Texture Mapping 3D Models of Indoor Environments with Noisy Camera Poses

Peter Cheng[a], Michael Anderson[a], Stewart He[b], and Avideh Zakhor[a]

[a]University of California, Berkeley;
[b]University of California, Davis

ABSTRACT

Automated 3D modeling of building interiors is used in applications such as virtual reality and environment mapping. Texturing these models allows for photo-realistic visualizations of the data collected by such modeling systems. While data acquisition times for mobile mapping systems are considerably shorter than for static ones, their recovered camera poses often suffer from inaccuracies, resulting in visible discontinuities when successive images are projected onto a surface for texturing. We present a method for texture mapping models of indoor environments that starts by selecting images whose camera poses are well-aligned in two dimensions. We then align images to geometry as well as to each other, producing visually consistent textures even in the presence of inaccurate surface geometry and noisy camera poses. Images are then composited into a final texture mosaic and projected onto surface geometry for visualization. The effectiveness of the proposed method 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 consequently automated 3D site modeling has garnered much interest in recent years.

The first step in automated 3D modeling is the physical scanning of the environment's geometry. An indoor modeling system must be able to recover its pose within an environment while simultaneously reconstructing the 3D structure of the environment itself.[1–4] 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 platform carried by an ambulatory human provides unique advantages over static or vehicular-based systems on wheels in terms of agility and portability, but can also result in larger localization error.[4] As a result, common methods for texture mapping generally produce poor results. In this paper, we present an approach to texture mapping 3D models of indoor environments in the presence of uncertainty and noise in camera poses. In particular, we consider data obtained from a human-operated backpack system, detailed in Section 2.

The overall block diagram for the proposed texture mapping procedure is shown in Figure 1, where the number attached to each box indicates the section in which the concept in the box is explained in this paper. First, the geometry of an environment is split into regions, each of which will be textured independently and

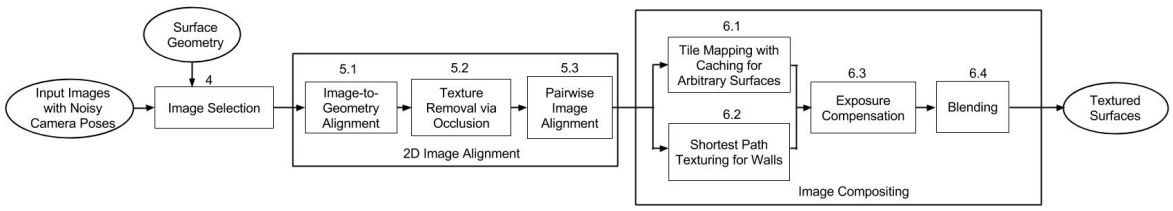


Figure 1: The proposed texture mapping procedure

in parallel. For each region, we begin by selecting a set of images that spans the entire region's surface with high resolution imagery. We then use recovered noisy camera poses to project these selected images onto the surface. These projections are rotated and translated in 2D, in order to maximally align them with the surface's geometry, as well as to each other, allowing us to handle both errors in geometry as well as camera poses. Finally, we demonstrate a tile-based as well as a shortest-path-based approach for compositing images, where the latter produces superior textures, but is only applicable for surfaces with optimal camera poses and lateral image coverage.

The remainder of the paper is organized as follows. Section 2 provides background on the data acquisition system and describes a region segmentation procedure. Section 3 covers existing approaches to image stitching, and their performance on our datasets. Section 4 explains how to downsample the set of available images by selecting those with the best orientation and distance from surfaces under consideration. Section 5 contains the proposed approach towards 2D image alignment, followed by Section 6, which describes two methods of selecting and compositing images to create the final texture. Sections 7 and 8 contain results and conclusions.

2. DATA ACQUISITION

This section contains background information describing the backpack-mounted system and its operation. The hardware and operation of the backpack system, as well as the postprocessing of acquired data, play an important role in motivating the approaches described in this paper. The following two sections will focus on image and raw data acquisition, and a method of partitioning recovered environment geometry, respectively.

2.1 Acquisition Hardware

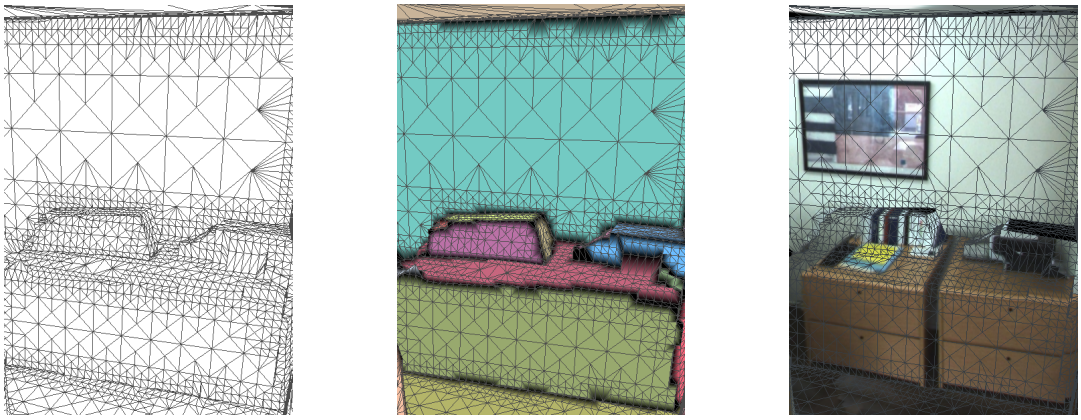
The backpack system used in this paper has a number of laser range scanners as well as side-facing cameras taking photos at a rate of 5 Hz.[4] In order to collect data, an operator wears the backpack system and traverses the environment to be scanned, walking in straight lines at a standard pace and making an effort to walk parallel to all walls in the area. Multiple localization and loop-closure algorithms are then applied to the raw data collected by the onboard sensors,[1,3,4] and the backpack is localized* over its data collection period. This involves recovering the 6 degrees of freedom for the backpack itself as well as the cameras rigidly mounted on it. These results are noisy, which motivates the approaches in this paper. Once this is complete, the data from the laser range scanners is used to generate a 3D point cloud of the surrounding environment. This point cloud is used to construct simple, coarse models,[5,6] as well as highly detailed models.[7] Regardless of resolution or size, the model, consisting of geometric surfaces in 3D space, along with the set of images captured by the backpack's cameras and their noisy 3D poses, are the input to the texture mapping problem.

