Automated Texture Mapping of 3D City Models With Oblique Aerial Imagery

Christian Frueh, Russell Sammon, and Avideh Zakhor

Department of Computer Science and Electrical Engineering
University of California, Berkeley

<frueh, sammon, avz>@eecs.berkeley.edu

This paper describes an approach to texture mapping a 3D city model obtained from aerial and ground-based laser scans with oblique aerial imagery. First, the images are automatically registered by matching 2D image lines with projections of 3D lines from the city model. Then, for each triangle in the model, the optimal image is selected by taking into account occlusion, image resolution, surface normal orientation, and coherence with neighboring triangles. Finally, the utilized texture patches from all images are combined into one texture atlas for compact representation and efficient rendering. We evaluate our approach on a data set of downtown Berkeley.

Keywords: Texture Mapping, 3D City Model, Image Registration

I. INTRODUCTION

Three-dimensional models of urban environments are useful in a variety of applications. They are typically represented either as Digital Surface Models (DSM) or triangular 3D meshes. A standard technique for obtaining the geometry of a large-scale city area in an automated or semi-automated way is to apply stereo vision techniques on aerial or satellite imagery [11]. DSMs and 3D models can also be obtained from Synthetic Aperture Radar (SAR) or from airborne laser scans [1]. Recently, ground-based data acquisition systems have been developed, which are capable of scanning the geometry of building facades as seen from the street level [5,20].

In applications such as urban planning, virtual heritage conservation, and computer gaming, it is desirable to capture not only the geometry, but also the visual appearance of an urban environment; this can be achieved by texture mapping a geometric model with acquired imagery. Besides resulting in a photo-realistic look, texture also creates the false impression of a higher level of geometric detail, a fact that is exploited in image-based rendering.

Except for stereo vision, the above-mentioned existing approaches do not incorporate aerial imagery as texture for photo-realistic rendering. Even in stereo vision, where the image/model registration is a byproduct of the model generation process, there are usually too few images available to cover all building sides, or the images are black and white and can therefore not be used as texture for photo-realistic rendering. Furthermore, they are usually top-down views that cannot be used for facades; facades, however, are essential for both walk-through and fly-through interactive rendering applications.

Thus, to achieve photo-realistic rendering, texture has to be acquired and mapped in a separate process. Previous approaches have decoupled the problem into texturing building tops using top-down views of the roofs, and texturing facades by utilizing ground-based images [1,12]. In these cases, roofs and terrain are often texture mapped with only one single top-down aerial image, and building sides not accessible from ground-level are not textured. However, to ensure that all sides of buildings in a city area are covered, it is necessary to utilize multiple images from different viewing angles. In this case, it is necessary to cope with most parts of a model being visible in several images, under varying viewing angles, resolutions, and potentially different lighting conditions.

In this paper, we propose ways to texture-map an existing 3D city model with multiple airborne images at oblique angles, capturing both facades and rooftops. Specifically, we acquire high-resolution aerial imagery from a helicopter using a standard digital camera; assuming rough initial knowledge of position and orientation, we register the images with the model by matching line segments. Then, for each triangle of the model, an optimal image is selected for texture by taking into account occlusion, image resolution, surface normal orientation, and coherence with neighboring triangles. Finally the utilized parts of all images are combined into a single texture atlas for compact representation and rendering.

The outline of this paper is as follows: Section II describes the registration of the aerial imagery. Section III details our image selection method, and Section IV describes assembling the selected image patches to a texture atlas. Finally, in Section V, we present results for a data set of downtown Berkeley.

II. AERIAL IMAGE REGISTRATION

The first step of our texture mapping procedure is the registration of the aerial images with the 3D model. These images are captured at a variety of different angles, including oblique perspectives that show the facades of buildings. We assume no knowledge of correspondences between portions of the aerial images and the 3D model, however we do assume an initial rough estimate of the camera's pose, which may be obtained using the Global Positioning System (GPS) and other sensors such as an inertial measurement system (INS), electronic compass, and/or electrolytic tilt sensors.

In the case of two uncalibrated images, this problem is commonly referred to as the simultaneous pose and correspondence problem. To cope with errors, existing solutions often use a probabilistic hypothesize-and-test approach such as RANSAC [4]. More recent research efforts have improved upon the original algorithm by reducing the search space, using faster pose estimation based on hypothesized correspondences, different features such as lines instead of points, and alternative methods of scoring matches [2, 12, 17]. However, since the success of such methods often relies on eliminating false correspondences by utilizing feature point signatures, they are not easily applicable to our problem, namely 2D image/3D model registration. Except for lines and corner points, no features present in the images actually correspond to features in the 3D model. Specifically, for our images, we have found that out of about 1600 features found by a Harris corner detector [10], less than 1 %, i.e. only 10-15, correspond to corners of buildings. This drastically reduces the probability of hypothesizing a correct point-to-point correspondence, making RANSAC-type approaches inapplicable.

