International Journal of Computer Vision 61(2), 159–184, 2005

Data Processing Algorithms for Generating Textured 3D Building Facade Meshes from Laser Scans and Camera Images

CHRISTIAN FRUEH, SIDDHARTH JAIN AND AVIDEH ZAKHOR
Video and Image Processing Laboratory, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley
frueh@eecs.berkeley.edu
morpheus@eecs.berkeley.edu
avz@eecs.berkeley.edu

Received May 29, 2003; Revised April 9, 2004; Accepted April 9, 2004

First online version published in October, 2004

Abstract. In this paper, we develop a set of data processing algorithms for generating textured facade meshes of cities from a series of vertical 2D surface scans and camera images, obtained by a laser scanner and digital camera while driving on public roads under normal traffic conditions. These processing steps are needed to cope with imperfections and non-idealities inherent in laser scanning systems such as occlusions and reflections from glass surfaces. The data is divided into easy-to-handle quasi-linear segments corresponding to approximately straight driving direction and sequential topological order of vertical laser scans; each segment is then transformed into a depth image. Dominant building structures are detected in the depth images, and points are classified into foreground and background layers. Large holes in the background layer, caused by occlusion from foreground layer objects, are filled in by planar or horizontal interpolation. The depth image is further processed by removing isolated points and filling remaining small holes. The foreground objects also leave holes in the texture of building facades, which are filled by horizontal and vertical interpolation in low frequency regions, or by a copy-paste method otherwise. We apply the above steps to a large set of data of downtown Berkeley with several million 3D points, in order to obtain texture-mapped 3D models.

Keywords: 3D city model, occlusion, hole filling, image restoration, texture synthesis, urban simulation

1. Introduction

Three-dimensional models of urban environments are useful in a variety of applications such as urban planning, training and simulation for urban terrorism scenarios, and virtual reality. Currently, the standard technique for creating large-scale city models in an automated or semi-automated way is to use stereo vision approaches on aerial or satellite images (Frere et al., 1998; Kim et al., 2001). In recent years, advances in resolution and accuracy of airborne laser scanners have also rendered them suitable for the generation of reasonable models (Haala and Brenner, 1997; Maas, 2001). Both approaches have the disadvantage that their resolution is only in the range of 1 to 2 feet, and more importantly, they can only capture the roofs of the buildings but not the facades. This essential disadvantage prohibits their use in photo realistic walk or drive-through applications.

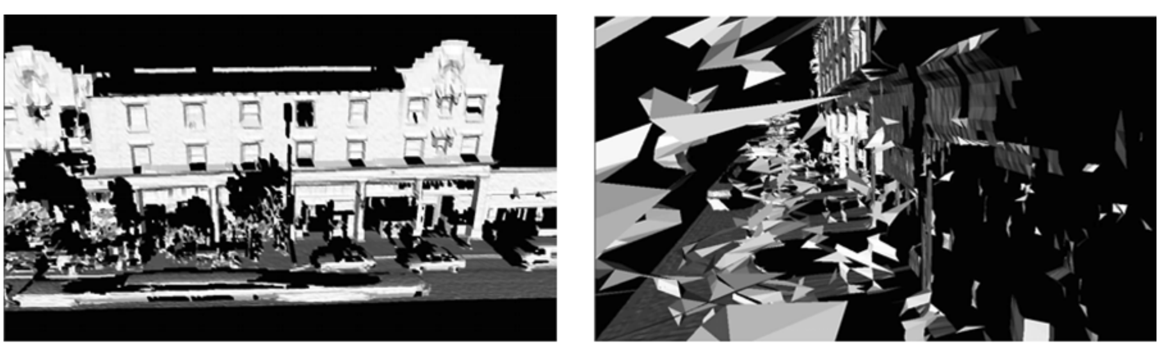
There exist a number of approaches to acquire the complementary ground-level data and to reconstruct building facades; however, these approaches are typically limited to one or few buildings. Debevec et al. (1996) propose to reconstruct buildings based on few camera images in a semi-automated way. Dick et al. (2001), Koch et al. (1999), and Wang et al.

(2002) apply automated vision-based techniques for localization and model reconstruction, but varying lighting conditions, the scale of the environment, and the complexity of outdoor scenes with many trees and glass surfaces generally pose enormous challenges to purely vision-based methods.

Stamos and Allen (2002) use a 3D laser scanner and Thrun et al. (2000) use 2D laser scanners mounted on a mobile robot to achieve complete automation, but the time required for data acquisition of an entire city is prohibitively large; in addition, the reliability of autonomous mobile robots in outdoor environments is a critical issue. In Zhao and Shibasaki (1999), use a vertical laser scanner mounted on a van, which is localized by using odometry, an inertial navigation system, and the Global Positioning System (GPS), and thus with limited accuracy. While GPS is by far the most common source of global position estimates in outdoor environments, even expensive high-end Differential GPS systems become inaccurate or erroneous in urban canyons where there are not enough satellites in a direct line of sight.

In previous work, we have developed a fast, automated data acquisition system capable of acquiring 3D geometry and texture data for an entire city at the ground level by using a combination of a horizontal and a vertical 2D laser scanners and a digital camera (Frueh et al., 2001; Frueh and Zakhor, 2001a). This system is mounted on a truck, moving at normal speeds on public roads, collecting data to be processed offline. It is similar to the one independently proposed by Zhao and Shibasaki (2001), which also use 2D laser scanners in horizontal and vertical configuration; however, our system differs from that of Zhao and Shibasaki (2001) in that we use a normal camera instead of a line camera. Both approaches have the advantage that data can be acquired continuously, rather than in a stop-and-go fashion, and are thus extremely fast; relative position changes are computed with centimeter accuracy by matching successive horizontal laser scans against each other. In Frueh and Zakhor (2001b), we proposed to use the particle-filtering-based Monte-Carlo Localization (Fox et al., 2000) to correct accumulating pose uncertainty by using airborne data such as an aerial photo or a digital surface model (DSM) as a map. An advantage of our approach is that both scan points and camera images are registered with airborne data, facilitating a subsequent fusion with models derived from this data (Frueh and Zakhor, 2003).

In this paper, we describe our approach to processing the globally registered scan points and camera images obtained in our ground-based data acquisition, and to creating detailed, textured 3D facade models. As there are many erroneous scan points, e.g. due to glass surfaces, and foreground objects partially occluding the desired buildings, the generation of a facade mesh is not straightforward. A simple triangulation of the raw scan points by connecting neighboring points whose distance is below a threshold value does not result in an acceptable reconstruction of the street scenery, as shown in Figs. 1(a) and (b). Even though the 3D structure can be easily recognized when viewed from a viewpoint near the original acquisition position as in Fig. 1(a), the mesh appears cluttered due to several reasons; first, there are holes and erroneous vertices due to reflections off the glass on windows; second, there are pieces of geometry "floating in the air", corresponding to partially captured objects or measurement errors. The mesh appears to be even more problematic when viewed from other viewpoints such as the one shown in Fig. 1(b); this is because in this case the large holes in the building facades caused by occluding foreground



(a) (b)

Figure 1. Triangulated raw points: (a) front view; (b) side view.

objects, such as cars and trees, become entirely visible. Furthermore, since the laser scan only captures the frontal view of foreground objects, they become almost unrecognizable when viewed sideways. As we drive by a street only once, it is not possible to use additional scans from other viewpoints to fill in gaps caused by occlusions, as is done in Curless and Levoy (1996) and Stamos and Allen (2002). Rather, we have to reconstruct occluded areas by using cues from neighboring scan points; as such, there has been little work to solve this problem (Stulp et al., 2001).

In this paper, we propose a class of data processing techniques to create visually appealing facade meshes by removing noisy foreground objects and filling holes in the geometry and texture of building facades. Our objectives are robustness and efficiency with regards to processing time, in order to ensure scalability to the enormous amount of data resulting from a city scan. The outline of this paper is as follows: In Section 2, we introduce our data acquisition system and position estimation; Section 3 discusses data subdivision and depth image generation schemes. We describe our strategy to transform the raw scans into a visually appealing facade mesh in Sections 4 through 6; Section 7 discusses foreground and background segmentation of images, automatic texture atlas generation, and texture synthesis. The experimental results are presented in Section 8.

2. Data acquisition and Position Estimation

As described in Frueh et al. (2001) and Frueh and Zakhor (2001a), we have developed a data acquisition system consisting of two Sick LMS 2D laser scanners, and a digital color camera with a wide-angle lens. As

Figure 2. Truck with data acquisition equipment.

seen in Fig. 2, this system is mounted on a rack approximately 3.6 meters high on top of a truck, in order to obtain measurements that are not obstructed by pedestrians and cars. The scanners have a 180° field of view with a resolution of 1°, a range of 80 meters and an accuracy of ±3.5 centimeters. Both 2D scanners face the same side of the street and are mounted at a 90-degree angle. The first scanner is mounted vertically with the scanning plane orthogonal to the driving direction, and scans the buildings and street scenery as the truck drives by. The data captured by this scanner is used for reconstructing 3D geometry as described in this paper. The second scanner is mounted horizontally and is used for determining the position of the truck for each vertical scan. Finally, the digital camera is used to acquire the appearance of the scanned building facades. It is oriented in the same direction as the scanners, with its center of projection approximately in the intersection line of the two scanning planes. All three devices are synchronized with each other using hardware-generated signals, and their coordinate systems are calibrated with respect to each other prior to the acquisition. Thus, we obtain long series of vertical scans, horizontal scans and camera images that are all associated with each other.

We introduce a Cartesian world coordinate system $[x, y, z]$ where x, y is the ground plane and z points into the sky. While our truck performs a 6 degree-of-freedom motion, its primary motion components are x, y, and θ (yaw), i.e. its two-dimensional (2D) motion. As described in detail in Frueh and Zakhor (2001a), we reconstruct the driven path and determine the global pose for each scan by using the horizontal laser scanner: First, an estimate of the 2D relative pose $(\Delta x, \Delta y, \Delta\theta)$ between each pair of subsequent scans is obtained via scan-to-scan matching; these relative estimates are concatenated to form a preliminary estimate for the driven path. Then, in order to correct the global pose error resulting from accumulation of error due to relative estimates, we utilize an aerial image or a DSM as a global map, and apply Monte-Carlo-Localization (Frueh and Zakhor, 2001b). Matching ground-based horizontal laser scans with edges in the global map, we track the vehicle and correct the preliminary path accordingly to obtain a globally registered 2D trajectory as shown in Fig. 3. As described in Frueh and Zakhor (2003), we obtain the secondary motion components z and pitch by utilizing the altitude information provided by the DSM, and the roll motion by correlating subsequent camera images, respectively.

P1: GDU
International Journal of Computer Vision KL3179-04/5384379 September 24, 2004 17:44

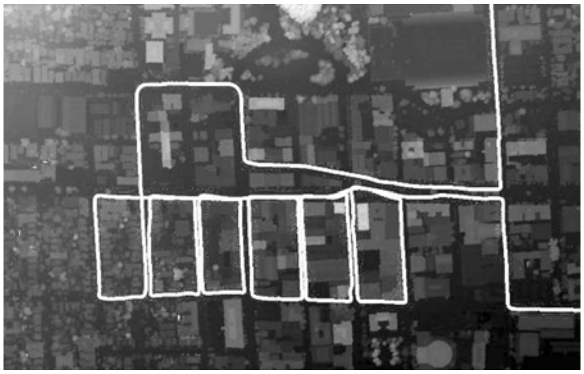


Figure 3. Driven path superimposed on top of a DSM.

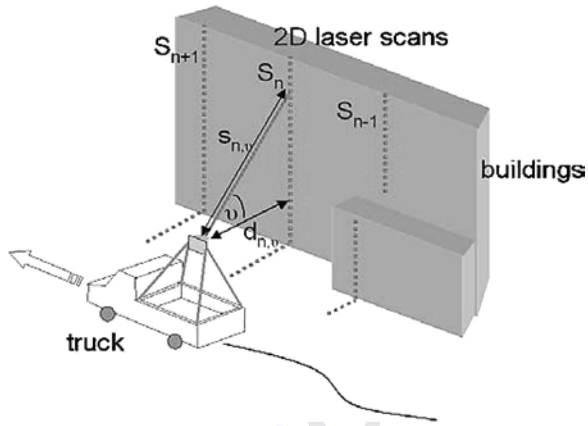


Figure 4. Scanning setup.

