Texture mapping 3D planar models of indoor environments with noisy camera poses

Peter Cheng, Michael Anderson, Stewart He, Avideh Zakhor

University of California, Berkeley

ABSTRACT

Automated 3D modeling of building interiors is used in applications such as virtual reality and environment mapping. Texturing these models allows for photorealistic visualizations of the data collected by such modeling systems. Camera poses obtained by these systems and used for texturing often suffer from inaccuracies however, resulting in visible discontinuities when successive images are projected adjacently onto a surface for texturing. Existing methods to stitch images together are often computationally expensive and work independently of pose estimates and geometry information. We propose a method to refine camera poses using both existing estimates and geometry information, followed by two different methods to composite images together, based on the uniformity of available images. The effectiveness of our methods is demonstrated on a number of different indoor environments.

Keywords: Texture Mapping, Reconstruction, Image Stitching, Mosaicing

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 a result, automated 3D site modeling has garnered much interest in recent years.

The first step in automated 3D modeling is the physical scanning of an environment's geometry. An indoor modeling system must recover its own poses within an environment while simultaneously reconstructing the 3D structure of the environment itself.[1–3] 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.

In this paper, we work with data obtained from a backpack-mounted system, carried by an ambulatory human. Such a system provides advantages over more common wheel-mounted systems in terms of agility and portability, but results in much higher localization error.[2] As a result, common methods for texture mapping generally produce poor results, leading to the development of the approaches contained in this paper. Besides localization information and images, environment geometry is required as well for texture mapping. In this paper we work with models obtained by fitting planar surfaces to point clouds generated by the backpack system.[4]A.

Though our texture mapping procedure was designed with the aforementioned system in mind, it is applicable to other systems, provided the required inputs of planar geometry, images with rough extrinsic pose estimates, and either known or estimated intrinsic camera calibration matrices. An important benefit of our method is its relatively low complexity and runtime, and independence of many steps, allowing for live tuning of parameters and easy accomodation of new input data given updates in localization or model-generation algorithms.

The remainder of the paper is organized as follows. Section 2 explains how input images are projected onto our geometry and a simple approach towards texturing. Section 3 covers more sophisticated existing approaches, using image matching and stitching, and their performance on our datasets. Section 4 contains our approach towards image alignment, followed by Section 5, which describes our two methods of compositing images. Section 6 contains results and conclusions.

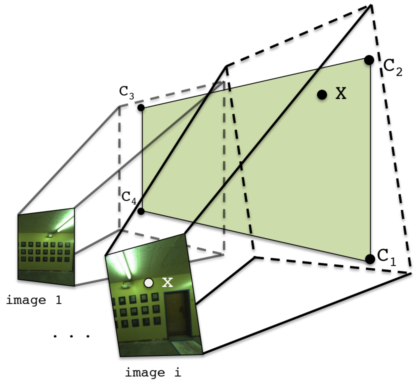


Figure 1: Planes are specified in 3D space by four corners C_1 to C_4. Images are related to each plane through the camera matrics $P_{1..m}$.

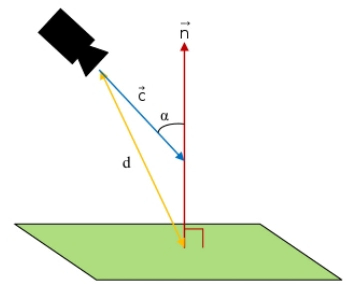


Figure 2: We minimize camera angle α and distance d by maximizing the scoring function $\frac{1}{d}(-1 \cdot \vec{c}) \cdot \vec{n}$

2. SIMPLE TEXTURE MAPPING

The geometry of the texture mapping process for a planar surface is shown in Figure 1. As described earlier, we are provided with a set of M images to texture a 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. A camera matrix P_i is composed of the camera's intrinsic parameters, containing 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 is taken. These extrinsic parameters are determined by the backpack hardware and localization algorithms[1–3] and are quite noisy.

Because our backpack system takes photos at a rate of 5 Hz, thousands of images are available for texturing each surface in our model. Our goal in designing a texture mapping process is to decide which of these images should be used, and where their contents should map onto the final texture, in order to minimize any visual discontinuities or seams that would suggest that the plane's texture is not composed of a single continuous image.