2.2 Geometry Partitioning

In order to effectively texture a 3D model, we first divide the model into a set of regions, each to be textured independently. The purpose of this is to ensure texture continuity along large uniform areas, while allowing texture boundaries to fall along natural environmental boundaries. Because textures may not match at such boundaries where two regions meet, it is important to minimize the visibility of boundaries. This is done by encouraging region boundaries to occur at sharp corners in the model geometry, where texture continuity is not important. For instance, in Figure 2(a), each side of the desk, objects on top of the desk, and the wall behind, should be textured separately. This is done by grouping the triangles comprising each such area into separate regions. Region grouping is performed by first designating all contiguous coplanar groups of triangles as different regions. Any regions that are less than $1m^2$ in size are then repeatedly joined to their largest neighboring regions, as long as the angle between them is under 90°. An example of such a grouping is shown in Figure 2(b). The same area with textures successfully applied to each region is shown in Figure 2(c).

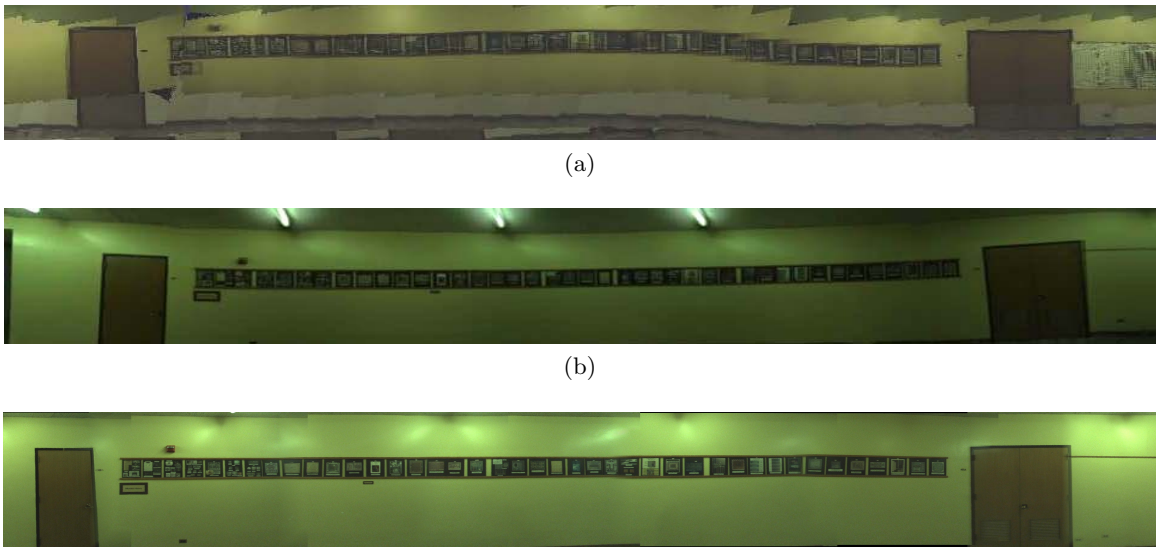
For the purposes of texturing, it is convenient to treat each region as planar. To accomplish this, a plane is fitted to each region. This plane corresponds to the largest contiguous section of coplanar triangles, as most regions are characterized by small outcroppings from a largely planar surface, such as objects on a table or

*In this paper, we use the terms localization and pose recovery interchangeably, in that they both refer to recovering position and orientation.



(a) (b) (c)

Figure 2: (a) Part of a model representing a desk and miscellaneous objects on top of the desk. (b) Triangles grouped into regions for texturing. (c) Final textured output.



(a) (b) (c)

Figure 3: Texture alignment via (a) the graph-based localization refinement algorithm, (b) the AutoStitch software package, and (c) the method proposed in this paper.

features protruding from a wall. The 3D surface composing a region is then projected onto its calculated plane, and the resulting 2D planar polygon is used as the texturing surface. This 2D planar approximation provides a simple, yet approximately accurate surface onto which images can be projected in order to generate textures. However, 3D surfaces are still retained in order to achieve accurate results when performing occlusion checks, and of course are used to generate the final textured model. Each region now has a set of triangles representing the original model geometry, as well as a planar approximation. The task now is to generate a texture for each region.

3. RELATED WORK

There are many existing approaches to stitching together multiple images to produce a larger, seamless image.[8–13] Generally, parts of images are matched to each other, by detecting feature points and identifying matches. Images are then transformed to maximizally align matches, often by computing homographies between pairs of images, or by iteratively adjusting camera poses in 1 to 6 degrees of freedom.

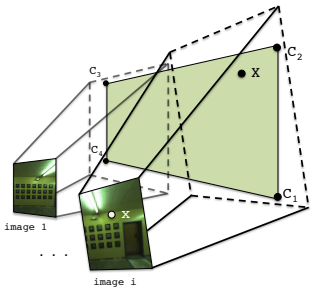


Figure 4: Images are related to each surface through the camera matrices $P_{1..m}$.

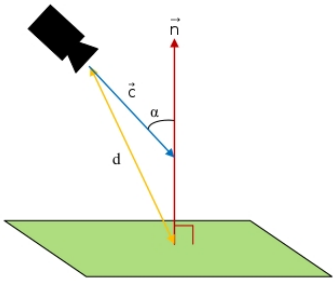


Figure 5: Camera angle α and distance d are minimized by maximizing the scoring function $\frac{1}{d}(-1 \cdot \vec{c}) \cdot \vec{n}$

Feature matching works best when unique visual references exist in the environment that can be detected in multiple images. Unfortunately, many indoor environments have a high prevalence of bare surfaces as well as repeating textures, such as with windows and doors. Additionally, our datasets often contain long 1-dimensional chains of images, such as in Figure 3, which often leads to error accumulation. For example, when matching a long chain of images via homography matrices, a pixel in the nth image must be translated into the first image's coordinates by multiplying by the 3×3 matrix $H_1 H_2 H_3 ... H_n$. Errors in each of these homography matrices is propagated to all further images, resulting in drift.

