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 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. Existing methods to stitch images together are often computationally expensive and work independently of pose estimates and geometry data. We present a method for texture mapping that reduces complexity by selecting images whose camera poses can be align in 2D. We perform alignment of images to geometry as well as images to each other, which produces visually consistent textures even in the presence of inaccurate surface geometry. We also present two different methods for selecting and compositing images into a single texture, depending on the geometry of the surface being textured. The effectiveness of these 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 such, 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 camera 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.

Mounting such devices on a platform carried by an ambulatory human provides unique advantages over vehicular-based systems on wheels in terms of agility and portability, but can also result in larger localization error.[1] As a result, common methods for texture mapping generally produce poor results.

In this paper, we present a number of approaches to texture mapping 3D models of indoor environments made of planar surfaces in the presence of uncertainty and noise in camera poses. In particular, we consider data obtained from a human-operated backpack system with a number of laser range scanners as well as 2 cameras facing left and right, each equipped with fisheye lenses reaching an approximately 180° field of view and taking photos at a rate of 5 Hz. Applying multiple localization and loop-closure algorithms on the raw data collected by the onboard sensors,[1–3] the backpack is localized[*] over its data collection period. This requires recovering the 6 degrees of freedom for the backpack as well as the cameras rigidly mounted on it. Once this is complete, the data from the laser range scanners is used to generate a 3D point cloud of the surrounding environment, from which a 3D planar model is created.[4] This model, consisting of 2D polygonal planes in 3D space, along with the set of images captured by the backpack's cameras and their noisy 3D poses, can be considered the input to our texture mapping problem.

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

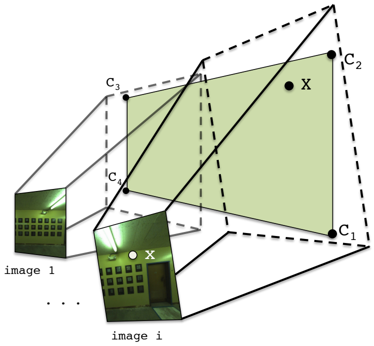


Figure 1: Surfaces to be textured are specified in 3D space by corners C_i. Images are related to each surface through the camera matrices $P_{1..m}$.

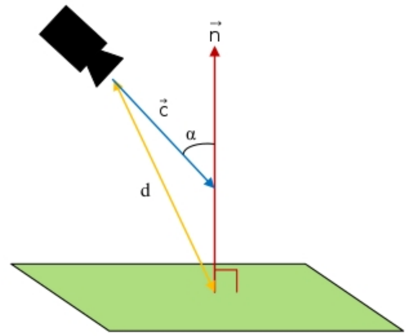


Figure 2: We minimize camera angle α and distance d by maximizing the scoring function $\frac{1}{d}(-1 \cdot \vec{c}) \cdot \vec{n}$

We perform texture mapping on each planar surface independently and in parallel. For each surface, we begin by obtaining a set of images that spans the entire surface with high resolution imagery. We then use our noisy camera poses to project these selected images onto the surface. These projections are then refined 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. For surfaces and camera poses at any location or orientation, we demonstrate a simple tile-based approach for sampling high-resolution portions of images and compositing them into a texture. In cases where cameras have consistently perpendicular viewing angles, we demonstrate a superior approach that leads to more seamless textures.

The remainder of the paper is organized as follows. Section 2 explains how images are projected onto geometry and shows a simple texture mapping procedure. Section 3 covers existing approaches to image stitching, and their performance on our datasets. Section 4 contains our approach towards efficient 2D image alignment, followed by Section 5, which describes two methods of selecting and compositing images to create the final texture. Section 6 contains results and conclusions.

2. SIMPLE TEXTURE MAPPING

Before describing our texture mapping procedure, we use this section to explain basic texture mapping and a method for selecting images for texturing. The geometry of the texture mapping process for a planar surface is shown in Figure 1. Given a set of M images to texture a target plane, camera matrix P_i for the ith image, 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 substantially noisy.