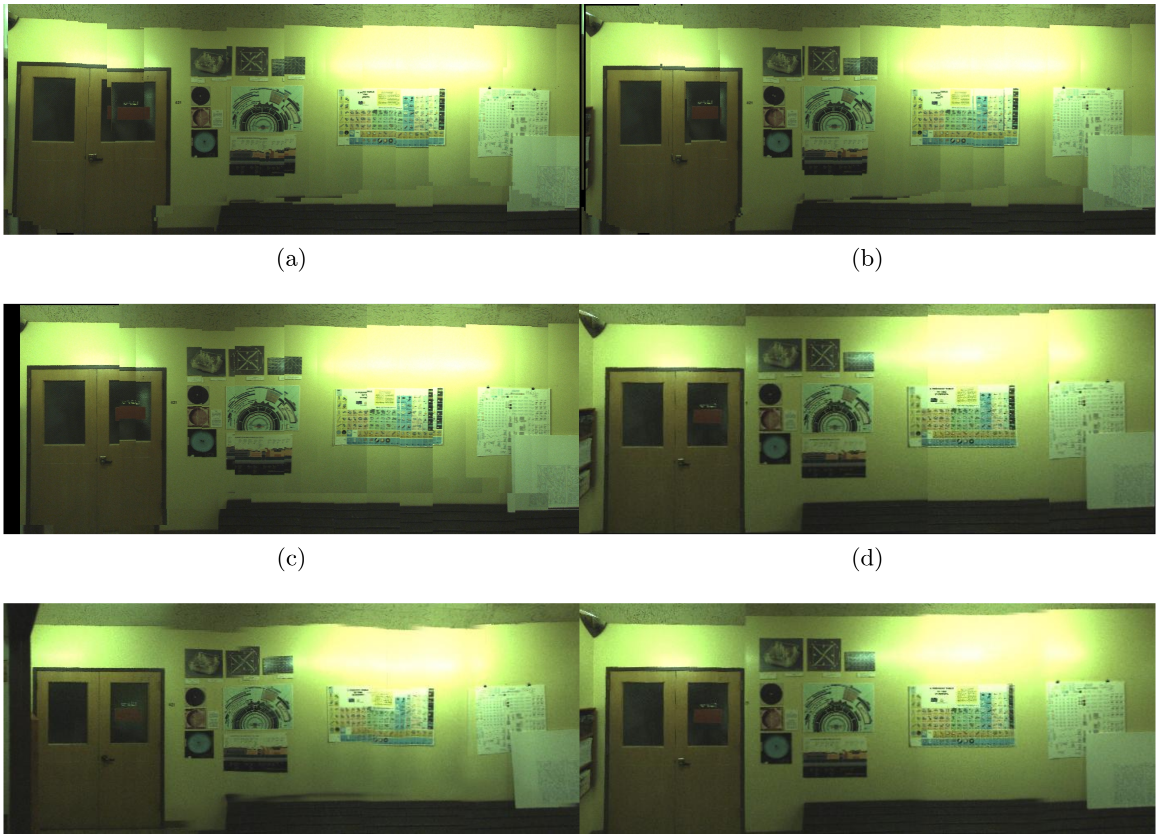
2.1 Tile-based Texture Mapping

Ignoring the fact that the camera matrices $P_{1..M}$ are inaccurate, a simple texturing approach can be performed by discretizing the target plane into small square tiles, in our case 5 pixels across, and choosing an image to texture each tile.

We choose to work with rectangular units to ensure that borders between any two distinct images in the final texture are either horizontal or vertical. Since most strong environmental features inside buildings are horizontal or vertical, any visible seams in our texture intersect them minimally and are less noticeable.

In order to select an image for texturing a 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 using the P_i camera matrices. 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, t, and every other plane.[5]

Once we have a list of candidate images for t, we define a scoring function in order to objectively select the best image for texturing t. Since resolution decreases and camera pose errors compound over distance, we wish to minimize the distance between cameras and the surfaces they texture. Additionally, we desire images that are projected perpendicularly onto the plane, maximizing the resolution and amount of useful texture available in their projections, as well as minimizing any parallax effects due to real-world geometry not accurately represented by our digital model. In other words, we wish to minimize the angle between the tile's normal vector and the camera axis for images selected for texture mapping. These two criteria can be met by maximizing the function $\frac{1}{d}(-1 \cdot \vec{c}) \cdot \vec{n}$ as shown in Figure 2. Specifically, d is the distance between the centers of a camera and a tile, and $\vec{n}$ and $\vec{c}$ are the directions of the plane's normal and the camera axis respectively.



(a) (b)

(c) (d)

(e) (f)

Figure 3: (a) Tile-based texturing. (b) Tile-based texturing after image alignment. (c) Tile-based texturing after image alignment with caching. (d) Shortest path texturing after image alignment). (e) Same as (c) with blending. (f) Same as (d) with blending.

