Alignment of 3D Point Clouds to Overhead Images

Ryan S. Kaminsky[1] Noah Snavely[2] Steven M. Seitz[1] Richard Szeliski[3]
University of Washington[1] Cornell University[2] Microsoft Research[3]

Abstract

We address the problem of automatically aligning structure-from-motion reconstructions to overhead images, such as satellite images, maps and floor plans, generated from an orthographic camera. We compute the optimal alignment using an objective function that matches 3D points to image edges and imposes free space constraints based on the visibility of points in each camera. We demonstrate the accuracy of our alignment algorithm on several outdoor and indoor scenes using both satellite and floor plan images. We also present an application of our technique, which uses a labeled overhead image to automatically tag the input photo collection with textual information.

1. Introduction

Recent progress in structure-from-motion (SfM) has led to robust techniques that can operate in extremely general conditions [20, 21]. However, a limitation of SfM is that the scene can only be recovered up to a similarity transformation (when camera intrinsics are available, or when autocalibration is used [24]). This paper addresses the problem of geo-referencing an SfM reconstruction by automatic alignment with a satellite image, floor plan, map, or other overhead view.

Solving this alignment problem has a number of interesting applications that we explore in this paper. For example, Figure 1 shows an image from the Photo Tourism paper [20] where an SfM reconstruction has been *manually* aligned to a satellite image, enabling an enhanced 3D browsing experience. Our objective is to *automate* this process and to explore a more general range of alignment tasks.

The fact that humans are able to perform this alignment task suggests they are using visual cues that an automated algorithm could also exploit. The most evident of these cues is *point-to-edge* agreement—the points in the SfM reconstruction, when properly aligned, coincide with boundaries of objects (i.e., edges) in the overhead image, as seen in Figure 1. This works because architectural scenes tend to contain vertical walls that appear as lines in overhead views.

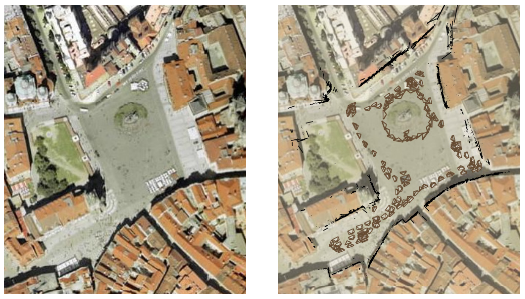


Figure 1. Left: satellite image of a town square in Prague, Czech Republic. Right: manual alignment of an SfM reconstruction (points and camera frustra) of the same area to the satellite image from Photo Tourism [20]. Our goal is to automate this process.

While this is indeed a powerful cue, image edges are notoriously noisy and often correspond to non-geometric factors such as shadows, surface markings, etc. Another powerful (but perhaps less obvious) cue is lack of boundaries, or *free-space*. The presence of a large open area, for example a square, park, or road, is a strong location cue that can be useful for the alignment task. Many such free-space regions can be detected both in the satellite image (relative lack of edges in the square of Figure 1), and in the SfM point cloud (visibility constraints due to 3D camera-point observations). Our alignment approach exploits both point-to-edge and free-space cues to obtain robust alignments over a wide range of scenes.

Our system offers a fully automated pipeline for creating geo-referenced scene models: given a user-specified search term (e.g., "Prague old town square"), the system downloads images from Flickr [8] using keyword search, matches and reconstructs the images via SfM, downloads a corresponding satellite image, and aligns the image and SfM point cloud. In the case of indoor scenes, our system also enables matching to blueprints and floor plans (although in this case, we assume the blueprint or floor plan is provided explicitly, as an additional input to the system).

Our approach is based on defining energy functions for the point-to-edge and free-space cues, and minimizing this energy using a discrete search over a 4D parameter space (2D position, orientation, and scale). In many cases, rough estimates of some of these parameters can be found by incorporating geotags (location information associated with some photos), and/or computing moments of the model and image; we show how such estimates enhance the performance of the method. Our approach works remarkably well in practice, and we show successful alignments for a wide range of indoor and outdoor scenes.

We also demonstrate how our technique can be used to automatically apply labels (or tags) to a photo collection given additional contextual information provided with the map or floor plan. Once an SfM reconstruction is registered with an overhead image, our auto-tagging application automatically transfers labels from the image to individual objects in the input photos.

2. Related Work

We seek to solve an alignment problem between a 3D point cloud and an overhead image; specifically, we align the 2D projection of the point cloud onto a ground plane with the overhead image. As such, this paper relates to previous work on 3D to 3D alignment, 3D to 2D alignment, and 2D to 2D alignment, as follows.

