LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT LCS TR-729

Automatic Extraction of Textured Vertical Facades from Pose Imagery

Satyan Coorg   Seth Teller

Computer Graphics Group

January, 1998

545 TECHNOLOGY SQUARE   CAMBRIDGE, MASSACHUSETTS 02139   (617) 253-5851

Automatic Extraction of Textured Vertical Facades from Pose Imagery*

SATYAN COORG   SETH TELLER
Computer Graphics Group
MIT Laboratory for Computer Science
{satyan,seth}@graphics.lcs.mit.edu

Abstract

Extracting 3-dimensional structure from real-world imagery and rendering it from unrestricted viewpoints is an important problem in computer vision, and increasingly, computer graphics. Despite many years of research, a system that automatically recovers realistic 3-D models from images remains elusive; most practical systems require significant human input. However, unlike automatic algorithms, human-assisted systems are not *scalable*, both in terms of the number of images and the complexity of the model being reconstructed.

This paper describes an automatic 3-D model extraction algorithm based on the following ideas:

- Each image in our input dataset is annotated with accurate estimates of camera position and orientation (*pose*), to become a fundamentally more powerful datum, a *pose image*. Pose information is used in our algorithm to constrain the reconstruction process, and to focus on processing only a portion of the dataset that is relevant to a given 3-D region.
- We exploit *geometric structure* inherent in typical urban environments. In particular, we focus on vertical facades, as they are common in urban scenes. The vertical facade extraction algorithm detects likely facade azimuths using image-space information of horizontal line segments, and locates them using a space-sweep algorithm.
- We exploit the availability of a *large set of observations* of each facade to design a simple texture estimation algorithm that is statistically robust with respect to illumination changes and occlusion.

We present results of the algorithm for a large pose image dataset (consisting of about four thousand images taken from eighty-one positions) of an urban office complex. Our algorithm was successful in recovering all significant vertical facades in the complex, as well as several neighboring facades.

1 Introduction

Three-dimensional modeling of existing urban architecture has numerous applications, including virtual environments [8, 13], urban planning, military simulation, etc. It is clearly attractive to employ photographs in the modeling process: the accuracy of the 3-D model is enhanced, and the resulting environments appear visually realistic.

*Funding for this research was provided in part by the Advanced Research Project Agency under contract DABT63-95-C-0009, and in part by Intel Corporation.

One approach to perform such modeling is to recover 3-D information from photographs automatically. This has been the focus of computer vision research for several years. The research has led to significant progress in designing algorithms that analyze a few images and recover 3-D structure from them. However, a drawback of many vision algorithms is their inability to scale to large sets of images, thereby limiting their applicability to large-scale 3-D modeling. Though some recent research attempts to address this drawback (see Section 1.2), a practical algorithm for automatically reconstructing a "computer graphics" 3-D model remains elusive.

An alternate approach has been to design semi-automatic modeling systems and allow a human to aid the reconstruction process (e.g., the user marks and corresponds image features). Though such systems can yield high quality results [8], they require the user to process each image in the input dataset, and thus do not scale for large sets of images or structures.

In this paper, we present a novel algorithm that automatically recovers 3-D information by analyzing a large set of images. The algorithm is based on three ideas:

- Each image is annotated with its *pose* – the (six degrees of freedom) position and orientation of the camera in a global coordinate system. Camera pose information is used in our algorithm in several ways. First, pose information provides useful geometric constraints that aid the reconstruction process. Second, pose information enables our algorithm to focus on processing only the (potentially small) portion of the data that is *relevant* for reconstructing a given 3-D region of interest. The algorithm thus scales with the number of images and reconstructed model size.
- We focus on extracting *vertical facades*, as they are common in urban scenes. As we demonstrate, this assumption leads to an extraction algorithm based on a "detect-and-verify" paradigm: existence of horizontal line segments on vertical facades is used to detect likely facade azimuths (i.e., angles made by facades with the $\mathbf{X}$ axis), and these azimuths are verified (and located) by a space-sweep algorithm.
- We exploit a *large set* of observations of a facade to design a simple and robust technique that synthesizes a single texture from its observed color values in various images.

The output of our algorithm, texture-mapped polygons, can directly be used to generate synthetic images of the virtual environment from novel viewpoints.

1.1 Dataset Acquisition

Our dataset consists of photographs acquired by a calibrated Kodak DCS 420 digital camera mounted with fixed optical center on an indexed pan-tilt head, itself attached to a tripod base. The tripod was manually positioned at eighty-one locations (*nodes*) among the buildings of an office complex. Typical inter-node distance was about 10 meters. At each node, the camera was rotated through a predetermined "tiling" of 50-70 orientations, yielding a roughly hemispherical field of view. Apart from avoiding inclement weather and darkness, no other restriction (e.g., selection of diffuse lighting conditions) was made on the days/times during which the dataset was acquired.

To provide camera pose for each node, initial position estimates were obtained with surveying instruments. Initial orientation estimates for each node were obtained by manual pointing of the pan-tilt head at some other node (marked by a second tripod and an orange ball). However, physical instrumentation alone does not produce pose estimates sufficiently accurate for direct incorporation into our reconstruction algorithm. To ameliorate this problem, initial pose data supplied by the instrumentation was *refined* using two optimization techniques. First, a *spherical mosaicing* technique accurately computed relative rotations between images in a single node using a quaternion-based correlation maximization algorithm, generating a single *spherical mosaic* for each node. Second, a global optimization algorithm based on a few semi-automatically generated correspondences between these spherical mosaics registered them in a global coordinate system, producing *pose mosaics*. Both these techniques are described in a separate paper [6].

1.2 Comparison with Related Work

Interactive modeling systems allow a user to identify geometric features in photographs and establish correspondences between them. The constraints generated by this process are then used to solve for 3-D structure. The more successful systems [1, 8] exploit geometric structure inherent in the environment, such as blocks in [8] or parallel and perpendicular lines [1]. While such systems can generate high quality results, the user must process each input image, e.g., by identifying edges (for correspondence) and occluded pixels (for texture computation). This makes them impractical for use on large datasets, such as ours. Such systems are also difficult to assess algorithmically, as a "human-in-the-loop" is performing non-algorithmic tasks.

Mosaicing [21, 22] seamlessly stitches together multiple images taken from the same viewpoint. As in QuickTime VR [4], such mosaics can be used for walkthroughs of virtual environments. However, as mosaics themselves do not contain 3-D information (i.e., depth), the range of viewpoints is severely limited. In our work, we use mosaics to organize a large dataset to significant engineering advantage, but recover facade geometry and texture, providing the user greater navigational flexibility.