In [13] and [16], the 2D/3D registration problem for small sculptures is solved by matching the silhouette derived from the 3D shape with the silhouette in the images. This method is not applicable to our problem, since (a) images show only a small subset of the entire model, and (b) our object, i.e. the city, cannot be placed in front of a contrasting background to detect the silhouette. In addition, midday shadows, slanted roofs, and irregular street layouts make it difficult to reliably determine principal axes of orientation necessary for the application of vanishing point models such as in [19] and [12]. The oblique nature of our images also prohibits assumptions such as the weak perspective projection model assumed in SoftPOSIT [2].

Our solution to the aerial image registration problem consists of finding 3D line segments in the model, i.e. "model lines", and matching them to 2D line segments in the aerial images, i.e. "image lines", by searching the poses around an initial pose obtained via GPS and INS. During the search, poses are rated using a line-to-line correspondence rating function that is robust to erroneous line segments. What complicates matters is that there are discrepancies between the 3D model and the 2D imagery: specifically, due to limited resolution and various model simplification and processing steps [7], the location of 3D model edges may be inaccurate; additionally, small features such as telephone poles, rooftop ventilation ducts, awnings, railings, and scaffoldings, which have been deliberately removed in the 3D model, are clearly visible in the 2D imagery.

A. Finding Line Segments in the 3D model

Finding 3D or "model' lines could be carried out on either the DSM or the triangular mesh. Since we are primarily interested in edges which are not occluded in the 2D imagery, we apply the following silhouette-based technique:

Using the initial camera pose obtained from GPS and INS, we generate the 2½D depth image corresponding to the captured camera image by projecting the triangular 3D model into a modified z-buffer. This buffer also stores the product of a triangle's normal vector with the camera's viewing direction at each pixel. Then, pixels are marked as edges if the depth value or the vector product of the pixel differ from that of at least one of its neighbors by greater than a threshold value. To avoid obtaining edges that are two pixels wide, only the pixel with the lower depth value is marked for either of these discontinuities. Next, we create a list of 3D line segments based on the edges marked in the depth image and the corresponding z value. Specifically, this is done by fitting straight lines to the edge pixels using a recursive endpoint subdivision algorithm [14]. It subdivides contours at their point of maximum deviation if the maximum deviation or average deviation of the contour exceeds threshold values; short line segments are discarded. As seen in the example in Figure 1(a), the resulting set of 3D line segments is a small subset of 3D model lines, which are in the view frustum, and not occluded for the initial pose. Since our position search space is small compared to the model distance, we assume that the above occlusion and frustum culled line set is also usable for all other poses in the search space.

B. Finding Line Segments in Aerial Images

To find 2D edges in the aerial imagery, we use a Canny edge detector; the edges are then divided into line segments using a recursive endpoint subdivision algorithm [14]. We remove line segments that are shorter than a threshold. This improves the ratio between "true" and "false" edges since false edges are rather short, and so are shadow edges when interrupted by street marks etc. However, there are still numerous false edges in the final selection, due to trees, vehicles, sidewalks, street marks and windows, as seen in Figure 1(b).

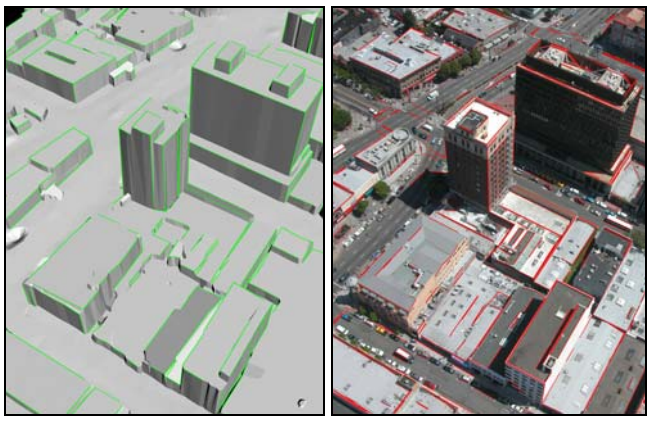

Figure 1: (a) Visible 3D lines of the model in green; (b) 2D lines in the image in red.

C. Line Matching Process

Assuming lens distortions to be negligible due to the large focal length, and using a simple camera model as in [14], the camera's pose consists of the 6 extrinsic and 5 intrinsic parameters. The extrinsic parameters are x, y, z, yaw, pitch, and roll. The intrinsic parameters are focal length, image size in pixels, center of projection, and pixel size in nanometers. Of these, focal length is unknown while the others are manually calibrated once, and remain fixed throughout the image acquisition process.

