B. Girod, G. Greiner, H. Niemann, H.-P. Seidel (Eds.)

Vision, Modeling, and Visualization 2000

Proceedings
November 22-24, 2000
Saarbrücken, Germany

AKA

Computer Assisted Reconstruction of Buildings from Photographic Data

M. Tarini* , P. Cignoni†, C. Rocchini‡ and R. Scopigno§

Istituto Elaborazione dell'Informazione - C. N. R.
via V. Alfieri 1,
56010 S. Giuliano T. (Pisa)
ITALY

Abstract

We present an approach to the reconstruction of 3D textured models of buildings starting from photographic data. We have defined a simple and easy to use interface that allows inexperienced users to be driven in the process of entering all the inputs (points, segments and so on, detected on the photo) necessary to define the geometry of the building. The basic building block is the rectangle, and mechanisms are provided to make the interaction intuitive, fast (in the order of few minutes for reconstructing a simple model of few tens of quads), and leading to relatively precise results. The system then builds the corresponding mesh, previewing it directly using perspective textures, or resampling rectified texture to export the reconstructed model into a common mesh format.

1 Introduction

In the field of automatic acquisition of 3D models, a special case is represented by building acquisition. Direct CAD modeling requires, even for a poorly detailed model, very large amount of data to be retrieved and input by expert users, and often leads to artificial-looking, error prone results. On the other hand, typical automatic acquisition tools (laser scanners, structured light, silhouette based techniques, stereography, TAC scans) appear far too complex, expensive, time consuming for the case (when not totally inadequate for targets as large as a buildings). For most applications purposes, in fact, the model of a building may be approximated with a very simple geometry, characterized by parallel flat surfaces and orthogonal lines. Those features may be exploited in order to reconstruct the model geometry even from a single, uncalibrated picture, with few "hints" from an human user. In the same time, a wealth of realistic detail can be retrieved from the same picture as textures, in a fully automatic way.

In order to obtain reasonably accurate textured meshes in the cheapest possible way, the main concern is to *minimize the effort* needed by the user (the computation time has proven short, and is anyway much cheaper). Another concern is *flexibility*: the approach should be able to deal directly with a great variety of buildings forms. Finally, we aim at the quality in the output standard meshes.

2 Related Work

Some other works have been presented to acquire building geometry, exploiting the regularity typical of many architectural styles which allows to infer geomery from the direct use of prospective geometry rules.

*Email: mtarini@di.unipi.it
†Email: cignoni@iei.pi.cnr.it
‡Email: rocchini@iei.pi.cnr.it
§Email: roberto.scopigno@cnuce.cnr.it

Some of these methods use, as input, a large set of closely spaced images [9, 1], while others concentrate on what can be recovered from a single, or a very few, images.

The technique presented in [5] integrates a method to acquire models in a particular format (using view dependent textures) as well as the method required to render them, leading to an integrated acquire-and-render tool (while our approach is aimed at producing exportable models). Also, in [5] multiple images are necessarily used, each with known position and calibration. The interface proposed by this approach looks quite complicated, consisting in identification of edges, manual matching of them to correspondent part of predefined 3D building blocks and definitions of relations between building blocks. Another, similar technique is presented in [7], using results from [3] and [4]. Here the camera calibration is computed as a part of the method, and therefore its knowledge is not needed, and single view can be used.

In all those papers, the focus is on what can be evinced from the photos and some auxiliary data (identification of lines, planes etc), and not on how a common user could introduce those required data.

Finally, the intelligent scissors method presented in in [8] can be integrated in our approach to identify, faster and better, lines on the photo, helping in the task of minimizing user effort, and improving result precision.

3 Overview

In the approach we are proposing the user interactively supplies all the info needed to reconstruct a 3D model starting from photographic data, in the form of a purely two-dimensional identification of point, lines and rectangles on the image. During this process the system uses all the data collected since then both to build the model (the user can see at any time what has been reconstructed) and to help the user in the subsequent input of new data (e.g. placing new points is an operation that is aware of previous inputs). In Section 4 and 5 we will (briefly) expose the techniques used to build the 3D textured model starting from photographic data and the user input, while in Section 6 we will describe the required used interaction and how the system exploits the collected data to help the user to supply new data.