Stereo vision [9] is the classic computer vision approach to recovering 3-D information from a few (usually two or three) pose-annotated images. This technique matches corresponding features (e.g., points or edges) across images, and locates 3-D features by triangulation. A drawback of this technique is the inherent trade-off between the inter-camera distance (baseline) and ease of matching: larger baselines allow more stable triangulation, and thus higher quality 3-D models, but matching across images from widely separated cameras is an extremely hard problem. *Multi-baseline* stereo (e.g., [14]) attempts to address this drawback by using several images simultaneously. However, in order to perform automatic matching, such algorithms require images to be taken from closely-spaced cameras under stable illumination conditions. Such conditions would be difficult, if not impossible, to achieve in extended outdoor environments.

Space-sweep techniques have been used recently [5, 20] to perform matching and reconstruction from an arbitrary number of images. Unlike traditional image-space algorithms that rely on correspondence between image features, these world-space algorithms traverse the entire 3-D region of interest and identify likely locations of 3-D features. In [5], a *counting* metric based on edge pixels is used to recover 3-D information from a few (seven) aerial images. In [20], a *correlation-based* metric using pixel colors is used to identify likely locations (and colors) of 3-D voxels, which can then be used to generate photo-realistic renderings. In this paper, we employ a related space-sweep technique, but our algorithm handles an arbitrary number of general camera positions, works in outdoor scenes with widely varying illumination (unlike [20]), and generates a 3-D model suited for graphics rendering (unlike [5]).

Image-based rendering systems [11, 17, 16] use multiple images to produce the image from a novel viewpoint using either interpolation or *disparity* (depth), or a combination of both. By using images directly, such systems generate visually realistic results, while partially avoiding the difficult task of 3-D reconstruction. However, current image-based rendering systems do not provide as much flexibility as a 3-D representation (such as the facades output by our algorithm), both in terms of navigation and editing. This is a drawback in many applications, such as when a user experiments with geometry and/or lighting conditions or applies simulation techniques like collision detection.

1.3 Overview

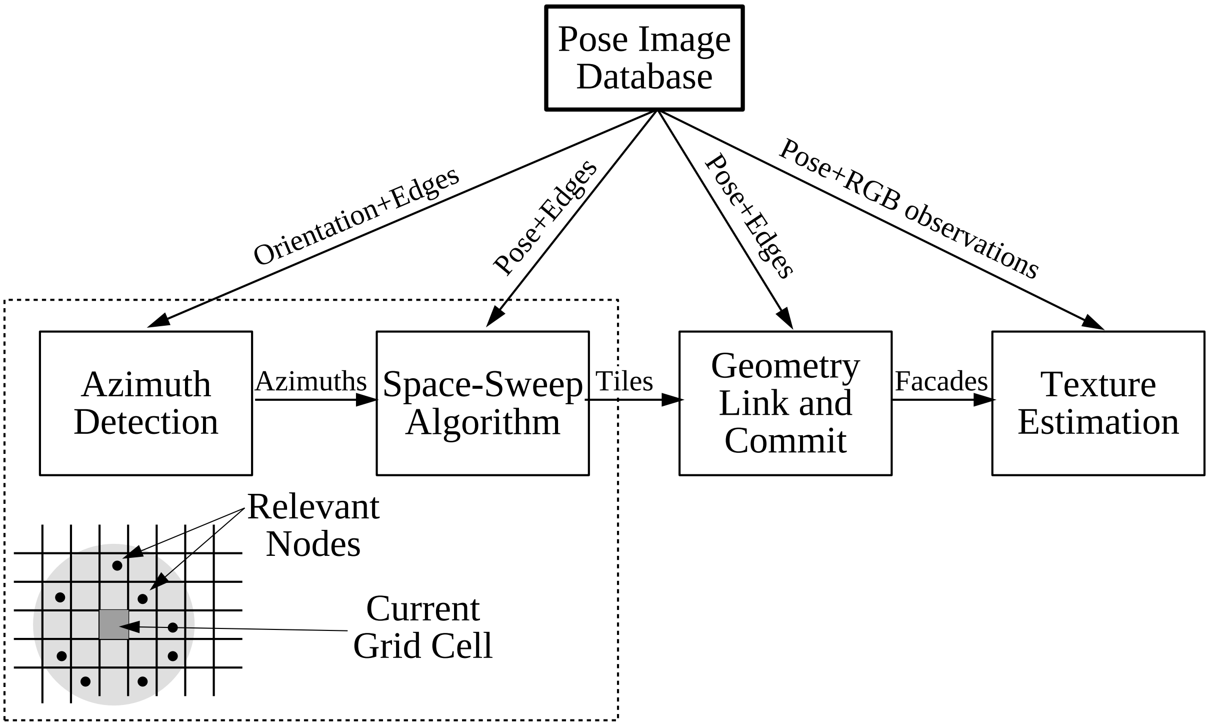


Figure 1: Overview of the Vertical Facade Extraction Algorithm.

Figure 1 shows a high-level overview of our algorithm. The algorithm executes in communication with a pose image database, which stores raw data (e.g., pose and images) as well as derived data (e.g., detected edges).

First, the algorithm divides the 3-D region of interest into a 2-D XY grid based on a user-supplied grid size G. Grid-based subdivision enables restriction of subsequent processing to nodes that are *relevant* to the grid cell (e.g., within some world-space distance D). This subdivision also decouples different parts of the scene, making the reconstruction process more robust. For each grid cell, the algorithm computes *tiles* – pieces of vertical facade – in a two step process. Likely tile azimuths are detected based on a histogramming technique (Section 2). These are verified and located using a space-sweep technique (Section 3), populating the grid cell with only those 3-D tiles that are supported by sufficient image evidence.

Second, the tile geometry recovered is linked to form complete facades; and many spurious facades in the model are removed by a *facade commitment* process (Section 4). Textures for the generated vertical facade geometry are computed from various observations (Section 5).

We report results of the algorithm on our dataset in Section 6 and conclude in Section 7.

2 Azimuth Detection

This section describes a basic technique used in our algorithm to identify likely azimuths of tiles, using horizontal line segments. Such line segments arise often in urban environments, e.g., from windows and facade boundaries. We first describe a simple technique to uniquely determine the azimuth of a tile using image-space edge information (Section 2.1). This technique is then applied to many *presumed* horizontal edges, and likely azimuths are identified by a histogramming technique (Section 2.2).

2.1 Estimating Azimuth from a Horizontal Line Segment

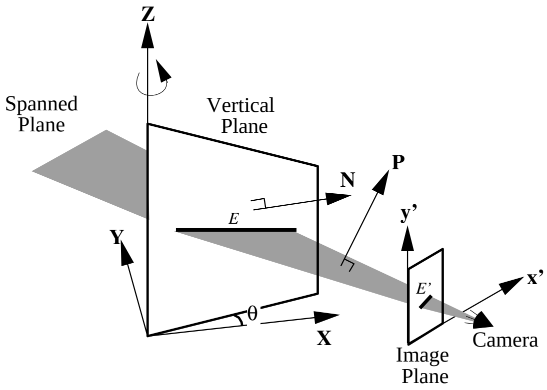


Figure 2: Deducing tile azimuth from a horizontal line segment.