We can rate a given pose by projecting the 3D model lines onto the captured aerial image, and comparing them to the 2D lines found in the image based on their slope and proximity in a manner similar to [12]. Specifically, the rating Q of a pose is computed as:

$$Q_{pose} = \sum_{i=0}^{M}\sum_{j=0}^{N} \|l_i\| \cdot S(l_i, L_j) \cdot D(l_i, L_j)$$

where l_i is the i^{th} 2D line segment found in the aerial image, M is the total number of 2D line segments, L_j is the 2D projection of the j^{th} 3D line segment, N is the total number of 3D line segments, $||l_i||$ is the length of the i^{th} 2D line segment found in the aerial image, $S(l_i,L_j)$ is a function of the slopes of lines l_i and L_j, and $D(l_i,L_j)$ is a function of the proximity of the endpoints of lines l_i and L_j. $S(l_i,L_j)$ and $D(l_i,L_j)$ are both determined as follows:

$$S(l_i, L_j) = \begin{cases} 0 & for\ \langle l_i, L_j \rangle < S_{\max} \\ \langle l_i, L_j \rangle & for\ \langle l_i, L_j \rangle \geq S_{\max} \end{cases}$$

$$D(l_i, L_j) = \begin{cases} 0 & for\ d(l_i, L_j) \geq D_{\max} \\ \dfrac{D_{\max} - d(l_i, L_j)}{D_{\max}} & for\ d(l_i, L_j) < D_{\max} \end{cases}$$

where $<l_i,L_j>$ is the dot product of the normals of lines l_i and L_j, S_{max} is a threshold value, $d(l_i,L_j)$ is the sum of the minimum distances of the endpoints of l_i to line segment L_j, and D_{max} is a threshold distance. The above pose rating function is designed to achieve a maximum value when both slope and position of the projected 3D model lines best match the 2D image lines. To accelerate the rating process, the projected model lines are stored in a spatiality-indexed data structure.

To find the camera pose, we optimize Q over the 7-dimensional parameter space around the initial pose. We have experimented with the following search patterns: (a) exhaustive search of the 7-dimensional parameter space; (b) steepest decent search. The pose with the highest rating is used for texture-mapping as described in the following section.

III. Texture Selection

In this section, we describe our approach to selecting the optimal image for texture mapping a specific triangle within the 3D geometric model. In doing so, we assume that for each 3D triangle there are multiple images available, and the task at hand is to choose the best one. We assume that all the images are taken within a short period of a few minutes, so that differences in lighting conditions are negligible. In order to identify which image region corresponds to a specific triangle, we utilize the pose obtained in the previous section. We compute the projection of the triangle's three corner points onto the image in order to obtain texture coordinates, and thus a 2D triangle in the image.

In general, a texture-mapped model looks best if viewed from a position close to the original image capture location. Using multiple view-dependent texture images [3] could potentially deliver a better visual performance than using only one single image; however, the drawbacks of this approach are its data redundancy, and the more complex view-dependent rendering. For every triangle, multiple texture maps have to be stored, resulting in enormous amounts of data for large-scale models. At the same time, the visual quality is only marginally better than a 'good' single texture map. Hence, for scalability reasons, we opt to use only one single texture map for each triangle, chosen among the texture map candidates based on the following criteria:

1. Resolution

We define the resolution as the number of pixels per area unit. Since our images are taken at oblique views rather than top-down views, there is a wide range of resolutions within an image, ranging from few centimeters per pixel for foreground facades to meters per pixels for far away buildings. However, to determine the resolution R_{ij} for each image I_i and triangle T_j, we divide the number of pixels within each projected 2D triangle by the area of the corresponding 3D triangle.

2. Occlusion

In order to determine what percentage of a 3D triangle T_j is visible in image I_i, we detect occlusions on a per-pixel basis in the images by using the z-buffer algorithm in two steps: In the first step, we allocate a z-buffer for each image and fill the z-buffer by projecting all 3D triangles into it.

Then, in a second step, we project each 3D triangle again, and compute the ratio η_{ij} between the number of pixels that are not occluded to the total number of pixels. Due to the cluttered nature of city environments, we have empirically observed that requiring 100%-visiblity has the undesirable consequence of often eliminating "good" image candidates. For instance, we have found that a few occluded pixels appear less disturbing to the human eye than utilizing a low resolution image or one taken from an extremely oblique view.

3. Viewing angle

For each triangle, the image used for texture mapping should ideally be taken from a direct, perpendicular view. This is because texture mapping a surface with a image taken at an extremely oblique angle results in large distortions. For an image I_i and triangle T_j, we compute the view direction $\vec{v}_{ij}$ as the vector between the camera's position as obtained in Section 2, and the center point of the triangle. We then use the scalar product between $\vec{v}_{ij}$ and the surface normal $\vec{n}_j$ of the triangle to quantify how direct the view of an image is.