4 Basic geometry

The basic element used in the mesh reconstruction is the matching between a the flat rectangle in 3D world (r^W from now on) and its projection visible on the photograph image r^I; we refer to the couple (r^W, r^I) simply with the term r. Each r^I shape is an irregular convex quad. At the end of the process, the acquired model is represented by a set of textured flat rectangles, one corresponding to each r^I. Rectangles are very versatile for a great variety of buildings (not all, though; for example, round surfaces, like columns, and pyramids could be dealt better with an approach based on solid blocks).

Each r^W lies on a plane belonging to a *set of parallel planes* (p from now on), which can host more that one r^W. Internally, each p is represented by two 2D points on the image, the vanishing points of two orthogonal lines on any plane of that p. A p can be identified by two distinct sets of 2D segments in the picture, each set containing projections of reciprocally parallel 3D segments; in each set the intersection of the prolonged 2D segments produces one of the two vanishing points.

With three[1] p's relative to three reciprocally orthogonal planes, the calibration of the camera can be evaluated [7, 10].

Given a calibration and a p it is possible to compute the 3D normal of its planes, and also the two orthogonal 3D directions of the two orthogonal lines on those planes. This means that we can correctly orient in 3D any r_i laying on that p. Still, an ambiguity remains on the position of a given r_i^W with respect to

[1] Two p are sufficient if the central point is assumed to be in the center of the picture.

other r^W's, and on its size: we do not know on which plane of a p a given r_i^W lays. This ambiguity cannot be solved without some additional data; infact, in theory, there is no difference, in the original photo, between an projection of a given rectangle and the projection of a smaller, nearer rectangle.

Also, not knowing absolute camera position, it is impossible to find any absolute position or size of the model: instead we aim to obtain a model in which all the positions and sizes are expressed with a coherent unit and in a common axis system. This can be solved by electing an r_b as base reference and by *locating* (i.e. resolving the aforesaid ambiguity) any other r with respect to it.

Locating the r_b^W means that we set one of its vertices as the origin of the whole model and we assign to r_b^W an arbitrary size (obviously respecting its aspect ratio). Then, given a r_i already located, any other r_j^W can be easily located with respect to r_i^W knowing the 2D position, on the photo, of a point lying:

1. on the intersection of the planes hosting r_j^W and r_i^W;
2. on plane hosting r_j^W and projection of that point on the plane hosting r_i^W.

In a great majority of cases, there are easily identified points to localize, in either way, all the r (the most common case, i.e. identification of a point in the intersection of two r, can be seen as a particular case of both 1 and 2 above).

5 Texture synthesis

Knowing the camera calibration, it is possible to compute (as in [7]) a mapping $m_i(h_{x,y})$ from each point h inside a given r_i into a point inside two-dimensional rectangle representing the same r_i as seen from front, i.e. removing the perspective distortions.

The inverse mapping $h_{x,y} = m_i^{-1}(k_{u,v})$ allows us to build for each rectangle r_i^W its texture t_i, finding for each of its texel position $k_{u,v}$, the corresponding position $h_{x,y}$ in the image (since $h_{x,y}$, in general, is not an integer value, rather then just rounding its components, anti-aliasing effect can be obtained by using its fractional parts as weights for averaging the appropriate four pixels of the original image).

The size and aspect-ratio of the textures t_i are not strictly dependent on the corresponding r_i^W but are chosen to be equal to the same number of pixel of the average of the two corresponding sides of r_i^I. For example, for a vertical r_i, the height of t_i is the average of the lengths, in the picture, of the left and the right edge of r_i^I. In this way we avoid to waste texture space [or original image information] by creating texture that are excessively large [or small].