Because the backpack system takes pictures at a rate of 5 Hz, hundreds of images are available for texturing each surface in the 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 provide an overview of a simple tile-based texture mapping procedure.

Ignoring the fact that the camera matrices $P_{1..M}$ are inaccurate, the most straightforward approach is to discretize the target plane into small square tiles, and choose an image to texture each tile directly. 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 the texture intersect them minimally and are less noticeable.

In order to select an image for texturing a tile t, we first gather a list of candidate images that contain all four of its corners, which we can rapidly 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 determined using standard ray-polygon intersection tests between the camera location, t, and every other surface.[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 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 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 2, 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.

As Figure 3(a) demonstrates, this approach results in image boundaries with abrupt discontinuities between tiles, due to significant misalignment between images, though most of it appears to be reconcilable using 2D transforms. Given the large number of input images, it is possible to develop a more sophisticated image selection procedure, as described in Section 5, however, a more significant and reliable improvement can be found through first improving the alignment of the images as shown in the next 2 sections.

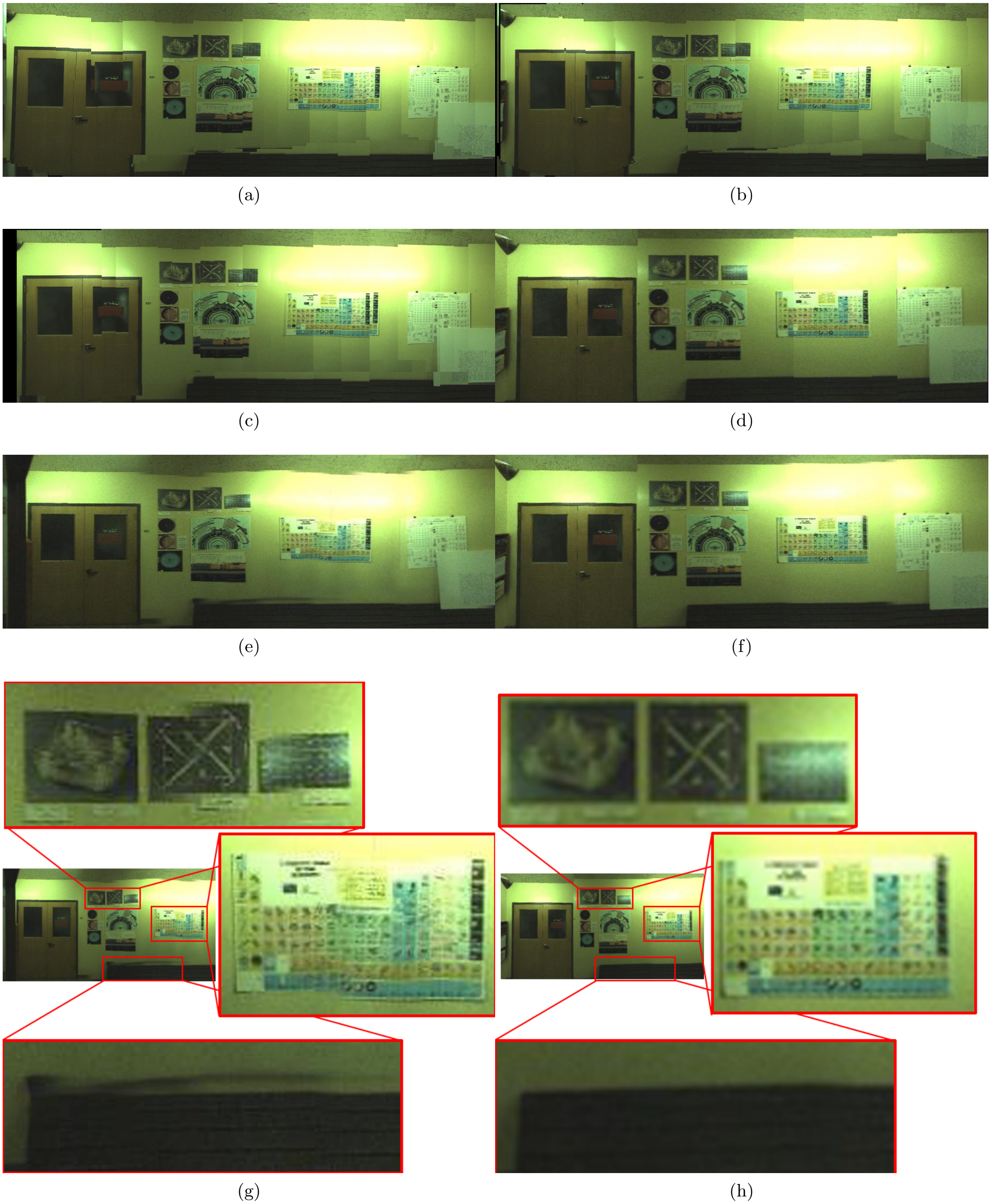
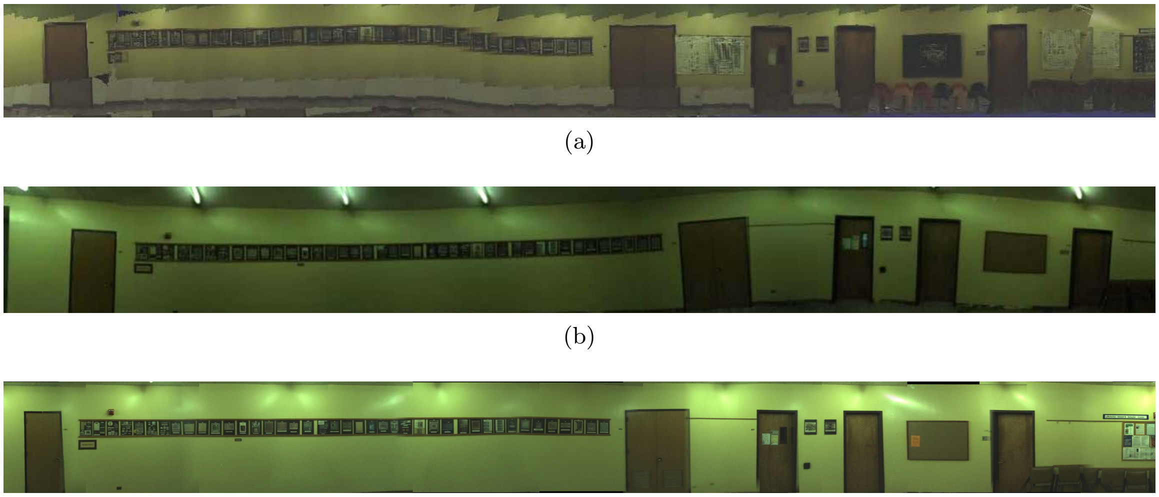


Figure 3: (a): Tile-based texturing. (b): Tile-based texturing after image alignment. (c): Tile-based texturing after image alignment with caching. (d): Shortest path texturing after image alignment). (e,f): Blending applied to (c) and (d). (g,h): Zoomed in views of discontinuities in (e) vs. in (f).



(a)

(b)

(c)

Figure 4: Texture alignment via (a) the graph-based localization refinement algorithm, (b) the AutoStitch software package, and (c) the proposed method.

3. EXISTING APPROACHES TO IMAGE ALIGNMENT

Stitching together multiple images to produce a larger, seamless image is a commonly performed task, with many successful approaches over the past years. Generally, parts of images are matched to each other, usually through direct comparisons or feature detection and matching. Images are then transformed to maximize matches, often by calculating homographies between pairs of images, or by iteratively adjusting camera poses in 1 to 6 degrees of freedom.