4. Coherence with neighboring triangles

If both the 3D model and the image/model registration were perfect, and brightness, color tone and resolution were identical across all images, there would be no visual seams across triangles with different texture source images. In practice however, there may be noticeable seams if the images are taken from opposite directions, and as such, it is desirable to minimize these visual effects wherever possible. Our 3D model contains imperfections such as geometry cracks across building facades, since it is derived from actual laser measurements with limited accuracy and resolution [7]. These cracks result in unexpected changes of surface normal orientation, which could adversely affect the viewing angle criteria mentioned earlier for a few triangles on an otherwise smooth surface. This could potentially result in two different images being chosen for neighboring triangles at the cracks, and thus in a visually unpleasant intensity discontinuity across the surface. Furthermore, due to measurement noise, surfaces in the 3D model are not perfectly smooth. If for such a surface two images are almost equally applicable for texture mapping, even small differences in normal orientations of the individual triangles can make a difference. As a result, the boundary between triangles with different texture sources could become jagged rather than a smooth seam.

While it is adequate to change the texture source image at true 3D discontinuities such as facade/roof boundaries, we attempt to reduce texture discontinuities across contiguous triangles all on a facade or all on a roof in the following manner: First, we classify each triangle into one of the types "facades", "rooftops", and "ground" based on its surface orientation. If the z-component is smaller than the x,y-component, we declare the triangle a facade triangle; otherwise, it is either a terrain triangle or a roof triangle, depending on the elevation of its corner points. The result of this classification process is shown in Figure 2. Then, for each triangle, we find all those triangles in the mesh, which are the same type and have a common edge, i.e. neighbors of the triangle. Hence, for each triangle we obtain a neighbor list, which contains the index of its neighbors of the same type. As seen shortly, this list is used to impose coherency across neighboring triangles during the image selection process.

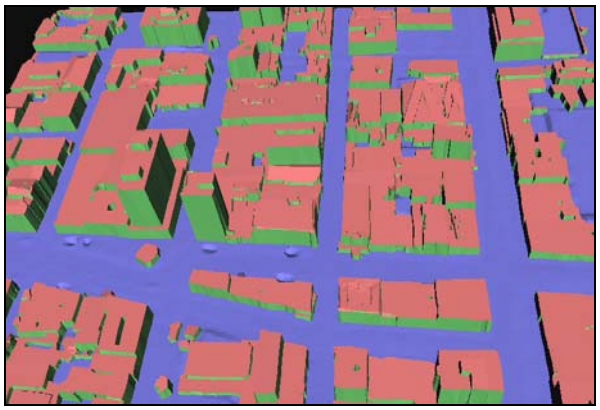

Figure 2: Classification of triangles into "roof" shown as red, "terrain" shown as blue, and "façade" shown as green

Using the above criteria, we apply the following procedure to determine the optimal image for texture mapping a 3D triangle T_j: First, for each image I_i, we calculate a score λ_{ij} defined as

$$\lambda_{ij} = R_{ij} \cdot \eta_{ij} \cdot \vec{n}_j \cdot \vec{v}_{ij}$$

We then assign the image with the highest score as preliminary texture for the triangle, and use the neighbor coherence criteria to accept or reject this selection: First, for each triangle T_j, we find the immediate neighboring triangles which are of the same type. Utilizing the neighbor list, we apply a tree search to determine the triangle's neighbors of the neighbors of the same type, and so on, all the way up to the N^{th} level neighbor of the same type. To finally decide which image to use for a triangle, we use a voting scheme similar to a median filter among the preliminarily assigned texture images: Assigning largest weight to the triangle T_j itself and smaller weights to farther away neighboring triangles of the same type, we discard the preliminarily assigned texture if the majority of votes favors a different image. In this manner, small 'islands' textured with a different image are assigned the same image as their surrounding neighbors, while the texture at true boundaries such as building corners remains unchanged. After choosing the image to be used for a given 3D triangle, the texture coordinates for each triangle's corner points are recomputed, specifying the texture for the mesh unambiguously.

IV. TEXTURE PACKING

In the previous section, we described a way of choosing the "appropriate" image triangle as texture for each 3D triangle. The larger the number of available images, the smaller the chance for a particular image to be chosen, and hence the lower the percentage of pixels actually used for texture mapping from each image. In the example image shown in Figure 3, the utilized texture areas are marked in green. This image is one out of 17 used to texture map a 10-block 3D city model; as seen, only about 17% of the pixels are actually utilized. Using the entire original images directly as texture for rendering is a waste of valuable graphics memory, and unnecessarily limits the model size. In this section, we briefly describe our approach to packing the model texture into one single texture atlas in order to optimize the model for rendering purposes.

