Texture Mapping Indoor Environments with Imperfect Camera Localization

Anonymous 3DIMPVT submission

Paper ID ****

Abstract

Automated 3D modeling of building interiors is useful in applications such as virtual reality and environment mapping. Applying realistic textures to these models is an important step in generating accurate visualizations of data gathered by modeling systems. Unfortunately, the localization of cameras in such systems often suffer from inaccuracies, resulting in visible discontinuties when different images are projected adjacently onto a plane for texturing. Previous approaches at minimizing these discontinuities do not robustly handle a wide range of camera locations and angles, and often suffer from error accumulation when stitching together multiple images. We propose two approaches for reducing discontinuities during texture mapping, one to robustly accomodate images of all orientations, and one to take advantage of optimal situations where images have uniform orientation.

1. Introduction

Three-dimensional modeling of indoor environments has a variety of applications such as training and simulation for disaster management, virtual heritage conservation, and mapping of hazardous sites. Manual construction of these digital models can be time consuming, and as such, automated 3D site modeling has garnered much interest in recent years.

The aim of this paper is to present a solution for texture mapping the 3D models generated by indoor modeling systems, with specific attention given to a human-operated system with higher localization errors and greater variance in camera locations.

The paper is organized as follows. Section 2 provides an overview of the backpack modeling system and data processing pipeline from which input data and examples used throughout this paper originate. Section 3 describes the general problem of texture mapping and examines two simple mapping approaches and their problems. Section 4 demonstrates two previous attempts at localization refinement, and demonstrates their inadequacies for our datasets. Section 5 presents our proposed approach to texture mapping, combining an improved localization refinement process with two methods for texture mapping based on the quality of images available. Section 6 contains results and conclusions.

2. Backpack Modeling System

The first step in automated 3D modeling is the physical scanning of the environment's geometry. An indoor modeling system must be able to calculate camera locations within an environment while simulatenously reconstructing the 3D structure of the environment itself. This is known as the simultaneous localization and mapping (SLAM) problem, and is generally solved by taking readings from laser range scanners, cameras, and inertial measurement units (IMUs) at multiple locations within the environment.

Mounting such devices on a human-carried platform provides unique advantages over vehicular-based systems in terms of agility and portability. Unfortunately, human-operated systems also suffer from a lack of automation and stability, resulting in much higher data variance and localization error. As a result, common methods for texture mapping generally produce poor results, as later shown in sections 3 and 4. Before discussing how to overcome these challenges, we will first provide an overview of the backpack modeling system from which our test data was obtained.

2.1. Data Acquisition Hardware

The backpack-mounted modeling system that captured our data contains five 2D laser range scanners, two cameras, an orientation sensor, and an IMU. The laser scanners are mounted orthogonally and have a 30-meter range and a 270° field of view. The two cameras are equipped with fisheye lenses, reaching an approximately 180° field of view, and are mounted with one facing left and one facing right. These cameras take images at the rate of 5 Hz. The orientation sensor provides orientation parameters at a rate of 180 Hz. The IMU provides highly accurate measurements of all 6 DOF at 200 Hz, and is used as a ground truth reference for localization processes.

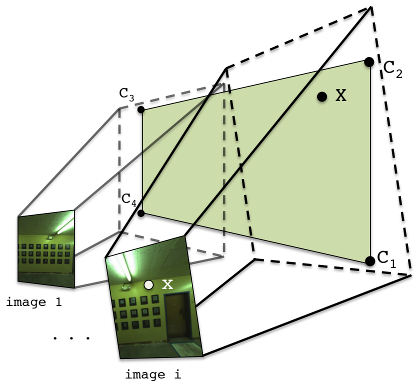

Figure 1: Planes are specified in 3D space by four corners C_1 to C_4. Images are related to each plane through the camera matrices $P_{1..M}$.

With the backpack actively scanning, the human operator wearing it takes great care to walk a path such that every wall in the desired indoor environment is passed and scanned lengthwise at least once.