As Figure 3(a) demonstrates, this approach leads to the best texture for each tile independently, but results in many image boundaries with abrupt discontinuities between tiles, due to significant misalignment between images. Given our high number of input images, a better image selection procedure could be used, as described in Section 5, but a more significant and reliable improvement can be found through the alignment of our images.

3. EXISTING APPROACHES TO IMAGE ALIGNMENT

Stitching together multiple images to produce a larger image is a commonly performed task, with many successful approaches over the past few decades. Parts of images are first matched to eachother, through direct pixel comparisons, or more commonly through feature detection and matching. Images are then adjusted to maximize matches, either by calculating homographies between pairs of images, or by modifying camera poses in 1 to 6 degrees of freedom.

Feature detection and matching works best when multiple unique visual references exist in the environment and are present within multiple images. In contrast, our indoor environments have a high prevalence of bare surfaces, as well as repeating textures that cause difficulty in disambiguating features. This lack of reliable reference points often results in errors when matching images together.

Additionally, our datasets often contain long chains of images, which leads to error accumulation when image correspondences are not accurate. For example, when matching a long chain of images through homographies, a pixel in the nth image is translated into the first image's coordinates by multiplying by the 3×3 matrices $H_1 H_2 H_3 ... H_n$. Errors in any of these homography matrices are propagated to all further images, resulting in

(a)

(b)

(c)

Figure 4: Texture alignment via (a) image mosaicing, (b) the graph-based localization refinement algorithm from,[1] and (c) the method proposed in this paper.

drift. Figure 4(a) shows the output of the AutoStitch software package, which performs homography-based image mosaicing.[6] Though AutoStitch performs well in areas with dense features, it has difficulty with even short segments of bare walls, and produces errors that cause straight lines to appear distorted. Many areas with fewer visual features or repeating texture patterns failed outright using AutoStitch.

Image projections can also be aligned by iteratively adjusting camera poses to maximize matches. This is generally done over 6 degrees of freedom.[2] The approach in the cited paper works well locally, but over larger areas results again in error accumulation and drift, as shown in Figure 4(b). Furthermore, it adjusts image locations using a long iterative process, which results in high runtime.

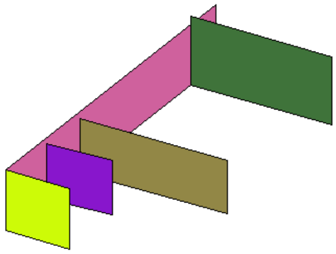
4. 2D IMAGE ALIGNMENT WITH GEOMETRY INFORMATION

In this section, we describe our method of efficient and robust image alignment. Our approach consists of three main parts. First, all images are projected onto the surface and lines within these projections are detected. These lines are then matched up with geometry-based lines comprising the surface's boundary and intersection with other surfaces. Second, occlusion checks are performed to remove invalid parts of each image for the target surface. These two steps are image-independent and performed in parallel. Third, we detect SIFT feature matches between pairs of images and solve a linear least squares problem in 2D to maximize matches.

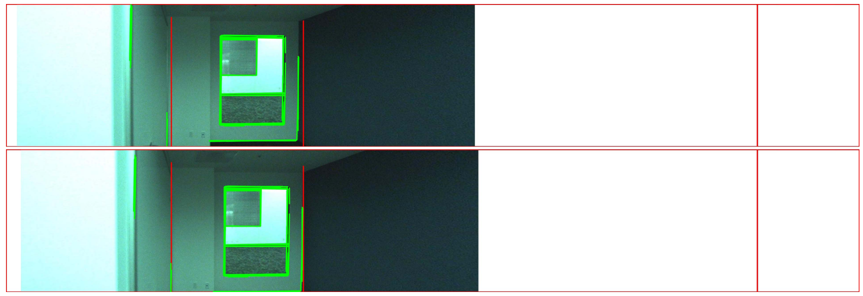
All of our following calculations and alignments are performed in 2D, partly for efficiency, partly because we have a large number of images to choose from, and partly because the nature of our input data is such that localization error chiefly occurs in two dimensions, in the same plane as the surface being projected onto. Recall that our data acquisition system is backpack-mounted, with cameras facing to the sides of the operator. The operator makes efforts to walk parallel to walls, and the localization and model-generation algorithms that provide our input conform the operator's path to be straight and parallel to detected surfaces.[3,4] Since the operator walks in as upright a position as possible, errors in roll, yaw, and the operator's distance from parallel surfaces are minimal. Thus, our highest errors stem from uncertainty in the operator's pitch, equivalent to rotation around the camera axes, as well as location along surfaces. These equate to 2D rotation and translation in a surface's plane.