The reconstructed textures t_i can be packed in a single rectangular texture. It has been chosen a simple, ad-hoc packing algorithm, that, even if it does not guarantee optimality, has proven to be efficient and produces compact textures.

6 Interface

The elements presented in the previous sections are very basic, but still are sufficient to reconstruct a textured mesh starting form a photo and some image elements (edges, corresponding point pairs, etc). The system interface, described in this section, supports an inexpert user in all the steps needed to perform the above atomic actions, and simplifies his/her work exploiting the redundancy that in many case is present in the data (as an example, a 2D point elected as a vertex of an r_i^I could also be a vertex of another r_j^I).

The proposed interface is based on a set of tools that drive the user in the process of 3D reconstruction of a building.

The first tool to be used, and initially the only available, is the *plane identifier*. The user is asked to determine on the photograph three sets of segments oriented in three orthogonal directions. Three vanishing points are detected by computing intersections of the extensions of those segment, and each pair is used to identify a different orthogonal p; the three p are in turn used to obtain the cal-

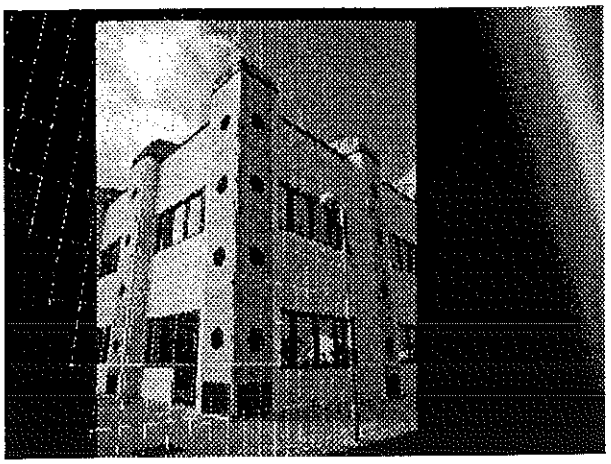

Figure 1: Visualization of currently selected plane set, on which a rectangular election will be drawn.

ibration for the given picture. Subsequent use of the plane identifier tools requires only two set of segments to be identified, and the third vanishing point is computed through the aforesaid calibration. In any case, only two segments per set are needed, but if more are provided then the redundancy is used by the system to improve precision.

Once at the first p triplet is determined, the *rectangle identifier* tool becomes available and can be used to determine r^I's. First of all, a p is selected, switching between all p's present at the moment (Figure 1 shows how the currently selected p is visualized). Then the user can interactively identify a r^I by a "rubber banding in the perspective space" (i.e. one vertex of the r^I is placed in the spot where the mouse is when the button is pressed, the opposite one is dragged around with the mouse until button is released, and meanwhile the other two vertexes are computed using intersections with the current p vanishing points).

With the *retouching* tool user can change a previously created r^I by selecting one of its vertices or sides and dragging it in a new position. Changing the position of a side changes the position of the two delimiting vertexes, while moving a vertex causes the automatic change of its two adjacent vertexes, so that the parallelism of edges of the corresponding r^W is respected.

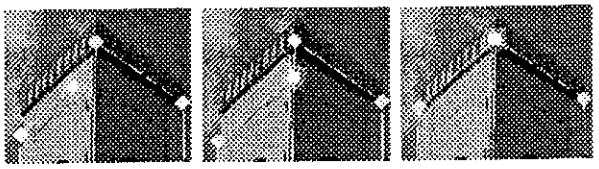

Figure 2: Visual feedback of a vertex, not being attracted (left), or being attracted to a line (middle), or to another vertex (right).

A snapping mechanism is provided to facilitate the definition of perfectly adjacent r's. Whenever a vertex is moved (or created) nearby another vertex or line, the former is "attracted" onto the latter, meaning that it is moved in the same position. There's a threshold distance for attracting element, varying with the zoom-ratio, and the attracting mechanism can be overridden keeping an apposite key pressed. Each point has a small square handle that allow interaction with it. The orientation of the handle provides a graphical feedback on whether the currently moved vertex (or side) is being attracted by a line, or by a vertex, of it is not attracted at all (see Figure 2).