2.2. Environment Reconstruction

Using data gathered by the onboard sensors and multiple localization and loop-closure algorithms, the backpack is first localized over its data collection period, and a 3D point cloud of the surrounding environment is constructed based on the laser scanner readings relative to the backpack [2]. Approximate normal vectors for each point in the point cloud are then calculated by gathering neighboring points within a small radius and processing them through principal component analysis. These normal vectors allow for the classification and grouping of adjacent points into structures such as walls, ceilings, floors, and staircases. A RANSAC algorithm is then employed to fit polygonal planes to these structured groupings of points, resulting in a fully planar model [7]. This model, consisting of multiple 2D polygonal planes in 3D space, along with the set of images captured by the backpack's camera, can be considered the input to our texture mapping problem.

3. Simple Texture Mapping

In all subsequent sections, we will discuss the process of texture mapping a single plane, as the texturing of each of our planes is independent and can be completed in parallel.

The geometry of the texture mapping process for a plane is shown in figure 1. As described in the previous section, we are provided with a set of M images with which we must texture our target plane. Each image has a camera matrix P_i for $i = 1..M$, which translates a 3D point in the world coordinate system to a 2D point or pixel in image i's coordinates. If the 3D world point is not contained in the image, the 2D point will simply be outside of the image boundaries. A camera matrix P_i is composed of the camera's intrinsic parameters, such as focal length and image center, as well as extrinsic parameters which specify the rotation and translation of the camera's position in 3D world coordinates at the time that image i was taken. These extrinsic parameters are determined by the backpack hardware and localization algorithms mentioned in Section 2. A point X on the plane in 3D space can be related to its corresponding pixel x in image i through the following equation:

$$x = project(P_i X)$$

where

$$X = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \text{ and } project(X) = \begin{pmatrix} x/z \\ y/z \end{pmatrix}$$

For the sake of simplicity, we treat all planes as rectangles by generating minimum bounding boxes for them. Since our final textures will be stored as standard rectangular images anyway, we can simply leave the area between plane boundary and bounding box untextured, or crop it out as needed. A plane to be textured is thus defined by a bounding box with corners C_i in world coordinates and a normal vector indicating the front facing side of the plane. Our goal is to texture this plane using images captured by the backpack, while eliminating any visual discontinuities or seams that would suggest that the plane's texture was not composed of a single continuous image.

3.1. Direct Mapping

Ignoring the fact that the camera matrices $P_{1..M}$ are inaccurate, we can texture the plane by discretizing it into small square tiles, generally about 5 pixels across, and picking an image to texture each tile with. Since we are provided with far more images than necessary to cover our plane, this process is a simple way of assigning texturess to locations on our plane. We choose to work with rectangular units to ensure that borders between any two distinct images in our final texture will be either horizontal or vertical. Since most environmental features inside buildings are horizontal or vertical, any seams in our texture will intersect them miniimally and be less noticeable.

In order to select an image for texturing tile t, we must first gather a list of candidate images that contain all four of its corners, which we can quickly check by projecting t into each image and using the projection method above. Furthermore, each candidate image must have been taken at a time when its camera had a clear line-of-sight to t, which can be calculated using standard ray-polygon intersection tests between the camera location, the center of t, and other planes, all in world coordinates.

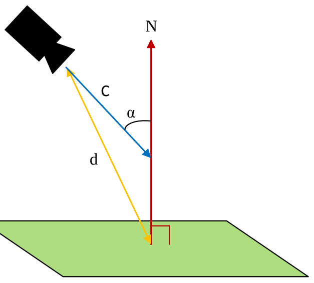

Figure 2: For each tile, we find the image that minimizes d and α by maximizing the following equation: $(1/d \cdot (-1 \cdot C) \cdot N)$

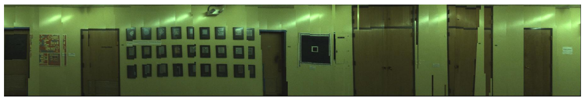

Figure 3: The result of direct texture mapping based on locally optimized textures.

Once we have a list of candidate images for t, we must define a scoring function in order to compare images and objectively select the best one. Since camera localization errors compound over distance, we wish to minimize the distance between cameras used for texturing and our plane. Additionally, we desire images that project squarely onto the plane, maximizing the resolution and amount of useful texture available in their projections.

A scoring function to maximize these two criteria is shown in figure 2.

As Figure 3 demonstrates, this approach leads to the best texture for each tile independently, but overall results in many image boundaries with abrupt discontinuities, due to significant misalignment between images.