While we use the full 6 degree-of-freedom pose to compute the final x, y, z coordinates of each scan point in the final model, we can for convenience and simplicity neglect the 3 secondary motion components for most of the intermediate processing steps described in the following sections of this paper. Furthermore, to reduce the amount of required processing and to partially compensate for the unpredictable, non-uniform speed of the truck, we do not utilize all the scans captured during slow motion; rather, we subsample the series of vertical scans such that the spacing between successive scans is roughly equidistant. Thus, in our processing steps described in this paper, we assume the scan data to be given as a series of roughly equally spaced vertical scans S_n with an associated tuple (x_n, y_n, θ_n) describing 2D position and orientation of the scanner in the world coordinate system during acquisition. Furthermore, we use $s_{n,\upsilon}$ to denote the distance measurement on a point in scan S_n with azimuth angle υ, and $d_{n,\upsilon} = \cos(\upsilon) \cdot s_{n,\upsilon}$ to denote the depth value of this point with respect to the scanner, i.e. its orthogonal projection into the ground plane, as shown in Fig. 4.

3. Data Subdivision and Depth Image Generation

3.1. Segmentation of the Driving Path into Quasi Linear Segments

The captured data during a 20-minute drive consists of tens of thousands of vertical scan columns. Since successive scans in time correspond to spatially close points, e.g. a building or a side of a street block, it is computationally advantageous not to process the entire data as one block, rather to split it into smaller segments to be processed separately. We impose the constraints that (a) path segments have low curvature, and (b) scan columns have a regular grid structure. This allows us to readily identify the neighbors to right, left, above and below for each point, and, as seen later, is essential for the generation of a depth image and segmentation operations.

Scan points for each truck position are obtained as we drive by the streets. During straight segments, the spatial order of the 2D scan rows is identical to the temporal order of the scans, forming a regular topology. Unfortunately, this order of scan points can be reversed during turns towards the scanner side of the car. Figure 5(a) and (b) show the scanning setup during such a turn, with scan planes indicated by the two dotted rays. During the two vertical scans, the truck performs not only a translation but also a rotation, making the scanner look slightly backwards during the second scan. If the targeted object is close enough, as shown in Fig. 5(a), the spatial order of scan points 1 and 2 is still the same as the temporal order of the scans; however, if the object is further away than a critical distance d_{crit}, the spatial order of the two scan points is reversed, as shown in Fig. 5(b).

For a given truck translation of Δs, and a rotation $\Delta\theta$ between successive scans, the critical distance can be computed as

$$d_{\text{crit}} = \frac{\Delta s}{\sin(\Delta\theta)}.$$

Thus, d_{crit} is the distance at which the second scanning plane intersects with the first scanning plane. For a particular scan point, the order with its predecessors should be reversed if its depth $d_{n,\upsilon}$ exceeds d_{crit}; this

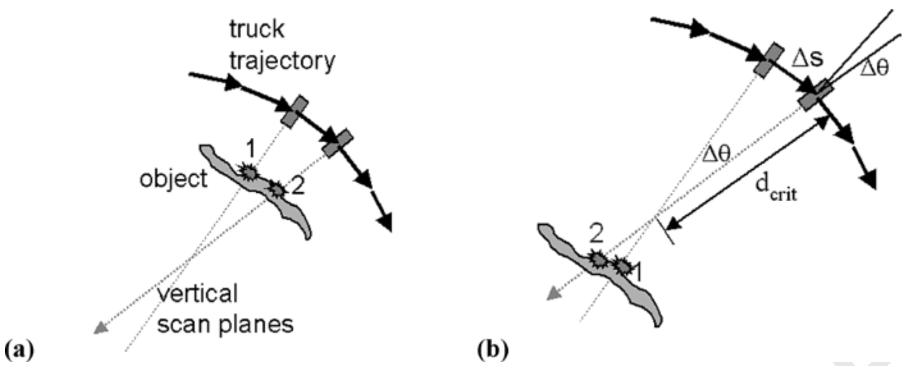


Figure 5. Scan geometry during a turn: (a) normal scan order for closer objects; (b) reversed scan order for farther objects.

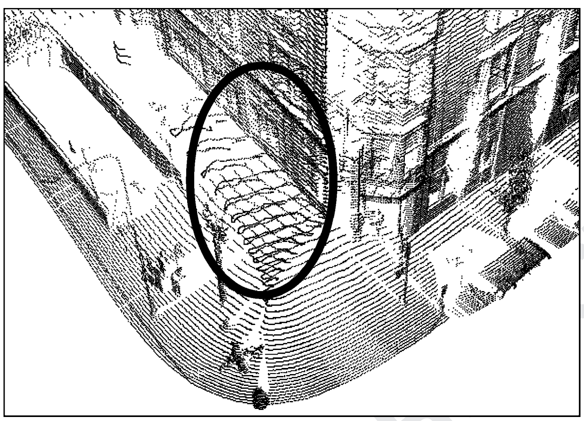


Figure 6. Scan points with reversed order.

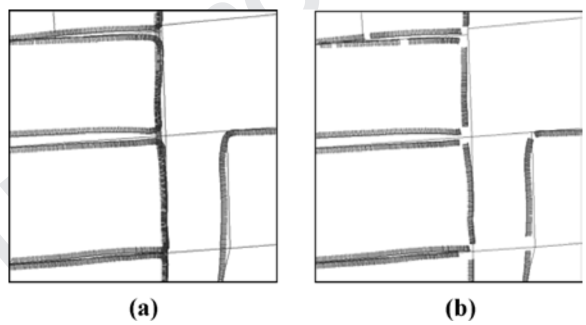


Figure 7. Driven path: (a) before segmentation; (b) after segmentation into quasi-linear segments.

means that its geometric location is somewhere in between points of previous scans. The effect of such order reversal can be seen in the marked area in Fig. 6. At the corner, the ground and the building walls are scanned twice, first from a direct view and then from an oblique angle, and hence with significantly lower accuracy. For the oblique points, the scans are out of order, destroying the regular topology between neighboring scan points.

Since the "out of order" scans obtained in these scenarios correspond to points that have already been captured by "in order" scans, and are therefore redundant, our approach is to discard them and use only "in order" scans. For typical values of displacement, turning angle, and distance of structures from our driving path, this occurs only in scans of turns with significant angular changes. By removing these "turn" scans and splitting the path at the "turning points", we obtain path segments with low curvature that can be considered as locally quasi-linear, and can therefore be conveniently processed as depth images, as described later in this section. In addition, to ensure that these segments are not too large for further processing, we subdivide them if they are larger then a certain size; specifically, in segments that are longer than 100 meters, we identify vertical scans that have the fewest scan points above street level, corresponding to gaps between buildings, and segment at these locations. Furthermore, we detect redundant path segments for areas captured multiple times due to multiple drive-bys, and use only one of them for reconstruction purposes. Figures 7(a) and (b) show an example of an original path, and the resulting path segments overlaid on a road map, respectively. The small lines perpendicular to the driving path indicate the scanning plane of the vertical scanner for each position.

3.2. *Converting Path Segments into Depth Images*

In the previous subsection, we described how to create path segments that are guaranteed to contain no scan

P1: GDU
International Journal of Computer Vision KL3179-04/5384379 September 24, 2004 17:44

pairs with permuted horizontal order. As the vertical order is inherent to the scan itself, all scan points of a segment form a 3D scan grid with regular, quadrilateral topology. This 3D scan grid allows us to transform the scan points into a depth image, i.e. a 2.5D representation where each pixel represents a scan point, and the gray value for each pixel is proportional to the depth of the scan point. The advantage of a depth image is its intuitively easy interpretation, and the increased processing speed the 2D domain provides. However, most operations that are performed on the depth image can be done just as well on the 3D point grid directly, only not as conveniently.

A depth image is typically used for representing data from 3D scanners. Even though the way the depth value is assigned to each pixel is dependent on the specific scanner, in most cases it is the distance between scan point and scanner origin, or its cosine with respect to the ground plane. As we expect mainly vertical structures, we choose the latter option and use the depth $d_{n,\upsilon} = \cos(\upsilon) \cdot s_{n,\upsilon}$ rather than the distance $s_{n,\upsilon}$, so that the depth image is basically a tilted height field. The advantage is that in this case points that lie on a vertical line, e.g. a building wall, have the same depth value, and are hence easy to detect and group. Note that our depth image differs from one that would be obtained from a normal 3D scanner, as it does not have a single center from which the scan points are measured; instead, there are different centers for each individual vertical column along the path segment. The obtained depth image is neither a polar nor a parallel projection; it resembles most to a cylindrical projection. Due to non-uniform driving speed and non-linear driving direction, these centers are in general not on a line, but on an arbitrary shaped, though low-curvature curve, and the spacing between them is not exactly uniform. Because of this, strictly speaking the grid position only specifies the topological order of the depth pixels, and not the exact 3D point coordinates. However, as topology and depth value are a good approximation for the exact 3D coordinates, especially within a small neighborhood, we choose to apply our data processing algorithms to the depth image, thereby facilitating use of standard image processing techniques such as region growing. Moreover, the actual 3D vertex coordinates are still kept and used for 3D operations such as plane fitting. Figure 8(a) shows an example of the 3D vertices of a scan grid, and Fig. 8(b) shows its corresponding depth image, with a gray scale proportional to $d_{n,\upsilon}$.

4. Properties of City Laser Scans

In this section, we briefly describe properties of scans taken in a city environment, resulting from the physics of a laser scanner as an active device measuring time-of-flight of light rays. It is essential to understand these properties and the resulting imperfections in distance measurement, since at times they lead to scan points that appear to be in contradiction with human eye perception or a camera. As the goal of our modeling approach is to generate a photo realistic model, we are interested in reconstructing what the human eye or a camera would observe while moving around in the city. As such, we discuss the discrepancies between these two different sensing modalities in this section.

4.1. Discrepancies Due to Different Resolution

The beam divergence of the laser scanner is about 15 milliradians (mrad) and the spacing, hence the angular resolution, is about 17 mrad. As such, this is much lower than the resolution of the camera image with about 2.1 mrad in the center and 1.4 mrad at the image borders. Therefore, small or thin objects, such as cables, fences, street signs, light posts and tree branches, are clearly visible in the camera image, but only partially captured in the scan. Hence they appear as "floating" vertices, as seen in the depth image in Fig. 9.

4.2. Discrepancies Due to the Measurement Physics

Camera and eye are passive sensors, capturing light from an external source; this is in contrast with a laser scanner, which is an active sensor, and uses light that it emits itself. This results in substantial differences in measurement of reflecting and semitransparent surfaces, which are in form of windows and glass fronts frequently present in urban environments. Typically, there is at least 4% of the light reflected at a single glass/air transition, so a total of at least 8% per window; if the window has a reflective coating, this can be larger. The camera typically sees a reflection of the sky or a nearby building on the window, often distorted or merged with objects behind the glass. Although most image processing algorithms would fail in this situation, the human brain is quite capable of identifying windows. In contrast, depending on the window reflectance, the laser beam is either entirely reflected, most times in a different direction from the laser itself,

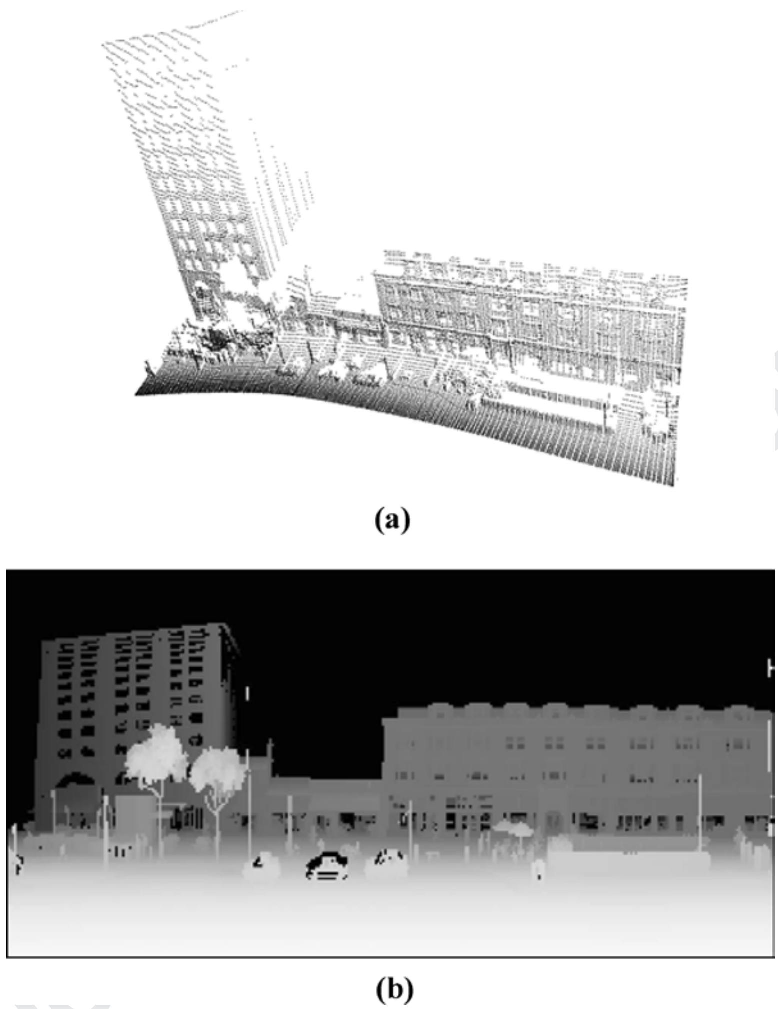


Figure 8. Scan grid representations: (a) 3D vertices; (b) depth image.