4.1 Geometry-based Alignment

After computing each image's projection onto the target surface, as described in Section 2, we detect line segments in the image projections using Hough transforms. Experience and intuition show that walls in indoor environments often contain linear features that are either horizontal or vertical, corresponding to doors, windows,



(a)



(b)

Figure 5: (a) When texturing the red surface, it makes sense to align images to surface boundaries and intersections with other surfaces. (b) Red lines are geometry-based lines, while green lines are detected in the image via Hough transform. Above is the original projection using the input camera poses, and below is the projection after being adjusted for alignment.

posters, etc. Thus, when texturing walls, we rotate images in 2D such that dominant lines are made to be horizontal or vertical. This can be further extrapolated by orienting lines with a wall's boundaries, for example in areas with a slanted roof.

At this point in time, image occlusions have not been accounted for. As a result, some image projections contain texture that should project to an adjacent surface, generally with a linear boundary where the two surfaces meet. If this linear boundary is detected by Hough transform, the image is rotated and shifted in 2D such that the visual boundary between two surfaces in an image projection matches the physical boundary in our digital model. An example of such an adjustment is in Figure 5

To perform the rotation and alignment, we use the RANSAC[11] framework. RANSAC allows us to determine an optimal rotation and translation to match up two groups of line segments, while accounting for outliers. First, a rotation angle is computed by gathering all pairs of image line segments and geometry line segments with less than 20 degrees between them. We then run the RANSAC algorithm by sampling 4 of these angles at a time, and using their average as the fitting function. Angles over 2 degrees from this average are considered outliers. Once this 2d rotation angle is determined, it is applied to the image, and projections are re-calculated and image-based lines are re-detected. Now, all pairs of image line segments and geometry line segments with less than 0.2 degree difference and less than 250 mm distance at their furthest points are gathered. RANSAC is then similarly employed to calculate optimal horizontal and vertical translations independently, by sampling 4 pairs at a time and considering an outlier threshold of 10 mm. If a translation is applied, we indicate that the image was aligned to geometry in one or both dimensions, which will reduce further shifting in Section 4.3.

4.2 Image Occlusion

Now that images have been aligned to geometry where possible, we perform a simple recursive occlusion procedure on each image. This is done by performing the intersection tests in section 2 in a regularly spaced grid. Where four corners of a rectangular region are occluded, texture is removed. Where no corners are occluded, nothing occurs. Where there is a mixture of both, the rectangular region is subdivided into four, and the same process is performed on each. By performing Section 4.1's alignment procedure before occlusion, texture belonging to other surfaces is accurately removed, which is necessary for the next section.

4.3 2D Feature Alignment

Our next step is to align overlapping images by searching for corresponding points between all pairs of overlapping images. We use SIFT features for their high detection rate, and choose to use feature alignment rather than pixel or intensity-based alignment due to the differences in lighting as well as possible occlusion among our images, both of which feature alignment is less sensitive to.[7–9] We use SiftGPU[10] for its high performance on both feature detection as well as pairwise matching. These matches determine d^x and d^y distances between each pair of features for two image projections, though these distances may not always be the same for different features.

Since indoor environments often contain repetitive features such as floor tiles or doors, we need to ensure that SIFT-based distances are reliable. First, we only perform alignment on parts of images that overlap given the original noisy poses. Second, we discard feature matches that correspond to an image distance greater than 40 pixels from what the noisy poses estimate. In order to utilize the remaining feature matches robustly, RANSAC[11] is again used to estimate the optimal $d_{i,j}^x$ and $d_{i,j}^y$ distances between two images i and j. For this application, the RANSAC fitting function finds the average distance between selected matches in a pair of images, and the distance function for a pair of points is chosen to be the difference between those points' SIFT match distance and the average distance computed by the fitting function. We use a 10 pixel threshold, so that SIFT matches are labeled as outliers if their horizontal or vertical distances are not within 10 pixels of the average distance computed by the fitting function.