3.2. Mapping with Caching

Since discontinuities occur where adjacent tiles select different images that do not match up well, it makes sense to take into account image choices made by neighboring tiles while selecting the best image for a tile. By using the same image across tile boundaries, we eliminate the discontinuity altogether. If this is not possible, using very similar images will result in less noticeable discontinuities.

Similar to a caching mechanism, we select the best image for a tile t by searching through two subsets of images for a good candidate, before searching through the entire set. The first subset of images is the set of images selected by adjacent tiles that have already been processed. We must first check which images can map to t, and then of those, we make a choice according to the same scoring function in figure 2. Rather than blindly reusing this image, we ensure it meets a predefined score threshold to be considered a good image. If no good image is found, we then check our second subset of images, which consists of images that were taken near the images in the first subset, both spatially and temporally. These images are not the same as the ones used for neighboring tiles, but they were taken at a similar location and time, suggesting that their localization and projection will be very similar. Again, if no good image is found according to a threshold, we then must search the entire set of candidate images.

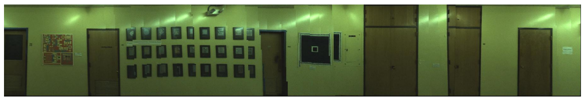

Figure 4: The result of adding a caching system to locally optimized textures.

The results of this caching approach are shown in figure 4. Discontinuities have been reduced overall, but the amount of remaining seams suggests that image selection alone cannot produce seamless textures. Camera matrices, or the image projections themselves will have to be adjusted in order to reliably generate clean textures.

4. Existing Approaches to Image-Aligned Texture Mapping

In order to produce photorealistic texture mapping, either camera matrices need to be refined such that their localization is pixel accurate, resulting in a perfect mapping, or image stitching techniques need to be applied to provide this illusion. Before examining these approaches, we first obtain a set of images to work with.

Rather than perform camera or image adjustments across many thousands of images, we choose to work with the more limited set of images corresponding to those chosen by the direct mapping approach, without caching. This set of images constitutes a good candidate set for generating a final seamless texture since it meets three important criteria. First, this set of images contains at least one image that covers each tile on our plane. Thus, unless extreme changes in localization occur, we can ensure that there will be no holes in our final texture. Second, since images are all selected according to the same scoring function in figure 2, we know that our images are all taken at as much of a head-on angle as possible and should project onto the plane in similar ways. Third, as a side result of the scoring function, selected images are only good candidates for the tiles near their center of projection. Thus, there should be plenty of overlap between selected images, allowing for some degree of shifting without resulting in holes, as well as area for blending between them. With this set of images, we now examine two approaches towards refining and combining their projections, before demonstrating our own.

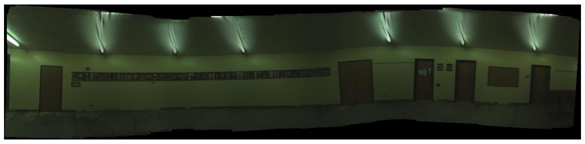
Figure 5: Image mosaicing.

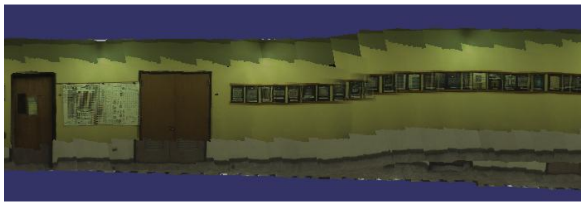
Figure 6: The graph-based localization refinement algorithm from [11] suffers from the problem of compounding errors.

4.1. Image Mosaicing

When images of a plane are taken from arbitrary overlapping positions, they are related by homography [4]. Thus, existing homography-based image mosaicing algorithms are applicable [1]. However, errors can compound when long chains of images are mosaiced together using these approaches. For example, a pixel in the nth image in the chain must be translated into the first image's coordinates by multiplying by the 3×3 matrix $H_1 H_2 H_3 ... H_n$. Any error in one of these homography matrices is propagated to all further images until the chain is broken. For some chains of images this can happen almost immediately due to erroneous correspondence matches and the resulting image mosaic is grossly misshapen.