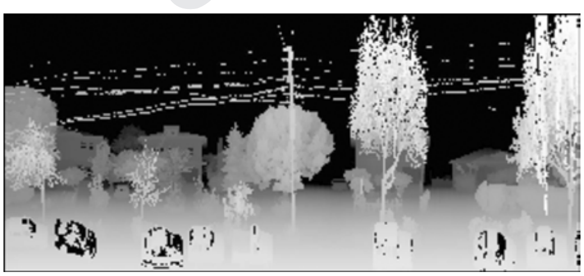


Figure 9. "Floating" vertices.

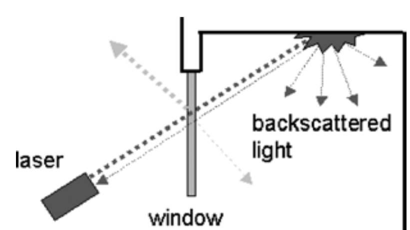


Figure 10. Laser measurement in case of a glass window.

resulting in no distance value, or is transmitted through the glass. In the latter case, if it hits a surface as shown in Fig. 10, the backscattered light travels again through the glass. The resulting surface reflections on the glass only weaken the laser beam intensity, eventually below the detection limit, but do not otherwise necessarily affect the distance measurement. To the laser, the window is quasi non-existent, and the measurement point is generally not on the window surface, unless the surface is orthogonal to the beam. In case of multi-reflections, the

P1: GDU
International Journal of Computer Vision KL3179-04/5384379 September 24, 2004 17:44

situation becomes even worse as the measured distance is almost random.

4.3. *Discrepancies Due to Different Scan and Viewpoints*

Laser and camera are both limited in that they can only detect the first visible/backscattering object along a measurement direction and as such cannot deal with occlusions. If there is an object in the foreground, such as a tree in front of a building, the laser cannot capture what is behind it; hence, generating a mesh from the obtained scan points results in a hole in the building. We refer to this type of mesh hole as *occlusion hole*. As the laser scan points resemble a cylindrical projection, but rendering is parallel or perspective, in presence of occlusions, it is impossible to reconstruct the original view without any holes, even for the viewpoints from which data was acquired. This is a special property of our fast 2D data acquisition method. An interesting fact is that the wide-angle camera images captured simultaneously with the scans often contain parts of the background invisible to the laser. These could be potentially used either to fill in geometry using stereo techniques, or to verify the validity of the filled in geometry obtained from using interpolation techniques.

For a photo realistic model, we need to devise techniques for detecting discrepancies between the two modalities, removing invalid scan points, and filling in holes, either due to occlusion or due to unpredictable surface properties; we will describe our approaches to these problems in the following sections.

5. Multi-Layer Representation

To ensure that the facade model looks reasonable from every viewpoint, it is necessary to complete the geometry for the building facades. Typically, our facades are 2 1/2 D objects rather than full 3D objects, and hence we introduce a representation based of multiple depth layers for the street scenery, similar to the one proposed in Chang and Zakhor (1999). Each depth layer is a scan grid, and the scan points of the original grid are assigned to exactly one of the layers. If at a certain grid location there is a point in a foreground layer, this location is empty in all layers behind it and needs to be filled in.

Even though the concept can be applied to an arbitrary number of layers, we found that it is in our case sufficient to generate only two, namely a foreground and a background layer. To assign a scan point to either one of the two layers we make the following assumptions about our environment: Main structures, i.e. buildings, are usually (a) vertical, and (b) extend over several feet in horizontal dimension. Furthermore, we assume that (c) building facades are roughly perpendicular to the driving direction and that (d) most scan points correspond to facades rather than to foreground objects, as it can occur in residential areas with houses hidden behind trees. Under these conditions, we can apply the following steps to identify foreground objects:

For each vertical scan *n* corresponding to a column in the depth image, we define the main depth as the depth value that occurs most frequently, as shown in Fig. 11. The scan vertices corresponding to the main depth lie on a vertical line, and the first assumption suggests that this is a main structure, such as a building, or perhaps other vertical objects, such as a street light or a tree trunk. With the second assumption, we filter out the

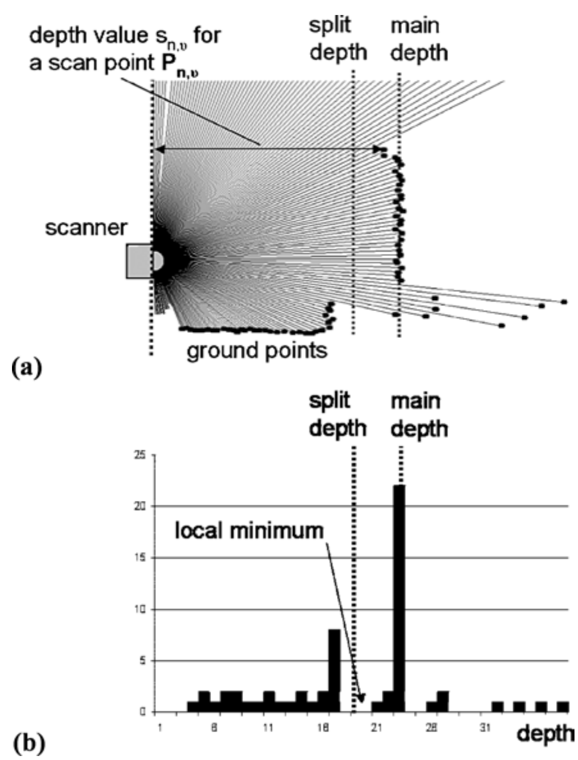


Figure 11. Main depth computation for a single scan n: (a) laser scan with rays indicating the laser beams and dots at the end the corresponding scan points; (b) computed depth histogram.

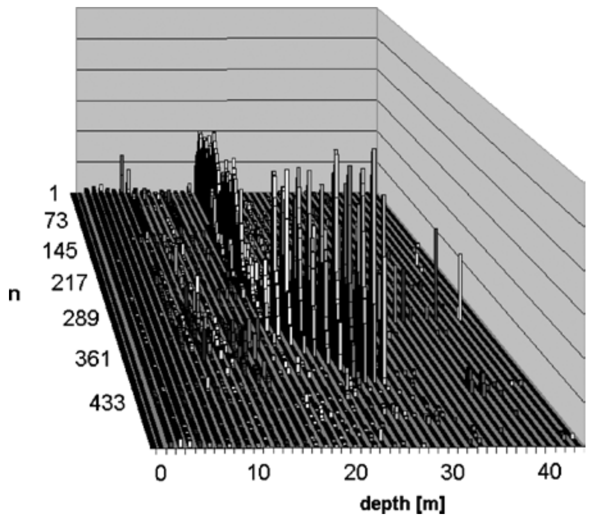


Figure 12. Two-dimensional histogram for all scans.

latter class of vertical objects. More specifically, our processing steps can be described as follows:

We sort all depth values $s_{n,\upsilon}$ for each column n of the depth image into a histogram as shown in Fig. 11(a) and (b), and detect the peak value and its corresponding depth. Applying this to all scans results in a 2D histogram as shown in Fig. 12, and an individual main depth value estimate for each scan. Based on the second assumption, isolated outliers are removed by applying a median filter on these main depth values across the scans, and a final depth value is assigned to each column n. We define a "split" depth, γ_n, for each column n, and set it to the first local minimum of the histogram occurring immediately before main depth, i.e. with a depth value smaller than the main depth. Taking the first minimum in the distribution instead of the main value itself has the advantage that points clearly belonging to foreground layers are splits off, whereas overhanging parts of buildings, for which the depth is slightly smaller than the main depth, are kept in the main layer where they logically belong to, as shown in Fig. 11.

A point can be identified as a ground point if its z coordinate has a small value and its neighbors in the same scan column have a similarly low z value. We prefer to include the ground in our models, and as such, assign ground points also to the background layer. Therefore, we split layers by assigning a scan point $P_{n,\upsilon}$ to the background layer, if $s_{n,\upsilon} > \gamma_n$ or $P_{n,\upsilon}$ is a ground point, and to the foreground layer otherwise. Figure 13 shows an example for the resulting foreground and background layers.

Since the steps described in this section assume the presence of vertical buildings, they cannot be expected

Figure 13. (a) Foreground layer; (b) background layer.

to work for segments that are dominated by trees; this also applies to the processing steps we introduce in the following sections. As our goal is to reconstruct buildings, path segments can be left unprocessed and included "as is" in the city model, if they do not contain any structure. A characteristic of a tree area is its fractal-like geometry, resulting in a large variance among adjacent depth values, or even more characteristically, many significant vector direction changes for the edges between connected mesh vertices. We define a coefficient for the fractal nature of a segment by counting vertices with direction changes greater than a specific angle, e.g. twenty degrees, and dividing them by the total number of vertices. If this coefficient is large, the segment is most likely a tree area and should not be made subject to the processing steps described in this section. This is for example the case for the segment shown in Fig. 9.

After splitting layers, all grid locations occupied in the foreground layer are missing in the background layer as the vertical laser does not capture any occluded geometry; in the next section we will describe an approach for filling these missing grid locations based on neighboring pixels. However, in our data acquisition system there are 3D vertices available from other sources, such as stereo vision and the horizontal scanner used for navigation. Thus, it is conceivable to use this additional information to fill some in the depth layers. Our approach to doing so is as follows:

Given a set of 3D vertices V_i obtained from a different modality, determine the closest scan direction for each vertex and hence the grid location (n, υ) it should be assigned to. As shown in Fig. 14, each V_i is assigned to the vertical scanning plane, S_n, with the smallest Euclidean distance, corresponding to column

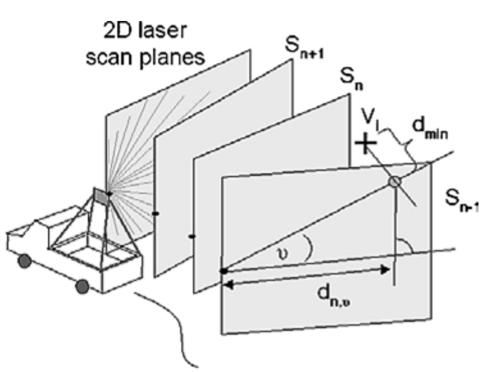


Figure 14. Sorting additional points into the layers.

Figure 15. Background layer after sorting in additional points from other modalities.

n in the depth image. Using simple trigonometry, the scanning angle under which this vertex appears in the scanning plane, and hence the depth image row υ, can be computed, as well as the depth $d_{n,\upsilon}$ of the pixel.

We can now use these additional vertices to fill in the holes. To begin with, all vertices that do not belong to background holes are discarded. If there is exactly one vertex falling onto a grid location, its depth is directly assigned to that grid location; for situations with multiple vertices, median depth value for this location is chosen. Figure 15 shows the background layer from Fig. 13(b) after sorting in 3D vertices from stereo vision and horizontal laser scans. As seen, some holes can be entirely filled in, and the size of others becomes smaller, e.g. the holes due to trees in the tall building on the left side. Note that this intermediate step is optional and depends on the availability of additional 3D data.

6. Background Layer Postprocessing and Mesh Generation

In this section, we will describe a strategy to remove erroneous scan points, and to fill in holes in the background layer. There exists a variety of successful hole filling approaches, for example based on fusing multiple scans taken from different positions (Curless and Levoy, 1996; Stamos and Allen, 2002). Most previous work on hole filling in the literature has been focused on reverse engineering applications, in which a 3D model of an object is obtained from multiple laser scans taken from different locations and orientations. Since these existing hole filling approaches are not applicable to our experimental setup, our approach is to estimate the actual geometry based on the surrounding environment and reasonable heuristics. One cannot expect this estimate to be accurate in *all* possible cases, rather to lead to an acceptable result in *most* cases, thus reducing the amount of further manual interventions and postprocessing drastically. Additionally, the estimated geometry could be made subject to further verification steps, such as consistency checks by applying stereo vision techniques to the intensity images captured by the camera.