We now use the RANSAC-calculated $d_{i,j}^x$ and $d_{i,j}^y$ distances between each pair of images to refine their positions using weighted linear least squares. There are a total of M^2 possible pairs of images, though we only generate distances between images with SIFT matches. Given these distances and the original image location estimates, we can solve a least squares problem ($\min_{\vec{\beta}} ||A\vec{\beta} - \vec{\gamma}||_2^2$) to estimate the location of the images on the plane. The M-dimensional vector $\vec{\beta}$ represents the unknown x location of each image on the plane for $1 \ldots M$. The optimal x and y locations are obtained in the same way, so we only consider the x locations here:

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

The $N \times M$ matrix A 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 a SIFT-based distance between images 1 and 2, images 1 and 3, images 2 and 3, etc.

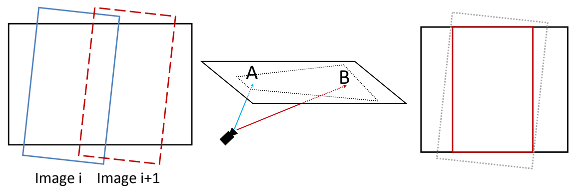
$$A = \begin{pmatrix} -1 & 1 & 0 & \cdots & 0 & 0 \\ -1 & 0 & 1 & \cdots & 0 & 0 \\ 0 & -1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & -1 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 0 \end{pmatrix}$$

If only relative distances between images are included, the absolute location of the images can not be calculated, and the matrix is rank deficient. From Section 4.1, a number of images are anchored to geometry points, in one or both dimensions, and thus their locations can be used to fix the rest in place. In case no anchor images exist, we simply arbitrarily pick an image, as we have no other reference points to work with. In the above matrix, the first image is set to be such an anchor, simply by placing a 1 in its column.

The N-dimensional observation vector $\vec{\gamma}$ is constructed using the SIFT-based distances generated in the RANSAC matching stage. Elements in the observation vector corresponding to anchor images are simply their locations as determined by the original noisy localization. Thus a $\vec{\gamma}$ corresponding to the above matrix can be written as:

$$\vec{\gamma}^T = \begin{pmatrix} d_{1,2}, & d_{1,3}, & d_{2,3}, & \ldots & d_{N-2,N-1}, & d_{N-1,N}, & x_1 \end{pmatrix}$$

The $\vec{\beta}$ that minimizes $||A\vec{\beta} - \vec{\gamma}||_2^2$ results in a set of image locations on the plane that best honors all the SIFT-based distance measurements between images. This solution however does not make use of our noisy camera poses, and will fail when no SIFT matches are found between one segment of the plane and another. To account for this, we add rows to the A matrix and observations to the $\vec{\gamma}$ vector corresponding to the original noisy distances. We then add weighting to our least squares problem where the SIFT distances and anchor values are given a high weight e.g. 1, while the noisy distances are given a smaller weight e.g. 0.01.



(a) (b) (c)

Figure 6: (a) Images for vertical planes are tilted, but their camera axes are more or less normal to their respective planes. (b) Camera axes for ceiling images are at large angles with respect to plane normals. (c) Wall images are cropped to be rectangular.

After completing this same process for the y dimension as well, and making the resultant shifts, our images overlap and match each other with far greater accuracy. Applying the simple mapping scheme in Section 2.1 to the same wall used in that section results in Figure 3(b), which has far fewer discontinuities, though errors due to lighting differences and repeating features are still visible.

5. IMAGE COMPOSITING

In this section, we revisit the tile-based texturing approach from Section 2, with an added caching mechanism to reduce image boundaries. This method works well given all manner of camera poses and surfaces, but for optimal cases where we have large sections of usable texture from images, we propose a superior method that further reduces image boundaries.

5.1 Tile-Mapping with Caching

From Section 2.1, we saw that discontinuities occur where adjacent tiles are textured by different images. Though Section 4's image alignment removes many such discontinuities, there are still cases where seams are visible due to imprecise matching or other factors such as model-based errors. To reduce the cases where this happens, it makes sense to take into account image selections made by neighboring tiles while texture mapping a given tile. By using the same image across tile boundaries, we can eliminate a discontinuity altogether. If this is not possible because a tile is not visible in images used by neighboring tiles, using similar images across tile boundaries also leads to less noticeable discontinuities.

Essentially a caching mechanism, we select the best image for a tile t by searching through two subsets of images for a viable candidate, before searching through the entire set of valid images. The first subset of images is the images selected by adjacent tiles that have already been textured. We must first check which of these images can map to t, and then of those, we make a choice according to the scoring function in Figure 2. Before reusing this image, we ensure it meets the criteria $\alpha < 45°$, in order to ensure a high resolution projection, with α as the camera angle as shown in Figure 2.