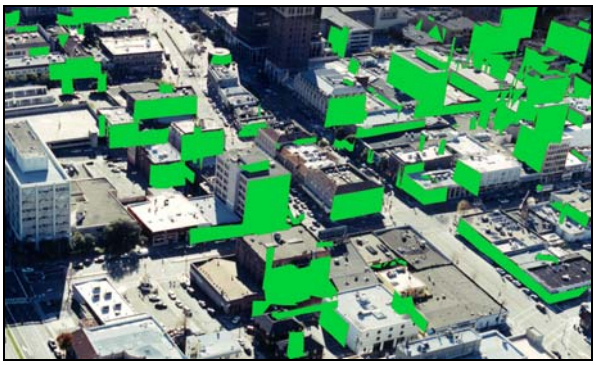

Figure 3: Used texture area marked green in the source image.

In general, it is not possible to warp a complex 3D shape into a simple rectangular 2D image without introducing severe distortions in resolution. Repacking the texture can preserve the resolution, but changes the spatial distribution of texture patches with respect to their original geometric location in the mesh. Packing texture on a per-triangle basis is inefficient for VRML and for rendering, since new vertices with different texture coordinates have to be defined for originally contiguous texture areas. Packing rectangles or quasi-quadratic texture charts has been described in [15] and [18], even though efficient packing has been shown to be an NP-hard problem. Our approach to creating a single texture atlas is to use a greedy algorithm which copies contiguous texture patches, and places them into available space in the atlas.

Specifically, we copy entire connected texture regions in the following manner: First, we project all 3D triangles into their corresponding texture images, and mark the utilized pixels, as illustrated in Figure 3. We then identify connected texture regions using a flood fill algorithm, and sort the resulting regions for all images according to their size. We create an empty rectangular texture atlas, and similar to packing a suitcase, we first place the largest region. We then determine the next largest region, search for an empty atlas space where it fits into, and copy it there. There are typically many possible destination locations for a region, and therefore, to choose the best destination, we minimize a cost function based on the following two criteria: (a) the distance of the destination to the top left corner of the atlas, and (b) the required graphics memory by placing the region at that destination location. The first criteria is intended to pack the regions as closely as possible to the top left corner; the second criteria is intended to penalize any destination location that increases utilized graphics memory to the next power-of-two. As a result, we obtain a render-efficient texture atlas that contains the utilized texture areas of all pictures packed without distortions. We finally assign the texture coordinates of each 3D triangle's vertex to the new corresponding locations in the atlas.

V. RESULTS

We have evaluated our proposed methods on a data set of Berkeley, California. We have two different 3D models available: One is an untextured airborne model generated only from airborne laser scans by creating, processing, and meshing a DSM. The other one is the result of a fusion process between the airborne model and ground-based façade models as described in our earlier work [6,7]. In the fused model, the lower parts of most street-facing facades are already texture-mapped with high-resolution imagery from the ground-based acquisition, so that only the upper facade parts and the rooftops need to be texture mapped.

We utilize 17 aerial images taken from a helicopter with a standard 5-Megapixel digital camera. Most images are taken from oblique angles with pitches of approximately 30-60 degrees, showing both the rooftops and facades of buildings.

A. Aerial Image Registration

Since we did not have access to an actual GPS or INS at the time of the helicopter flight, we simulate this data by utilizing pose information from a manual registration of the images via correspondence points. More specifically, we assume our initial orientation, i.e. roll, pitch, and yaw, to be known within a 10° range, and our initial position, i.e. x, y and z, to be known within a 20 meter range, and we randomly select a pose within this range as hypothetical sensor readout. We consider these values to be realistically accurate for a mid-tier GPS/INS system. Furthermore, we obtain an initial focal length estimate from the digital camera.

We have empirically found that the exhaustive search can find the correct pose for all the images reliably, while steepest decent by itself can become easily trapped in a local minimum. Even exhaustive search can miss the rating peak occurring near the true pose unless the sampling intervals are sufficiently small. Specifically, due to the presence of numerous misleading lines in the images, we have found that the coarsest sampling intervals for exhaustive search to detect the true rating peak are 0.5° in roll, 0.25° in pitch and yaw, and 10 meters in x, y, and z. The maximum granularity for yaw and pitch is smaller than that of the roll angle, since these angles result in an x-,

respective y-shift of all projected 3D lines in the image at the same time, while the roll angle results in a rotation which only marginally affects lines near the image center. Figure 4 shows the rating function around the true pose. As seen, step sizes larger than the above numbers can potentially miss the peak, resulting in an erroneous local maximum far away from the true pose.

We are able to find the correct pose for all 17 images if we sample the parameter space with a step size of 0.5° for roll, 0.2° for pitch and yaw, 10 meter for x, y, and z, and 0.1 millimeter for focal length. However, with a search range of 10° for orientation, 20 m for position, and 1mm for focal length, this results in $21\times41\times41\times3\times3\times3\times11 \cong 10$ million poses to query per image, which on average takes 25 hours to complete on a 2.0 GHz Pentium IV computer, and hence is impractical. However, such an exhaustive search becomes drastically more feasible if the search space is smaller. For example, if we had assumed a more accurate differential GPS, used a fixed focal length, and assumed the orientation range to be 5°, we would only need to compute $11\times21\times21$ poses, and the computation time would drop to 40 seconds per image. Thus, for more accurate sensors, this method can efficiently be applied to remove residual errors.

