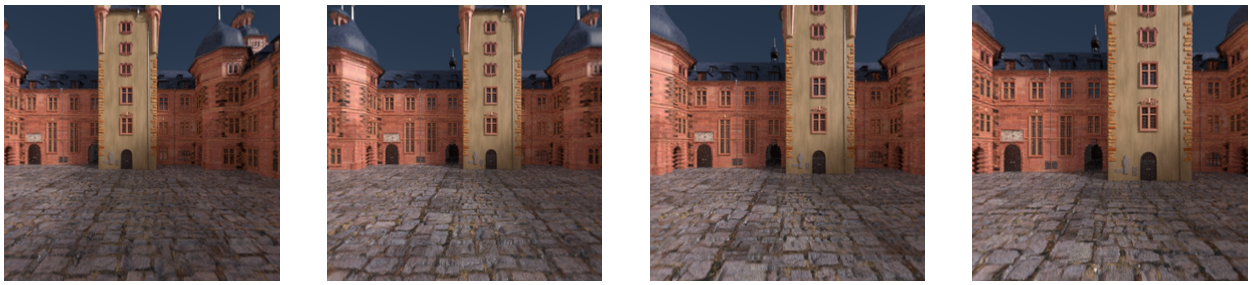
7 View synthesis

7.1 Definitions

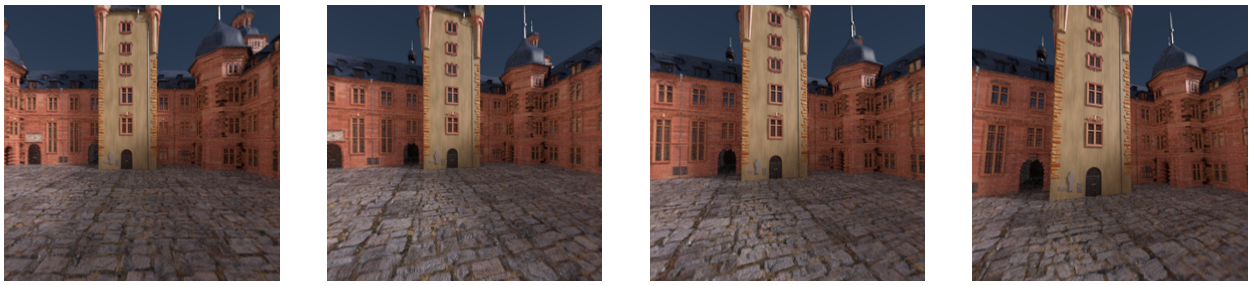
In view synthesis, all 360 images used for synthesis are named *source view* while the synthesized 360 image is called the *target view*. When we project pixels from *source views* to the *target view*, there could be lots of pixels falling into the same pixel of the *target view*. Those pixels are called *candidates* for the pixel of the *target view*.

7.2 Problem formalizing

After obtaining good 360 depth maps, we can synthesize new 360 images. Given N source views $I_i, i = 1, 2, ..., N$ together with their poses $[R_{src,i}|t_{src,i}]$ and depth maps d_i, we would like to synthesize a target view T whose pose $[R_T|t_T]$ and camera parameters are available. To this end, firstly, all pixels of I_i with valid depth are projected to T according to $[R_{src,i}|t_{src,i}]$ and $[R_T|t_T]$. Then synthesis costs are computed to evaluate the costs of each candidate of a pixel of T. Finally, with different criteria, different candidates are chosen to synthesize T.



(a) reference view (b) original src view 1 (c) original src view 2 (d) original src view 3



(e) reference view (f) selected src view 1 (g) selected src view 2 (h) selected src view 3

Figure 9: Above are the reference view and the three original/selected views. We can find that after the view selection, all selected views are toward the reference view which is helpful for dense reconstruction.

The pipeline can be found in the Algorithm3.

Data: N source view I_i together with their poses $[R_{src,i}|t_{src,i}]$ and depth maps $d_i, i = 1, 2, ..., N$; the pose $[R_T|t_T]$ of the target view T;

Result: Synthesized T and its depth map;

1. Project all pixel of I_i with valid depth to T, as in 7.3.1;
2. Compute cost volume for pixels of T, as in 7.3.2;
3. Aggregate the cost volume to get the best candidate for pixel of T, as in 7.3.3;
4. Generate textures and depth map for T.