The problem of aligning two 3D models is well studied, with perhaps the best known method being Iterative Closest Points (ICP) [4, 26]. ICP alternates between estimating a point correspondence and solving for the best rigid alignment between the models, repeating until convergence. While ICP can also be defined in 2D, a limitation of ICP is that it requires a good initialization to converge. Additionally, there is not an easy way to factor in visibility constraints in the ICP procedure, as we advocate in this paper. Finally, while ICP has been used to align SfM-derived point clouds to 3D sensor data [27], our approach can handle alignments to much noisier types of data, such as edge maps derived from satellite images.

Classical work on object recognition addresses the problem of matching projections of 3D models to 2D images. Examples (among many) include the alignment work of Huttenlocher and Ullman [12] and Lowe's viewpoint consistency constraint [16] which seek to match projections of a known 3D model to 2D edge images. While we also seek to align a 3D model to a 2D image, there are a number of important differences which make it difficult to apply these methods to our problem. First, traditional methods assume a clean, correct 3D model with known contours that produce edges when projected. We are not aware of prior recognition work that operates on SfM-derived 3D point clouds—these point clouds represent a very incomplete and sparse scene representation. Second, prior work has focused on simpler scenes typically consisting of a single object, where we seek to align images of large urban environments. And lastly, the nature of our problem enables us to reduce the alignment problem to 2D by operating on overhead views.

The third category of related work is 2D shape to image matching, another well-studied topic in the literature, with popular methods that include chamfer matching, Hausdorff matching [11], and shape context [3]. These and many others could be used to help solve our problem, indeed, we incorporate a chamfer term in our edge cost. These prior methods rely on shape descriptors for matching. In this paper, we additionally exploit *free-space* information—not available to the above methods—that is derived from the distribution of camera centers and feature points. We have found that this visibility information is crucial in many cases to find correct alignments.

Also related to our work are techniques that correlate SfM data to maps. Robertson and Cipolla treat a map as an affine view of a scene, and use manually-specified correspondences between ground-level images and a map to recover more accurate scene geometry [18]. Levin and Szeliski use a rough, user-sketched map to refine a camera path computed using SfM, by solving for a temporal correspondence between the video path and the map [14]. Again, these techniques correlate SfM geometry to clean, manually-specified map data, and are not directly applicable to unsegmented satellite imagery.

There is also a body of related work in georegistering laser scans. Ding et al. [5] align LiDAR scans with oblique aerial imagery by detecting and matching corners, while Früh and Zakhor [10, 9] register aerial and ground-level scans. The dense 3D geometry used in these techniques allow for much more robust detection of geometric primitives such as edges and corners for matching. Our challenge is to handle much sparser point clouds and cluttered, real-world aerial imagery. Additionally, we introduce a visibility term that can significantly improve registration.

3. Problem Formulation

Our system takes two inputs. The first is an SfM reconstruction of a scene, consisting of a set of reconstructed 3D points, $P = \{p_1, p_2, \ldots, p_n\}$, a set of recovered camera locations, $C = \{c_1, c_2, \ldots, c_m\}$, and, for each camera, a list of 3D points visible to that camera (represented as a visibility operator V_{ij} equal to 1 when point p_i is visible in camera c_j and 0 otherwise). The second input is a binary, overhead image $B(x, y)$ of the same scene representing important structures (e.g., walls). B can be obtained from a variety of sources, e.g., an edgemap computed from a satellite or aerial image or a vector street map (for outdoor scenes), or a floor plan (for indoor scenes). B is assumed to be an orthographic view looking directly down on the scene.

In a preprocessing step, we project the point cloud onto the ground plane to create a 2D point cloud, producing a *point cloud image*. The ground plane normal (i.e., the scene up vector) is computed using the method of Szeliski [23]. In this way, we reduce the problem to a 2D to 2D alignment

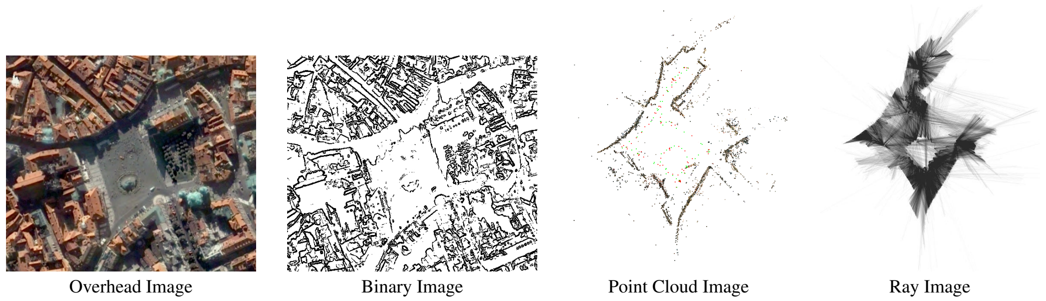


Figure 2. Example of inputs to the edge cost and free-space cost for the Prague dataset. The left two figures show the overhead image and the binary image B created from it. The two figures on the right demonstrate a point cloud viewed from above and the associated ray image R with inverted values for visualization (lighter pixels have lower values, while darker pixels have higher values). The binary image and the point cloud are inputs to the edge cost, while the binary image and the ray image are used for the free-space cost.

problem. In the remainder of the paper, we treat each point $p \in P$ as a (projected) 2D point (x, y).