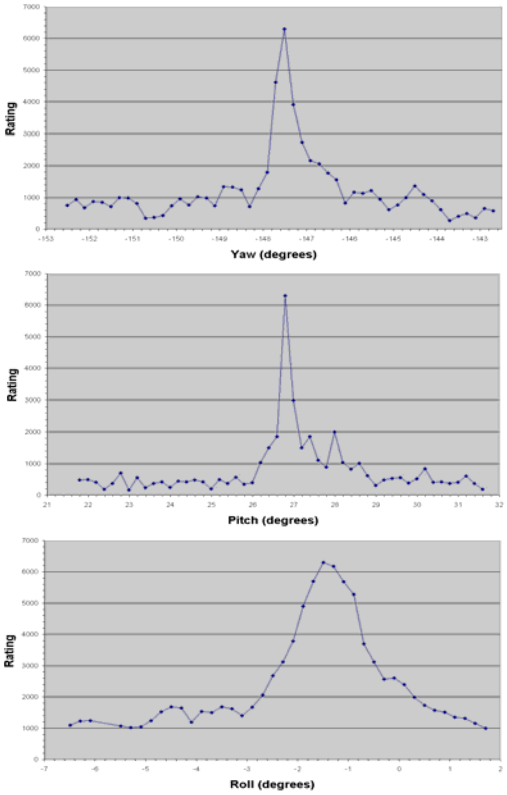

Figure 4: Pose rating as a function of (a) yaw, (b) pitch, and (c) roll. The step size of this search is 0.2°.

Figure 5 shows an example for 2D image and 3D model lines computed for (a) a random pose and (b) for the best pose found. As seen, the best pose is well suited for texture mapping the model. However, even for the correct pose, not all image lines and 3D lines match perfectly due to inaccuracies in the 3D model such as erroneous building dimensions due to overhanging roofs etc.

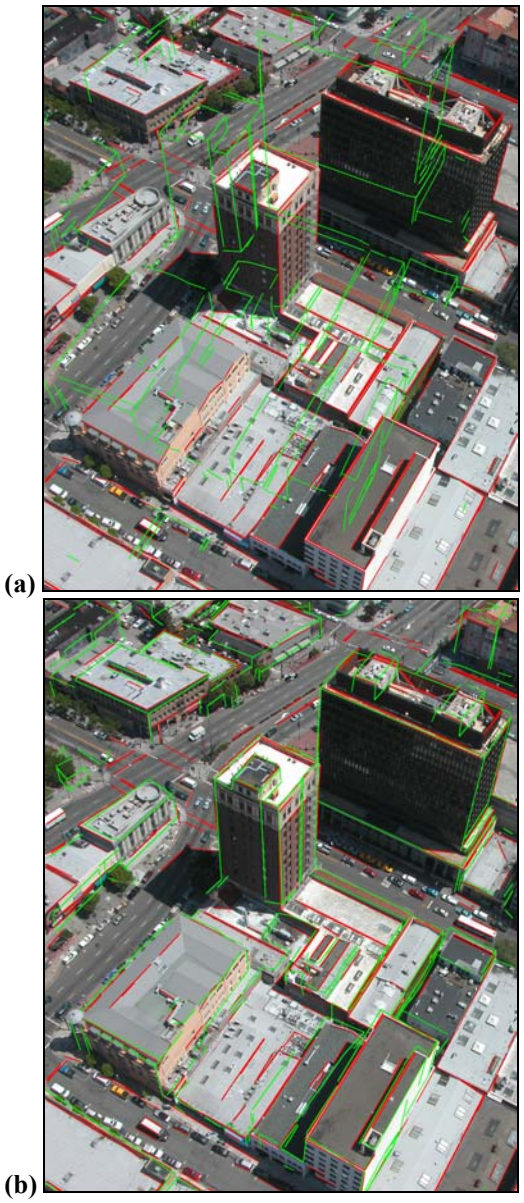

Figure 5: Resulting pose; (a) random pose within the search range; (b) best pose found. 3D Model lines projected onto the image are drawn in green, and lines found in the 2D image are drawn in red.