Figure 5 shows the output of the AutoStitch software package which does homography-based image mosaicing. This plane is nearly a best-case scenerio with many features spread uniformly across it. Even so, the mosaicing produces errors that causes straight lines to appear as waves on the plane. This image was generated after careful hand tuning as well. Many planes that had fewer features simply failed outright. This leads to the conclusion that image mosaicing is not a robust enough solution for reliably texture mapping our dataset.

4.2. Image-Based 3D Localization Refinement

Another approach is to refine the camera matrices using image correspondences to guide the process. Each image's camera matrix has 6 degrees of freedom that can be adjusted. Previous work on this problem attempted to refine camera matrices by solving a non-linear optimization problem [5]. This process is specific to the backpack system which generated our dataset, as it must be run during backpack localization[5, 2]. Unfortunately, this approach suffers from a similar error propagation problem shown in Figure 6. In our new approach, we choose also to refine the placement of images using image correspondences. However, we do so in two dimensions on the plane whereas this previous work did so over all 6 degrees of freedom. Refining in two dimensions on the plane is less flexible in that it does not address projection errors, however, it provides significant benefits while avoiding the error propagation problem.

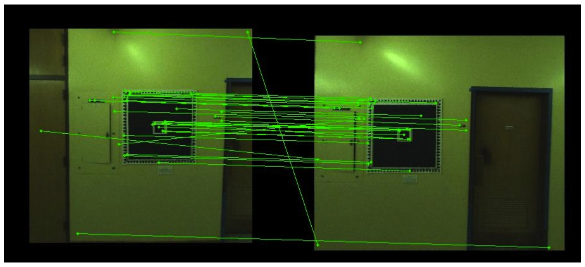
Figure 7: SIFT feature matches between overlapping images.

5. Proposed Approach

Our proposed approach towards texture mapping is a two-step process. First, with the same input set of images as described in section 4, we perform image rotation and shifting in order to maximize SIFT feature matches between images. We then project these images onto our plane, applying textures either with the tile caching method from section 3.2, or with the more specialized method described in 5.4.

5.1. Image Projection and Rotation

Our localization approach begins with the projection of all our images onto separate copies of our plane, such that no projected data is covered up and lost. This is done in the same way as the approaches in section 3. We then perform rotations on these projections, as adjacent images with different orientations will result in strong discontinuities.

These rotations are accomplished by using Hough transforms, which detect the presence and orientation of linear features in our images. Rather than try and match the orientation of such features in each image, we simply apply rotations such that the strongest near-vertical features are made completely vertical. This is effective for indoor modeling, since strong features in indoor scenes usually consist of parallel vertical lines corresponding to doors, wall panels, rectangular frames, etc. If features in the environment are not vertical, or are not parallel to eachother, this step should be skipped.

5.2. Robust SIFT Feature Matching using RANSAC

Our next step is to try and fix misalignment between overlapping images. We do this by first searching for corresponding points between all pairs of overlapping images using SIFT feature matching [6]. An illustration of this is given in Figure 7. The SIFT matches allow us to determine x and y distances between two images on the plane for each set of features, which will allow us to reevaluate where each image should be projected.

Since indoor environments often contain repetitive features however, such as floor tiles or doors, we need to ensure that our SIFT-based distances are reliable. In order to diminish the effect of incorrect matches and outliers, the RANSAC framework [3] is used for a robust estimate of the true x and y distances between the two images. The RANSAC framework attempts to build a consensus among the distances between every pair of SIFT matches, and tries to generate the horizontal and vertical distance between them while ignoring the influence of outliers that would skew the results. The framework handles the consensus-building machinery, and only requires that two functions be specified: the fitting function and the distance function. These functions are called for random subsets of the SIFT matches until the best set of inliers is found. For this application, the fitting function simply finds the average distance between matches. If the matches are exactly correct and the image is frontal and planar then the distances for various SIFT feature matches should be the same. Our distance function for a pair of points is the difference between those points' SIFT match distance and the average distance computed by the fitting function. We specified a 10 pixel outlier threshold to the framework. This means that a SIFT match is labeled as an outlier if its horizontal or vertical distance is not within 10 pixels of the average distance computed by the fitting function.