Given these inputs, the goal is to correctly align the SfM reconstruction (in the form of the point cloud) to the overhead image. In particular, we seek the 2D similarity transform T that best maps the reconstruction to B. T has four degrees of freedom: a 2D translation (x, y), an in-plane rotation θ, and a uniform scale s. To make these parameters explicit, we refer to a given transform as $T_{x,y,\theta,s}$. To find $T_{x,y,\theta,s}$, we define an energy function (called the *alignment cost*, $A(x, y, \theta, s)$) that measures the fit of a transformed reconstruction to the overhead image B.

The alignment cost consists of two terms: an *edge cost*, E, and a *free-space cost*, F. The edge cost measures how closely points in the point cloud image correspond to edges in the overhead image B, while the free-space cost estimates how many free-space violations occur given the known visibility relationship between points and cameras. We now discuss the two costs in detail.

3.1. The edge cost

The edge cost is based on the idea that points in the SfM reconstruction, when properly aligned, coincide with boundaries of walls and other objects (i.e., edges) in the overhead image, and is similar to the objective function commonly used in ICP. We define the edge cost to be the average of the L_2 distance between each (transformed) point in the point cloud image and the spatially closest point (i.e., nearest pixel with value 1) in the binary image B:

$$E(i, j, \theta, s) = \frac{1}{n} \sum_{p \in P} \min_{(x,y) \in B_1} ||T_{i,j,\theta,s}(p) - (x, y)||_2 \quad (1)$$

where B_1 is the set of pixels (x, y) such that $B(x, y) = 1$. Figure 2 shows an example of a binary image created from a satellite image, as well as a point cloud.

Lower values of the edge cost indicate that more of the feature points lie on or close to edges, implying a potentially better alignment. To compute E efficiently, we first compute a distance transform DT_B of B [6]. The distance transform DT_B is a lookup table mapping a pixel (x, y) to the distance to the nearest pixel with value 1 in B.

There are several limitations to using only the edge cost for alignment. First, overhead images derived from satellite images have many extraneous edges, compared to the points in the SfM reconstruction. For example, the binary image B may contain several edges on a building's roof, but the roof may not be visible from the position of the cameras that created the 3D point cloud. Thus, the transform with the minimum edge cost might correspond to an incorrect alignment that happens to have many accidental matches between points and extraneous edges. Second, the scale parameter of the transform would be optimal when the point cloud converges to a single pixel, which is then translated onto an edge in the binary image B. To address these limitations we introduce the free-space cost.

3.2. The free-space cost

The free-space cost is based on the idea that the path between a camera and any point visible to the camera should be free of occluders, as otherwise the visibility of the point would be violated. To compute the free-space cost, we first create an image we call the *ray image*, R for each scale we are searching over. This image is created by tracing a ray from each camera location c_j in the point cloud image to each the point p_i that is deemed visible to that camera during the SfM computation (i.e., $V_{ij} = 1$). Each pixel in the ray image along the line segment from c_j to p_i is incremented. Once all source locations are processed, each pixel $R(x, y)$ in the ray image will have the value of k, the number of rays that pass through that pixel or 0 if no rays pass through it. Figure 2 shows an example of a ray image.

Next, for each transform of the point cloud image, we superimpose the transformed ray image R onto the binary image B. The free-space cost, F is computed by examining each edge pixel in the binary image B and adding the value of the pixel at the corresponding location in the ray image

R. F is the sum of the pixelwise product of the transformed ray image and the binary image. The intuition is that if an edge in the overhead image lies on a pixel in the ray image that is part of any ray ($R(x, y) > 0$), the edge should be penalized as an occluder that violates the free-space assumption. The free-space cost F is thus defined as:

$$F(i, j, \theta, s) = \frac{1}{n} \sum_{(x,y)} R(T_{i,j,\theta,s}(x, y))B(x, y) \quad (2)$$

where the normalization factor n is the number of pixels in the binary image B where $B(x, y) = 1$.