B. Texture Selection

Applying the algorithms described in Section III on the 17 images, we select the "best" image for each of the 3D mesh triangles and compute the texture coordinates. Figure 6 shows color-coded the spatial distribution of the images used on the 3D model. Figure 6(a) shows the preliminary image assignment computed based on resolution, occlusion, and viewing angle. Figure 6(b) shows the final image assignment after applying the neighbor coherence criteria. As seen in the area marked by the white circle, the image assignment pattern is substantially less fragmented after applying the neighbor coherence filter. A close-up view of

this area is shown in Figure 7(a), along with the resulting texture mapped model in Figure 7(b). As seen, except for triangles for which only extreme oblique views were available, seams are mostly invisible or unnoticeable. This is because all images were acquired within one half hour period, and thus under very similar lighting conditions.

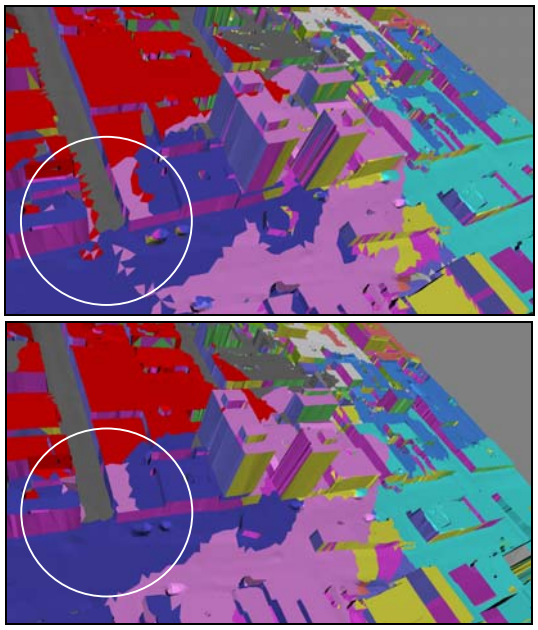

Figure 6: Spatial distribution of images utilized for texture mapping; a) as obtained by using the resolution, occlusion and viewing angle; b) after applying the neighbor coherence constraint. Different colors indicate different source images.

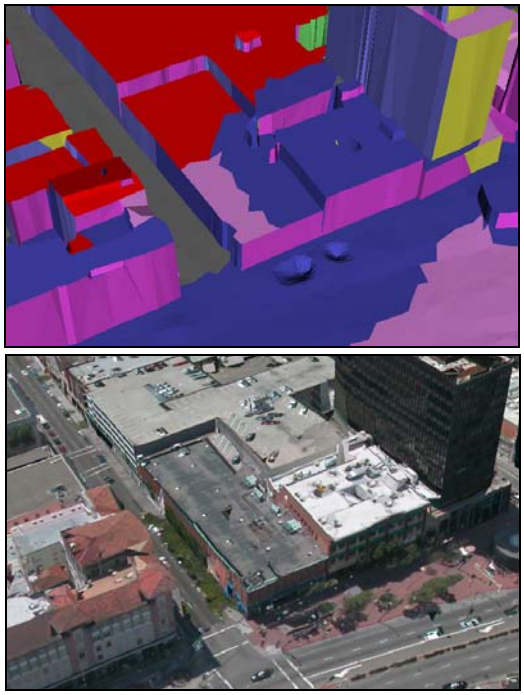

Figure 7: Close-up view of an area texture-mapped with multiple images. (a) Source images color-coded; (b) textures applied to the model.

C. Texture Packing

Next, to facilitate interactive rendering, we apply the texture packing algorithms described in Section IV to the selected image patches. Figure 8 shows the resulting texture atlas. As seen, its spatial structure does not correspond to the spatial structure of the 3D mesh at all. The larger contiguous texture patches are near the top-left corner, since they have been placed in first. The shown texture atlas is 24 Mbytes; despite some remaining empty areas towards the bottom, this is a reduction of a factor of 7 as compared to the combined texture size of 192 Mbytes for all the 17 original images. Nonetheless, for the rendered model, there is no visual difference between the original images and the packed texture, since no distortions have been introduced.

Figure 8: Packed texture.

D. Final 3D model

Figures 9 and 10 show the texture-mapped airborne-only and the fused model, respectively. The shown viewing positions are not identical to any of the camera positions, and the views show regions covered by several different images.

Figure 9: Texture mapped airborne model. The shown viewing positions are not identical to any camera position.

As seen, areas textured with different images align nicely with each other; furthermore, the aerial images align with the ground-based model, which was created

independently and texture mapped automatically using ground-based images [6]. While seams between different aerial images are for the most part invisible, one can clearly notice seams between ground-based and airborne texture as shown in Figure 10. This is not surprising, since (a) the resolution of the ground-based texture is about an order of magnitude higher, and (b) ground-based and airborne image acquisitions were made at different times of the day and months apart from each other, and thus under completely different lighting conditions. Also, there are 3D objects such as cars, ventilation ducts, or trees, which are not included in the 3D model, but visible in the images and thus texture-mapped onto the mesh surface. While these 'flat' features greatly contribute to the level of photo-realism for a wide range of viewing angles, they appear somewhat distorted if rendered from an extremely oblique view. This can be seen for the cars on top of the parking structure in Figure 10(a). Both models can be downloaded from our web site [8] and viewed interactively as VRML.