During the drawing operation, to cancel the attraction is sufficient to move the attracted vertex away from the attractor. However, when the button is released, the "attraction", if present, becomes definitive and the two element become "glued" one to the other, meaning that any subsequent change of one of them affects the other (or the others). More than two vertexes can be glued in the same position. If necessary, "gluing" can be undone by moving one of them away from the other(s) while pressing another apposite key.

This automatic movement of glued elements, together with the movement of a vertex triggered by the movement of one vertex on the same r^I (see above), and movement of vertexes due to attraction, causes a sort of cascade effect of updates (see Figure 3). Retouching a vertex, for example, can cause the movement of two vertex on the same r_i^I, which in turn causes the movement of a side of another r_j^I glued to one of them, and so on. The final combined result is seen by the user as the

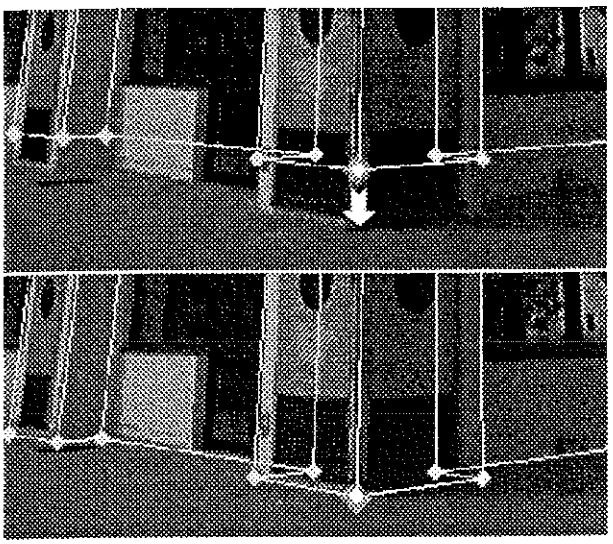

Figure 3: Example of cascade effect: moving a single vertex, as the arrow shows, causes several other vertexes to move accordingly.

consequence of that single editing operation (intermediate step of Figure 4, for example, is never seen).

To avoid a infinite recursion, first of all we forbid vertex to move (and cause cascade movements) if its destination is very near (a fraction of a pixel) to its current position. This prevents vertexes to mutually cause small displacements as a consequence of rounding errors.

To deal with the other cases of infinite recursion, an easy way would be to limit each vertex to be moved at most once as a consequence of a single user operation. But, this would not allow situations like the one shown in figure 4, where one change is propagated forth and back to vertex being moved by user, which proved to be very useful in order to quickly determine precise r's. Therefore, we preferred to assign a priority to each vertex movement, depending on the nature of the triggering original event (from lower to highest: user determined, attraction to a line, attraction to a vertex). For a given input, a vertex will only move as a consequence of a movement with higher priority of the one that caused its last movement.

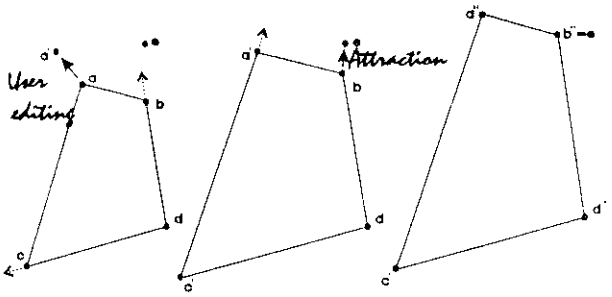

Figure 4: Example of back-cascade effect. Left: users moves vertex a in a', causing also the movement of b in b' and c in c'. Middle: now b' is near enough to another vertex e, and therefore is moved over it, in b''; this causes also d to move in d' and a' to move for the second time, in a''.