The basic idea is our azimuth estimation technique is that, when pose information (specifically, orientation) is known, the direction of a horizontal edge is completely determined by the 2-D line equation of its projection. Figure 2 illustrates this idea. Let a 3-D horizontal line segment E on a tile project to a 2-D edge E' in some pose image. In the figure, the vertical plane through E makes an angle θ with the $\mathbf{X}$ axis, i.e., its *azimuth* is θ. The normal $\mathbf{N}$ to this plane is $[\sin\theta, -\cos\theta, 0]^T$. Let $\mathbf{P} = [p_x, p_y, p_z]^T$ be the normal to the plane spanned by E' and the camera. $\mathbf{P}$ (in global coordinates) is determined by the orientation of the camera (a 3×3 rotation matrix $\mathbf{R}$) and the 2-D image-space line equation $ax' + by' + c = 0$ of E' (where $a^2 + b^2 + c^2 = 1$):

$$\mathbf{P} = \mathbf{R}^{-1} \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

Then, the direction of E is given by:

$$\mathbf{P} \times \mathbf{N} = [p_z \cos\theta, -p_z \sin\theta, (-p_x \cos\theta - p_y \sin\theta)]$$

As E is horizontal, the z component vanishes. This yields two solutions for the orientation (when either $p_x \neq 0$ or $p_y \neq 0$):

$$\theta = \begin{cases} \tan^{-1} \frac{-p_x}{p_y} & \text{or} \\ \tan^{-1} \frac{-p_x}{p_y} + \pi & \end{cases} \tag{1}$$

Of these, the correct azimuth is the one that faces the viewing direction. If $p_x = 0$ and $p_y = 0$, the camera has observed a horizontal line segment at the same height, and no information can be determined. As most of the nodes in our dataset are taken from positions near ground level, this case arises very rarely.

Figure 9 (color plate) shows the results of applying this technique to two nodes. In this figure, vertical edges were first identified by thresholding ($p_z < 0.01$). For the rest, an azimuth θ is estimated using Equation 1. Note that horizontal edges from the same facade are assigned the same azimuth, even across nodes. This property is not true of azimuths computed from non-horizontal edges (e.g., tree edges); thus they tend to be uncorrelated both within a node and across different nodes.

Unlike computer vision techniques that recover 3-D orientation information from aggregate properties of a *set* of edges (e.g., moments in *shape from texture* [23] and common intersection points in *vanishing point estimation* [2]), this technique recovers 3-D orientation from a single (presumed horizontal) edge. A technique similar to ours has been used in photogrammetry to generate buildings from *monocular* (i.e., single) aerial images [7]; the novelty of our approach is the use multiple nodes to discount non-correlated azimuths, and robust azimuth verification, which is described in the next section.

2.2 Histogramming Azimuths

Our technique to identify likely azimuths is based on the following idea: 2-D edges arising from truly (i.e., world-space) horizontal line segments will be assigned identical azimuths in different nodes, whereas azimuths of non-horizontal edges will vary with node position. Thus histogramming reinforces true azimuths; the rest tend to be unsupported, as they are uncorrelated across nodes.

This idea is used in the following algorithm, which reports a set of likely tile azimuths in a grid cell C, from edges in C's relevant nodes $1 \ldots k$:

Azimuths $A = \phi$.
for node $i \in 1 \ldots k$ **do**
  Let C project to image-space region C' in node i.
  Compute tile azimuths of each non-vertical edge in C' using Equation 1.
  Histogram azimuths (weighted by edge length) into buckets[1].
  Identify the most populated bucket and add its representative azimuth (e.g., median) to A.
endfor
Histogram A and report each bucket that contains at least three nodes.

Informally, the algorithm picks a dominant azimuth from each node, then reports azimuths that are supported by several nodes. These azimuths are then verified by the space-sweep algorithm described in the next section.

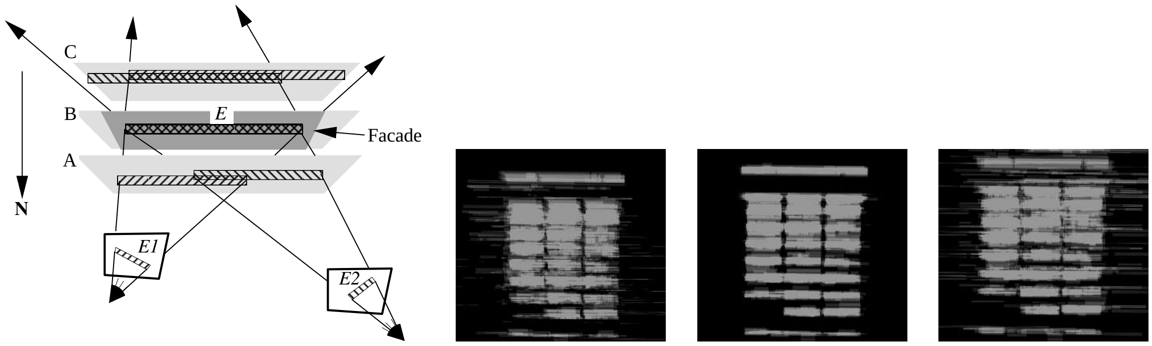
3 The Space-Sweep Algorithm

The space-sweep algorithm to locate and verify tiles is based on an *incidence counting* idea, related to that proposed by Collins [5]. If any *sparse* set of features in several nodes is projected into 3-D, regions with high *incidence* of such projections correspond to likely locations of 3-D features, for the following reasons. If a 3-D feature is present in multiple nodes, then projections through the corresponding 2-D features pass near it, increasing its incidence count. Conversely, as the set of features are sparse, it is rare that unrelated projections pass through the same 3-D region by chance.

Given an azimuth θ, the sparse features in our algorithm are edges that are likely to lie on tiles with azimuth θ. Note that it is straightforward to identify such edges in each node; they are edges whose computed azimuth is θ. For example, in Figure 9, red edges are used for tiles with azimuth$=-\frac{\pi}{2}$ (i.e., normal $= [-1, 0, 0]^T$).

Using such edges, Figure 3 shows the application of incidence counting to tile location. Part (a) shows three planes with common normal $\mathbf{N}$ and different offsets, with projected (and blurred) edges $E1$ and $E2$ from two nodes. Note that correlation between horizontal line segments is greatest at the position of the plane which corresponds to the tile location.

[1] We use overlapping θ buckets of size 3 degrees.



A B C

(a) (b)

Figure 3: This figures shows three different positions A, B, and C of a vertical plane. Part (a) shows a schematic with two nodes. Part (b) shows close-ups of the planes with line segments from a facade in our dataset. Correlation between different line segments is indicated by brightness.