If no satisfactory image is found in the first subset, we check a second subset of images, consisting of those taken near the ones in the first subset, both spatially and temporally. These images are not the same as the ones used for neighboring tiles, but are taken at a similar location and time, suggesting that their localization and projection are quite similar, and thus likely matched more cleanly. If no viable image is found according to the same criteria as before, we search the entire set of candidate images, selecting based on the same scoring function from Figure 2.

The result of this caching approach is shown in Figure 3(c), where seams are now reduced as compared to Figure 3(b).

As mentioned earlier, our data comes from a mobile backpack system. Human operators can not carry the backpack in a perfectly upright position and are bent forwards at 15 to 20 degrees with respect to the vertical direction. As a result, cameras facing sideways are head on with respect to vertical walls, while cameras oriented towards the top or bottom of the backpack are at an angle with respect to horizontal floors and ceilings. This is depicted in Figures 6(a) and 6(b). These oblique camera angles for horizontal surfaces translate into textures

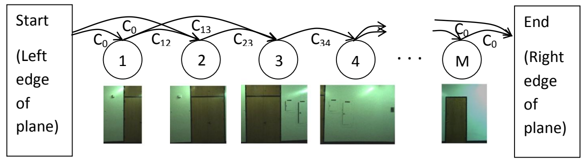


Figure 7: DAG construction for the image selection process.

that span large areas once projected, as shown in Figure 6(b). Using the tile-based texture mapping criteria from Figure 2, such projections have highly varying scores depending on the location of a tile on the plane. Thus, the tiling approach in this section is a good choice for texturing floors and ceilings, as it uses the parts of image projections that maximize resolution and accuracy for their respective plane locations, e.g. areas near point A and not near point B, in Figure 6(b).

5.2 Shortest Path Texturing

For vertical walls, most images are taken from close distances and head-on angles, resulting in high resolution fronto-parallel projections. As a result, for each tile on a wall plane, the scoring function of Figure 2 is relatively flat with respect to candidate images, as they are all more or less head on. Thus, the scoring function is less significant for walls, and it is conceivable to use a different texturing strategy to directly minimize visible seams when texturing them. This is done by choosing the smallest possible set of images that (a) covers the entire plane and (b) minimizes the visibility of borders between them. A straightforward cost function that accomplishes the latter is the sum of squared differences (SSD) of pixels in overlapping regions between all pairs of images. Minimizing this cost function encourages image boundaries to occur either in featureless areas, such as bare walls, or in areas where images match extremely well.

The first step in this procedure is to obtain a list of images useful for texturing the wall. A simple way to do this is to use the images selected by the tiling process in Section 2.1. Such a list of images is guaranteed to cover the entire wall, and consists of desired camera poses overall.

To cover the entirety of a plane, our problem can be defined as minimally covering a polygon i.e. the planar surface, using other polygons of arbitrary geometry i.e. image projections, with the added constraint of minimizing the cost function between chosen images. This is a complex problem, though we can take a number of steps to simplify it.

Given that wall-texture candidate images are taken from more or less head-on angles, and knowing that only minor rotations are made in Section 4, we can reason that their projections onto the plane are approximately rectangular. By cropping them all to be rectangular, as shown in Figure 6(c), our problem becomes the conceptually simpler one of filling a polygon with rectangles, such that the sum of all costs between each pair of rectangles is minimal. We thus also retain the advantages of working with rectangular units, as explained in Section 2.1.

The operator's path, and the location and orientation of the cameras on the acquisition backpack is such that images nearly always contain the entirety of the floor to ceiling range of wall planes. Images therefore rarely need to be projected with one above another when texturing wall planes. In essence, we need only to ensure lateral coverage of wall planes, e.g. from left to right, 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 the SSD cost function, and solve a simple shortest path problem to find an optimal subset of images with regard to the SSD cost function.[12]

Figure 7 demonstrates the construction of a DAG from overlapping images of a hallway wall. 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 overlapping images. The weights of these edges are determined by the SSD cost function. 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(right) artificial node has directed edges with equal and arbitrary cost C_0 to(from) all images that meet the left(right) border of the plane. We now solve the shortest path problem from the start node to the end node. This results in a set of images completely covering the plane horizontally, while minimizing the cost of seams between images.

In rare cases where the vertical dimension of the plane is not entirely covered by one or more chosen images, we are left with holes where no images are selected to texture. Since these holes are rare, and generally fairly small, we use a greedy approach, repeatedly filling the hole with images that result in the lowest SSD costs in a blending region around the hole, as discussed in Section 5.3. This method is not as optimal as a true 2D-coverage solution would be, but it is a fast approximation, and adequately handles the few holes we encounter.