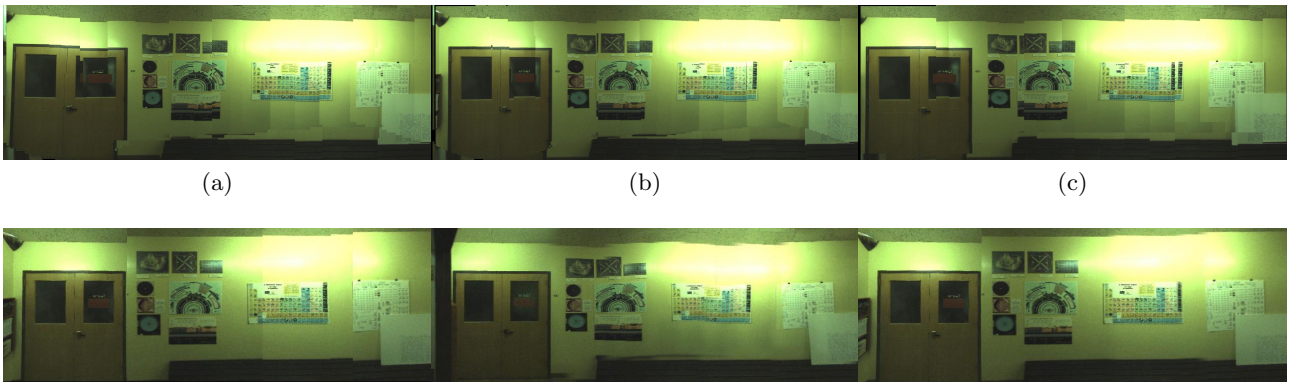
Two methods based on feature matching and homography calculation are shown in Figure 3. Figure 3(a) depicts an attempt at integrating image stitching with the iterative global localization algorithm used to localize the backpack system detailed in Section 2.[4] Figure 3(b) depicts an output image from AutoStitch, which is software based on research in the related area of panorama generation.[14,15] Both methods were thoroughly tuned for the shown example, but due to the factors mentioned in the previous paragraph, they still show significant amounts of drift and distortion, as well as loss of alignment to the environment geometry.

4. IMAGE SELECTION

The geometry of the texture mapping process for a region, as described in Section 2.2, is shown in Figure 4. Given a set of images to texture a target surface, camera matrix P_i for the ith image transforms 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 was taken. These extrinsic parameters are determined by the backpack hardware and the corresponding localization algorithms[1,3,4] and are noisy.

Since the backpack system takes pictures at a rate of 5 Hz, thousands of images are available for texturing each surface in the 3D model. Our objective in designing a texture mapping process is to determine 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. In the remainder of this section, we propose an image subsampling procedure to obtain a set of images for use in all further steps.

Our criteria for image subsampling has the following stipulations. First, we want a set of images such that their projections together cover up the entirety of our target surface, so as to avoid holes in the final texture. Second, we desire our final texture to have high resolution throughout; thus every location on the target surface should have at least one image that contains high resolution imagery for it. A straightforward way to accomplish these goals is to discretize the target surface into small square tiles, and for each tile, select the image that has the best viewing angle and resolution for texturing that tile. In order to select the image that can best texture a tile t, we first gather a list of candidate images that contain all four of its corners; we can rapidly check this 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 determined using standard



(a) (b) (c)

(d) (e) (f)

Figure 6: (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,f) Blending applied to (c) and (d).

ray-polygon intersection tests between the camera location, t, and every other surface, or via the k-d tree from Section 2.2 if present.[16]

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 become more pronounced with distance, we wish to minimize the distance between cameras and the surfaces they texture. Additionally, we desire images that are projected perpendicularly, rather than obliquely, 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 the digital 3D model. In other words, we wish to minimize the angle between the tile's normal vector and the camera axis for images selected for texturing that tile. These two criteria can be met by maximizing the function $\frac{1}{d}(-1 \cdot \vec{c}) \cdot \vec{n}$ as shown in Figure 5, where 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.

When the images selected for each tile are used directly to texture their respective tiles, image boundaries with abrupt discontinuities between tiles are visible, as shown in Figure 6(a). While it is clear that camera pose inaccuracies are too severe for such a simple approach to work, the selected images all appear to contain optimal camera angles with high resolution, and much of their misalignment appears reconcilable using 2D transforms. This procedure generally selects around 10% of the possible images that could be used for texturing a surface, and not only reduces the computational complexity of the remaining steps, but also selects the most promising images for the remaining steps.

5. 2D IMAGE ALIGNMENT

In this section, we describe our method for efficient and robust image alignment. Rather than register all of our images in 3D, as many state-of-the-art techniques for image stitching do, we instead align a subset of images in 2D; this subset corresponds to the images selected by the image selection procedure described in Section 4.

Applying 2D alignments to this set of images works well for the following reasons. First, the nature of our input data and the selected images is such that localization error chiefly occurs in two dimensions, which correspond to the plane of most surfaces being projected onto. This is because the backpack operator, while scanning an environment, generally walks parallel to walls. Furthermore, the backpack system maintains a consistent distance from floors and ceilings during data collection. As a result, the translational error of camera poses is quite minimal in dimensions perpendicular to walls, ceilings and floors, which comprise the majority of surfaces being textured.

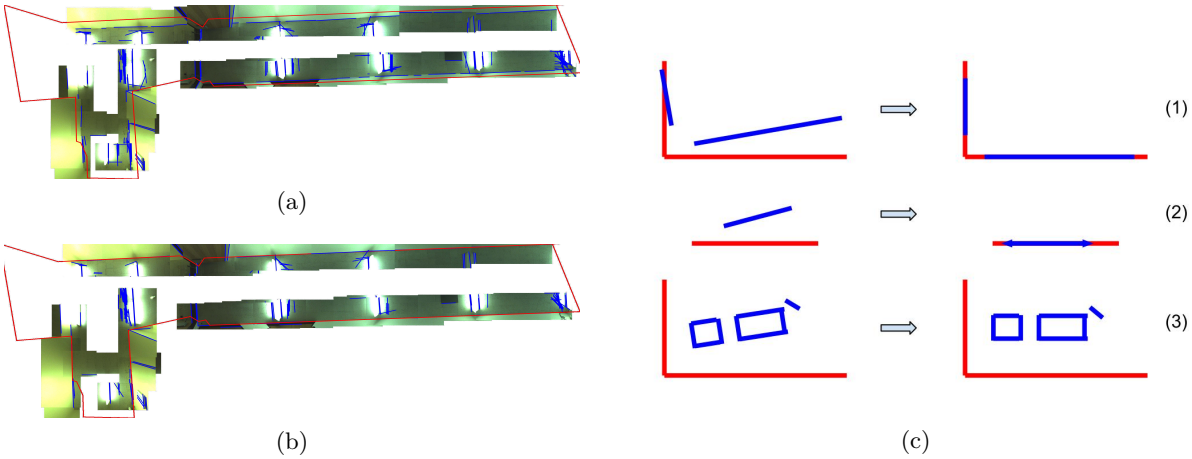


Figure 7: Images projected onto a ceiling surface, where geometry-based lines corresponding to the ceiling's boundary are shown in red. Image-based lines detected by Hough transform in the image projections are shown in blue; (a) images projected with their original noisy camera poses; (b) after image alignment to maximize line matches between images and geometry; (c) examples of matching lines in cases with ≥ 2 line pairs, 1 line pair, and zero line pairs, from top to bottom.