(a)

(b)

Figure 10: Texture-mapped fused model. The arrows indicate the horizontal boundaries between texture from aerial images, i.e. the upper part, and texture from the ground-based data acquisition, i.e. the lower part.

VI. Conclusions and Future Work

We have presented an approach to texture map an existing 3D city model with aerial imagery. While our approach is automated and results in visually acceptable models, there are several problems that could be addressed in future work: First, applying stereo vision techniques on the images can potentially identify geometry errors in the model such as the ones caused by overhanging roofs. Second, rather than selecting one single "best" image for texture mapping a triangle, the final texture could be a blend of several images, so as to avoid seams in situations with images of different brightness and contrast. Third, the problem of handling texture for simplification and creating lower LODs remains to be solved.

VII. Acknowledgements

This work was sponsored by Army Research Office contract DAAD19-00-1-0352.

VIII. References

[1] Brenner C., Haala N., and Fritsch D., "Towards fully automated 3D city model generation", Workshop on Automatic Extraction of Man-Made Objects from Aerial and Space Images III, 2001

[2] David P., DeMenthon D., Duraiswami R., and Samet H., "Simultaneous Pose and Correspondence Determination using Line Features", Computer Vision and Pattern Recognition Conf., Madison, WI, vol. II, p.424-431.

[3] Debevec P. E., Taylor C. J., and Malik J.: "Modeling and Rendering Architecture from Photographs", ACM SIGGRAPH 1996

[4] Fischler M.A. and Bolles R.C. "Random sample consensus: A paradigm for model fitting with application to image analysis and automated cartography", Communications of the ACM, 24(6):381-395, 1981.

[5] Frueh C. and Zakhor A., "An Automated Method for Large-Scale, Ground-Based City Model Acquisition", International Journal of Computer Vision, October 2004, vol. 60, p. 5-24

[6] Frueh C. and Zakhor A., "Data Processing Algorithms for Generating Textured 3D Building Façade Meshes From Laser Scans and Camera Images", Proc. 3D Data Processing, Visualization and Transmission, Padua, Italy, June 2002, p. 834 - 847

[7] Frueh C. and Zakhor A., "Constructing 3D City Models by Merging Ground-Based and Airborne Views", IEEE Conference on Computer Vision and Pattern Recognition, Madison, WI, p. II-562 – 69, June 2003.

[8] http://www-video.eecs.berkeley.edu/~frueh/3d

[9] Garland M. and Heckbert P., "Surface Simplification Using Quadric Error Metrics", SIGGRAPH '97, Los Angeles, 1997, p. 209-216

[10] Harris C. J. and Stephens M. , "A combined corner and edge detector", Proc. 4th Alvey Vision Conference, Manchester, pages 147-151

[11] Kim Z., Huertas A., and Nevatia R., "Automatic description of buildings with complex rooftops from multiple images," IEEE Conference on Computer Vision and Pattern Recognition, Kauai, 2001, p. 272-279

[12] Lee S. C., Jung S. K., and Nevatia R., "Automatic pose estimation of complex 3D building models", Workshop on Application of Computer Vision, 2002.

[13] Lensch H.P.A., Heidrich W., and Seidel H-P., "Automated texture registration and stitching for real world models", Proc. Eighth Pacific Conference on Computer Graphics and Applications, pp.317-452, Hong-Kong, 2000

[14] Lowe, D. G., "Three-dimensional object recognition from single two-dimensional images", Artificial Intelligence, 31:355--395, 1987.

[15] Murata H., Fujiyoshi K., Nakatake S., and Kajitani Y., "Rectangle-packing-based module placement", Int. Conf. on Computer-Aided Design", pp.472-9, San Jose, 1995

[16] Neugebauer, P.J. and Klein K., "Texturing 3D Models of Real World Objects from Multiple Unregistered Photographic Views", Proc. Eurographics '99, Milan, pp. 245-56, 1999

[17] Olson C.F., "Efficient Pose Clustering Using a Randomized Algorithm", *International Journal of Computer Vision*, 23(2):131-147, June 1997.

[18] Sander P.V., Snyder J., Gortler S.J., and Hoppe H., "Texture mapping progressive meshes", SIGGRAPH 2001, pp.409-16, Los Angeles, 2001

[19] Stamos I. and Allen P. K., "Geometry and Texture Recovery of Scenes of Large Scale", Computer Vision and Image Understanding (CVIU), V. 88, N. 2, Nov. 2002, pp. 94-118.

[20] Zhao H. and Shibasaki R., "Reconstructing a textured CAD model of an urban environment using vehicle-borne laser range scanners and line cameras", Machine Vision and Applications 14 1, pp. 35-41