Feature matching has a number of advantages over direct matching that make it more suitable for our often non-planar data, and rotational differences.[6] Feature matching however, works best when multiple unique visual references exist in the environment that can be detected in multiple images. In contrast, indoor environments have a high prevalence of bare surfaces, as well as repeating textures, such as similar windows, doors, and wall-mounted decorations, 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, corresponding to long hallways and corridors as shown in Figure 4, which leads to error accumulation when image correspondences are not accurate. For example, when matching a long chain of images through homography, a pixel in the nth image must be translated into the first image's coordinates by multiplying by the 3×3 matrix $H_1 H_2 H_3 ... H_n$. Any error in one of these homography matrices is propagated to all further images, resulting in drift.

In prior work, we experimented with integrating image stitching with an iterative localization algorithm.[1] When run on long chains of images, especially where features are sparse, this approach produces distorted textures, as seen in Figure 4(a). Furthermore, this approach is not closed-form, and its iterative camera adjustment process over large datasets led to prohibitively long computation time.

The AutoStitch software package performs homography-based alignment as well, with additional provisions that attempt to reduce drift and increase efficiency.[7,8] Though AutoStitch also performs well in areas with dense features, it can not handle areas without features, and has trouble aligning wall sections with even short segments of bare texture. The example in Figure 4(b) was generated after many rounds of manual tuning, and areas with fewer visual features or repeating texture patterns simply failed outright with AutoStitch.

4. 2D IMAGE ALIGNMENT

In this section, we describe our method for efficient and robust image alignment. Rather than attempt to register all of our images in 3D, as many state-of-the-art techniques for image stitching do, we instead select a subset of images that can be aligned in 2D. The subset of images we select for alignment and subsequent texturing corresponds to the images selected by the tile-based texturing procedure described in Section 2.

We have found that applying 2D alignments to this set of images works well for a few 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 the surface being projected onto. This is because our backpack operator, during data acquisition, makes efforts to walk within a few meters of, and parallel to all walls being scanned. As a result, the translational error of camera poses is quite minimal in the dimension perpendicular to the surface being textured. Furthermore, because our set of images was selected to have minimal distance from our surface, rotational errors have a lesser effect, except for rotation around the axis perpendicular to the surface being textured. Therefore, the majority of both our translation and rotational errors occur in the 2 dimensions of the surface's plane. This of course does not apply for ceilings and floors; however, the lack of features on most ceilings and floors means that 3D misalignment does not often have a visible effect, beyond what can be fixed in 2D.

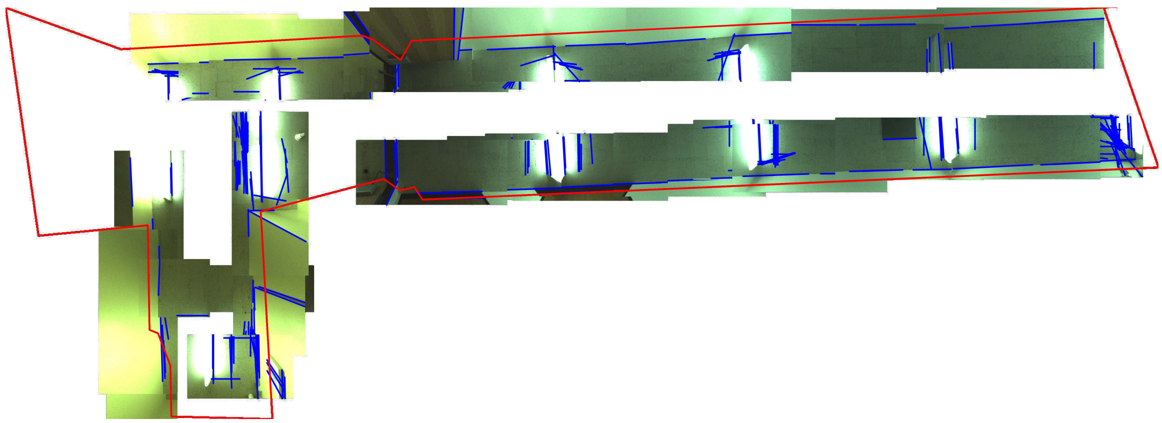
Our proposed 2D alignment procedure consists of three parts. First, images are projected onto the surface and lines within these projections are detected. These lines are then used to transform images such that they match geometry-based lines composing the boundaries of the surface being textured. Second, 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. Each step will now be explained in detail.