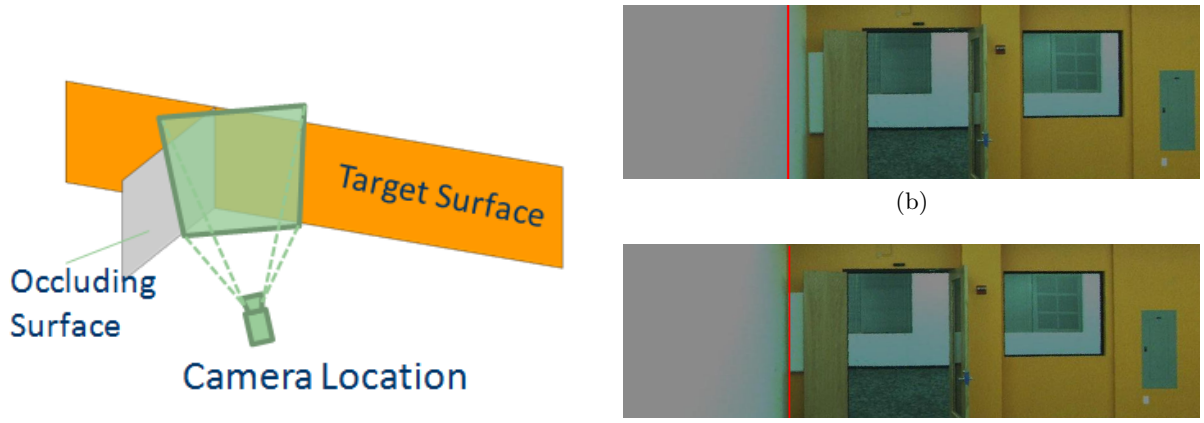
Our proposed 2D alignment procedure consists of three parts, as shown in the diagram in Figure 1. First, images are projected onto the surface and lines within these projected images are detected. Images are then transformed such that these lines match geometric lines composing the boundaries of the surface being textured. Second, occlusion checks are performed to remove invalid parts of each image for the target surface. Third, SIFT feature matches are detected between pairs of images, and a weighted linear least squares problem is solved in order to maximize all image and geometry-based alignments.

5.1 Geometry-based Alignment

After computing each image's projection onto the target surface, as described in Section 4, we can obtain a set of image-based line segments by using Hough transforms to detect lines in the image projections. We also gather a set of geometry-based lines, which correspond to the lines comprising the target surface's border, as well as lines formed where other surfaces intersect the target surface. An example of these lines is shown in red for a ceiling surface in Figure 7(a). Given perfect camera poses and surface geometry, the lines in images corresponding to corners between surfaces should match up exactly with corners in the 3D model. By inducing such lines to match, we can fit camera poses more accurately to the surface, and therefore to each other as well.

To align images to surface geometry, we collect pairs of image-based and geometry-based line segments, which are within distance and angular thresholds of each other. We have found a distance threshold of 250 mm and an angular difference threshold of 10° to work well for our datasets. For each pair of lines, we compute the angular difference between the pair's image and geometry lines. If there are 2 or more pairs with angular differences within 1°, we select the two pairs with the longest noncollinear image-based lines, and rotate the image such that the lines in the pair with the longest image-based line become parallel. We then find a translation such that the lines in that same pair overlap. This translation has ambiguity in the dimension along the matched lines, which is resolved by matching the midpoint of the image-based line in the second pair to its corresponding geometry-based line. This is shown in case (1) of Figure 7(c). Thus, with 2 or more pairs, it is possible to obtain a fixed rotation and translation for geometry alignment, which are saved for usage in Section 5.3.

If there are not 2 or more pairs with similar angular differences, we select the pair with the longest image-based line, which corresponds to a strong linear visual feature, and apply a rotation and the minimal translation to match the pair's lines. This translation's ambiguity however, can not be resolved, but is also saved to be used in Section 5.3. This is shown in case (2) of Figure 7(c). Finally, in the case where there are no line pairs, we can still rotate images in order to exploit patterns in indoor environments, as shown in case (3) of Figure 7(c). For



(a) (b) (c)

Figure 8: (a) The image from the camera in this diagram contains texture that belongs to the gray occluding surface, which should not be projected onto the orange target surface; (b) without geometry alignment, texture to the left of the red line would be removed, which would leave some erroneous texture projected onto our target surface; (c) after geometry alignment, the image is shifted, resulting in the correct amount of texture being removed.

instance, doors, windows, furniture, ceiling lights, etc. tend to have linear edges that are parallel to the edges of the surfaces they are on. Thus, we choose to minimize the angle between image-based lines and geometry-based lines regardless of distance. We use the RANSAC framework to compute a rotation angle that best accomplishes this while ignoring outliers.[17]

After these steps, image projections line up well with target surfaces, as shown in Figure 7(b), which is considerably more aligned than Figure 7(a). This procedure reconciles both errors in camera poses as well as in geometry, and results in sharp, continuous borders across images, which is crucial when checking for occlusion.

5.2 Image Occlusion

In order to correctly texture surfaces, it is important to detect and remove parts of image projections containing texture for occluding surfaces. For instance, in Figure 8(a), an image used to texture the orange target surface also contains part of a gray occluding surface. We remove this incorrect texture by performing ray-polygon intersection tests between the camera location and every surface in our model except the target surface.[16] If using planar approximations for regions, as explained in Section 2.2, the k-d tree is used instead. These intersection tests are performed repeatedly in a recursive grid, until all occlusions are detected and corresponding occluded texture is removed. Occlusion checking works entirely with geometry, so by ensuring that images match geometry using 5.1's alignment procedure, texture belonging to other surfaces is accurately removed, as seen in Figure 8(b) vs. 8(c).