5.2.1 Refining Image Positions using Least Squares

There are a total of M^2 possible pairs of images, though we only generate distances between images that overlap at SIFT feature points. Given these distances and the original image location estimates, we can solve a least squares problem ($\min_\beta ||\beta X - y||_2^2$) to estimate the correct location of the images on the plane. The vector β of unknowns represent the correct x and y locations of each image on the plane from $1 \dots M$. The optimal x and y locations are calculated in the same way, so we will only consider the x locations here:

$$\beta = \begin{pmatrix} x_1, & x_2, & x_3, & \cdots & x_{M-1}, & x_M \end{pmatrix}$$

The matrix X is constructed with one row for each pair of images with measured distances produced by the SIFT matching stage. A row in the matrix has a -1 and 1 in the columns corresponding to the two images in the pair. For example, the matrix below indicates that we generated a SIFT-based distance between images 1 and 2, images 1 and 3, images 2 and 3, etc.

$$X = \begin{pmatrix} -1 & 1 & 0 & \cdots & 0 & 0 \\ -1 & 0 & 1 & \cdots & 0 & 0 \\ 0 & -1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & -1 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 0 \end{pmatrix}$$

If only relative distances between images are included then there is no way to determine the absolute location of any of the images and the matrix becomes rank deficient. To fix this we choose the first image to serve as the anchor for the rest, meaning all the absolute distances are based on its original location. This is done by adding a row with a 1 in the first column and the rest zeros.

Finally, the observation vector y is constructed using the SIFT-based distances generated earlier in the matching stage. The distances are denoted as $d_1 \dots d_N$ for N SIFT-based distances. The last element in the observation vector is the location of the first image determined by its original localization.

$$y^T = \begin{pmatrix} d_{1,2}, & d_{1,3}, & d_{2,3}, & \dots & d_{N-2,N-1}, & d_{N-1,N}, & x_1 \end{pmatrix}$$

The β that minimizes $||\beta X - y||_2^2$ results in a set of image locations on the plane that best honors all the SIFT-based distance measurements between images. In practice there are often cases where there is a break in the chain of images, meaning that no SIFT matches were found between one segment of the plane and another. In this case we add rows to the X matrix and observations to the y vector that contain the original x and y distance estimates generated by the localization algorithm. Another way to do this is to add rows for all neighboring pairs of images and solve a weighted least squares problem where the SIFT distances are given a higher weight i.e. 1, and the distances generated by the localization algorithm are given a smaller weight i.e. 0.01.

After completing this same process for the y dimension as well, and making the resultant shifts, our image projections should overlap and match eachother with far greater accuracy.

5.3. Relocalized Mapping with Caching

Now that our images have been relocalized for much greater accuracy relative to eachother, we can revisit the cached mapping approach from section 3.2. In figure 8, we see the wall from section 3.2, texture mapped using the same caching method after our localization refinement process.

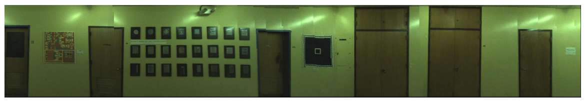

Figure 8: Localization refinement results in signifcantly fewer discontinuities in the final texture.

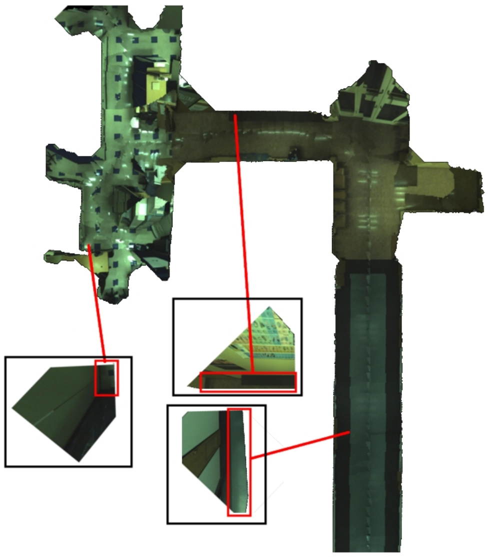

Figure 9: The tiling method ensures that only the most optimal portions of each image are used. This allows it to successfully handle all manner of plane and image geometry, as shown in this floor texture.