Our data typically exhibits the following characteristics:

- Occlusion holes, such as those caused by a tree, are large and can extend over substantial parts of a building.
- A significant number of scan points surrounding a hole may be erroneous due to glass surfaces.
- In general, a spline surface filling is unsuitable, as building structures are usually piecewise planar with sharp discontinuities.
- The size of data set resulting from a city scan is huge, and therefore the processing time per hole should be kept to a minimum.

Based on the above observations, we propose the following steps for data completion.

6.1. Detecting and Removing Erroneous Scan Points in the Background layer

We assume that erroneous scan points are due to glass surfaces, i.e. the laser measured either an internal wall/object, or a completely random distance due to multi-reflections. Either way, the depth of the scan points measured through the glass is substantially greater than the depth of the building wall, and hence these points are candidates for removal. Since glass windows are usually framed by the wall, we remove the candidate points only if they are embedded among a number of scan points at main depth. An example of the effect of this step can be seen by comparing the windows of the original image in Fig. 16(a) with the processed background layer in Fig. 16(b).

6.2. Segmenting the Occluding Foreground Layer into Objects

In order to determine holes in the background layer caused by occlusion, we segment the occluding foreground layer into objects and project segmentation onto the background layer. This way, holes can be filled in one "object" at a time, rather than all at the same time; this approach has the advantage that more localized

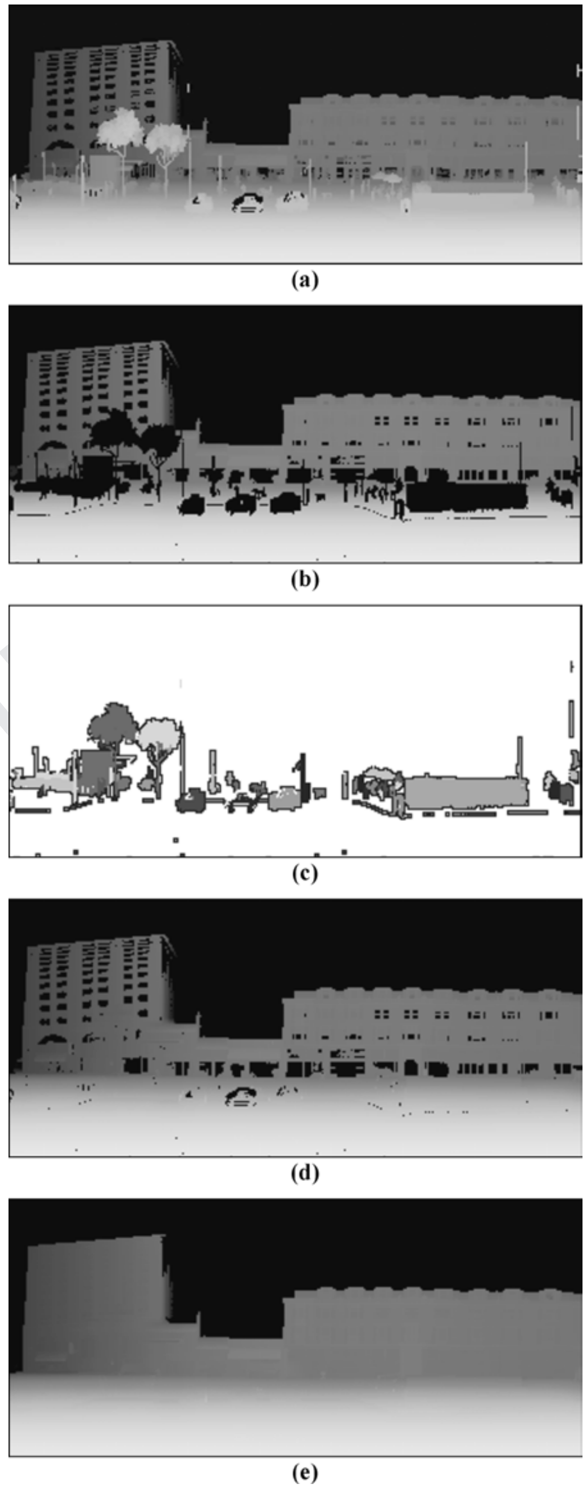


Figure 16. Processing steps of depth image. (a) Initial depth image. (b) Background layer after removing invalid scan points. (c) Foreground layer segmented. (d) Occlusion holes filled. (e) Final background layer after filling remaining holes.

P1: GDU
International Journal of Computer Vision KL3179-04/5384379 September 24, 2004 17:44

hole filling algorithms are more likely to result in visually pleasing models than global ones. We segment the foreground layer by taking a random seed point that does not yet belong to a region, and applying a region growing algorithm that iteratively adds neighboring pixels if their depth discontinuity or their local curvature is small enough. This is repeated until all pixels are assigned to a region, and the result is a region map as shown in Fig. 16(c). For each foreground region, we determine boundary points on the background layer; these are all the valid pixels in the background layer that are close to hole pixels caused by the occluding object.

6.3. Filling Occlusion Holes in the Background Layer for Each Region

As the foreground objects are located in front of main structures and in most cases stand on the ground, they occlude not only parts of a building, but also parts of the ground. Specifically, an occlusion hole caused by a low object, such as a car, with a large distance to the main structure behind it, is typically located only in the ground and not in the main structure. This is because the laser scanner is mounted on top of a rack, and as such has a top down view of the car. As a plane is a good approximation to the ground, we fill in the ground section of an occlusion hole by the ground plane. Therefore, for each depth image column, i.e. each scan, we compute the intersection point between the line through the main depth scan points and the line through ground scan points. The angle υ'_n at which this point appears in the scan marks the virtual boundary between ground part and structure part of the scan; we fill in structure points above and ground points below this boundary differently.

Applying a RANSAC algorithm, we find the plane with the maximum consensus, i.e. maximum number of ground boundary points on it, as the optimal ground plane for that local neighborhood. Each hole pixel with $\upsilon < \upsilon'_n$ is then filled in with a depth value according to this plane. It is possible to apply the same technique for the structure hole pixels, i.e. the pixels with $\upsilon > \upsilon'_n$, by finding the optimal plane through the structure boundary points and filling in the hole pixels accordingly. However, we have found that in contrast to the ground, surrounding building pixels do not often lie on a plane. Instead, there are discontinuities due to occluded boundaries and building features such as marquees or lintels, in most cases extending horizontally across the building. Therefore, rather than filling holes with a plane, we fill in structure holes line by line horizontally, in such a way that the depth value at each pixel is the linear interpolation between the closest right and left structure boundary point, if they both exist; otherwise no value is filled in. In a second phase, a similar interpolation is done vertically, using the already filled in points as valid boundary points. This method is not only simple and therefore computationally efficient, it also takes into account the surrounding horizontal features of the building in the interpolation. The resulting background layer is shown in Fig. 16(d).

6.4. Postprocessing the Background Layer

The resulting depth image and the corresponding 3D vertices can be improved by removing scan points that remain isolated, and by filling small holes surrounded by geometry using linear interpolation between neighboring depth pixels. The final background layer after applying all processing steps is shown in Fig. 16(e).

In order to create a mesh, each depth pixel can be transformed back into a 3D vertex, and each vertex $P_{n,\upsilon}$ is connected to a depth image neighbor $P_{n+\Delta n,\upsilon+\Delta\upsilon}$ if

$$|s_{n+\Delta n,\upsilon+\Delta\upsilon} - s_{n,\upsilon}| < s_{\max} \quad \text{or if} \quad \cos\varphi > \cos\varphi_{\max}$$

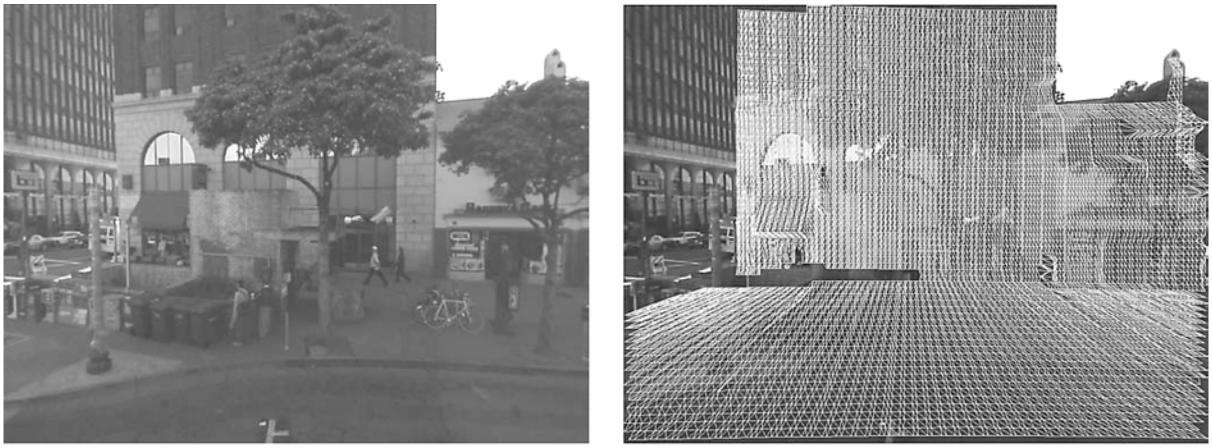
with

$$\cos\varphi = \frac{(\vec{P}_{n-\Delta n,\upsilon-\Delta\upsilon} - \vec{P}_{n,\upsilon}) \cdot (\vec{P}_{n,\upsilon} - \vec{P}_{n+\Delta n,\upsilon+\Delta\upsilon})}{|\vec{P}_{n-\Delta n,\upsilon-\Delta\upsilon} - \vec{P}_{n,\upsilon}| \cdot |\vec{P}_{n,\upsilon} - \vec{P}_{n+\Delta n,\upsilon+\Delta\upsilon}|}$$

Intuitively, neighbors are connected if their depth difference does not exceed a threshold $s_{\max}$ or the local angle between neighboring points is smaller than threshold angle $\varphi_{\max}$. The second criteria is intended to connect neighboring points that are on a line, even if their depth difference exceeds $s_{\max}$. The resulting quadrilateral mesh is split into triangles, and mesh simplification tools such as Qslim (Garland and Heckbert, 1997) can be applied to reduce the number of triangles.

7. Atlas Generation for Texture Mapping

As photorealism cannot be achieved by using geometry alone, we need to enhance our model with texture data. To achieve this, we equip our data acquisition system with a digital color camera with a wide-angle lens. The



(a) (b)

Figure 17. Background mesh triangles projected onto camera images. (a) Camera image. (b) Hole filled background mesh projected onto the image and shown as white triangles; occluded background triangles project onto foreground objects. The texture of foreground objects such as the trees should not be used for texturing background triangles corresponding to the building facade.

camera is synchronized with the two laser scanners, and is calibrated against the laser scanners' coordinate system; hence, the camera position can be computed for all images. After calibrating the camera and removing lens distortion in the images, each 3D vertex can be mapped to its corresponding pixel in an intensity image by a simple projective transformation. As the 3D mesh triangles are small compared to their distance to the camera, perspective distortions within a triangle can be neglected, and each mesh triangle can be mapped to a triangle in the picture by applying the projective transformation to its vertices.

As described in Section 4, camera and laser scanners have different viewpoints during data acquisition, and in most camera pictures, at least some mesh triangles of the background layer are occluded by foreground objects; this is particularly true for triangles that consist of filled-in points. An example of this is shown in Fig. 17 where occluded background triangles project onto foreground objects such as the tree. The background triangles are marked in white in Fig. 17. Although the pixel location of the projected background triangles is correct, some of the corresponding texture triangles merely correspond to the foreground objects, and thus should not be used for texture mapping the background triangles.

In this section, we address the problem of segmenting out the foreground regions in the images so that their texture is not used for the background mesh triangles. After segmentation, multiple images are combined into a single texture atlas; we then propose a number of techniques to fill in the texture holes in the atlas resulting from foreground occlusion. The resulting hole filled atlas is finally used for texture mapping the background mesh.

7.1. Foreground/Background Segmentation in the Images

A simple way of segmenting out the foreground objects is to project the foreground mesh onto the camera images and mark out the projected triangles and vertices. While this process works adequately in most cases, it could miss out some parts of the foreground objects such as those shown in Fig. 18, where projected foreground geometry is marked in white. As seen in the figure, some small portions of the foreground tree are incorrectly considered as background. This is due to following reasons:

1. The foreground scan points are not dense enough for segmenting the image with pixel accuracy, especially at the boundaries of foreground objects.
2. The camera captures side views of foreground objects whereas the laser scanner captures a direct view, as illustrated in Fig. 19. Hence, some foreground geometry does not appear in the laser scans and as such cannot be marked as foreground.