7 Cutting rectangles

Given a defined r and selecting the appropriate tool, the user can cut a polygonal shape in it, dividing it in two distinct parts.

One of the two parts can then optionally be deleted, producing in the r an hole or an irregularly shaped contour (depending on which part is trashed). The internal part, if not trashed, can be "pushed in" or "popped out" along the normal of the r^W, adding to the final model, in the process, a quad for each side of the polygonal shape. This can be useful both for quickly defining some classes of topological detail (like the typical recesses of windows, doors, and the like), and also to increase the number of building forms that can be dealt with. During 3D model construction, irregular polygonal shapes need retriangulation.

The drawing of the polygonal shape inside a r, for a better precision, is done on a temporary, rectified image representing the r being cut rather then directly on the original photograph, and with the aid of mechanisms to draw perfectly vertical or horizontal lines.

To avoid confusion, the shape, whether extruded in/out, or deleted, is a feature totally below the level of r's, meaning, for example, that extra points added at the polygonal vertexes, or the extra quads used for the extrusion effect, do not attract other r^I vertexes.

8 3D model reconstruction

The system supposes that elements (r^l vertexes or sides) "glued" one to another are actually coincident (or laying one on the other) in the 3D model. In this manner, the information needed to "locate" (see Section 4) one r to another r is obtained without any additional user input.

Still, sometimes some r^l's are not "glued" to any other r^l in the model. To discover if this is the case (which actually happens only rarely), before computing the 3D model, the system computes a connection graph $\mathcal{A}$ (nodes representing r^l's and arcs connecting two r^l's with at least an element of the first glued to an element of the other), and, if $\mathcal{A}$ is not connected, it finds out its biggest connected subgraph $\mathcal{A}'$. Any r^l not in $\mathcal{A}'$ is ignored until the user adds one or more "connection lines" (l^l, a new entity consisting in a single line drawn, using the suitable tool, on the current plane).

At any moment, the user can see the model resulting from the currently defined r^l's. In fact, once the user has input the data, the only time consuming operation of model reconstruction is the resampling one, used to build the textures for the model (see Section 9 for some quantification). Using projective texture (see [6]), the resampling step can be avoided at all: the original image is directly used as texture, allowing the user to see the ongoing model in real time.

Resampling and packing of textures must eventually be done in order to export the result in common 3D mesh formats that does not support projective textures (like VRML or other modeling package formats).

9 Results and Conclusion

A prototypal application (Figure 5), which uses just a photograph at a time, has been developed to test the interface usability and

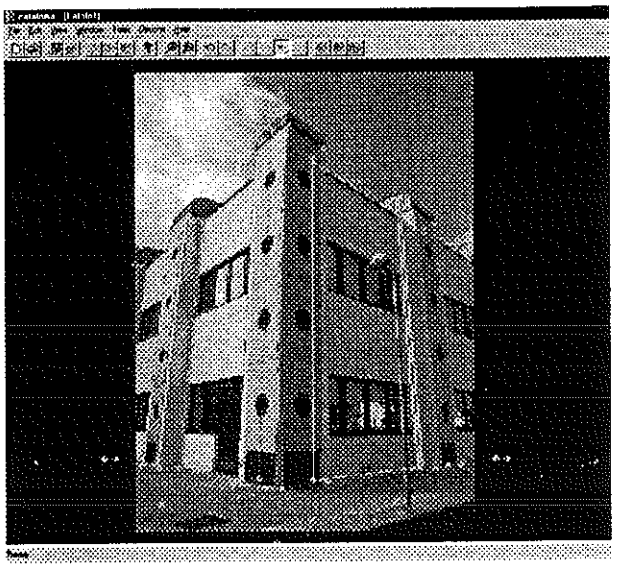

Figure 5: A snapshot of the application.

the method efficiency[2].

All the geometric computation are easily done in interactive time on a PC[3].