Lower values of this cost imply fewer occlusions between a camera and its feature points, which implies a better alignment. Also, note that we are computing the free-space cost over translations, rotations and scales. Had we only been searching over translations, we could have simply computed the convolution of the binary image and ray image or, similarly, the convolution of the point cloud image and the distance transform for the edge cost. Instead, we use a coarse-to-fine approach as described in Section 4.

There are limitations to the free-space cost as well. First, while we assume that satellite images are orthographic, in practice they usually exhibit a slight angle. This can cause higher penalties on the free-space cost as the edges of the angled vertical surfaces "creep" into free space areas of the ray image. Second, we assume that if a ray passes through an edge in the binary image, there is a free-space violation. However, this may not be the case if the ray passes *over*, rather than through, an object, or if the ground plane is textured. Despite these limitations, we observe that the free-space cost for the best alignment is usually still better than the cost for incorrect alignments. Like the edge cost, the free-space cost also prefers transforms with smaller scales, as scaling the entire reconstruction down to a small, textureless area of the overhead image can result in a very low free-space cost. In the next section, we combine both costs to reduce the impact of the limitations of each cost.

3.3. The alignment cost

The free-space and edge costs each tend to balance out the limitations of the other. While the edge cost seeks to maximize the match of points in the point cloud to edges, the free-space cost minimizes the violation of free space constraints by edges. In particular, the problem with each cost preferring small scales is neutralized by considering both costs, which also yields more robust alignments overall. Thus, the final alignment cost is a linear combination of F and E, combined using a weighting factor α:

$$A(i, j, \theta, s) = \alpha\, F(i, j, \theta, s) + (1 - \alpha)\, E(i, j, \theta, s) \quad (3)$$

where $0 \leq \alpha \leq 1$. We seek to minimize this function to compute the optimal alignment of the point cloud image to the overhead image.

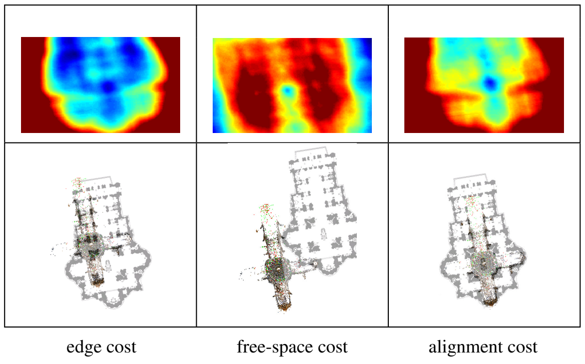


Figure 3. Comparison of the components of the alignment cost for St. Peter's dataset. Scale and rotation are fixed for easier visualization. The first row shows cost maps, which plot the cost for each x and y translation. The second row shows the best scoring alignment. The range blue to red represents low to high costs. Neither the edge cost nor free-space cost alone are sufficient to determine the optimum alignment, which the combined alignment cost does find. The blue spot near the center is the correct alignment.

Figure 3 shows a visualization of the alignment cost and its individual components. Notice that the edge cost prefers an alignment where the dense feature points in the dome are aligned over the edges of a side chamber, without regard for the free space. Additionally, although we could have ruled out the free-space cost false minimum by decreasing the search range, this example illustrates that the free-space cost by itself is susceptible to false minima due to large empty regions of the binary image. Requiring that the point cloud also align with image edges solves this problem, as demonstrated by the selection of the optimal alignment based on the alignment cost.

4. Alignment Algorithm

The optimal alignment could be found by computing the alignment cost for all possible transforms of the point cloud in a brute force fashion and choosing the transform receiving the minimum cost. However, this search space can be very large. For a 1000 by 1000 image, searching across 180 rotations and 10 scales results in 1.8 billion possible cost computations. By using a slightly more efficient approach, we can reduce this space while not significantly impacting the results. In this section we detail the algorithm and show our results on several datasets. In the next section, we show how using prior information can improve our results.

To reduce the search space of possible transforms, we first search at a coarse granularity to identify areas where the optimal alignment may reside. This involves using larger steps when searching the space for the optimal translation. For example, instead of searching at single pixel increments, we search at intervals of 10 to 20 pixels during the coarse search phase. This particular interval range worked