5.3 2D Feature Alignment

The next step is to align the selected images from Section 4 for each surface to each other by searching for corresponding feature points between all pairs of overlapping images. We use feature alignment rather than pixel or intensity-based alignment due to the differences in lighting as well as possible occlusion among images, both of which feature alignment is less sensitive to.[8,18,19] We use SiftGPU[20] for its high performance on both feature detection as well as pairwise matching. These matches determine dx and dy 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 align parts of images that overlap given the original noisy poses. Second, we discard feature matches that correspond to an image distance greater than 200 mm from what the noisy poses estimate. RANSAC[17] is then used to obtain a consensus from the the remaining feature matches.

We now use the feature-based distances between each pair of images as well as geometry alignment results from Section 5.1 to refine all image positions using a weighted linear least squares approach. An example setup for such a problem $\min_{\vec{\beta}} ||W^{\frac{1}{2}}(A\vec{\beta} - \vec{\gamma})||_2^2$ with 3 images is as follows.

$$A = \begin{pmatrix} -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & -m_2 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad \vec{\beta} = \begin{pmatrix} x_1, \\ x_2, \\ x_3, \\ y_1, \\ y_2, \\ y_3 \end{pmatrix} \quad \vec{\gamma} = \begin{pmatrix} dx_{1,2}, \\ dy_{1,2}, \\ dx_{2,3}, \\ dy_{2,3}, \\ -m_2 gx_2 + gy_2, \\ gx_1, \\ gy_1, \\ tx_1, \\ ty_1, \\ tx_2, \\ ty_2, \\ tx_3, \\ ty_3 \end{pmatrix} \quad \vec{W} = \begin{pmatrix} 1, \\ 1, \\ 1, \\ 1, \\ 1, \\ 1, \\ 1, \\ 0.01, \\ 0.01, \\ 0.01, \\ 0.01, \\ 0.01, \\ 0.01 \end{pmatrix}$$

The variables to solve for are the x_i and y_i positions of images, while equations are the feature-based distances between pairs of images, images fixed to geometry with 0 or 1 degrees of freedom, and the original noisy camera poses. In this scenario, a feature-based distance of $dx_{1,2}$, $dy_{1,2}$ was calculated between images 1 and 2. This corresponds to the first and second row of A, while the third and fourth row of A represent the same for images 2 and 3. Rows 5 through 7 correspond to results of the geometry alignment procedure in Section 5.1. Specifically, row 5 corresponds to a geometry-based constraint of image 2's location to a line of slope m_2, passing through point gx_2, gy_2, while rows 6 and 7 correspond to a fixed location for image 1 without any degrees of freedom. Rows 8 through 13 correspond to the original camera locations for each image (tx_i, ty_i).

The original camera poses are needed due to lack of feature matches in all images, or lack of enough geometry alignment results to generate a single solution. Since it is desirable to minimally use the original noisy poses, we assign to them a weighting factor of 0.01, while all other equations are weighted at 1.

Since this problem is linear, it can be solved efficiently; after applying the resulting shifts, images overlap and match each other with far greater accuracy. Using the simple tile-based texturing scheme from Section 4 on these adjusted images results in Figure 6(b), which has far fewer discontinuities than in 6(a), though some 3D error as well as lighting differences and parallax effects are still visible.

6. IMAGE COMPOSITING

In Section 4 we described an image selection method, that when used to create a texture, resulted in many visible discontinuities. In Section 6.1, we will refine that approach with an added caching mechanism to improve texture continuity. This method works well for all arbitrary camera poses and surfaces. However, for cases where images have consistently perpendicular viewing angles to surfaces under consideration, such as walls, we develop an alternative method in Section 6.2 which further reduces visual artifacts. Both of these approaches are then combined with an exposure compensation and blending scheme in order to produce final textures for each surface.

6.1 Tile-Mapping with Caching

For the simple texturing method described in Section 4, discontinuities occur where adjacent tiles are textured by different images. Though Section 5'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 this, we implement a spatiotemporal caching mechanism to take into account image selections made by neighboring tiles while texture mapping a given tile. By using the same image across tile boundaries, it is possible to eliminate a discontinuity altogether. If a tile is not visible in images used by neighboring tiles, using similar images across tile boundaries still leads to less noticeable discontinuities.

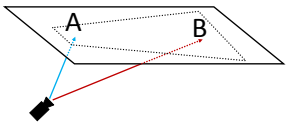


Figure 9: Camera axes for horizontal surfaces are at large angles with respect to plane normals.

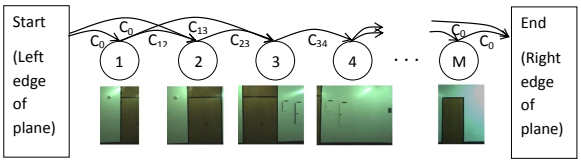


Figure 10: DAG construction for the image selection process.

The best image for a tile t is selected by searching through two subsets of images for a viable candidate, before searching through the entire set of valid images obtained in Section 4. The first subset of images is those selected by adjacent tiles that have already been textured. We must first check which of these images contain texture for t, and then of those, we make a choice according to the scoring function in Figure 5. Before reusing this image, we check the criteria $\alpha < 45°$, in order to ensure a high resolution projection, with α as the camera angle as shown in Figure 5.

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. We use the noisy camera poses to determine spatial proximity. 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 match more seamlessly. If no viable image is found according to the $\alpha < 45°$ criteria, we search the entire set of candidate images from Section 4, selecting based on the same scoring function from Figure 5.

The result of applying this caching approach to the images for the surface in Figure 6(a) is shown in Figure 6(c), where seams are considerably reduced as compared to Figure 6(b). However, some discontinuities are still present, as visible in the posters on the wall with breaks in their borders.

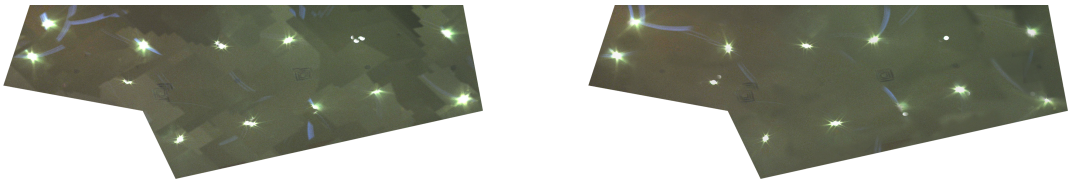
6.2 Shortest Path Texturing

As mentioned earlier, our data comes from a mobile backpack system with side-facing cameras. While these cameras have a wide-angle field of view,they are often pointed directly at vertical surfaces during data acquisition. This means that image projections on vertical surfaces tend to have higher, and more consistent resolution, while the more oblique projections onto horizontal surfaces result in textures spanning large areas, such as in Figure 9. Using the tile-based texture mapping criteria from Figure 5, such projections have highly varying scores depending on the location of a tile on the plane. Thus, the tiling approach in Section 6.1 is an appropriate choice for texturing horizontal surfaces, as it uses the parts of image projections that maximize resolution for their respective plane locations, e.g. areas near point A and not near point B, in Figure 9.