Figure 3-(b) shows close-ups of three similar planes, but with several line segments from a real facade. In the figure, segments on the plane are blended together, so that regions with high incidence (and therefore high correlation) appear brighter. Note that the image of the plane appears brightest and sharpest at position B, i.e., if the plane is at the facade's location. Our technique to locate tiles is based on this idea. In our implementation, we use a correlation function defined in Section 3.1, and a space-sweep algorithm described in Section 3.2 that locates tiles using the maxima of this function.

3.1 Correlation Function

Given a plane and its associated horizontal line segments, the correlation function computes a measure of the extent of overlap between different line segments on the plane. Each line segment L is blurred into a *rectangle* of half-width σ_L to alleviate small errors in camera pose and smooth the peaks of the correlation function[2].

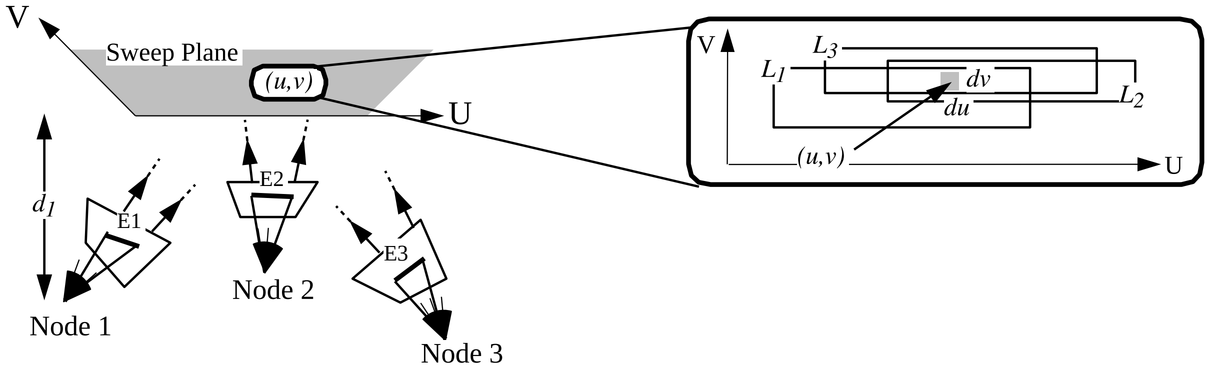


Figure 4: Parameters in the correlation function. The figure on the left shows edges $E1$, $E2$, and $E3$ from three nodes projected onto a plane. The figure on the right is a blowup of a small section of the plane. It depicts overlap of the rectangles L_1, L_2, and L_3 corresponding to the edges.

[2]We use $\sigma_l = 0.01d$, where d is the perpendicular distance between the node generating l and the vertical plane.

$$O = \int \int \sum_{i=1}^{k} w_i(u,v) \sum_{i=1}^{k} \frac{1}{d_i^2} du dv \tag{2}$$

Consider the function defined in Equation 2 (see also Figure 4). In this equation, u and v range over the plane. The point (u, v) is inside rectangles $L_1 \dots L_k$ arising from nodes $1 \dots k$; the perpendicular distance from the plane to node i is given by d_i; and $w_i(u, v)$ is the weight contributed by L_i to plane element (u, v). In our implementation, we use a triangular filter for $w_i(u, v)$, as it is both efficient to evaluate and possesses a well-defined peak (Figure 5):

$$w_i(u,v) = \max(1 - \frac{|v - L_v|}{\sigma_L}, 0)$$

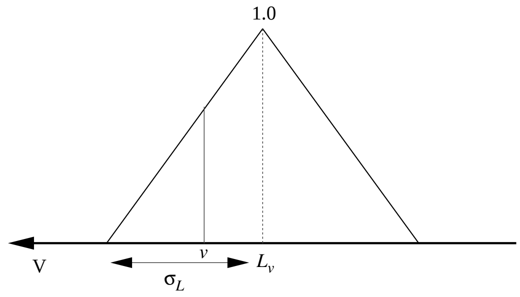


Figure 5: Triangular filter for edge overlap computation.

This correlation function has several properties that make it useful for locating vertical tiles. First, the function favors overlap between line segments arising from different nodes: "cross-terms" w_i/d_j^2 such that $i \neq j$ arise when L_i overlaps with L_j, and contribute to O. An example is shown in Figure 6, in which Equation 2 was evaluated for different plane offsets using the line segments shown in Figure 3-(b). Note that, due to the combined effect of many line segments, the function has a well-defined peak at the facade's location.

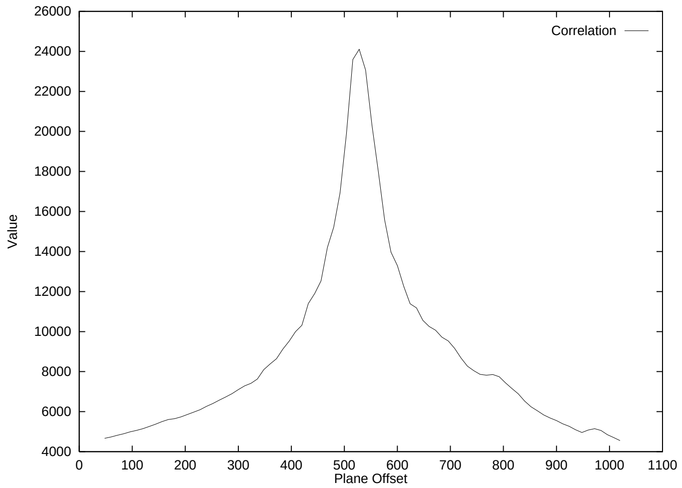


Figure 6: Correlation function values.

Second, there is no built-in bias toward planes of larger area; instead, the function downweights the area of each plane-element (u, v) by the squared-distance from the source node. This is advantageous, as such a bias would favor tiles farther away from the nodes. In effect, the function measures correlation in each node's image-space, and aggregates correlation to yield the total value.

Third, the correlation function can be evaluated efficiently by exploiting the rectangular geometric structure inherent in Equation 2. The evaluation technique is a straightforward modification of a segment-tree based plane-sweep algorithm that computes the total area of m rectangles in optimal $O(m \log m)$ time [18].