4.1 Geometry-based Alignment

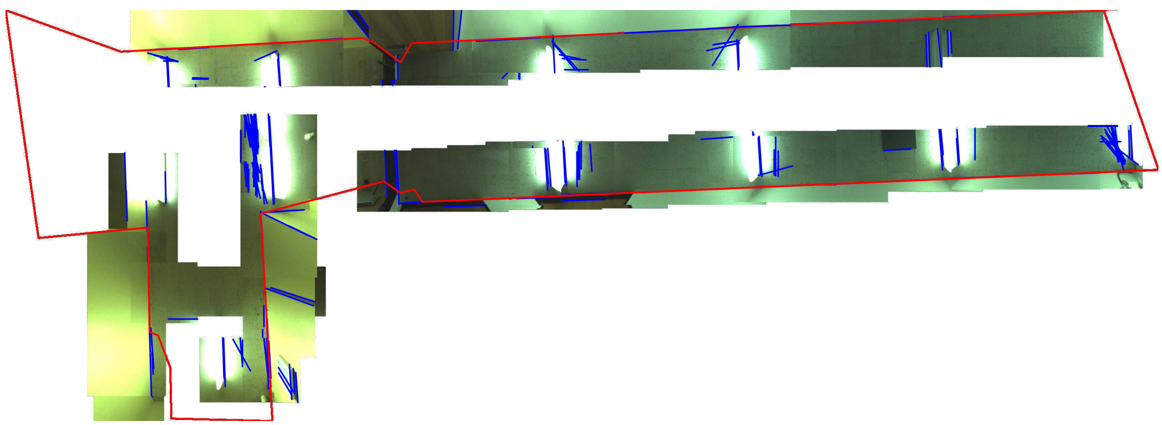
After computing each image's projection onto the target surface, as described in Section 2, we obtain a set of image-based line segments by using Hough transforms to detect lines in the image projections. We then gather a set of geometry-based lines, which correspond to the lines comprising the target surface's border, as well as lines formed where other surfaces intersect the target surface. An example of these lines is shown for a ceiling surface in Figure 5. Given perfect camera poses and surface geometry, the lines in images corresponding to corners between surfaces should match up exactly with borders and intersections of surfaces. By inducing such lines to match, we fit our camera poses more accurately to our surface, and therefore to eachother.

To perform geometry-based alignment, we collect pairs of image-based and geometry-based lines, which are within a distance and orientation difference threshold of each other. We have found a distance threshold of 250 mm and an orientation difference threshold of $10°$ to work well for our datasets. In practice, we rarely have more than 2 such pairs, but if so, we select the 2 noncollinear pairs with the most similar orientation difference and continue. With 2 noncollinear pairs, we select the pair with the longer image-based line, and perform a rotation of the image such that the lines in that pair become parallel. We then perform a translation such that the lines in that same pair overlap. This translation has ambiguity in the dimension along that line, which is resolved by matching the midpoint of the image-based line in the second pair to its corresponding geometry-based line. Thus, with 2 or more pairs, we can obtain a fixed rotation and translation, which we save for usage in Section 4.3. If we only have 1 pair, we perform a rotation and a minimal translation to match its lines. This translation's ambiguity however, can not be resolved, but is saved also to be used in Section 4.3. Finally, in the case where we have no matches, we can still perform rotations in order to exploit patterns in indoor environments. For instance, doors, windows, furniture, etc. all tend to have linear edges that are parallel to the edges of the surfaces they are on. Similarly, lights, visible interior walls, etc. which are visible in floor and ceiling images, all tend to be parallel to exterior walls corresponding to the edges of ceiling and floor surfaces. Thus, we choose to minimize the angle between image-based lines and any geometry-based line. We do this using the RANSAC framework, which successfully ignores outliers.[9]

After these steps are performed, our image projections line up well with our target surface, as shown in Figure 5. 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 performing occlusion checks.