For vertical surfaces however, which make up a large portion of most models, images are usually taken from close distances and head-on angles, resulting in high resolution rectangular projections. As a result, for each tile on a wall plane, the scoring function of Figure 5 is relatively flat with respect to candidate images, as they are all more or less head on. Since the scoring function is less discriminative for walls, we devise a different texturing strategy to directly minimize visible seams when texturing them. This is done by choosing a subset of images such that their projections (a) cover the entire plane and (b) have minimal differences in overlapping regions. 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.

To find the subset of images that covers the plane with minimal cost, we form a shortest path problem. This can be done because the wide-angle cameras have full floor-to-ceiling coverage of walls, and the backpack operator logically only moves horizontally. We thus are only concerned with lateral coverage when working with wall surfaces. We can therefore construct a horizontal 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.[21]

Figure 10 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



(a) (b)

Figure 11: (a) A ceiling texture composed of images taken with varying exposures, with significant brightness differences. (b) Exposure compensation applied to the same set of images from (a) by applying computed gains to each image

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.

We have now (a) mapped every location on the plane to at least one image, (b) decreased the number of texturing images, generally retaining around 20% of the image subset obtained in Section 4, and (c) decreased the discontinuities at each image border. As seen in Figure 6(d), this shortest path method has fewer visible discontinuities than Figure 6(c) corresponding to the tile caching approach[†]. This is especially evident when comparing the posters in the images. 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 our models, we apply the shortest path method on surfaces where images are taken at optimal angles and provide full coverage along one dimension. Floors, ceilings, and smaller complex objects such as furniture, given their many images taken at oblique angles, are textured using the tile caching method of Section 6.1.

6.3 Exposure Compensation

Before blending images together, exposure compensation is applied to equalize brightness among neighboring images. For the images in this report, cameras are set to have automatic exposure, which means that images of the same object may have different brightness levels. While successful image alignment and blending can reduce sharp seams between adjacent images, there may still be noticeable brightness gradients between images, particularly in areas near light sources. This can be seen in Figure 11(a), where the ceiling texture has patches of clearly differing brightness. To diminish this effect, a gain can be computed and applied to each image, with the effect of linearly scaling the brightness of each image. The goal is to compute a gain for each image, such that the brightness of an area is consistent across image boundaries.

Similar to the image position refinement procedure in Section 5.3, exposure compensation is performed simulatenously across all images present in a single region. To begin, a relative gain R_{ij} is computed for each pair of overlapping images I_i and I_j, which have a set of pixels common to both images, labeled as P_{ij}. The relative gain R_{ij} is then obtained by calculating the scaling factor between the average intensity of P_{ij} in I_i and the average intensity of P_{ij} in I_j.

[†]In Figure 6(d), we arbitrarily chose one image for texturing where images overlap, as blending will be discussed in section 6.4.

As before, these pair-wise observations can be combined and solved as a least-squares optimization problem. In this case, observations do not need to be weighted, and so the formulation is $\min_{\vec{b}}||(X\vec{b}-\vec{a})||_2^2$. An example setup for 3 images, where each image overlaps with both of the other images, is shown below.

$$X = \begin{pmatrix} R_{12} & -1 & 0 \\ R_{13} & 0 & -1 \\ 0 & R_{23} & -1 \\ 1 & 1 & 1 \end{pmatrix} \quad \vec{b} = \begin{pmatrix} G_1, \\ G_2, \\ G_3 \end{pmatrix} \vec{a} = \begin{pmatrix} 0, \\ 0, \\ 0, \\ 3 \end{pmatrix}$$

In this problem, $\vec{b}$ is the $N \times 1$ vector of gains G_i to be solved for, where N is the number of images. Matrix X is an observation matrix corresponding to the relative intensities of overlapping images. In particular, the number of rows of X is one plus the number of unique pairs of overlapping images, and the number of columns of X is equal to N. Each row of X, except the last one, contains an R_{ij} value which is placed in column i, as well as a -1 value in column j, with zeroes elsewhere. As explained above, R_{ij} is the relative gain between images i and j, and therefore $G_i R_{ij} - G_j = 0$. Thus, all values of $\vec{a}$, which is a vector with size equal to the number of rows of X, are set to 0, except for the last one.

The last row of X and corresponding last entry of $\vec{a}$ are used to constrain the problem, as without them, all computed gains would be allowed to scale by any amount. To keep gains at reasonable values, the final observation is used: $\Sigma G_i = N$, where N is the number of images. This has the effect of setting the average of all gains to 1, and is represented by the final row of ones in X, and the corresponding N value in $\vec{a}$.

For the 81 source images that compose the ceiling location in Figure 11, the corresponding X matrix has 1343 rows, and the optimization problem takes under a second to solve, on a 2.8 GHz dual core machine. The result of computing and applying gains can be seen in Figure 11(b). As compared to Figure 11(a), the effect is clear, as dark/bright patches are no longer present, and the overall brightness of the image has not changed significantly.