3.2 Maxima Location and Tile Generation

Given a grid cell C, the azimuth θ of a tile, and a set of nodes $1 \ldots k$, the space-sweep algorithm locates tiles as follows:

/* *Project edges from each node onto canonical planes* */
for node $i \in 1 \ldots k$ **do**
    Let C project to image-space region C' in node i.
    Let $\mathbf{P}_i$ be a plane with azimuth θ and distance 1 from node i.
    Project each edge in C' with azimuth θ onto plane $\mathbf{P}_i$.
endfor
/* *Sweep plane through C locating and generating tiles* */
for offset d in C (with step size S) **do**
    Let plane $\mathbf{P} = (\theta, d)$.
    Reproject all edges in $\mathbf{P}_1 \ldots \mathbf{P}_k$ to $\mathbf{P}$.
    /* *Identify high-incidence regions and tiles corresponding to local maxima* */
    if $O(d) > O(d+S)$ and $O(d) > O(d-S)$ **then**
        Identify regions with incidence $> K$ in $\mathbf{P}$ and corresponding node edges.
        Extrude high-incidence regions vertically to a ground plane, producing tile rectangles.
    endif
endfor

In the first phase, the algorithm identifies edges in each node that are likely to lie on the tile by comparing their azimuths with θ. For efficiency, such edges are projected to a canonical plane $\mathbf{P}_i$ (with azimuth θ) corresponding to node i. This intermediate projection permits a simple transformation – a scaling plus a 2-D translation – to be used to construct horizontal line segments on any other plane $\mathbf{P}$ with the same azimuth.

Next, the algorithm discretizes, using step size S, the set of all possible plane offsets corresponding to grid cell C. At each plane offset, it computes horizontal line segment positions and evaluates the correlation function. For each local maximum of the correlation function, it identifies regions on the plane that correspond to a tile by thresholding on incidence K, i.e., regions on the plane (if any) that overlap more than K rectangles (weighted using the triangular filter). In addition, it also identifies node edges that *support* (i.e., give rise to) the tile, by thresholding on the extent of overlap[3] with identified 3-D regions on the tile. Finally, the 3-D region information generated is converted to rectangles by extruding them onto a *ground plane*, and combining overlapping rectangles to produce tiles. The ground plane is estimated by using the z values of camera node positions.

[3] We use a threshold of $0.8l$ for an edge with length l, i.e., if more than 80% of the edge overlaps with high incidence regions identified on the tile.

The complexity of this algorithm for a single grid cell is $O(\frac{G}{S}ke(\log k + \log e))$, where G is the grid size, S is the step size, k is the number of nodes, and e is number of edges of a node that lie in the cell's projection. Thus, if the number of nodes (and edges) relevant to a grid cell is bounded by some constant, the complexity of the algorithm scales linearly with the XY area of the 3-D region of interest.

4 Geometry Link and Commit

The space-sweep algorithm of the previous section populates a set of grid cells with tiles likely to exist in the 3-D world. In this section, we describe two techniques to enhance the quality of the generated 3-D model. We first describe a technique to link tiles from different grid cells to form complete facades. Then, in Section 4.1, we use a priority ordering-based heuristic to eliminate many spurious facades reported by the space-sweep algorithm.

The algorithm to form facades from tiles is straightforward: it first links tiles with similar azimuths and offsets across neighboring grid cells, then performs a depth-first-search to identify connected components of tiles. Connected tiles are then merged into a single facade, and the azimuth and location of the combined facade are recomputed, for greater accuracy, using the algorithms of Section 2 and Section 3.

4.1 Facade Commitment

In this section, we describe a technique to eliminate spurious facades present in the recovered 3-D model. Our technique is based on the idea of facade *commitment*, which enforces the following constraint: a facade commited to the model *precludes* edges that gave rise to it from supporting other facades. Such enforcement can result in the removal of other facades, if the number of observations supporting them falls below the incidence threshold K of Section 3.

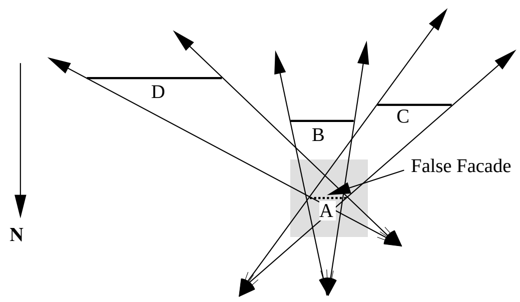


Figure 7: This figure shows one way in which a spurious facade can result by the interaction between unrelated facades of the same azimuth.

Figure 7 shows (in 2-D) an example of how a spurious facade might arise. As the three real facades B, C, and D have the same normal $\mathbf{N}$, there is sufficient evidence to report (and verify) even the spurious facade A located in the shaded grid cell. Such spurious facades can arise even from a single facade, due to the interaction between repeated texture on the facade.

Our spurious facade elimination technique consists of the following steps. First, the algorithm orders all facades using some criterion that favors real facades over spurious ones. Facades are then committed to the model in this order. Edges giving rise to facades earlier in the ordering are removed from later ones. For example, in Figure 7, if the ordering favors either B, C, or D over A, it would preclude at least one of A's observations from supporting it, resulting in A's removal.

We considered two possible facade orderings:

1. Order facades according to *occlusion*, similar to [20]. That is, if A and B are two facades such that $A < B$ in the ordering, only A can occlude B from any node, not vice-versa. Such an ordering exists for facades with the same normal – in Figure 7, it is simply the ordering opposite $\mathbf{N}$. Unfortunately, as this ordering favors locations closer to the nodes, it tends to select spurious facades. For example, in Figure 7, facade A would be selected, possibly suppressing the real facades.

2. Order facades according to *higher reported observations* (measured by total length of horizontal segments on the facade). This ordering heuristic favors facades that are larger in area and/or have more line segments. In Figure 7, each of the facades B, C, D is larger than A, resulting in A's removal after facade commitment.

In practice, we have observed that the first ordering tends to break up facades into several pieces before their actual location, whereas the second, by favoring larger facades, tends to retain real facades. Our results in Section 6 are computed using the second ordering.

5 Texture Estimation

Thus far, we have described an algorithm that recovers vertical facade geometry from pose imagery. In order to enhance realism of the generated geometry, we now describe an algorithm to synthesize a single (diffuse RGB) texture for a facade given its various observations. Note that the set of nodes that observe the facade are known, having been reported by facade extraction algorithm.

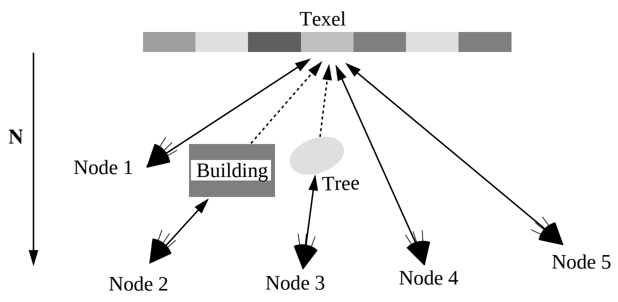


Figure 8: This figure illustrates the problem of texture estimation from various node observations.