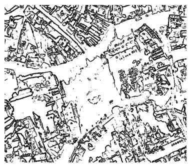
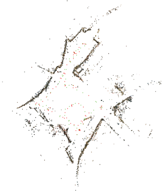
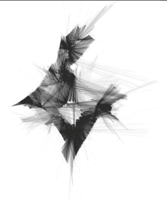
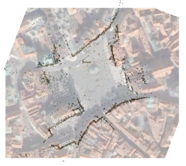
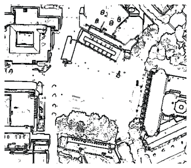
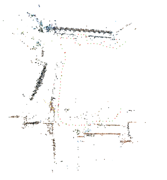
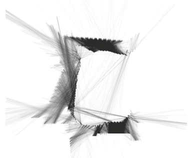
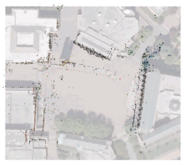
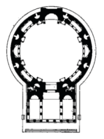
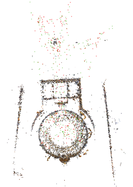
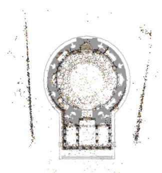
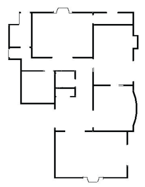
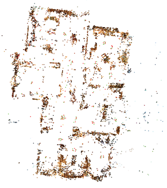
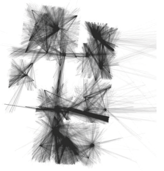
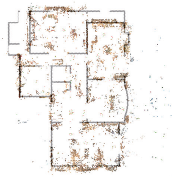
Dataset Info.	Overhead Image	Binary Image	Point Cloud	Ray Image	Alignment Result
Prague # of images 168 # of points 15647 Error % 0.43 Time 67m					
Red Square (UW) # of images 114 # of points 11870 Error % 1.09 Time 78m					
Pantheon # of images 592 # of points 89904 Error % 0.28 Time 67m					
House # of images 158 # of points 13467 Error % 0.65 Time 19m					

Figure 4. Successful alignment results from the algorithm in Section 4. First two rows show outdoor scenes with satellite images, last two rows show indoor scenes with floor plans. Zoom into the PDF file to see alignment results (last column) at higher resolution. More challenging cases are presented in the next section.

well for our datasets, but we could try a more classic octave sampling approach in the future. During this phase, we also keep track of the top k (5 to 10 proved effective) transforms $T_{i,j,\theta,s}$ that produce the smallest alignment costs.

For the refinement step, we adopt the standard coarse-to-fine approach and search within a window surrounding the k transforms that the coarse search identified as possible optimal alignments. We use a window size large enough to guarantee that we search the space of possible transforms within the "step size." For example, if our coarse step size was 10 pixels for the x and y translations, in the refinement phase, we search with a range of ± 10 pixels, a 20×20 search window. The rightmost column of Figure 3 shows a visualization of the alignment cost at single pixel translation increments over all rotations and at constant scale. This visualization implies a descent toward the minimal alignment cost (also the optimal alignment) that makes our coarse-to-fine search likely to succeed for many datasets.

We show the results of our system on several datasets in Figure 4. The first column provides statistics on each dataset as well as information about the alignment, including the average pixel error as a percentage of the image height. We compute this by finding the average distance between corresponding 3D points in the ground truth (obtained through manual registration) and computed alignments. The next four columns display the inputs as described in Section 3. The final column shows the resulting alignment of the point cloud overlaid onto the overhead image from the second column.

5. Exploiting Prior Alignment Information

Our system performed well on the datasets presented in Figure 4, but did not perform as well on others. The 5^{th} columns of Figure 5 and the 4^{th} column of Figure 6 show the incorrect alignments our system produces. We now discuss these failures, as well as methods to overcome them.

5.1. Using GPS information

A growing number of photos have embedded location information in the form of latitude/longitude *geotags*. Photos can be geotagged in a number of ways, typically via GPS-

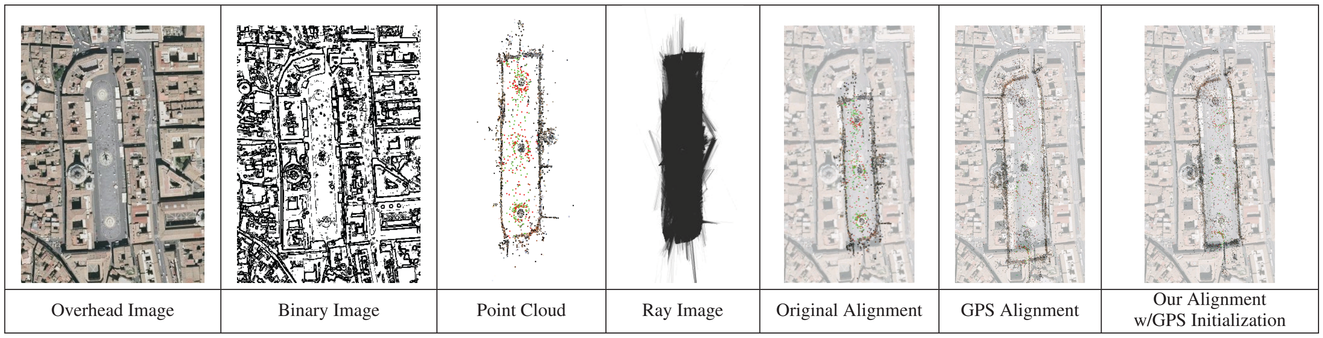


Figure 5. Inputs and results for the Piazza dataset (532 images, 74233 points). Our algorithm incorrectly computed an alignment with a 17.05% error, mainly due to the 180 degree rotation error (shown in column 5). The alignment computed using only GPS is shown in column 6 and has an error of 2.01%. The final column shows that our GPS-initialized results produce the best alignment (0.45% error).