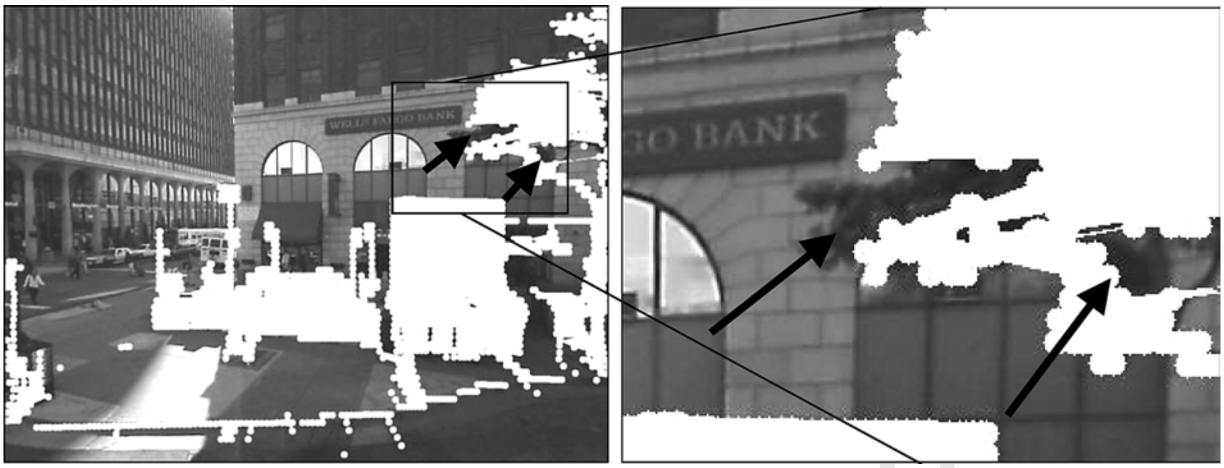


Figure 18. Identifying foreground in images by projection of the foreground mesh. White denotes the projected foreground and thus image areas not to be used for texture mapping of facades.

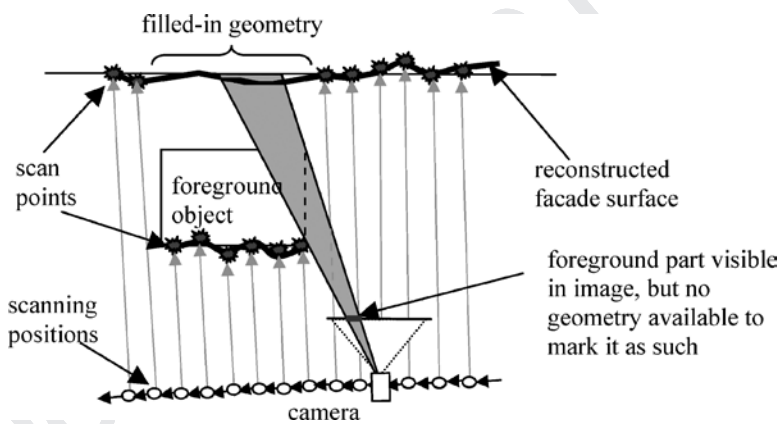


Figure 19. Some foreground objects at oblique viewing angle are not entirely marked in camera images.

To overcome this problem, we have developed a second, more sophisticated method for pixel-accurate foreground segmentation based on the use of correspondence error. The overview of our approach is as follows:

After splitting the scan points into the foreground and background layers, the foreground scan points are projected onto the images. A flood-filling algorithm is applied to all the pixels within a window centered at each of the projected foreground pixels using cues of color constancy and correspondence error. The color at every pixel in the window is compared to that of the center pixel. If the colors are in agreement, and the correspondence error value at the test pixel is close or higher than the value at the center pixel, the test pixel is assigned to the foreground.

In what follows we describe the notion of correspondence error in more detail. Let $I = \{I_1, I_2, \ldots, I_n\}$ denote the set of camera images available for a quasi-linear path segment. Consider two consecutive images I_{c-1} and I_c. Consider a 3D point **x** belonging to the background mesh obtained after geometry hole filling described in Section 7. **x** is projected to the images I_{c-1} and I_c using the available camera position. Assuming that the projected point is within the clip region of both images, let its coordinates in I_{c-1} and I_c be denoted by u_{c-1} and u_c respectively. If **x** is not occluded by any foreground object in an image, then its pixel coordinates in the image belong to the background and represent **x**; otherwise its pixel coordinates correspond to the occluding foreground object. This leads to three cases described below, and illustrated in Fig. 20:

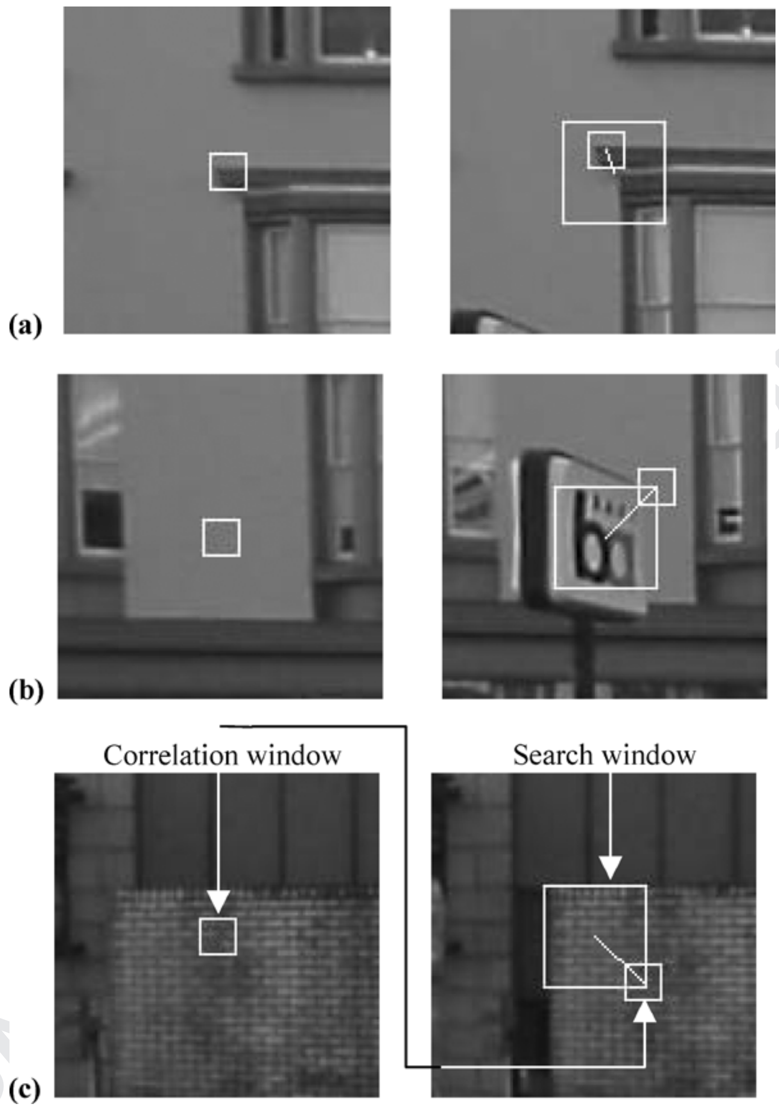


Figure 20. Illustration of correspondence error. (a) background scan point is unoccluded in both images. (b) background scan point occluded in one of the images. (c) background scan point occluded in both images. The search window and correlation window are marked for clarity. The line represents the correspondence error vector. The correlation window slides in the search window in order to find the best matching window.

1. **x** is occluded in neither images as shown in Fig. 20(a); u_{c-1}, and u_c both belong to the background. If the camera position is known precisely, u_c would be the correspondence point for u_{c-1}. In practice, the camera position is known only approximately, and taking u_{c-1} as a reference, its correspondence point in I_c can be located close to u_c.
2. **x** is occluded only in one of the images as shown in Fig. 20(b); one of u_{c-1} or u_c belongs to a foreground object due to occlusion of point **x**, and the other belongs to the background.
3. Point **x** is occluded in both images as shown in Fig. 20(c), and both u_{c-1} and u_c belong to foreground objects.

In all three cases the best matching pixel to u_{c-1} in I_c, denoted by $u_{c-1,c}$, is found by searching in a window centered around u_c, and performing color correlation as illustrated in Fig. 20. The length of vector $\mathbf{v}(u_c, u_{c-1,c})$ then denotes the correspondence error between u_{c-1} and u_c. If $|\mathbf{v}(u_c, u_{c-1,c})|$ is large, one or both of u_{c-1} and u_c belong to a foreground object resulting in cases 2 or 3. In the next step when images I_c and I_{c+1} are considered, $\mathbf{v}(u_{c+1}, u_{c,c+1})$ is computed and we define the correspondence error at pixel u_c as:

$$\varepsilon(u_c) = \max(|\mathbf{v}(u_c, u_{c-1,c})|, |\mathbf{v}(u_{c+1}, u_{c,c+1})|)$$

Intuitively, if the correspondence error at a pixel is large the pixel likely belongs to a foreground object. The above equation is used to compute the correspondence error at all the pixels corresponding to projected background scan points. To compute the correspondence error at all other pixels within the window centered at each of the projected foreground scan points, we apply nearest neighbor interpolation. Each pixel in the window is declared to be foreground if (a) its color is in agreement with the center pixel, and (b) its correspondence error value is close or higher than the value at the center pixel.

The max operation in the above equation has the effect of not missing out any foreground pixels. Even though this approach results in large values of correspondence error at some background pixels corresponding to case 2 above, we choose to adopt it for following reasons:

1. The flood filling algorithm is applied to projected foreground scan points only within a square window w, the size of which is 61×61 pixels in our case; so if a background pixel has a high value of ε but has no projected foreground scan point within a neighborhood equal to size of w, it is never subjected to flood filling and thus never marked as foreground.
2. Marking non-foreground pixels as foreground is not as problematic as leaving foreground pixels unmarked. This is because the same 3D point is observed in multiple camera images, and even though it may be incorrectly classified as foreground in some images, it is likely to be correctly classified as background in others. On the other hand incorrect assignment of foreground pixels to the background and using then for texturing, results in a erroneous texture as discussed before.

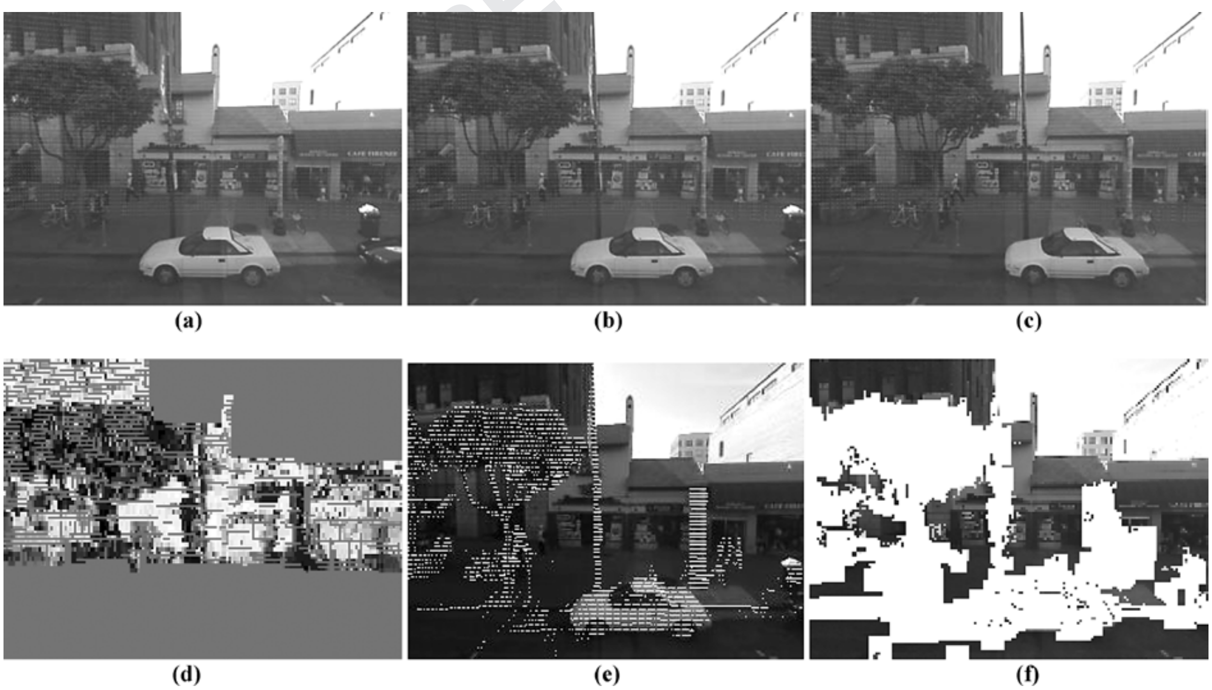


Figure 21. (a), (b), (c) sequence of three camera images I_{c-1}, Ic, I_{c+1}. (d) correspondence error for I_c shown as gray values. White corresponds to low value and black corresponds to high value of ε. Red pixels are pixels where no background scan points projected. ε is not computed at these pixels. (e) Foreground scan points marked as white pixels. (f) Foreground regions of I_c marked as white, using color constancy and correspondence error. The green triangles are the triangles used for texture mapping/atlas generation from this image.