It should be evident that our localization adjustments resulted in a major improvement. A great strength of our tiling approach for image selection is that it extracts the areas with good camera angles from each image, and thus can handle images of all types. As demonstrated on the floor plane in figure 9, we can run this process even with extremely poor image projections, where the amount of quality texture for each image is limited. Additionally, since our plane is discretized, we can handle arbitrarily complex plane geometry as well.

Given that the backpack system's hardware and the data collection process strive for optimal wall texturing however, we can aim for a less conservative approach when texturing walls, and simply fall back to this more robust approach as needed.

5.4. Texture Mapping with Seam Minimization

Since we now have higher confidence in the quality of image boundaries, we explore an alternate method for texture mapping that greatly reduces seams in cases where we have an abundance of optimal images. The cached tile approach used previously selected images based on individual and neighboring quality of image selections, in an effort to reduce seams between tiles. In a more optimal case, as the backpack system provides for walls, images are all taken from close distances and have near-rectangular projections and thus less deviation between scores according to the scoring function in figure 2. Thus, we can reason that rather than selecting the set of best images, since all images are near in quality, we should instead select the best set of images, such that the selection together results in the cleanest final texture. We will accomplish this by using entire images where possible, defining a cost function between images, and seeking to minimize total cost.

5.5. Occlusion Masking

Before we proceed with this approach, we need to ensure that our images contain only content that should be mapped onto the target plane in question. The tiling approach used previously checked occlusion for each tile as it was being textured, but we plan on using entire images, so we need to perform occlusion checks over the entirety of each image, so we know which areas are available for texture mapping.

Fortunately, by virtue of our indoor environments, the vast majority of surface geometry is either horizontal or vertical, with high amounts of right angles. This means that after masking out occluded areas, our image projections will remain largely rectangular. We can thus be efficient by recursively splitting each image into rectangular pieces, and performing the same occlusion checks used in the tiling process where needed. To actually occlude out rectangles, we simply remove their texture, as we will ensure that untextured areas are never chosen for texture mapping.

5.6. Inter-Image Cost Functions

In order to objectively decide which set of images results in the cleanest texture, we need a cost function such that we can evaluate the visibiilty of seams between images in our set. A straightforward cost function that accomplishes this is the sum of squared pixel differences in overlapping regions between all pairs of images. Minimizing this cost function encourages image boundaries to occur in featureless areas, such as bare walls, as well as in areas where images match extremely well.

Another possible cost function is overall edge energy, i.e. the sum of the smoothed gradient over seams. Minimizing this encourages image boundaries to be placed in featureless areas even more than the first cost function. For the results shown throughout this paper, the first cost function was used, as the regular occurrence of featureless areas was not guaranteed in our datasets.

5.7. Image Selection

Now that we have a cost function defined, we mechanically select the set of images for which the overall cost func-

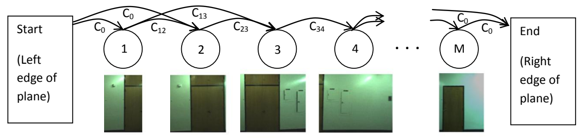

Figure 10: DAG construction for the image selection process.

tion is minimized. Since we aim to cover the entirety of our plane, our problem is to minimally cover a polygon(our plane), using other polygons of arbitrary geometry (our image projections), with the added constraint of minimizing our cost function between chosen images. This is a complex problem, though we can take a number of steps to simplify it. Returning again to the optimality of our situation when texturing walls, we can make a quick simplification for the sake of efficiency. Given that our wall-texture candidate images were all taken from a head-on angle, and assuming only minor rotations were made during localization refinement, we can reason that their projections onto the plane are approximately rectangular. By discarding the minor excess texture and cropping them all to be rectangular, our problem becomes the conceptually simpler problem of filling a polygon with rectangles, such that the sum of all edge costs between each pair of rectangles is minimal. We thus also retain axis-aligned image boundaries, along with their advantages explained in section 3.

By taking two more shortcut we can simplify our problem down even further. Since this approach is already assuming optimal images, we know it will be applied on walls, which are the focus of the backpack modeling system. For wall planes, which are nearly always rectangular, care is taken so that images contain the entirety of their floor-to-ceiling range, and thus we do not have wall images such that one should be projected vertically above the other. In essence, we need only to ensure horizontal coverage of our planes, as our images provide full vertical coverage themselves. We can thus construct a Directed Acyclic Graph (DAG) from the images, with edge costs defined by our cost function above, and solve a simple shortest path problem to find an optimal subset of images with regard to the cost functions.