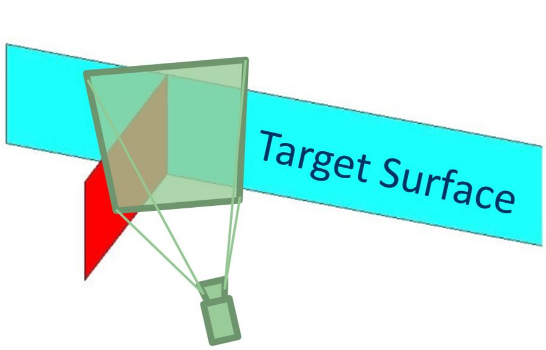


(a)

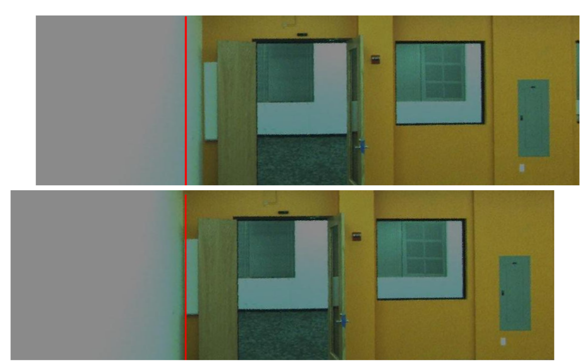


(b)

Figure 5: 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) shows images projected with their original noisy camera poses, while (b) is after images have been aligned to maximize line matches between images and geometry.



(a)



(b)

Figure 6: (a) The green image contains texture that belongs to the red surface, which should not be projected onto the blue target surface. (b) Above, 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. Below, after geometry alignment, the correct amount of texture is removed.

4.2 Image Occlusion

In order to properly texture surfaces such as the one shown in Figure ??, it is important to detect when parts of image projections contain texture belonging to other surfaces and should therefore not be used. This can be tested by recursively performing ray-polygon intersection tests between camera locations and other surfaces in a regularly spaced grid.[5] Where four corners of a rectangular region are occluded, texture is removed. Where no corners are occluded, the recursion stops. 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

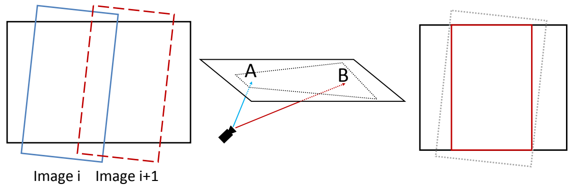
Our next step is to align images to each other by searching for corresponding feature points between all pairs of overlapping images. We use feature alignment rather than pixel or intensity-based alignment due to the differences in lighting as well as possible occlusion among our images, both of which feature alignment is less sensitive to.[6,10,11] We use SiftGPU[12] 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 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 200 mm from what the noisy poses estimate. In order to utilize the remaining feature matches robustly, RANSAC[9] is again used to estimate the optimal $dx_{i,j}$ and $dy_{i,j}$ distances between two images i and j. We use a 5 mm threshold for RANSAC, so that SIFT matches are labeled as outliers if their distance is not within 5 mm of the sampled average distance.

We now use the feature-based distances between each pair of images as well as our previous geometry alignment results to refine all image positions using a weighted linear least squares solution. The variables we wish 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. The original camera poses are needed because we sometimes are not able to detect feature matches in all of our images, or we lack enough geometry alignment results to generate a single solution. Because we want to minimally use our original poses, we give them a weighting factor of 0.01, while all other equations are weighted at 1. An example setup for a weighted linear least squares problem $\min_{\vec{\beta}} ||W^{\frac{1}{2}}(A\vec{\beta} - \vec{\gamma})||_2^2$ with 3 images is shown below.