P1: GDU
International Journal of Computer Vision KL3179-04/5384379 September 24, 2004 17:44

Figures 21(a)–(c) show a sequence of three camera images, and Fig. 21(d) shows the correspondence error for the center image shown as gray values; the gray values have been scaled so that 0 or black corresponds to maximum value of ε, and 255 or white corresponds to minimum value of ε. The correspondence error has been computed for each projected background scan point. A 7×7 window is centered at each projected background scan point, and ε at all pixels in the window has been determined using nearest neighbor interpolation. The red pixels denote those for which ε has not been computed or interpolated in the image. The image looks like a roughly segmented foreground and background. Figure 21(e) shows the projected foreground scan points marked as white pixels.[1] Figure 21(f) shows the foreground segmentation using flood-filling with color and correspondence error comparisons as explained in this section. The foreground has been marked in white color. The green triangles are the triangles used for texture mapping/atlas generation from this image. As seen, there are some background pixels that have been incorrectly assigned to the foreground. This can be attributed to the fact that our algorithm has been purposely biased to maximize the size of foreground region in order to avoid erroneously assigning background pixels to foreground.

7.2. *Texture Atlas Generation*

Since most parts of a camera image correspond to either foreground objects, or facade areas visible in other images at a more direct view, we can reduce the amount of texture imagery by extracting only the parts actually used. The vertical laser scanner results in a vertical column of scan points, and triangulation of the scan points thus results in a mesh with a row-column structure as can be seen in Fig. 17(b). The inherent row-column structure of the triangular mesh permits to assemble a new artificial image with a corresponding row-column structure, and reserved spaces for each texture triangle. This so-called *texture atlas* is created by performing the following steps: (a) Determining the inter-column and inter-row spacing for each consecutive column and row pair in the mesh and using this to reserve space in the atlas. (b) Warping each texture triangle to fit to the corresponding reserved space in the atlas and copying it into the atlas. (c) Setting texture coordinates of the mesh triangles to the location in the atlas.

Since in this manner the mesh topology of the triangles is preserved and adjacent triangles align automatically due to the warping process, the resulting texture atlas resembles a mosaic image. While the atlas image might not visually look precisely proportionate due to slightly non-uniform spacing between vertical scans, these distortions are inverted by the graphics card hardware during the rendering process, and are thus negligible.

Figures 22(a) and (b) illustrate the atlas generation: From the acquired stream of images, the utilized texture triangles are copied into the texture atlas as symbolized by the arrows. In this illustration, only five original images are shown; in this example we have actually combined 58 images of 1024×768 pixels size to create a texture atlas of 3180×540 pixels. Thus, the texture size is reduced from 45.6 million pixels to 1.7 million pixels, while the resolution remains the same. If occluding foreground objects and building facade are too close, some facade triangles might not be visible in any of the captured imagery, and hence cannot be texture mapped at all. This leaves visually unpleasant *holes* in the texture atlas, and hence in final rendering of the 3D models. In the following, we propose ways of synthesizing plausible artificial texture for these holes.

7.3. *Hole Filling of the Atlas*

Early work relating to disocclusion in images was done by Nitzberg et al. (1993). Significant improvements to this were made in Masnou and Morel (1998) and Ballester et al. (2000, 2001). These methods are capable of filling in small holes in non-textured regions and essentially deal with *local Inpainting*; they thus cannot be used for filling in large holes or holes in textured regions (Chan and Shen, 2001). We propose a simple and efficient method of hole filling that first completes regions of low spatial frequency by interpolating the values of surrounding pixels, and then uses a copy-paste method to synthesize artificial texture for the holes. In what follows, we explain the above steps in more detail.

Horizontal and Vertical Interpolation. Our proposed algorithm first fills in holes in regions of low variance using linear interpolation of surrounding pixel values. A generalized two-dimensional (2D) linear interpolation is not advantageous over a one-dimensional (1D) interpolation in a man-made environment where features are usually either horizontal or vertical e.g. curbs run across the streets horizontally, edges of facades are vertical, banners on buildings are horizontal.

Figure 22. (a) Images obtained after foreground segmentation are combined to create a texture atlas. In this illustration only five images are shown, whereas in this particular example 58 images were combined to create the texture atlas. (b) Atlas with texture holes for the facade portions that were not visible in any image. (c) Artificial texture is synthesized in the texture holes to result in a filled in atlas that is finally used for texturing the background mesh.

One-dimensional interpolation is simple, and is able to recover most sharp discontinuities and gradients. We perform 1D horizontal interpolation in the following way: for each row, pairs of pixels between which RGB information is missing are detected. The missing values are filled in by a linear interpolation of the boundary pixels if (a) the boundary pixels at the two ends have similar values, and (b) the variances around the boundaries are low at both ends. We follow this by vertical interpolation in which for each column the missing values are interpolated vertically.

Figure 23(a) shows part of a texture atlas with holes marked in red. Figure 23(b) shows the image after a pass of 1D horizontal interpolation. As seen, horizontal edges such as the blue curb are completed. Figure 23(c) shows the image after horizontal and vertical interpolation. We find the interpolation process to be simple, fast, and to complete the low frequency regions well.

The Copy-Paste Method. Assuming that building facades are highly repetitive, we fill holes that could not be filled by horizontal and vertical interpolation, by copying and pasting blocks from other parts of the image. This approach is similar to the one proposed in Efros and Freeman (2001) where a large image is created with a texture similar to a given template. In our copy-paste method the image is scanned pixel by pixel in raster scan order, and pixels at the boundary of holes

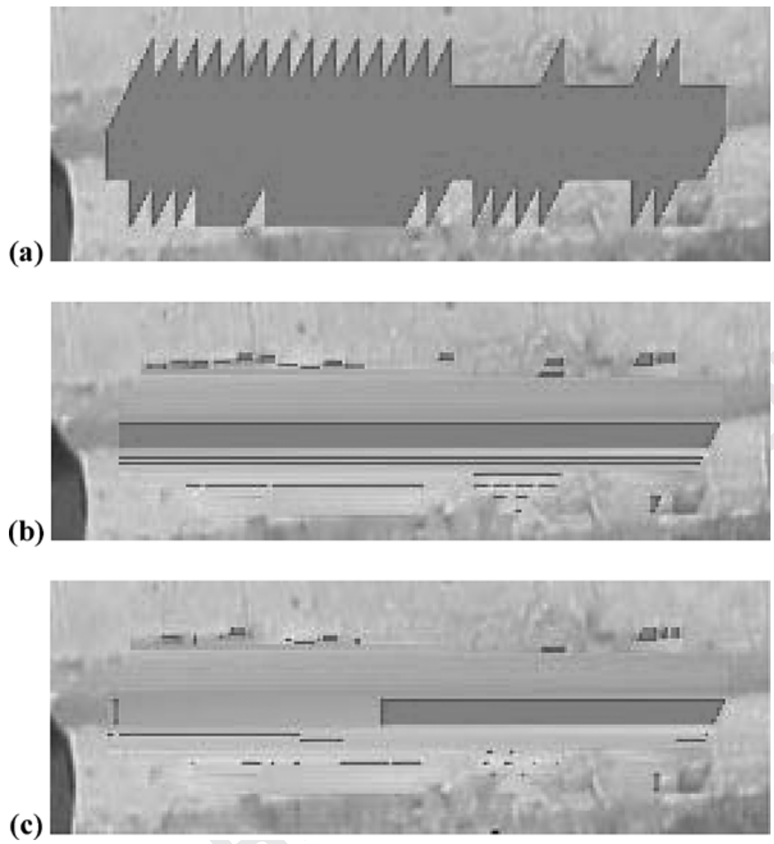


Figure 23. (a) part of a texture atlas with holes marked in red (b) after horizontal interpolation (c) after horizontal and vertical interpolation.

are stored in an array to be processed. A square window w of size $(2M + 1) \times (2M + 1)$ pixels is centered at a hole pixel p, and the atlas is searched for a window denoted by *bestmatch*(w) which (a) has the same size as w, (b) does not contain more than 10% hole pixels, and (c) matches best with w. If the difference between w and *bestmatch*(w) is below a threshold, the *bestmatch* is classified as a good match to w and hole pixels of w are replaced with corresponding pixels in *bestmatch*(w). The method is illustrated in Fig. 24.

For the method to work well, we need a suitable metric that accurately measures the perceptual difference between two windows, an efficient search process that finds the *bestmatch* of a window w, a decision rule that classifies whether the *bestmatch* found is good enough, and a strategy to deal with cases when the *bestmatch* of a window w is not a good match. In our proposed scheme, the difference between two windows consists of two components: (a) the sum of color differences of corresponding pixels in the two windows, and (b) the number of outliers for the pair of windows. These components are weighted appropriately to compute the resulting difference. An efficient search is performed by constructing a hierarchy of Gaussian pyramids, and performing an exhaustive search at a coarse level to find a few good matches, which are then successively refined at finer levels of the hierarchy. In cases when no good match is found the window size is changed adaptively. If a window of size $(2M + 1) \times (2M + 1)$ does not result in a good match, the algorithm finds the *bestmatch* for a smaller window of size $(M + 1) \times (M + 1)$ and this process continues until the window size becomes too small, in our case 9×9 pixels. If no good match is found even after reducing the window size, the hole pixels are filled by averaging the known neighbors provided the pixel variance of the neighbors is low; otherwise the colors of hole pixels are set to the value of randomly chosen neighbors.

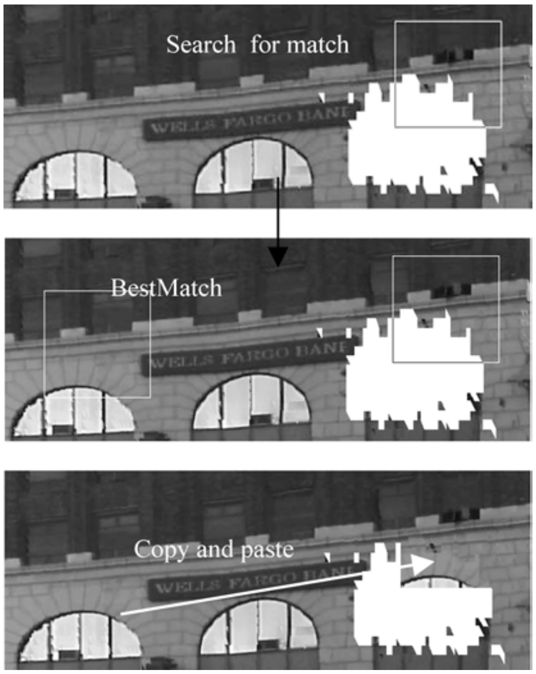


Figure 24. Illustrating the copy-paste method.

8. Results

We drove our equipped truck on a 6769 meters long path in downtown Berkeley, starting from Blake street through Telegraph avenue, and in loops around the downtown blocks. During this 24-minute-drive, we captured 107,082 vertical scans, consisting of 14,973,064 scan points. For 11 minutes of driving time in the downtown area, we also recorded a total of 7,200 camera images. Applying the described path splitting techniques, we divide the driven path into 73 segments, as shown in Fig. 25 overlaid with a road map. There is no need for further manual subdivision, even at Shattuck Avenue, where Berkeley's street grid structure is not preserved.

8.1. Geometry Reconstruction

For each of the 73 segments, we generate two meshes for comparison: the first mesh is obtained directly from the raw scans, and the second one from the depth image to which we have applied the postprocessing steps described in previous sections. For 12 out of the 73 segments, additional 3D vertices derived from stereo vision techniques are available, and hence, sorting in

Figure 25. Entire path after split in quasi-linear segments.

these 3D points into the layers based on Section 5 does fill some of the holes. For these specific holes, we have compared the results based on stereo vision vertices with those based on interpolation alone as described in Section 6, and have found no substantial difference; often the interpolated mesh vertices appear to be more visually appealing, as they are less noisy than the stereo vision based vertices. Figure 26(a) shows an example before processing, and Fig. 26(b) shows the tree holes completely filled in by stereo vision vertices. As seen, the outline of the original holes can still be recognized in Fig. 26(b), whereas the points generated by interpolation alone are almost indistinguishable from the surrounding geometry, as seen in Fig. 26(c).