equipped cameras and mobile phones or photo-sharing sites such as Flickr and Picasa Web Albums that allow users to drag images onto a map. When available, we can exploit geotags to compute a rough initial alignment of the scene to the model. Our approach is to solve for a 2D similarity transform between the SfM-derived camera positions and the geotagged positions. The geotags are first converted to meters as described in [1]. Because geotags can be very inaccurate (especially when manually specified), we use a RANSAC [7] approach to reduce the effect of outliers.

Given this initial alignment, we search a range of 75% to 125% of the initial scale estimate and $\pm$ 25 degrees of the initial rotation. Figure 5 shows the inputs of the Piazza dataset and compares the results of our original algorithm, GPS alignment only and our algorithm using GPS initialization. The symmetry of the scene causes a 180 degree error in the rotation parameter, resulting in the selection of a sub-optimal scale. The point cloud for this dataset was constructed using 532 images, of which 23 were geotagged and 15 used as inliers to calculate the GPS alignment. The original algorithm ran in 59 minutes, while the GPS-initialized version completed in 19 minutes.

The accuracy of GPS is on the order of 5-10 meters (often larger in confined areas or poor environmental conditions). Given enough GPS tags, this accuracy is sufficient for reasonable initialization of the transformation. However, while the initialization we get from geotags is often quite close to the true solution, for some datasets the initial scale or orientation is far from correct. These errors stem from the fact that many existing geotags do not actually come from GPS, but from manual placement of images on a map. Thus, the accuracy of tags within a datset can vary widely, depending on the proportion of true GPS tags and the accuracy of manual placement. Cameras with built-in GPS (e.g. the Nikon P60000 and Ricoh 500SE) are beginning to appear, however, so we anticipate that a greater number of accurate geotags will be available in the future.

5.2. Estimating scale from floor plans

We can use characteristics of indoor scenes to obtain a reasonable estimate for scale. In general, floor plans are well segmented into the area of interest and the same structures (walls) are likely to be present in both the floor plan and an indoor point cloud. By contrast, satellite images contain many structures that are not represented in the point cloud. We can use the fact that indoor point clouds and floor plans often cover similar areas to aid in scale estimation.

We first compute the mean and standard deviation for the point cloud and the binary image B. We then use the ratio of the standard deviation of the binary image to that of the point cloud as an initial scale estimate and search in the range of 50% to 125% of this value. The lower part of this range is larger because the point cloud usually encompasses only a subset of the area of the floor plan. Thus the ratio is likely to over-estimate the scale of the point cloud, hence the lower search bound. Figure 6 demonstrates the improvement on the St. Peter's dataset when using the prior scale information as part of our search. The original algorithm took 52 minutes to run, compared to only 26 minutes when searching over the smaller scale range.

6. Alignment Pipeline

When GPS tags are available for an outdoor image collection, then the entire pipeline, including downloading the satellite image, is completely automated. A user simply enters a search term on Flickr corresponding to the site of interest (e.g., "Prague Old Town Square"), and the system downloads the images, finds the geotags, computes the initial alignment using RANSAC, downloads the corresponding satellite image with the specified range of latitude/longitude values, extracts an edge image, and runs our alignment algorithm on this image with GPS initialization.

To automatically acquire a satellite image given the initial RANSAC alignment, we compute an area of interest by using the point cloud and camera positions to find the