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

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 4.1. 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 and 9 correspond to the original camera pose for image 1, and original poses for images 2 and 3 are omitted for brevity.

Because our problem is linear, it is easily and quickly solved, and after applying the resulting 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 parallax effects are still visible.



(a) (b) (c)

Figure 7: (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.

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 images have consistently perpendicular viewing angles, we propose a superior method that selectively 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). However, some discontinuities are still present, as visible in the posters on the wall with breaks in their borders.

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 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).

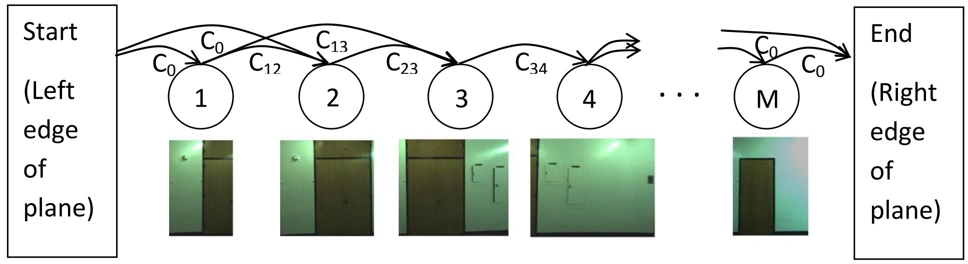


Figure 8: DAG construction for the image selection process.

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.

In the general case, 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. 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 crop our image projections to be rectangular with minimal texture loss. Furthermore, because our fisheye camera lenses have full floor-to-ceiling coverage of nearly all walls, and our backpack operator logically only moves horizontally, we only need to ensure lateral coverage of our wall planes. 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.[13]

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 very 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. 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 200 mm 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. The remaining images in Figure 3 highlight differences between the two methods, showing that Figure 3(f) has the best visual quality and the best texturing approach 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. These results are as accurate and clean as the best results we obtained using the two algorithms mentioned in Section 3. High resolution colored texture comparisons, as well as video and interactive walkthroughs of full models are available at [‡].

As mentioned earlier, our approach is quite efficient. The top wall in Figure 8(a) was generated with 7543×776 pixels, and spans a 40-meter long wall. Given 41000 input images of the entire dataset, a 2.8GHz dual-core consumer-grade laptop takes approximately a minute to pick 36 candidate images, followed by under a minute to perform both image alignment and the shortest path texturing method, though over 75% of that time is spent calculating SIFT matches within the SiftGPU framework, which could feasibly be split into a separate preprocessing step. While not real-time, the process is capable of generating quick updates after changes in various parameters or modifications to input data, and if integrated directly into a modeling system, could provide live visualization as data is collected. Our full models consist of our input model file, textures that we generate, and a mapping of image points to model vertices. The models shown in Figure 8 are roughly 20 MB in size, and are visualized using the OpenSceneGraph toolkit,[14] which allows for export to many common model formats, as well as efficient visualization, even in web browsers or mobile devices.

[†] 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/indoormapping`

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 similar systems across multiple environments.

While our method works well on our datasets, it is likely to fail in areas where 3D error is much higher. A technique for resolving camera error in 3D that would be a logical progression of our approach is to perform 3D matching between image lines and geometry in 3D, which can be done reasonably efficiently.[15,16] Using linear features on top of our SIFT features is also likely to result in improved matches, as indoor scenes often have long, unbroken lines spanning multiple images.[17] Finally, our blending procedure is very basic, and applying more sophisticated methods of blending as well as normalization would benefit our final visual quality, and more robustly handle motion-based or parallax errors.

REFERENCES

[1] 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).

[2] 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).

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

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

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

[9] 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).

[10] 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).

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

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

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

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

[15] 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).

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

[17] 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).