Figure 10 demonstrates the construction of a DAG from overlapping images of a long hallway. Images are sorted by horizontal location left to right, and become nodes in a graph. Directed edges are placed in the graph from left to right between images that overlap. The weights of these edges are determined by the cost functions discussed previously. Next, we add two artificial nodes, one start node representing the left border of the plane, and one end node representing the right border of the plane. The left artificial node has directed edges with equal cost C_0 to all images

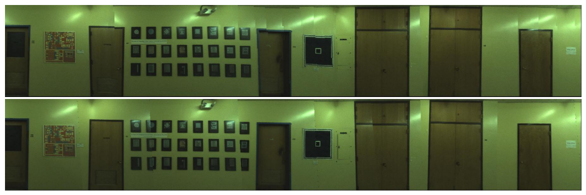

Figure 11: The tile caching approach is presented above the seam minimization approach.

that meet the left border of the plane, and the right artificial node has directed edges from all images with that meet the right border of the plane, with equal cost C_0 as well.

We now solve the standard shortest path problem from the start node to the end node. This provides a set of images that completely covers the plane horizontally, while minimizing the cost of the seams between images.

In rare cases where the vertical dimension of the plane is not entirely covered by a chosen image, we are left with a hole where no image was chosen to texture. Rather than reverting to a 2D-coverage problem, we can elect to simply fill the hole by selecting images to fill it in a greedy fashion with respect to edge costs of the same cost function.

With this completed, we have now mapped every location on our plane to at least one image, and have minimized the amount of images, as well as the discontinuity between their borders. In the next section, we will apply blending between images where they overlap, but for the sake of comparison with the unblended tile caching method in section 3.2, we arbitrarily pick one image for texturing where images overlap. Figure 11 compares the tile caching method against this seam minimization method.

Though both methods provide quite accurate texturing thanks to the localization refinement process, the seam minimization approach results in fewer visible discontinuities, since it directly reduces the cost of each image boundary, while the tile caching method uses a scoring function that only approximates this effect. Furthermore, seam minimization guarantees the best selection of images, while the tile caching method may select images early on that turn out to be poor choices once subsequent tiles have been processed.

In the context of the backpack modeling system, we apply the seam minimization approach on walls, due to its superior output when provided with optimal images. Floors and ceilings however, given their suboptimal images, as shown in figure 9, can only be textured using the tile caching method.

5.8. Blending

Until now, blending of images has not been used, for the sake of clear comparisons between texturing methods. We will now apply the same blending process on our two

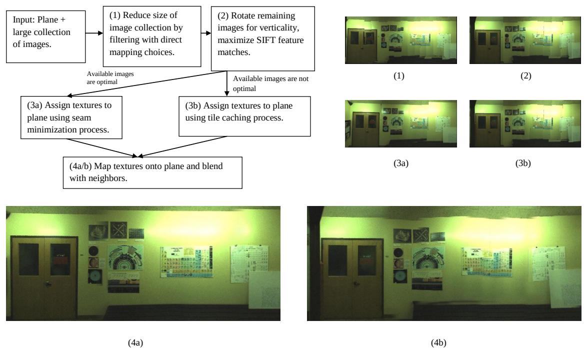
Input: Plane + large collection of images.
(1) Reduce size of image collection by filtering with direct mapping choices.
(2) Rotate remaining images for verticality, maximize SIFT feature matches.
Available images are optimal
Available images are not optimal
(3a) Assign textures to plane using seam minimization process.
(3b) Assign textures to plane using tile caching process.
(4a/b) Map textures onto plane and blend with neighbors.
(1)
(2)
(3a)
(3b)
(4a)
(4b)

Figure 12: Our final texture processing pipeline, with the final output of both approaches.

final texturing methods: localization refinement followed by either tile caching or seam minimization.