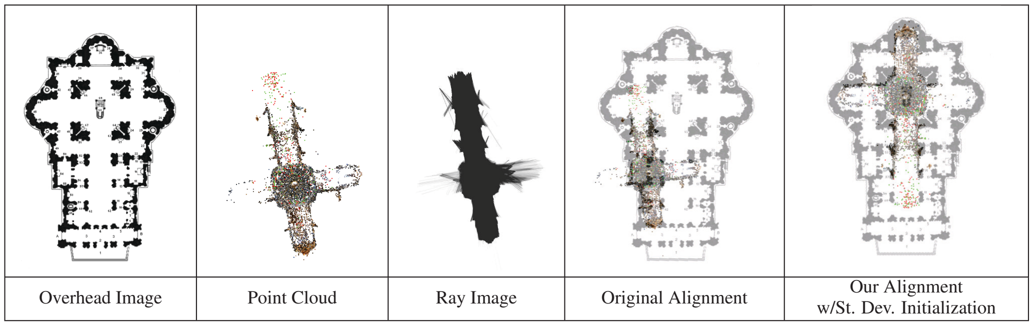


Figure 6. Inputs and results for the St. Peter's dataset (380 images, 43507 points). Our algorithm computes a poor alignment (27.6% error) for St.Peters when not using scale initialization (shown in column 5). The final column shows our improved alignment algorithm when using the standard deviation for scale initialization (0.42% error).

bounding box (in latitude/longitude coordinates) that contains 90% of the points. This bounding box is then expanded by a factor of two. We download the corresponding satellite image from Live Search Maps [15] using the ModestMaps API [17].

7. Auto-Tagging Photos

Once we have registered an SfM reconstruction to a floor plan, map, or satellite image, we can leverage the wealth of additional context that accompanies the map for a host of applications (including the Photo Tourism application [20] shown in Figure 1). In this section, we focus on one application in particular: using tagged objects in the map to automatically tag objects in the input photos.

There are many rich sources of tags for overhead images: road and point-of-interest data on Internet mapping sites, region tags on sites like like Wikimapia [2], textural and geotag information on Wikipedia [25], to name a few. In addition, floor plans of famous buildings are often annotated with locations and descriptions of objects.

For example, the floor plan for St. Peter's used to register the St. Peter's reconstruction (shown in Figure 6) is annotated with numbers indicating locations of important statues and monuments, and the webpage where we obtained this image has a key describing each object [22]. We use this information to automatically tag the photos in the St. Peter's dataset. Several autotagged photos are shown in Figure 7.

To autotag each image, we check, for each tag (number on the map), whether any 3D points visible to that image are physically close to the location of that tag on the map (we use a threshold of 6 pixels in the coordinate system of the floor plan, though this could also be specified in meters given the dimensions of the building; we could also use a footprint of the object if known). If such points exist, we compute the centroid of their projections into the image, and draw the name of the object at that image position. We restrict 3D points considered to be lower than a certain height (we assume objects in the scene are close to the ground).

While autotagging of photos has also been explored by other authors [13, 19], the novelty of our approach is the ability to automatically import data from existing *2D* maps and annotated floor plans, resources which are very prevalent on the Internet.

8. Discussion

We have presented a system for aligning SfM-derived 3D point clouds with overhead images (e.g. satellite maps and floor plans). We develop an objective function that exploits the characteristics of architectural datasets that leads to accurate alignments. Our results improve when computing prior information based on GPS information and statistics. Our system performs well on almost all of our datasets.

While we've found this approach to work remarkably well across a broad range of indoor and outdoor scenes, there are also important limitations and failure cases. Figure 8 shows one such failure case where the presence of many image edges in the interior of the Colosseum violates our 2D free-space model (as described in Section 3.2). A more sophisticated free-space model that takes into account the 3D structure of the point cloud and cameras could remedy this problem. Our implementation is a prototype that runs on Java and is not optimized for speed, resulting in run times that sometimes take hours. There are many opportunities for accelerating the algorithm which could be explored in the future (e.g., by extracting descriptors or performing convolutions in the Fourier domain).

Acknowledgements

This work was supported in part by National Science Foundation grant IIS-0811878, the Office of Naval Research, the University of Washington Animation Research Labs, and Microsoft Corporation.

References

[1] http://www.uwgb.edu/dutchs/usefuldata/utmformulas.htm. 6

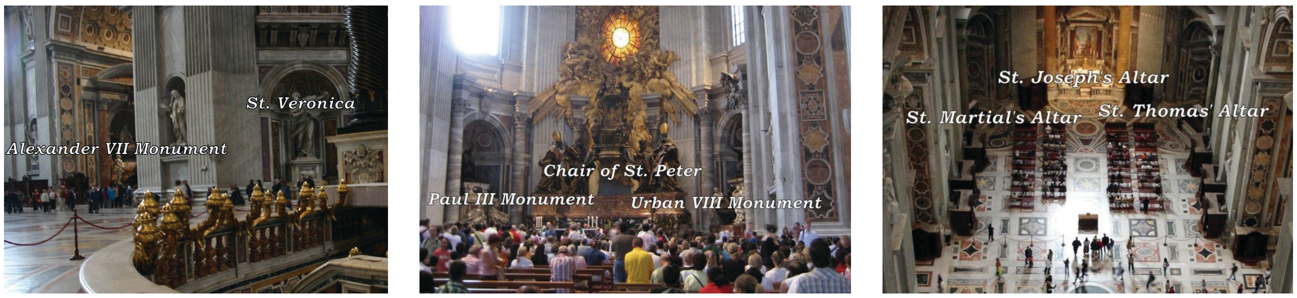


Figure 7. Several images from St. Peter's Basilica automatically tagged by transferring information from the floor plan.