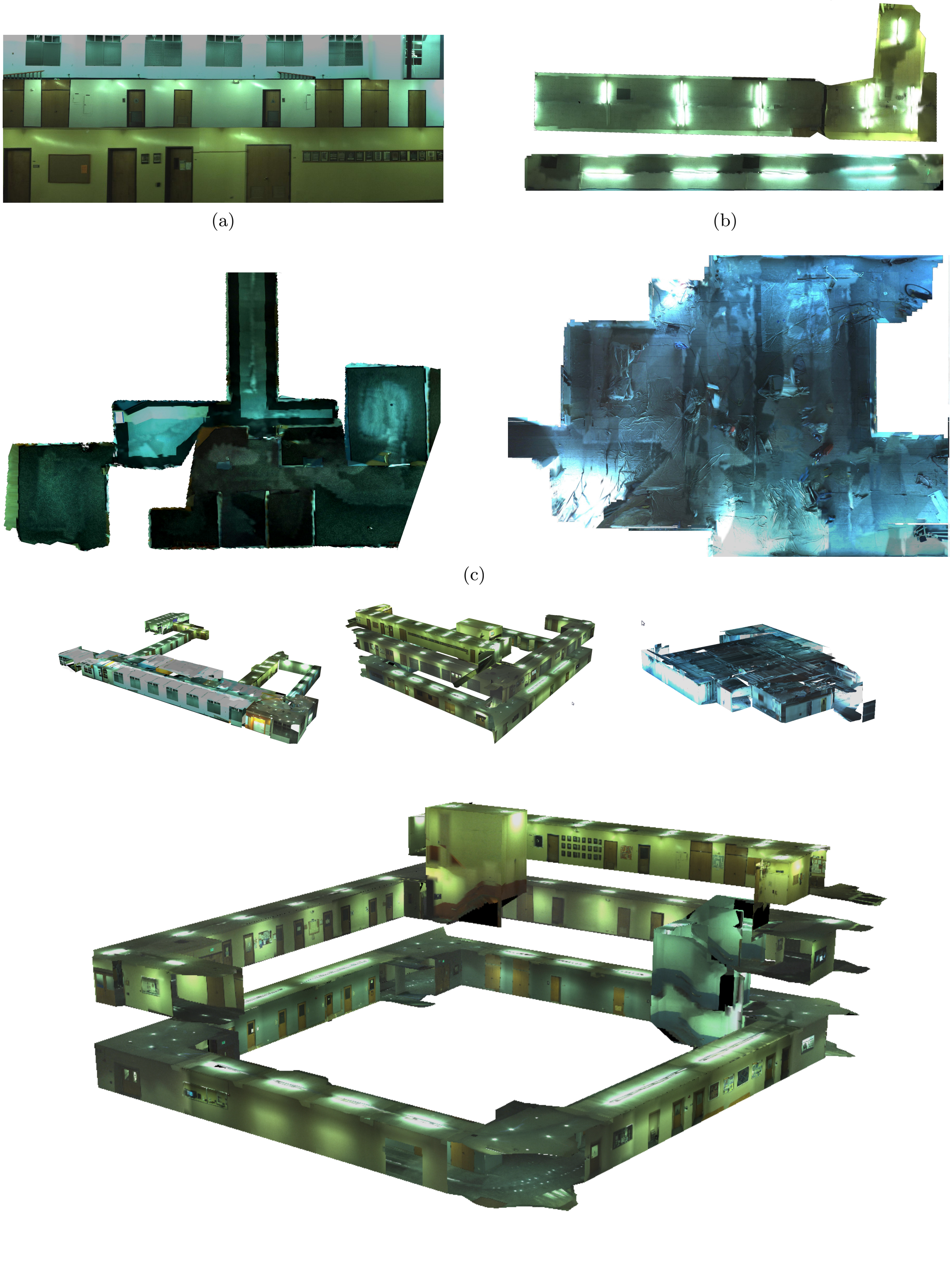


Figure 9: Examples of our final texture mapping output for (a) walls, (b) ceilings, (c) floors, (d) full models.