Algorithm 3: View synthesis

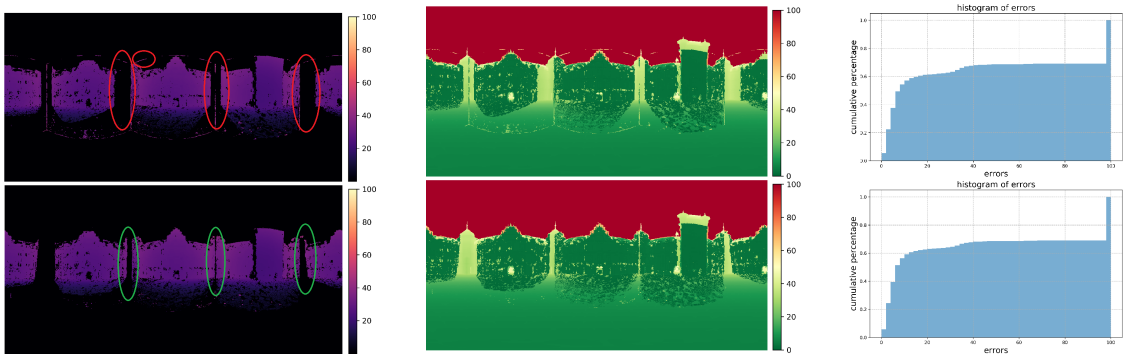
7.3 Synthesis procedure

7.3.1 Projection

Given the poses of N source views $[R_{src,i}, t_{src,i}], i = 1, 2, ..., N$ and the pose of the target view $[R_T, t_T]$, the projection of a pixel $P_{src,i}$ from source view i to the target view can be written as:

$$P_T = R_T * R_{src,i}^{-1} * P_{src,i} + t_T - R_T * R_{src,i}^{-1} * t_{src,i}$$

where P_T represents the 3D point $P_{src,i}$ under the coordinates of the target view. R_T and R_{src} are the rotation matrix from world to the target view and to source views. t_T and t_{src} are the corresponding translation vectors from world to the target view and source view.



(a) depth map from the 3 360 images with and without view selection

(b) corresponding error maps

(c) cumulative errors distributions

Figure 10: The top 3 figures are for the scene 1 in Figure4a without view selection while the bottom 3 figures have view selection. (a) Comparing to the depth map estimated without view selection (the top figure), using view selection (the bottom figure) makes the depth map more complete and more smooth. (b) Corresponding error maps for depth maps in (a).

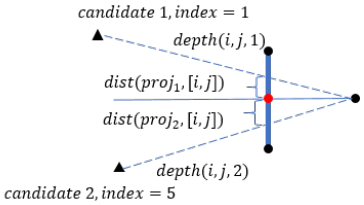
7.3.2 Synthesis costs

After projecting all pixels with vaild depth from source views to the target view as in 7.3.1, a cost volume is created to make the final textures and depth map as good as possible. For every pixel on the target view, the cost volume stores the costs of its candidates. The way of computing the costs is given below which is a weighted sum of three terms: the cost from the COLMAP, the distance between a 3D point and the ray pointing from camera center to the pixel corresponding to the 3D point as well as the normalized distance between a 3D point to the camera center.

$$Cost(i,j,k) = W1 * cost_{colmap}(k) + W2 * dist(proj_k, [i,j]) + W3 * \frac{depth(i,j,k) - \overline{depth}(i,j)}{\sigma_{depth}(i,j)}$$

where the $Cost(i,j,k)$ means the cost of the k-th candidate for the pixel at i-th row and j-th column of the target view. The $cost_{colmap}(k)$ means the cost given by COLMAP. $dist(proj_k, [i,j])$ is the distance between the projected point and the pixel center, which is proportion to the distance between the projected 3D points and rays pointing from the camera center to the pixel of the target view. the third part is the distance between the projected 3D point and the camera center of the target view. $\overline{depth}(i,j)$ is the average depth of all candidates for pixel (i,j) and $\sigma_{depth}(i,j)$ is the variance. $W1, W2, W3$ are predefined weights to aggregate the three terms. Figure 11a displays these terms.

With the cost volume, candidates for pixels of the target view can be sorted. After that, there come two choices to convert costs to textures: synthesizing view with the best candidates or conducting further filtering to make depth and textures more consistent.



(a) cost terms

1.1	0.8	1.01	1.9
0.4	0.7	0.59	0.8
0.75	0.33	1.03	1.08
1.4	1.51	0.87	0.66

(b) costs volume

1	4	2	2
2	1	4	3
1	4	2	2
4	1	3	2

(c) indices volume

Figure 11: (a)A 2D example to show the terms considered in synthesis costs. The vertical line segment represents a pixel of the target view and the red dot is the center of the pixel. The two triangles mean 3D points projected from source views to the target view and the right dot is the camera center. The *index* indicates the source view from which a candidate comes. (b) an 1 layer example of costs volume. 1 layer means each pixel has exactly one candidate. Each element represents the cost computed as in 7.3.2 (c) a 1 layer example of indices volume and each element represents the index of the source view that the candidate is from.