Figure 8 illustrates the process of estimating a texture for a facade from its observations. Note that each observation can report very different colors for a texel on the facade, due to occlusion, obliqueness, illumination variation, etc. Figure 10 (color plate) shows such effects in several nodes of our dataset which were used to identify a facade in our office complex. The relevant pixels from the nodes are shown *rectified*, i.e., projected (and clipped) onto the facade. Note the significant variations in pixel color due to changes in illumination and the effects of shadows and reflections. In addition, parts of the facade are occluded by various objects (e.g., trees) in most nodes, making it difficult to determine (and use) a single "best" node, or even interpolate between various nodes based on viewing direction [8]. Instead, it is necessary to combine the information present in all relevant nodes to compute a single facade texture. This is a formidable task due to the sheer size of pixel data that has to be processed, even for a human-assisted modeling system.

One possible automatic approach would be to fit the the various observations onto a standard reflectance model and recover coefficients of the model. An example of this technique is [19], where the authors estimate specular and diffuse coefficients of a small object from many images. However, such algorithms tend to be less effective when there is significant occlusion and illumination variation. Instead, we employ a technique based on median statistics, making it robust with respect to such variations.

5.1 The Median Texture

Our technique divides the facade into a (area dependent) number of texels, and computes a single diffuse value for each texel from values of all its observations. The method consists of two steps: first, it converts RGB color values into an appropriate color space; second, it picks the "best" color value for the texel, and converts the result back to RGB.

As raw RGB color of the same texel can vary significantly in each node, we use a color space where at least some of the components are stable under different lighting conditions. In our implementation, we use the CIE xyY color representation [10], which decouples chromaticity x, y from luminance Y. Under illumination by (predominantly) white sunlight, the luminance Y of a texel can vary significantly, but its chromaticity remains reasonably stable.

We use the *median* to compute a single x and y value for each texel from observed x and y values in various nodes. The median possesses useful robustness properties, i.e., it is less sensitive to *outliers* than is simple averaging [12]. This is important in our application as outliers are fairly common, typically arising from occlusion by foliage or other structures. Fortunately, such outliers do not appear at the same texel on different rectified nodes (i.e., they exhibit *parallax* [15]), allowing the median to generate a good estimate of the texel's chromaticity.

We also use the median to select a single luminance value for the texel. Even though luminance values vary significantly, we have observed that the median luminance reasonably reflects the luminance of the texel under "average" lighting conditions. In any event, the differences in illumination are normalized away, so that the generated texture can be subjected to different lighting conditions.

Weighted Observations

Note that the observation *quality* of a texel varies across different nodes. For example, nodes that view a texel obliquely or from a distance generally yield poor observations. Thus we use the *weighted* median in our algorithm. The median v_m of values $v_1 < v_2 \ldots < v_k$ weighted by $w_1 \ldots w_k$ is given by the index m, where:

$$m = \max\{j \text{ such that } \sum_{i=1}^{j} w_i < \frac{\sum_{i=1}^{k} w_i}{2}\}$$

For the weights w_i, we use the dot product $\mathbf{N} \cdot \mathbf{V}$, where $\mathbf{N}$ is the facade's normal, and $\mathbf{V}$ is the vector direction from the texel center to the node. This downweights nodes that view the texel obliquely. We have observed that incorporating a distance term (e.g., weighting by the texel's solid angle), tends to pick the node that is closest to the texel, losing much of the robustness of the median. In any case, nodes that are far from the texel do not contribute observations, as they are excluded by the geometry extraction algorithm.

Figure 11-(a) (color plate) shows the results of the median texture algorithm using these weights. Note the automatic removal of most occlusion from the the texture; the luminance pattern on the texture is also reasonably approximated. Compared to Figure 10, the median texture has less occlusion, fewer changes in luminance across the texture (e.g., due to shadows), and fewer view-dependent effects (e.g., reflection). The texture is slightly blurred due to small errors in camera pose; the next section describes a technique to ameliorate this problem.

5.2 Texture Sharpening

Our iterative technique for "sharpening" the median texture consists of two steps. The first step *rewarps* each rectified node to achieve a higher correlation with the median texture. The second step recomputes the median using the new values. These steps are repeated until the median values converge.

Our rewarping technique is a correlation-optimization based on luminance values, identical to mosaicing optimization [21, 22]. Briefly, the optimization uses a 8-parameter, 2-D projective transformation described by a 3×3 matrix $\mathbf{M}$:

$$u = \frac{\mathbf{M}_{00}x+\mathbf{M}_{01}y+\mathbf{M}_{02}}{\mathbf{M}_{20}x+\mathbf{M}_{21}y+1} \quad v = \frac{\mathbf{M}_{10}x+\mathbf{M}_{11}y+\mathbf{M}_{12}}{\mathbf{M}_{20}x+\mathbf{M}_{21}y+1}$$

In our algorithm, u, v are texture coordinates, and x, y coordinatize the rectified node. Starting from an identity matrix, our algorithm computes $\mathbf{M}_i$ for each node i by minimizing the sum-of-squared differences of luminances:

$$\sum_{x_i, y_i} [Y'(u,v) - Y_i(x_i, y_i)]^2$$

To compensate for illumination variation between a rectified node and the median texture, we normalize the luminance values by equalizing the average and standard deviation for each 64×64 patch on the rectified node with the corresponding patch on the median texture.

Figure 11-(b) (color plate) shows the results of this process. Note the significant improvement in the quality of the texture. Even though the color of each texel is computed independently, straight lines on the facade are clearly demarcated. Despite this optimization, it is possible for some blurring to persist in the texture. This is caused by parts of the facade extruding out of its plane. One possible solution would be to construct disparity maps to capture such extrusions, as in [8].

6 Results

We have implemented the algorithm (and associated visualization) described in this paper in about 5000 lines of C++ code. In addition to extracting all significant vertical facades in the office complex (the primary focus of the dataset), the algorithm surprised us by automatically extracting several neighboring facades. Details on the extraction process and algorithm execution times are provided below.

Nodes	81
Images	~ 4000
Resolution	762×506 pixels

Table 1: Input characteristics.

Table 1 lists the characteristics of the input dataset. The facade extraction algorithm considered edges detected on six faces of a cubical environment map representing a node. Each face of the environment map was generated at 1024×1024 resolution by resampling the input images. Edge pixels were detected using the Canny edge detector [3] and converted to line segments by linking pixels with similar gradient orientation. Approximately 1000 edges were computed for each cube face (ignoring edges less than 10 pixels in length).

Some of the important parameters supplied to the algorithm are listed in Table 2. The grid size G should be approximately equal to the size of the smallest facade, to avoid interaction between different facades during tile reconstruction. We use a grid size of 10 meters for this dataset. The minimum weighted incidence value of 3.0 usually implies that a facade must be observed by at least five or six nodes to be successfully extracted.