We have found our approach to work well in the downtown areas, where there are clear building structures and few trees. However, in residential areas, where the buildings are often almost completely hidden behind trees, it is difficult to accurately estimate the geometry. As we do not have the ground truth to compare with, and as our main concern is the visual quality of the generated model, we have manually inspected the results and subjectively determined the degree to which the proposed postprocessing procedures have improved the visual appearance. The evaluation results for all 73 segments before and after postprocessing techniques described in this paper are shown in Table 1; the postprocessing does not utilize auxiliary 3D vertices from horizontal laser scanner or the camera. Even though 8% of all processed segments appear visually worse than the original, the overall quality of the facade models is significantly improved. The important downtown segments are in most

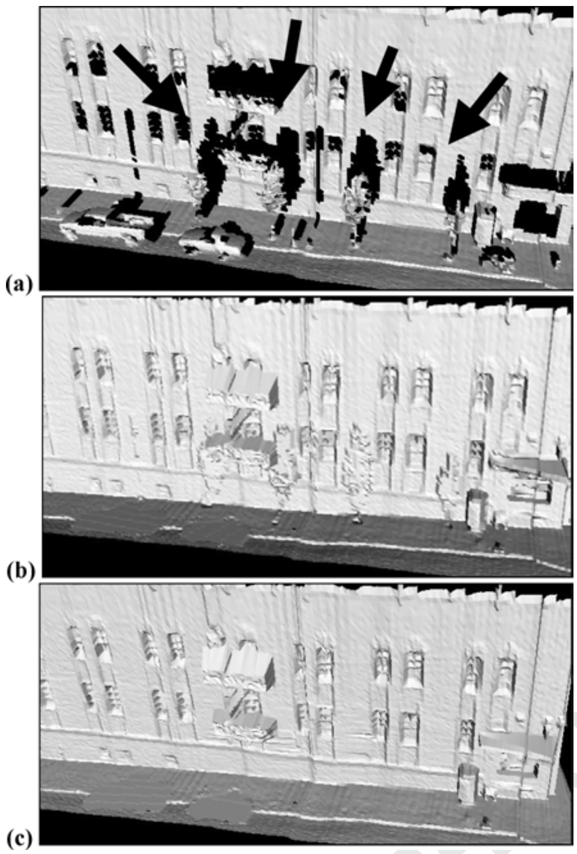


Figure 26. Hole filling. (a) Original mesh with holes behind occluding trees; (b) filled by sorting in additional 3D points using stereo vision; (c) filled by using the interpolation techniques of Section 6.

cases ready to use and do not require further manual intervention.

The few problematic segments all occur in residential areas, consisting mainly of trees. The tree detection algorithm described in Section 5 classifies ten segments as "critical" in that too many trees are present; all six problematic segments corresponding to "worse" and "significantly worse" rows in Table 1 are among them, yet none of the improved segments in rows 1 and 2 are detected as critical. This is significant because it shows that (a) all problematic segments correspond to regions with a large number of trees, and (b) they can be successfully detected and hence not be subjected to the proposed steps. Table 2 shows the evaluation results if only non-critical segments are processed. As seen, the postprocessing steps described in this paper together with the tree detection algorithm improve over 80% of the segments, and never result in degradations for any of the segments.

Table 1. Visual comparison of the processed mesh vs. the original mesh for all 73 segments.

Significantly better	35	48%
Better	17	23%
Same	15	21%
Worse	5	7%
Significantly worse	1	1%
Total	73	100%

Table 2. Visual comparison of the processed mesh vs. the original mesh for the segments automatically classified as non-tree-areas.

Significantly better	35	56%
Better	17	27%
Same	11	17%
Worse	0	0%
Significantly worse	0	0%
Total	63	100%

In Fig. 27 we show before and after examples, and the corresponding classifications according to Tables 1 and 2. As seen, except for pair "f", the proposed postprocessing steps result in visually pleasing models. Pair f in Fig. 27 is classified by our tree detection algorithm as critical, and hence, should be left "as is" rather than processed.

8.2. *Texture Reconstruction*

For 29 path segments or $3\frac{1}{2}$ city blocks, we recorded camera images for texture mapping, and hence we reconstruct texture atlases as described in Section 7. Most facade triangles which were occluded in the direct view could be texture mapped from some other image with an oblique view. Only 1.7% of the triangles were not visible in any image, and therefore required texture synthesis.

Figure 28 demonstrates our texture synthesis algorithm. Figure 28(a) shows a closer view of the facade together with holes caused by occlusion from foreground objects. The holes are marked in white. Figure 28(b) shows the result using the hole filling technique described in Section 7. As seen, the synthesized texture improves the visual appearance of the model. For comparison purposes, Fig. 28 (c) shows the image resulting from the inpainting algorithm described in Bertalmio et al. (2000). A local algorithm such as inpainting only uses the information contained in a thin band around the

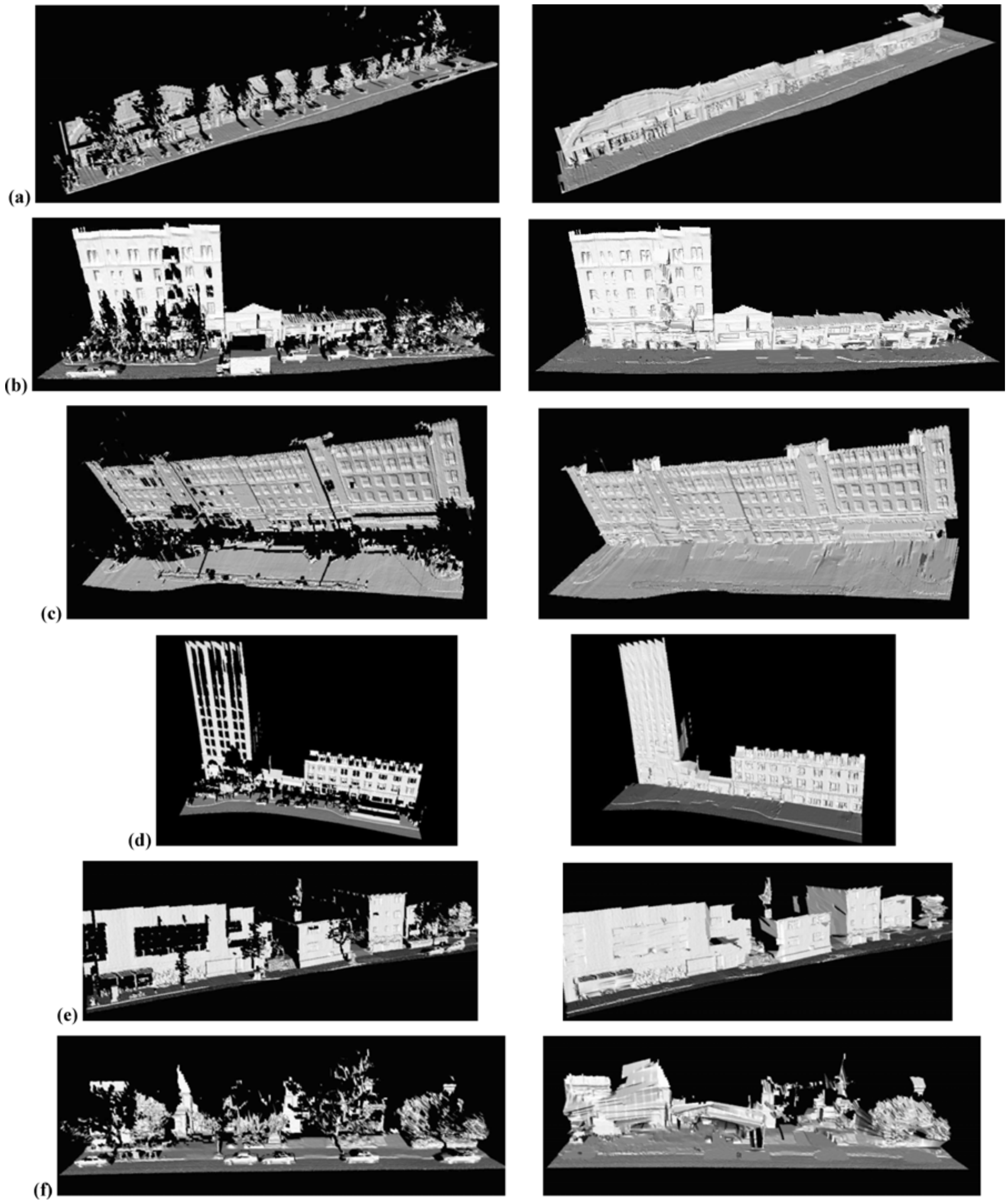


Figure 27. Generated meshes, left side original, right side after the proposed foreground removal and hole filling procedure. The classification for the visual impression is "significantly better" for the first four image pairs, "better" for pair e and "worse" for pair f.

hole, and hence interpolation of surrounding boundary values cannot possibly reconstruct the window arch or the brick pattern on the wall. The copy-paste method on the other hand, is able to reconstruct the window arch and brick pattern by copying and pasting from other parts of the image.

In Fig. 29 we apply the texture atlas of Fig. 28 to the geometry shown in Fig. 27(d) and compare the model

(a) (b) (c)

Figure 28. (a) part of texture atlas with holes marked in white; (b) hole filled atlas using the copy-paste method described in Section 7; (c) result of Inpainting.

with and without the data processing algorithms described in this paper. Figure 29(a) shows the model without any processing, Fig. 29(b) the same model after our proposed geometry processing, and Fig. 29(c) the model after both geometry processing and texture synthesis. Note that in the large facade area occluded by the two trees on the left part of the original mesh, geometry has been filled in; while most of it could be texture mapped using oblique camera views, a few remaining triangles could only be textured via synthesis. As seen, the visual difference between the original mesh and the processed mesh is striking and appears to be even larger than in Fig. 27(d). This is because texture distracts the human eye from missing details and geometry imperfections introduced by hole filling algorithms. Finally, Fig. 30 shows the facade model for the entire $3\frac{1}{2}$ city blocks area.

8.3. *Complexity and Processing Time*

Table 3 shows the processing time measured on a 2 GHz Pentium 4 PC for the automated reconstruction of the $3\frac{1}{2}$ complete street blocks of downtown Berkeley shown in Fig. 30. Without the texture synthesis technique of Section 7, thus leaving 1.7% of the triangles untextured, the processing time for the model reconstruction is 2 hours and 17 minutes. Due to the size of the texture, our texture synthesis algorithm is much slower, with processing time varying between <1 min and 8 hours per segment, depending on the number and the size of the holes. If quality is more important than processing speed, the entire model can be reconstructed with texture synthesis in about 23 hours.

Our approach is not only fast, but also automated: Besides the driving, which took 11 minutes for the model shown, the only manual step in our modeling approach is one mouse click needed to enter the approximate starting position in the digital surface map for Monte-Carlo Localization, which is needed once

Table 3. Processing times for $3\frac{1}{2}$ downtown Berkeley blocks.

Processing Times for Automated Reconstruction on 2 GHz Pentium 4	
Data conversion	14 min
Path reconstruction based on scan matching and global correction with Monte Carlo Localization (with DSM and 5,000 particles)	70 min
Path segmentation	1 min
Geometry reconstruction	6 min
Texture mapping and atlas generation	27 min
Texture synthesis for atlas holes (including pixel-accurate image foreground removal)	20 h 51 min
Model optimization for rendering	19 min
Total model generation time without texture synthesis	2 h 17 min
Total model generation time with texture synthesis	23 h 08 min

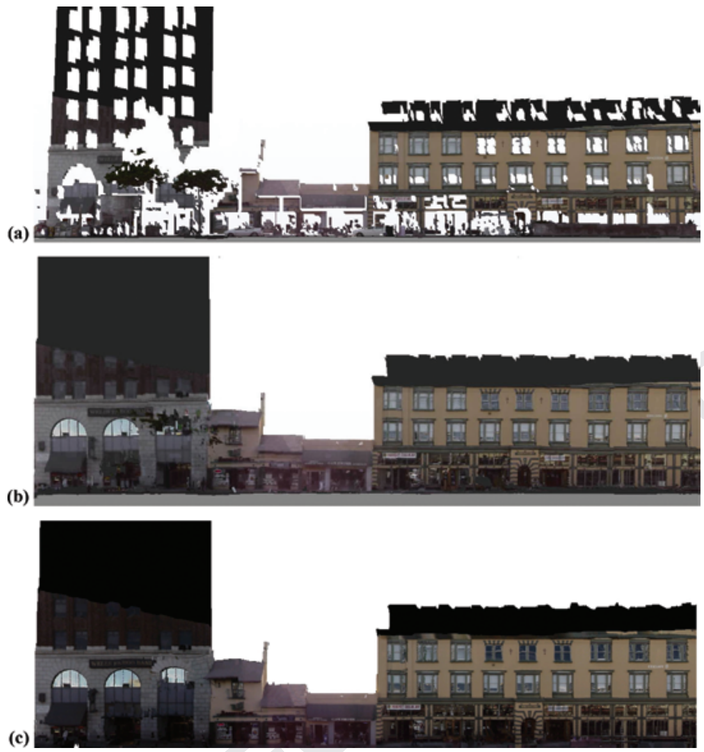


Figure 29. Textured facade mesh: (a) without any processing; (b) with geometry processing; and (c) with geometry processing, pixel-accurate foreground removal and texture synthesis.