With this completed, we have now mapped every location on the plane to at least one image, and have minimized the number of images, as well as the discontinuities at their borders. As seen in Figure 3(d), this shortest path method has fewer visible discontinuities than Figure 3(c) corresponding to the tile caching approach*. This is especially evident when comparing the posters in the images, which have clear misalignment over seams in Figure 3(c), but are much more aligned in Figure 3(d). This shortest path approach approach directly reduces the cost of each image boundary, while the tile caching method uses a scoring function that only approximates this effect. Furthermore, this approach guarantees the best selection of images to minimize seams, while the sequential tile caching method may select images early on that turn out to be poor choices once subsequent tiles have been processed. This shortest path approach is also far less intensive in terms of memory usage and runtime, both during texture generation and rendering, as it does not require discretizing planes or images.

When texturing an entire 3D planar model, we apply the shortest path method on walls, due to its superior output when provided with head-on images. Floors and ceilings however, given their many images taken at oblique angles, are textured using the tile caching method.

5.3 Blending

We now apply a blending procedure to both texturing methods. Although the image alignment steps and image selection in both methods attempt to minimize all mismatches between images, there are occasional unavoidable discontinuities in the final texture due to different lighting conditions or inaccuracies in model geometry. 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 needed for display. For example, for a rendered tile $l_1 \times l_1$, we can associate it with a texture $(l_1 + l_2) \times (l_1 + l_2)$ in size. We have found $l_2 = \frac{l_1}{2}$ to provide a balance between blending and keeping features unblurred. For the shortest path method, we have already ensured overlap between images. To enforce consistent blending however, we add a minimum required overlap distance of 40 px while solving the shortest path problem in Section 5.2. Additionally, if images overlap in a region greater than the overlap distance, we only apply blending over an area equal to the overlap distance.

After performing linear alpha blending across overlapping regions, our texture mapping process is complete. Figures 3(e) and 3(f) show the blended versions of Figures 3(c) and 3(d) respectively. It is clear that the shortest path approach exhibits much better alignment and fewer seams than the tile-caching method, as Figure 3(f) clearly has the best visual quality among the textures in Figure 3.

6. RESULTS AND CONCLUSIONS

Examples of ceilings and floors textured with the tile caching approach, and walls textured with the shortest path approach, are displayed in Figure 8. High resolution texture comparisons, as well as further examples and interactive walkthroughs are available at †.

As mentioned earlier, our approach is quite efficient. The top wall in Figure 8(a) was generated with 7543 $\times$ 776 pixels, and spans a 40-meter long wall. Given 41000 input images, a 2.8GHz consumer-grade laptop

*In Figure 3(d), we arbitrarily chose one image for texturing where images overlap, as blending will be discussed in section 5.3.

†`http://www.eecs.berkeley.edu/~pcheng/textureMapping`

takes approximately a minute to pick 36 candidate images, followed by another minute to perform both image alignment and the shortest path texturing method, with the majority of computation time spent calculating SIFT matches within the SiftGPU framework. While not quite real-time visualization, the process is capable of generating quick updates after changes in various parameters or modifications to input data.

In this paper, we have developed an approach to texture mapping models with noisy camera localization data. We are able to refine image locations based on geometry references and feature matching, and robustly handle outliers. Using the tile-based mapping approach, we can texture both simple rectangular walls as well as complex floor and ceiling geometry. We also implemented a shortest path texturing method that produces seamless textures on planes where multiple head-on images are available. Both of these approaches are highly modular, and easily tunable for different environments and acquisition systems.

REFERENCES

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

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

[3] Kua, J., Corso, N., and Zakhor, A., "Automatic loop closure detection using multiple cameras for 3d indoor localization," in [*IS&T/SPIE Electronic Imaging*], (2012).

[4] Sanchez, V. and Zakhor, A., "Planar 3d modeling of building interiors from point cloud data," in [*Internation Conference on Image Processing*], (2012).

[5] Glassner, A. et al., [*An introduction to ray tracing*], Academic Press (1989).

[6] Brown, M. and Lowe, D., "Autostitch." `http://www.cs.bath.ac.uk/brown/autostitch/autostitch.html`.

[7] Lowe, D., "Object recognition from local scale-invariant features," in [*Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*], **2**, 1150–1157, Ieee (1999).