Although our preprocessing steps and image selections in either method attempt to minimize all mismatches between images, there are cases, where due to different lighting conditions or inaccuracies in geometry or projection, where we simply have unavoidable discontinuities. These can however be treated and smoothed over by applying alpha blending over image seams. Whether the units we are blending are rectangularly-cropped images or rectangular tiles, we can apply the same blending procedure, as long as we have a guaranteed overlap between units to blend over.

For the tile caching method, we can ensure overlap by texturing a larger tile than we really have. For example, for a tile $l_1 x l_1$ in size, we can make sure to associate it with a texture $(l_1+l_2)x(l_1+l_2)$ in size. For the seam minimization method, we have already ensured overlap between images. To enforce consistent blending however, it is beneficial to add a required overlap distance while solving the shortest path problem in section 5.7. If images do not overlap in a region at least the length of this overlap distance, we do not consider them to overlap at all. If images overlap in a region greater than the overlap distance, we will only apply blending over an area equal to the overlap distance.

Our alpha blending technique is widely used, and blends pixels linearly across overlapping regions. For each pixel within such a region, we simply scale its intensity (by a factor from 0 to 1), based on its distance from the image's border, with a cap set at a fixed distance from the border. In this way, we interpolate between two overlapping images, providing a gradual transition between them. Our texture mapping process is now complete, and in figure 12, we review our entire pipeline and demonstrate the final blended output of both approaches.

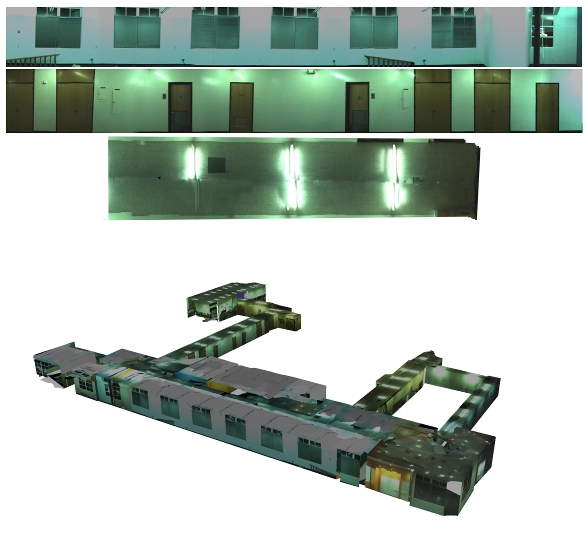

Figure 13: Examples of our final texture mapping output for walls, floors, and ceilings

6. Results and Conclusions

In this paper, we have shown how to accurately texture map models, even when provided with imprecise camera localization data. We were able to refine image locations based on feature mapping, and robustly handle outliers. We generalized one approach to texture mapping to any manner of planes and images, and successfully textured both simple rectangular walls as well as complex floor and ceiling geometry. We also presented an optimized texturing method that takes advantage of our localization refinement process and produces cleaner textures on planes where multiple head-on images are available. Each of these approaches is highly modular, and easily tunable for different environments and acquisition hardware.

Ceilings and floors textured with the robust approach, and walls textured with the optimized approach, are displayed in figure 13. A more detailed walkthrough demonstrating fully textured 3D models using the approaches in this paper is available in the accompanying video to this paper.

References

[1] M. Brown and D. Lowe. Automatic panoramic image stitching using invariant features. *International Journal of Computer Vision*, 74(1):59–73, 2007. 4

[2] G. Chen, J. Kua, S. Shum, N. Naikal, M. Carlberg, and A. Zakhor. Indoor localization algorithms for a human-operated backpack system. In *Int. Symp. on 3D Data, Processing, Visualization and Transmission (3DPVT)*. Citeseer, 2010. 2, 4

[3] M. Fischler and R. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analy-

sis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 5

[4] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*, volume 2. Cambridge Univ Press, 2000. 4

[5] T. Liu, M. Carlberg, G. Chen, J. Chen, J. Kua, and A. Zakhor. Indoor localization and visualization using a human-operated backpack system. In *Indoor Positioning and Indoor Navigation (IPIN), 2010 International Conference on*, pages 1–10. IEEE, 2010. 4

[6] D. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150–1157. Ieee, 1999. 5

[7] V. Sanchez and A. Zakhor. Planar 3d modeling of building interiors from point cloud data. In *Internation Conference on Image Processing*, 2012. 2