6.4 Blending

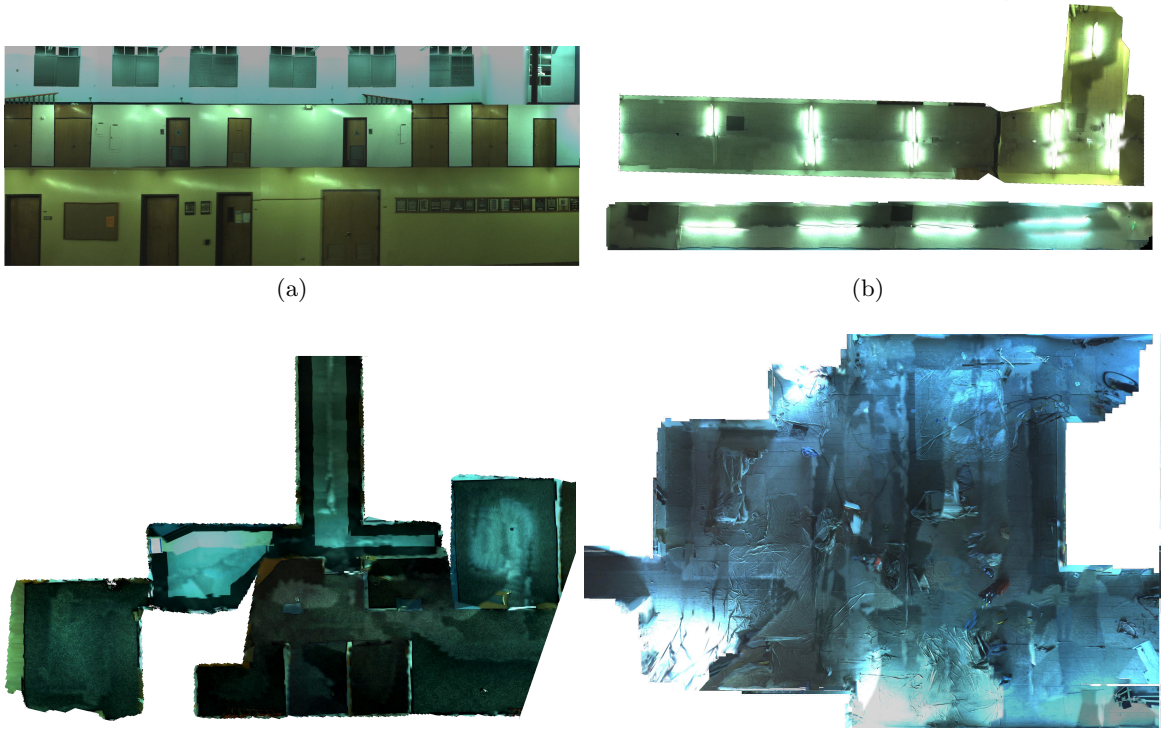
We now apply a blending procedure to the texturing methods in Sections 6.1 and 6.2. Although the image alignment and selection steps in both methods attempt to minimize all mismatches between images, and exposure compensation minimizes brightness differences, there are occasional unavoidable discontinuities in the final texture due to parallax effects or inaccuracies in model geometry. These can however be treated and smoothed over by applying alpha blending over image seams. Whether the units to be blended are rectangularly-cropped images or rectangular tiles, we can apply the same blending procedure, as long as there is a guaranteed overlap between units to blend over.

For the tile caching method of Section 6.1, we can ensure overlap by texturing a larger tile than needed for display. For example, for a rendered tile of size $l_1 \times l_1$, 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 good balance between blending and keeping features unblurred at image boundaries. For the shortest path method, we already have ensured overlap between images. To enforce consistent blending however, we add a minimum required overlap of images of 200 mm while solving the shortest path problem in Section 6.2. 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 linear alpha blending across overlapping regions, the texture mapping process is complete. Figures 6(e) and 6(f) show the blended versions of Figures 6(c) and 6(d) respectively. By comparing all images, it is clear that Figure 6(f) has the best visual quality and the best texturing approach among the textures in Figure 6.

7. RESULTS

This section contains results of the proposed texture mapping process on a number of different environments, with geometry generated using three different methods. Further images of individual textured surfaces and entire textured models, as well as videos and interactive walkthroughs can also be found at ‡.



(a) (b)

(c)

Figure 12: Examples of our final texture mapping output for (a) walls, (b) ceilings, (c) floors

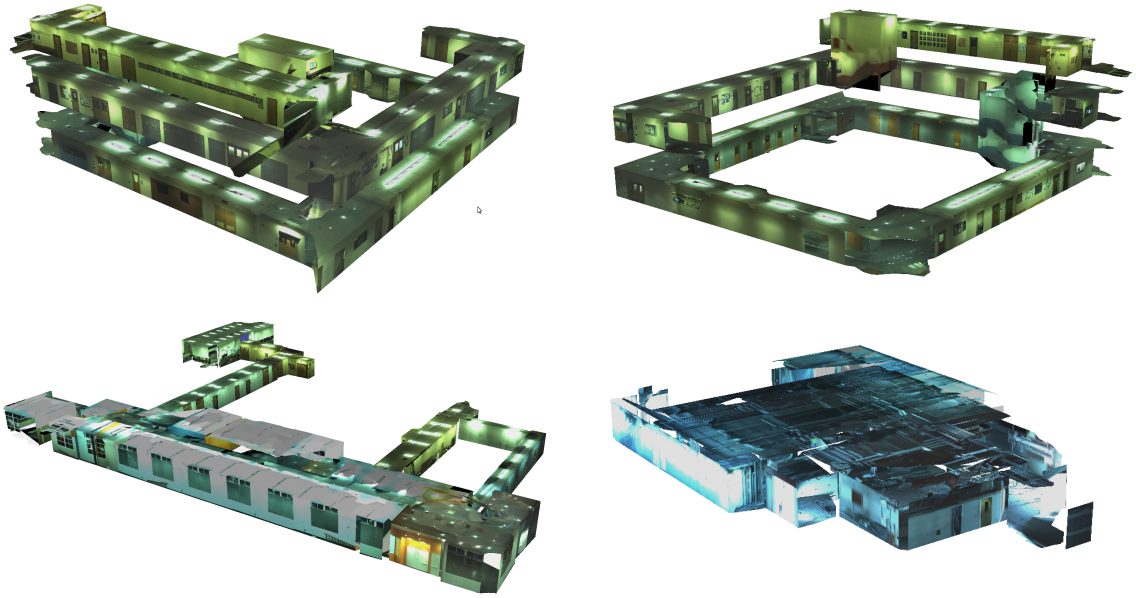


Figure 13: Textured models from the PCA-based approach.