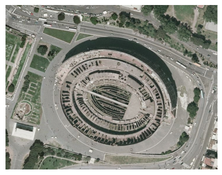
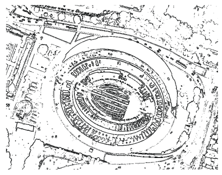
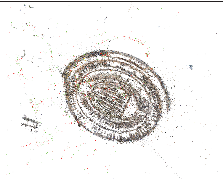
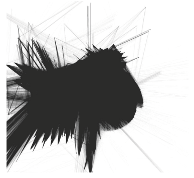
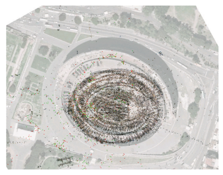
				
Overhead Image	Binary Image	Point Cloud	Ray Image	Our Alignment w/GPS Initialization

Figure 8. Inputs and results for Colosseum dataset (1016 images, 390587 points). Our alignment produces an error of 2.33% even when using GPS-initialization (column 5). This failure case demonstrates a limitation when areas of free space coincide with heavy edges.

[2] http://www.wikimapia.org/. 7

[3] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. 24:509–522, 2002. 2

[4] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, 1992. 2

[5] M. Ding, K. Lyngbaek, and A. Zakhor. Automatic registration of aerial imagery with untextured 3d lidar models. *CVPR 2008.*, pages 1–8, June 2008. 2

[6] P. F. Felzenszwalb and D. P. Huttenlocher. Distance transforms of sampled functions. *Tech. Rep., Cornell Computing and Information Science*, 2004. 3

[7] M. Fischler and R. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Readings in computer vision: issues, problems, principles, and paradigms*, pages 726–740, 1987. 6

[8] Flickr. `http://www.flickr.com`. 1

[9] C. Fruh and A. Zakhor. 3d model generation for cities using aerial photographs and ground level laser scans. pages II:31–38, 2001. 2

[10] C. Früh and A. Zakhor. An automated method for large-scale, ground-based city model acquisition. *Int. J. Comput. Vision*, 60(1):5–24, 2004. 2

[11] D. P. Huttenlocher, G. A. Kl, and W. J. Rucklidge. Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:850–863, 1993. 2

[12] D. P. Huttenlocher and S. Ullman. Recognizing solid objects by alignment with an image. *Int. J. Comput. Vision*, 5(2):195–212, 1990. 2

[13] J. Kopf, B. Neubert, B. Chen, M. Cohen, D. Cohen-Or, O. Deussen, M. Uyttendaele, and D. Lischinski. Deep photo: Model-based photograph enhancement and viewing. In *SIGGRAPH Asia Conf. Proc.*, 2008 (to appear). 7

[14] A. Levin and R. Szeliski. Visual odometry and map correlation. In *CVPR*, pages 611–618, 2004. 2

[15] Live Search Maps. `http://maps.live.com`. 7

[16] D. G. Lowe. The viewpoint consistency constraint. *Int. J. Comput. Vision*, 1(1):57–72, 1987. 2

[17] Modest Maps. `http://modestmaps.com`. 7

[18] D. P. Robertson and R. Cipolla. Building architectural models from many views using map constraints. In *Proc. ECCV*, volume II, pages 155–169, 2002. 2

[19] I. Simon and S. M. Seitz. Scene segmentation using the wisdom of crowds. In *Proc. ECCV*, 2008. 7

[20] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3D. In *SIGGRAPH*, pages 835–846, 2006. 1, 7

[21] N. Snavely, S. M. Seitz, and R. Szeliski. Skeletal sets for efficient structure from motion. In *Proc. CVPR*, 2008. 1

[22] St. Peter's Basilica floor plan. `http://www.stuardtclarkesrome.com/floor.htm`. 7

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

[24] B. Triggs. Autocalibration and the absolute quadric. In *CVPR*, pages 609–614, 1997. 1

[25] Wikipedia. `http://www.wikipedia.org`. 7

[26] Z. Zhang. Iterative point matching for registration of free-form curves and surfaces. *Int. J. Comput. Vision*, 13(2):119–152, 1994. 2

[27] W. Zhao, D. Nister, and S. Hsu. Alignment of continuous video onto 3d point clouds. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 27(8):1305–1318, 2005. 2