7.3.3 Cost/indices volume to texture

To get the best candidate for each pixel, winner-takes-all is the most straight-forward method, i.e. for every pixel of the target view, only the candidate with the lowest cost is kept. However, as we can see from Figure 12e and 13e, it increases the inconsistency of the depth map. But we can also find that the inconsistency is similar to noise such as salt and pepper noise. Thus we can treat the depth map as a regular image and filter the inconsistency out.

As the distortion of 360 images is very strong, filtering directly on depth space or texture space could make the synthesized view worse. Inspired by the fact that in general, it is more likely to have smooth depth and good texture if all pixels in a patch of the target view come from the same source image, we implement median filter on the indices map. Indices volume is similar to the cost volume but elements in the volume are indices indicating the source view from which a candidate comes. As shown in Figure 11c. Therefore, filters such as median filter and total variation filter can be used.

Since the filtered indices could be invalid, e.g. a pixel of the target view is not observed by the source view indicated by the filtered index, a post-process is necessary. For example, we filter out the invalid indices by setting them to -1 if the corresponding pixels are not observed by anyone of the source views. Differently, if the pixels are observed by at least one source view, we replace these invalid indices with their original indices.

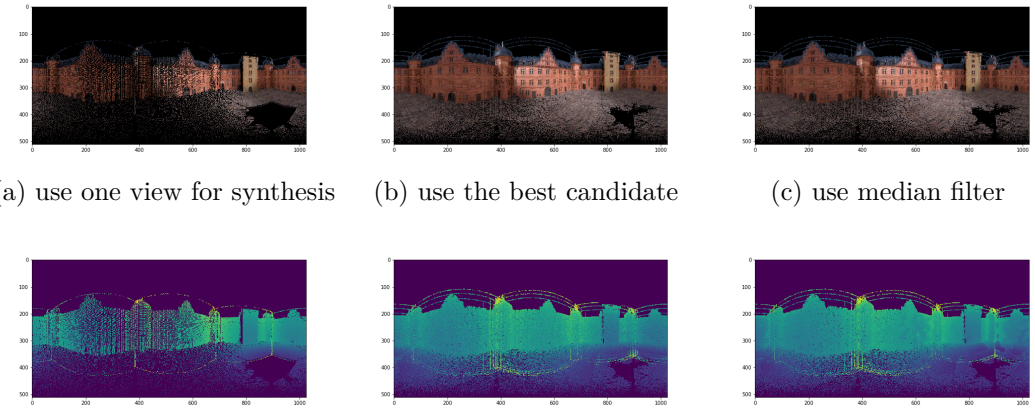
Finally, for each pixel of the target view, there is at most one final candidate, the one with lowest cost or the one given by the filter. Copying the texture of the candidate to the pixel synthesizes the target view.

7.4 Experiments

We accomplish two experiments: one has view selection while the other does not. The settings are the same as scene 2 shown in Figure 4b and the view to be synthesized is at [-8,8,8].

Figure 12 shows the synthesized view together with the corresponding depth map. They are generated from depth maps estimated in 5, i.e. there is no view selection. From the figure, we can see that using more views for synthesis increases the RMSE from 10.71 to 12.60. But after the median filtering, the RMSE decreases to 12.45. Thus, median filtering reduces depth noises. Besides, when generating textures from only one view, there are a lot of holes as the synthesized view is far away from any of the existing views. In fact, only 26.2% of the pixels have valid textures and depth. In contrast, when we leverage all views, no matter using the best candidate or using the median filter, the target view is more complete and around 48.24% of pixels are filled. The PSNR of the three synthesized views are very close, around 30.65

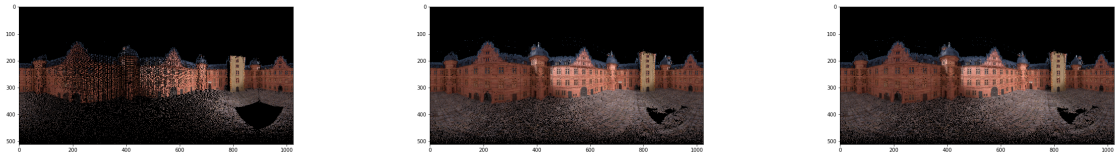
Views in Figure 13 are generated from depth maps of 6.3, so depth maps are estimated with view selection. Comparing corresponding results in Figure 13 and 12, we can find that, with the help of view selection, the synthesis view has fewer outliers, e.g. the wrong textures from the boundary of cubic maps are much less than those without view selection, more details around roofs and on grounds and better consistency on depth maps. Quantitatively, the RMSE for depth maps of all the three synthesized views are below 10, better than the depth without view selection; similarly, the PSNR and completeness of textures remain at the same level as textures without view selection.