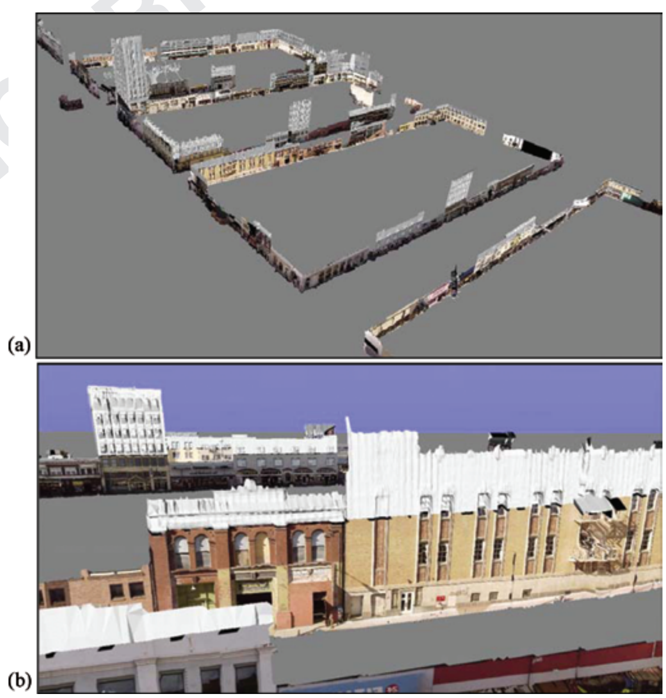


Figure 30. Reconstructed facade models: (a) overview; (b) close-up view.

P1: GDU
International Journal of Computer Vision KL3179-04/5384379 September 24, 2004 17:44

at the beginning of a model acquisition, and could be automated by using a low-cost GPS.

8.4. *Accuracy, Limitations, and Possible Failure Scenarios*

We have demonstrated that our approach is capable of reconstructing facade models for a large-scale urban area. Since we do not have access to ground-truth geometry or texture data, it is difficult, if not impossible, to assess the accuracy of the reconstructed models. However, the following observations can be made:

The accuracy of the reconstructed model depends on (a) the accuracy of the raw scan points and (b) errors made during hole filling and mesh reconstruction. The vertical scan points have a basic random error of $\sigma_s = \pm 3.5$ centimeters due to the scanner's measurement noise. As determined in Frueh (2002), the horizontal scan matching is accurate to within $\sigma_x = \sigma_y = 1$ cm for successive horizontal scans, which are on average about 1 meter apart. Thus, the relative position accuracy for a path corresponding to N matched horizontal scans, or about N meters, is $\sigma_N = \sqrt{N \cdot (\sigma_x^2 + \sigma_y^2)}$. Therefore, the total uncertainty between 2 scan points p1, p2 recorded within N meters of driving can be estimated to $\sigma_{p1,p2} = \sqrt{N \cdot (\sigma_x^2 + \sigma_y^2) + 2\sigma_s^2}$. For example for a 10 meter wide facade, $\sigma_{p1,p2}$ is 6.67 centimeter. Additionally, our Monte-Carlo-Localization-based approach utilizes a DSM to correct drift-like global pose offsets in the vehicle's path by redistributing correction vectors among the relative motion estimates. By virtue of the parameters chosen in our Monte-Carlo localization, these correction vectors are designed to be of the same order of magnitude as σ_x. While the correction vectors are intended to compensate for errors made during the horizontal scan matching, they can add to the uncertainty due to inaccuracies in the DSM itself. Thus, our models are accurate, locally to about $\sigma_{p1,p2}$, e.g. few centimeters, and globally to the accuracy of the DSM as a global map, e.g. one meter.

Errors made during hole filling and mesh reconstruction can be severe, depending on the scene and the amount of geometry that needs to be "invented". First, facades perpendicular to the driving direction or entirely occluded by large foreground objects are invisible to the laser scanner and hence not even result in a hole to be filled in—such structures do not appear in the model at all. Similarly, facades that are nearly all glass without surrounding solid walls would not provide enough vertical scan points to be recognized as a facade and would therefore not be reconstructed. Second, complicated facade objects such as fences, fire escapes, or wires cannot be adequately reconstructed; due to their non-contiguous structure, corresponding scan points are classified as outliers and removed. Third, it is obvious that even a human operator can be wrong in filling a hole, since clues at the boundaries might be misleading; this is more so for an automated hole filling algorithm such as ours, which is based on interpolation and hence implicitly assumes a rather simple geometric structure. Forth, and most importantly, there are scenes for which a simple foreground/facade layer concept is not sufficient. Examples of these are more complex staged building structures with porches, pillars, oriels, or non-vertical walls, and residential areas with many trees. In these cases, our assumptions of Section 5 do not hold true any longer; using histogram analysis to separate the scan points into either foreground or facades is inadequate and results in oddly reconstructed models as seen in Fig. 27(f).

As a matter of fact, for complicated structures which differ substantially from a foreground/background scenario, our drive-by approach with one single vertical scanner does not provide enough data to successfully reconstruct a satisfactory model and hence is inapplicable. Fortunately however, as demonstrated for downtown Berkeley, the street scenery in most downtown areas consists of a foreground/background composition. As a solution to more complicated structures, multiple vertical laser scanners could be mounted at different orientations; similar to merging 3D scans taken from multiple viewpoints, these oblique scans could be used if direct scans are not sufficient.

9. Conclusions

We have proposed a method to reconstruct building facade meshes from large laser surface scans and camera images, even in presence of occlusion. Future work will focus on using color and texture cues to verify filled-in geometry. Additionally, foreground objects could be classified and replaced by appropriate generics.

Acknowledgment

This work was sponsored by Army Research Office under contract DAAD19-00-1-0352. We wish to thank Sick, Inc. for their support. We also wish to thank John Flynn for providing the stereo vision results.

Note

1. The original image is more than 4 times larger in each dimension. This image is produced by subsampling the original image in a special way. Each white pixel corresponding to a foreground scan point in the original image is retained as a white pixel in the subsampled image. This gives a false impression that the density of foreground scan points is very high. On the other hand if the image is subsampled in the normal fashion, there would almost be no white pixels left in the subsampled image.

References

Ballester, C., Bertalmio, M., Caselles, V., Sapiro, G., and Verdera, J. 2001. Filling in by joint interpolation of vector fields and gray levels. In *IEEE Transactions on Image Processing*, pp. 1200–1211.

Au: Pls. provide year

Ballester, C., Caselles, V., Verdera, J., Bertalmio, M., and Sapiro, G. A variational model for filling-in gray level and color images. In *Proc. 8th IEEE Int'l Conference on Computer Vision*, vol. 1, pp. 10–16.

Bertalmio, M., Sapiro, G., Ballester, C., and Caselles, V. 2000. Image inpainting. In *Proc. SIGGRAPH* 2000, pp. 417–424.

Chan, T. and Shen, J. 2001. Mathematical models for local nontexture inpaintings. *SIAM Journal on Applied Mathematics* 62(3):1019–1043.

Chang, N.L. and Zakhor, A. 1999. A multivalued representation for view synthesis. In *Proc. Int'l Conference on Image Processing*, Kobe, Japan, vol. 2, pp. 505–509.

Curless, B. and Levoy, M. A volumetric method for building complex models from range images. In *SIGGRAPH*, New Orleans, pp. 303–312.

Debevec, P.E., Taylor, C.J., and Malik, J. 1996. Modeling and rendering architecture from photographs. In *Proc. ACM SIGGRAPH*.

Dick, A., Torr, P., Ruffle, S., and Cipolla, R. 2001. Combining single view recognition and multiple view stereo for architectural scenes. In *International Conference on Computer Vision*, Vancouver, Canada, pp. 268–274.

Efros, A. and Freeman, W. 2001. Image quilting for texture synthesis and transfer. In *Proc. SIGGRAPH*, pp. 341–346.

Fox, D., Thrun, S., Dellaert, F., and Burgard, W. 2000. Particle filters for mobile robot localization. In *Sequential Monte Carlo Methods in Practice*, A. Doucet, N. de Freitas, and N. Gordon (Eds.), Springer Verlag, New York.

Frere, D., Vandekerckhove, J., Moons, T., and Van Gool, L. 1998. Automatic modelling and 3D reconstruction of urban buildings from aerial imagery. In *IEEE International Geoscience and Remote Sensing Symposium Proceedings*, Seattle, pp. 2593–2596.

Frueh, C. 2002. Automated 3D Model Generation for Urban Environments. Ph.D. Thesis, University of Karlsruhe.

Frueh, C., Flynn, J., Foroosh, H., and Zakhor, A. 2001. Fast 3D model generation in urban environments. In *Workshop on the Convergence of Graphics, Vision, and Video (CGVV'01)*, Berkeley, USA.

Frueh, C. and Zakhor, A. 2001a. Fast 3D model generation in urban environments. In *IEEE Conf. on Multisensor Fusion and Integration for Intelligent Systems*, Baden-Baden, Germany, pp. 165–170.

Frueh, C. and Zakhor, A. 2001b. 3D model generation of cities using aerial photographs and ground level laser scans. In *Computer Vision and Pattern Recognition*, Hawaii, USA, vol. 2.2. pp. II-31-8.

Frueh, C. and Zakhor, A. 2003. Constructing 3D city models by merging ground-based and airborne views. *IEEE Computer Graphics and Applications*, Special Issue Nov/Dec. pp. 52–61.

Garland, M. and Heckbert, P. 1997. Surface *Simplification Using Quadric Error Metrics*. In *SIGGRAPH '97*, Los Angeles, pp. 209–216.

Haala, N. and Brenner, C. 1997. Generation of 3D city models from airborne laser scanning data. In *Proc. EARSEL Workshop on LIDAR Remote Sensing on Land and Sea*, Tallin, Esonia, pp. 105–112.

Kim, Z., Huertas, A., and Nevatia, R. 2001. Automatic description of Buildings with complex rooftops from multiple images. In *Computer Vision and Pattern Recognition*, Kauai, pp. 272–279.

Koch, R., Pollefeys, M., and van Gool, L. 1999. Realistic 3D Scene modeling from uncalibrated image sequences. In *ICIP'99*, Kobe, Japan, Oct., pp. II: 500–504.

Maas, H.-G. 2001. The suitability of airborne laser scanner data for automatic 3D object reconstruction. 3. In *Int'l Workshop on Automatic Extraction of Man-Made Objects*, Ascona, Switzerland.

Masnou, S. and Morel, J. 1998. Level-lines based disocclusion. In *5th IEEE Int'l Conference on Image Processing*, pp. 259–263.

Nitzberg, M., Mumford, D., and Shiota, T. 1993. *Filtering, Segmentation and Depth*. Springer-Verlag: Berlin.

Stamos, I. and Allen, P.K. 2002. Geometry and texture recovery of scenes of large scale. In *Computer Vision and Image Understanding (CVIU)*, 88(2):94–118.

Stulp, F., Dell'Acqua, F., and Fisher, R.B. 2001. Reconstruction of surfaces behind occlusions in range images. In *Proc. 3rd Int. Conf. on 3-D Digital Imaging and Modeling*, Montreal, Canada, pp. 232–239.

Thrun, S., Burgard, W., and Fox, D. 2000. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping. In *Proc. of International Conference on Robotics and Automation*, San Francisco, vol. 1. pp. 321–328.

Wang, X., Totaro, S., Taillandier, F., Hanson, A., and Teller, S. 2002. Recovering facade texture and microstructure from real-world images. In *Proc. 2nd International Workshop on Texture Analysis and Synthesis in Conjunction with European Conference on Computer Vision*, pp. 145–149.

Zhao, H. and Shibasaki, R. 1999. A system for reconstructing urban 3D objects using ground-based range and CCD images. In *Proc. of International Workshop on Urban Multi-Media/3D Mapping*, Tokyo.

Zhao, H. and Shibasaki, R. 2001. Reconstructing urban 3D model using vehicle-borne laser range scanners. In *Proc. of the 3rd International Conference on 3D Digital Imaging and Modeling*, Quebec, Canada.