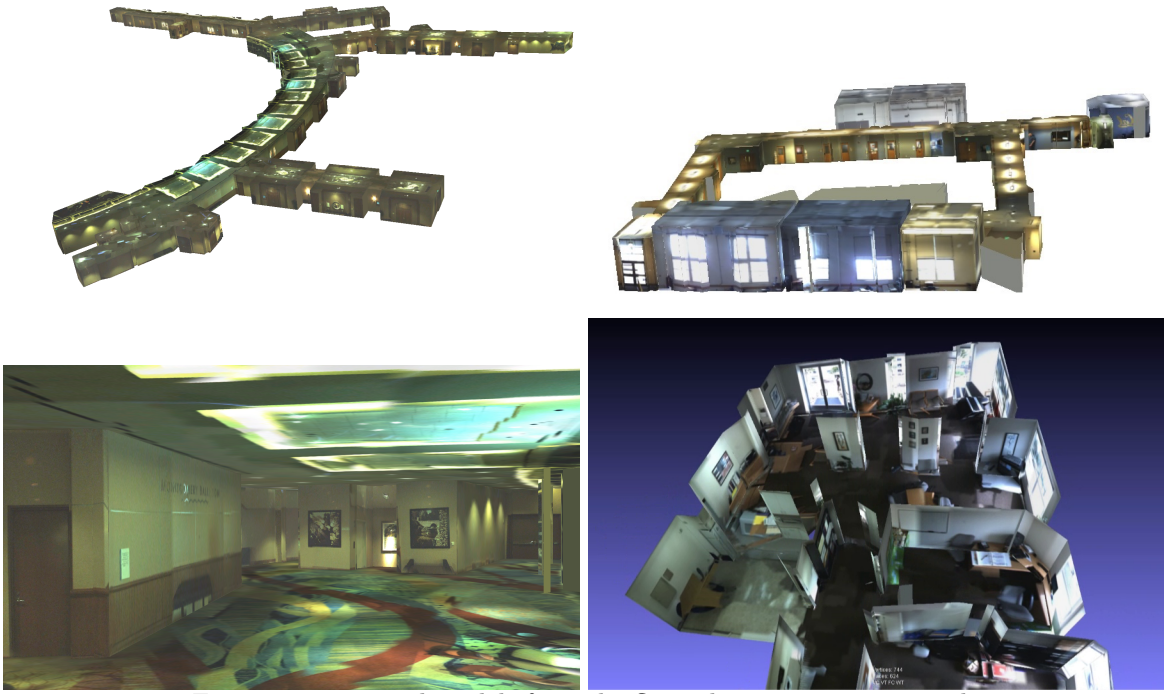


Figure 14: Textured models from the floor plan extrusion approach.

Figure 15: Textured models from the voxel-carving approach.

7.1 Examples

The texture mapping procedure in this paper was tested with three different geometry-reconstruction methods, a PCA-based plane-fitting method,[5] a floor plan extrusion method,[6] and a voxel-carving mesh generation method.[7] The first two methods generate very similar low-resolution models, containing only major walls, ceilings and floors. The third method generates high-resolution models, and attempts to reconstruct all scanned objects in the environment. Examples of individually textured regions are shown in Figure 12, while fully texture mapped models generated by the PCA-based, floor plan extrusion, and voxel-carving approaches, can be seen in Figures 13, 14, and 15 respectively.

7.2 Runtime

As mentioned earlier, our approach is quite efficient. The top wall in Figure 12(a) is a single region, and its texture was generated with 7543×776 pixels, spanning a 40 meter long wall. Given 41000 input images in the dataset, a 2.8 GHz dual-core machine takes under 10 seconds to choose 36 candidate images and perform the full texturing process. The tile-caching approach takes a similar amount of time as well, with the much larger floor textures in Figure 12(c) taking 5-10 minutes to generate. While not real-time, our process is capable of generating fast updates after changes in various parameters or modifications to input data, and if integrated directly into a 3D modeling system, could provide quick visual feedback as data is collected.

7.3 Visualization

Our full models consist of an input model file, generated textures, and a mapping of image points to 3D model vertices. The textured models shown throughout this paper range from 20 MB in size to over 400 MB in a compressed format, with textures of sizes over 500 megapixels. These models are visualized using the OpenSceneGraph toolkit,[22] which allows for export to many common model formats, as well as interactive visualization, even in web browsers or mobile devices.

8. CONCLUSION

In this paper, we have developed an approach to texture mapping models of indoor environments 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 large planar features as well as smaller, more complex surfaces. 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 similar systems across multiple environments.

Our method is likely to fail however in scenarios where 3D error is large. A logical progression of our approach to resolve camera error in 3D is to perform matching between image lines and geometry in 3D, which can be done reasonably efficiently.[23,24] Using linear features in addition to SIFT features is also likely to result in improved matches, as indoor scenes often have long, unbroken lines spanning multiple images.[25] Finally, the blending procedure is quite basic, and applying more sophisticated methods of blending as well as image boundary selection would benefit the final visual quality, and more robustly handle motion-based or parallax errors.

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

[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).

[‡] `http://www-video.eecs.berkeley.edu/research/indoor/`

[4] 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).

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

[6] Turner, E. and Zakhor, A., "Watertight as-built architectural floor plans generated from laser range data," in [*3DIMPVT*], (2012).

[7] Turner, E. and Zakhor, A., "Watertight planar surface meshing of indoor point-clouds with voxel carving," in [*3DV*], (2013).

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

[9] Agarwala, A., Agrawala, M., Cohen, M., Salesin, D., and Szeliski, R., "Photographing long scenes with multi-viewpoint panoramas," in [*ACM Transactions on Graphics (TOG)*], **25**, 853–861, ACM (2006).

[10] Wang, L., Kang, S., Szeliski, R., and Shum, H., "Optimal texture map reconstruction from multiple views," in [*Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*], **1**, I–347, IEEE (2001).

[11] Coorg, S. and Teller, S., "Matching and pose refinement with camera pose estimates," in [*Proceedings of the 1997 Image Understanding Workshop*], (1997).

[12] Debevec, P., Taylor, C., and Malik, J., "Modeling and rendering architecture from photographs: A hybrid geometry-and image-based approach," in [*Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*], 11–20, ACM (1996).

[13] Bernardini, F., Martin, I., and Rushmeier, H., "High-quality texture reconstruction from multiple scans," *Visualization and Computer Graphics, IEEE Transactions on* **7**(4), 318–332 (2001).

[14] Brown, M. and Lowe, D. G., "Automatic panoramic image stitching using invariant features," *Int. J. Comput. Vision* **74**, 59–73 (Aug. 2007).

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

[16] Glassner, A. S., [*An introduction to ray tracing*], Academic Press (1989).

[17] 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).

[18] 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).

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

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

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

[22] "Openscenegraph." `http://www.openscenegraph.org/projects/osg`.

[23] Elqursh, A. and Elgammal, A., "Line-based relative pose estimation," in [*Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*], 3049 –3056 (june 2011).

[24] Koeck, J. and Zhang, W., "Extraction, matching and pose recovery based on dominant rectangular structures," (2005).

[25] Ansar, A. and Daniilidis, K., "Linear pose estimation from points or lines," *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **25**, 578 – 589 (may 2003).