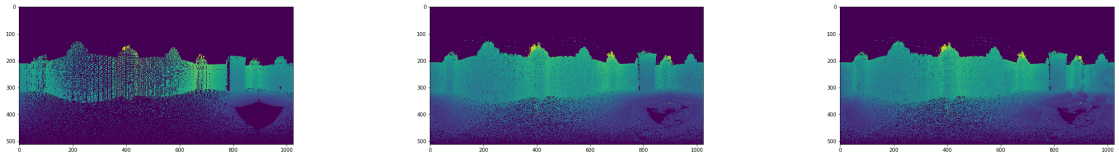
(a) use one view for synthesis (b) use the best candidate (c) use median filter

(d) depth map of (a) (e) depth map of (b) (f) depth map of (c)

Figure 12: Results of view synthesis on depth maps estimated without view selection. From left to right, the RMSE of depth maps are 10.71, 12.60 and 12.45; the PSNR of textures are 30.65, 30.63 and 30.63; the completeness of textures are 26.20%, 48.24% and 48.24%



(a) use one view for synthesis (b) use the best candidate (c) use median filter



(d) depth map of (a) (e) depth map of (b) (f) depth map of (c)

Figure 13: Results of view synthesis on depth maps estimated with view selection. From left to right, the RMSE of depth maps are 9.66, 9.990 and 9.89; the PSNR of textures are 30.68, 30.62 and 30.62. the completeness of textures are 26.21%, 48.60% and 48.60%

8 Conclusion

In this semester project, cubic maps and patchmatching stereo in COLMAP are used for 360 image depth estimation. To improve the performance, a similarity and triangulation angle based view selection method is implemented. Finally, to synthesize smooth depth map and good textures for 360 camera at any pose, a view synthesis algorithm is realized.

However, there are three main drawbacks. During the depth estimation, we have to project 360 images to cubic maps with interpolation and project cubic depth maps back to 360 camera sphere with nearest neighbour. But interpolation and nearest neighbour introduce errors. In addition, it is computationally expensive to do the forward and backward projection especially on high resolution 360 images, which can be avoided if we estimate depth directly from 360 images.

The second problem is that we rely on the patchmatching stereo[2], to estimate geometric depth which requires all photometric depth maps which is time-consuming. In the future, it would be better to implement geometric filtering by ourselves.

Finally, as we use sparse features detection algorithm to select view, once there are not many textures in a scene, the view selection might diverge or converge to a wrong view. Therefore, it would be better to explore some other view selection algorithms such as learning based methods, because they consider semantic information rather than low-level feature points.

References

[1] S. Adhyapak, N. Kehtarnavaz, and M. Nadin. "Stereo matching via selective multiple windows". In: *Journal of Electronic Imaging* 16.1 (2007), p. 013012.

[2] M. Bleyer. "PatchMatch Stereo - Stereo Matching with Slanted Support Windows". In: *British Machine Vision Conference*. 2011.

[3] P. Henry et al. "RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments". In: *The International Journal of Robotics Research* 31.5 (2012), pp. 647–663.

[4] S. Im et al. "All-around depth from small motion with a spherical panoramic camera". In: *European Conference on Computer Vision*. Springer. 2016, pp. 156–172.

[5] M. Lourenco, J.P. Barreto, and F. Vasconcelos. "sRD-SIFT: Keypoint detection and matching in images with radial distortion". In: *IEEE Transactions on Robotics* 28.3 (2012), pp. 752–760.

[6] J. Masci et al. "Descriptor learning for omnidirectional image matching". In: *Registration and Recognition in Images and Videos*. Springer, 2014, pp. 49–62.

[7] J.L. Schönberger and J. Frahm. "Structure-from-Motion Revisited". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[8] J.L. Schönberger et al. "Pixelwise View Selection for Unstructured Multi-View Stereo". In: *European Conference on Computer Vision (ECCV)*. 2016.

[9] K. Tateno et al. "Cnn-slam: Real-time dense monocular slam with learned depth prediction". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 6243–6252.

[10] L. Valgaerts et al. "Dense versus sparse approaches for estimating the fundamental matrix". In: *International Journal of Computer Vision* 96.2 (2012), pp. 212–234.

[11] E. Wang et al. "Tri-SIFT: A Triangulation-Based Detection and Matching Algorithm for Fish-Eye Images". In: *Information* 9.12 (2018), p. 299.

[12] N. Zioulis et al. "OmniDepth: Dense Depth Estimation for Indoors Spherical Panoramas". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 448–465.