Area of 3-D region of interest	$\sim 500m \times 500m$
Grid Size G	$\sim 10m$
Far Distance D	$\sim 100m$
Step Size S	$\sim 0.1m$
Minimum Weighted Incidence K	3.0

Table 2: Parameters supplied to the extraction algorithm. All lengths are given in meters.

The facade extraction algorithm took about seven hours on an SGI O2 workstation with one R10000 processor, most of which was spent in the space-sweep algorithm. The space-sweep algorithm recovered approximately 2000 tiles. The model consists of about 140 facades after tile linking and facade commitment (Figure 12-(a)). After removing facades with area less than $100m^2$, the model consists of about 40 facades (Figure 12-(b)). The horizontal lines used to generate this model are shown in Figure 12-(c). Figure 13 shows the extracted facade geometry in wireframe, co-located with the input data (shown as texture-mapped spheres).

The recovered facade geometry was further processed using the following steps:

- Adjacent facades (those with end-points in the same grid cell) with different orientations were linked, modifying facade boundaries to the edge formed by intersecting adjacent facades. Also, a "roof" was added to each connected set of facades after equalizing facade heights to the maximum height in the set.
- A 2-D Delaunay triangulation of the camera XY positions was computed, and an approximation to the ground terrain was constructed using ground heights obtained from camera poses.

Textures were computed to $0.1m$ resolution, with the largest texture containing 1024×512 pixels. Texture computation took about ten hours. Textures for non-vertical geometry (i.e., roof and ground polygons) were extracted from a single aerial pose image of the complex.

Figure 14 shows one frame of a real-time virtual flythrough of the final reconstructed model on an SGI O2 workstation.

7 Conclusion

This paper described an algorithm that extracts urban vertical by detecting likely vertical facade orientations from horizontal edges, and locating these using space-sweep. In addition, we described a simple and robust technique for computing a texture-map for each recovered facade. We presented results on a large pose image dataset consisting of over four thousand images. To our knowledge, ours is the first system to automatically analyze such a large set of images, and produce a realistic 3-D model suitable for computer graphics rendering.

Our algorithm has other desirable properties for automatic model extraction. It uses all information from relevant images, yet scales to an arbitrary number of images and model size. It exploits geometric constraints inherent in the 3-D environement. Finally, it is robust with respect to occlusion and changes in illumination. We believe that these properties will be crucial for future systems that perform practical, large-scale reconstruction.

Several improvements are possible for the generated model. Our algorithm typically overestimates the extent of vertical facades (e.g., by extruding to the ground or choosing the maximum possible height for the facade). This could be corrected by clipping the model to the edges of computed textures. Also, textures could be augmented with disparity maps to model small extrusions, as in [8].

Other areas for future work include designing pose imagery-based extraction algorithms to recover more general geometry (e.g., arbitrarily oriented facades and non-facade structures), and recovering more general reflectance properties (e.g., specular coefficients, BRDFS) of the model.

References

[1] S. Becker and J. V. Michael Bove. Semiautomatic 3-d model extraction from uncalibrated 2-d camera views. In *Proceedings of Visual Data Exploration and Analysis II, SPIE Vol. 2410*, pages 447–461, 1995.

[2] B. Brillault-O'Mahony. New method for vanishing point detection. *Image Understanding*, 54:289–300, 1991.

[3] F. J. Canny. A computational approach to edge detection. *IEEE Trans PAMI*, 8(6):679–698, 1986.

[4] S. E. Chen. Quicktime VR - an image-based approach to virtual environment navigation. In *SIGGRAPH 95 Conference Proceedings*, pages 29–38, Aug. 1995.

[5] R. Collins. A space-sweep approach to true multi-image matching. In *CVPR96*, pages 358–363, 1996.

[6] S. Coorg, N. Master, and S. Teller. Acquisition of a large pose-mosaic dataset. Technical Report TM-568, Laboratory for Computer Science, MIT, 1998.

[7] J. D. M. McKeown, C. McGlone, S. D. Cochran, Y. C. Hsieh, M. Roux, and J. Shufelt. Automatic cartographic feature extraction using photogrammetric principles. In *Digital Photogrammetry*, pages 195–212. ASPRS, 1997.

[8] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *SIGGRAPH 96 Conference Proceedings*, pages 11–20, Aug. 1996.

[9] U. R. Dhond and J. K. Aggarwal. Structure from stereo – A review. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(6):1489–1510, Nov.-Dec. 1989.

[10] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics, Principles and Practice, Second Edition*. Addison-Wesley, Reading, Massachusetts, 1990.

[11] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In *SIGGRAPH 96 Conference Proceedings*, pages 43–54, Aug. 1996.

[12] P. J. Huber. *Robust Statistics*. Wiley, 1981.

[13] W. Jepson, R. Liggett, and S. Friedman. An environment for real-time urban simulation. In *Proc. 1995 Symposium on Interactive 3D Graphics*, pages 165–166, 1995.

[14] S. B. Kang and R. Szeliski. 3-D scene recovery using omnidirectional multibaseline stereo. In *International Conference on Computer Vision and Pattern Recognition*, pages 364–370, San Francisco, CA, June 1996.

[15] R. Kumar, P. Anandan, and K. Hanna. Shape recovery from multiple views: a parallax based approach. In *ARPA Image Understanding Workshop, Monterey, CA*, Nov. 1994.

[16] M. Levoy and P. Hanrahan. Light field rendering. In *SIGGRAPH 96 Conference Proceedings*, pages 31–42, Aug. 1996.

[17] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. In *SIGGRAPH 95 Conference Proceedings*, pages 39–46, Aug. 1995.

[18] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction*. Springer-Verlag, 1985.

[19] Y. Sato, M. D. Wheeler, and K. Ikeuchi. Object shape and reflectance modeling from observation. In *SIGGRAPH 97 Conference Proceedings*, pages 379–388, Aug. 1997.

[20] S. Seitz and C. Dyer. Photorealistic scene reconstruction by voxel coloring. In *CVPR97*, pages 1067–1073, 1997.

[21] R. Szeliski. Video mosaics for virtual environments. *IEEE Computer Graphics and Applications*, 16(2):22–30, Mar. 1996.

[22] R. Szeliski and H. Shum. Creating full-view panoramic mosaics and texture-mapped 3D models. In *SIGGRAPH 97 Conference Proceedings*, pages 251–258, Aug. 1997.

[23] A. P. Witkin. Recovering Surface Shape and Orientation from Texture. *Artificial Intelligence*, 17(1-3):17–45, Aug. 1981.

Figure 9: This figure shows two nodes (four faces of a cubical environment map) along with their (long) edges. Vertical edges are colored *blue*, and other edges are colored with (absolute values of) the tile normal derived from their azimuth (e.g., *red* is $[1, 0, 0]^T$, *green* is $[0, 1, 0]^T$, etc.).