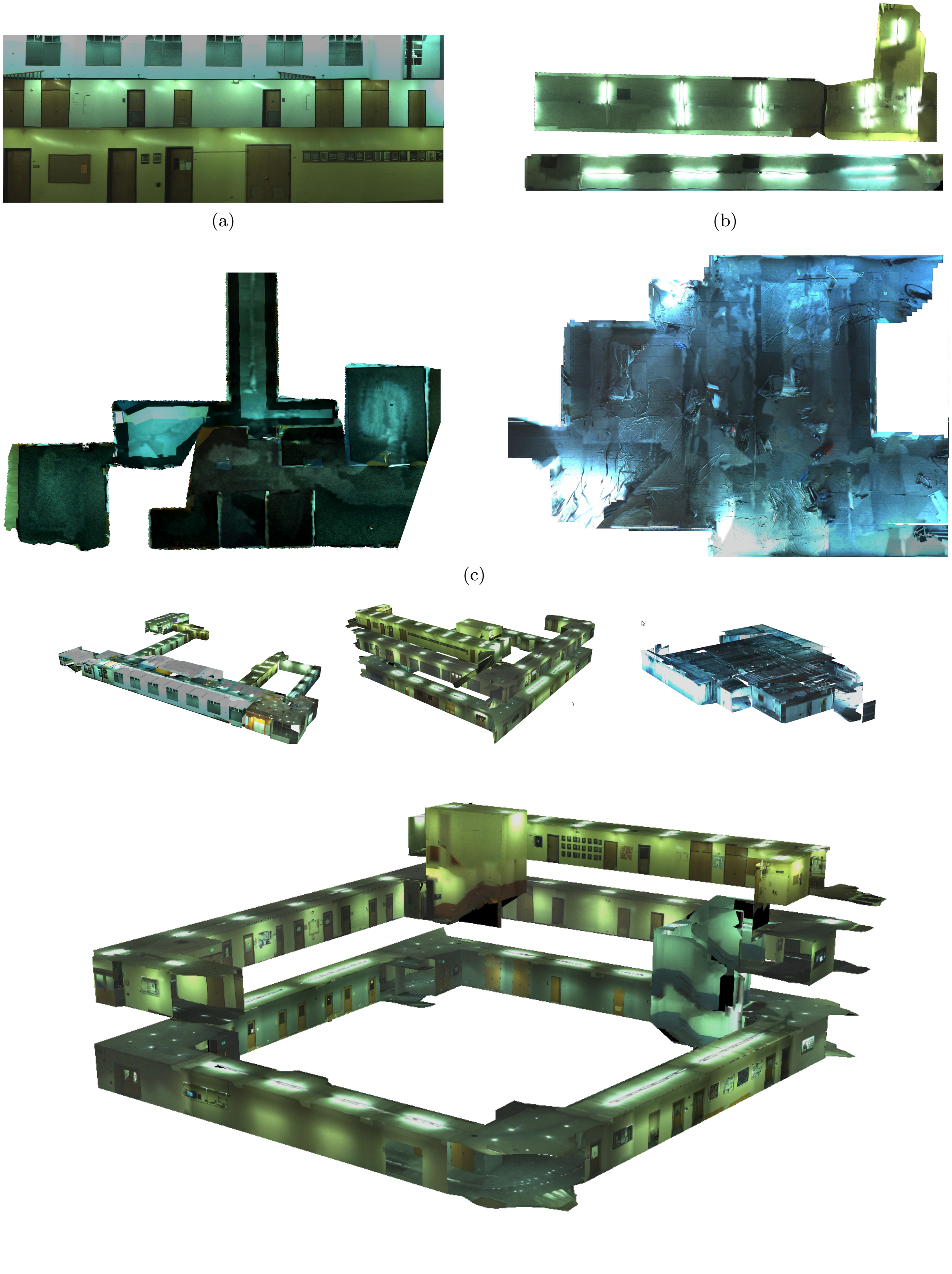
[8] Mikolajczyk, K. and Schmid, C., "A performance evaluation of local descriptors," *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **27**(10), 1615–1630 (2005).

[9] Szeliski, R., "Image alignment and stitching: A tutorial," *Foundations and Trends® in Computer Graphics and Vision* **2**(1), 1–104 (2006).

[10] Wu, C., "Siftgpu." `http://www.cs.unc.edu/~ccwu/siftgpu/`.

[11] Fischler, M. and Bolles, R., "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM* **24**(6), 381–395 (1981).

[12] Dijkstra, E., "A note on two problems in connexion graphs," *Numerische Mathematik* **1**, 269–271 (1959).



(d)

Figure 8: Examples of our final texture mapping output for (a) walls, (b) ceilings, (c) floors, (d) full models.