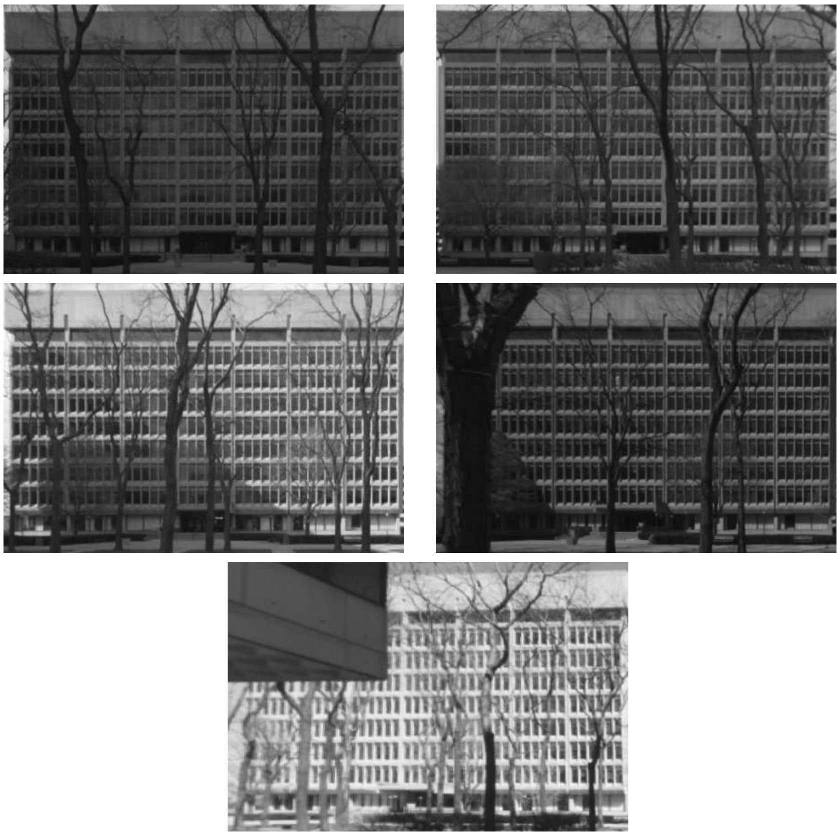


Figure 10: This figure shows (the relevant portions of) five nodes rectified to a facade. Note the significant differences in illumination, and the effects of shadows, reflections, and occlusion.

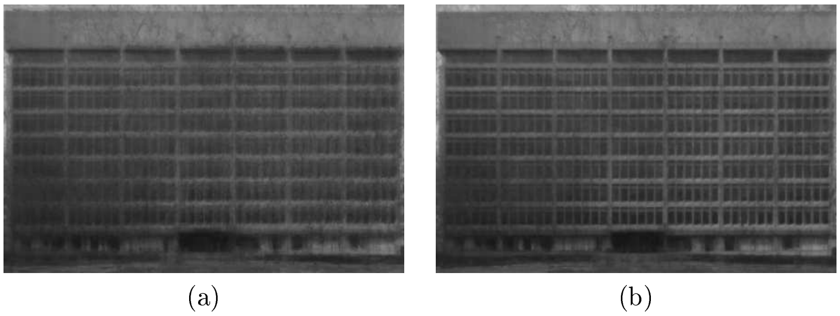


Figure 11: Part (a) shows the median texture. Part (b) shows the median texture after sharpening. Note the removal of occlusion, shadows, luminance variations, etc. from the texture.

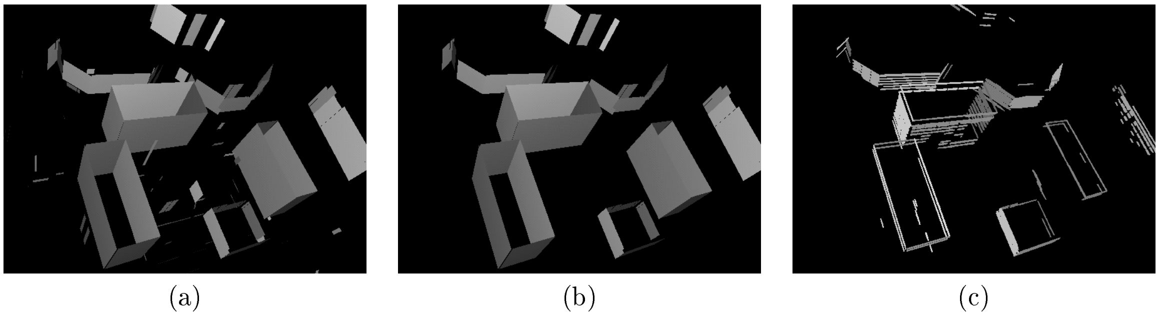


Figure 12: Part (a) shows the recovered vertical facades. Part (b) shows the model after removal of small facades, and Part (c) shows the horizontal lines that generate this model.

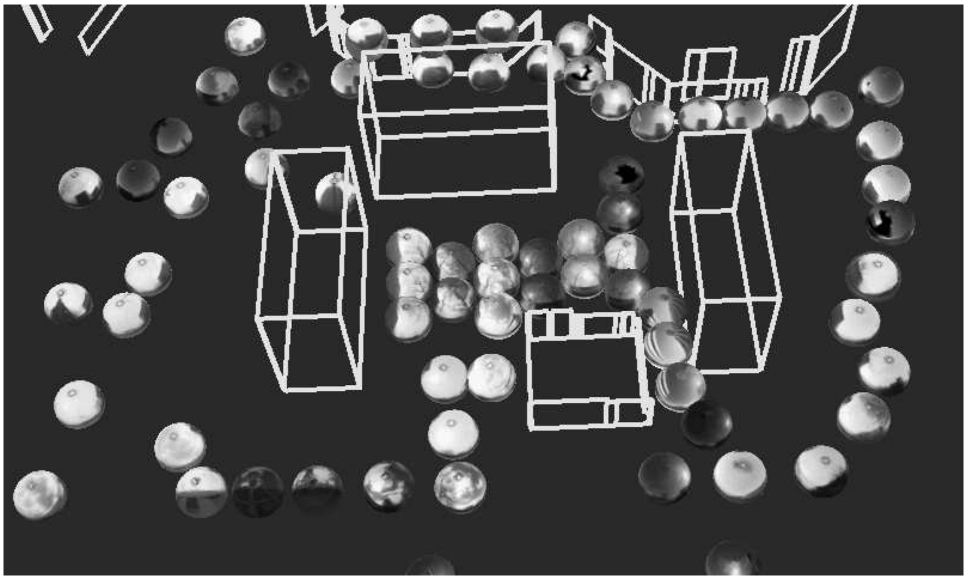


Figure 13: The extracted vertical facades (in wireframe) co-located with input pose mosaics (drawn as spheres).

Figure 14: An aerial view of the final model.