Texture synthesis is only slightly more time consuming: only 1.65 seconds where necessary to resample, using antialiasing, the full resolution texture[4] shown in Figure 6.

More importantly, it has been possible to test that the prototypal application requires just about three minutes of user time to reconstruct a simple geometry like the one shown in Figure 6.

The method proposed, based on flat rectangles as basic building blocks, identified on predefined planes, with the attracting and gluing mechanisms, has advantages and drawbacks:

- Rectangles are very versatile for a great variety of buildings. Unfortunately, they do not fit for curve surface, but none of the proposed block-based approaches is able to manage this type of architectural elements.
- Often many parts of a building, even

[2]The part about cutting quads described in Section 7 is still under implementation.

[3]We used a 350 MHz Pentium II, with 64 Mb of RAM

[4]sized 1024 × 1024, but with only about 600K resampled texel, the rest being wasted for packing and for making both texture coordinate a integer power of 2

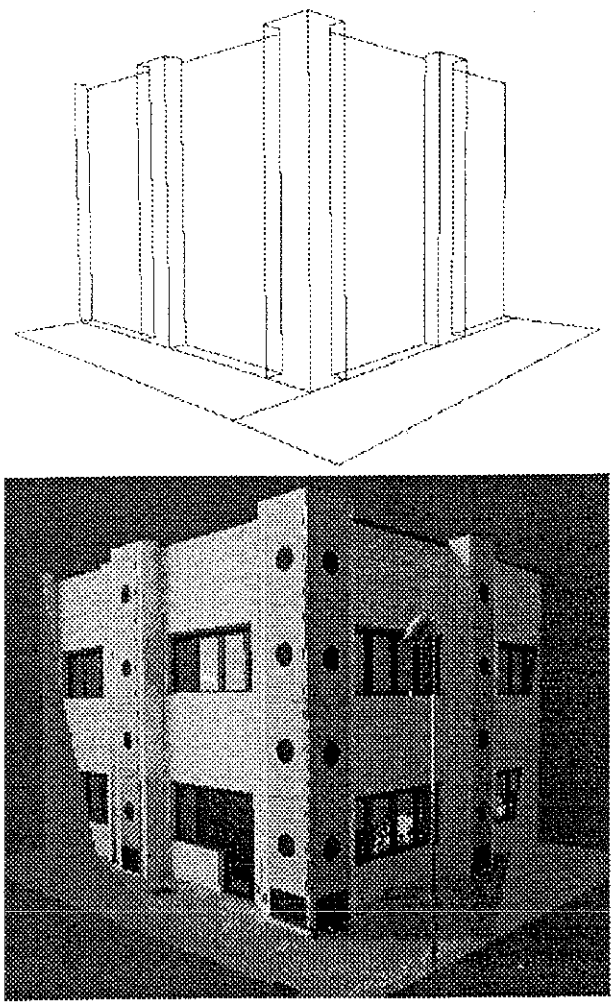

Figure 6: Above, the reconstructed geometry (21 quads). Below, a rectified, resampled and packed 1024x1024 texture is added.

when far apart, are oriented in the same direction, and therefore it pays to define once for all those directions (i.e. p's), to simplify the definition of r (which then requires only two opposite vertexes).

- The gluing mechanism not only aid the user, and allows its inputs to be integrated, but also improves precision: in fact, a position of an element, e.g. the side of an r, can be determined (even when totally occluded) by looking at the positions that other elements assume as a consequence of a possibly long chain of cascade effects.

10 Future work

The presented approach could be enriched by a number of features.

A possible extension in case of multiple photos is as follows:

1. Apply the method to the first photo, obtaining a mesh m_1.
2. Reproject the current mesh into the next photo obtaining an automatic initial set of input data for it.
3. Complete, using the user interface, the set of data with information identifiable from the current photo, and obtain a more complete mesh m_2.
4. Repeat from second step, until all photos are processed.

The second step requires an alignment problem to be solved (if the camera position is to be left unknown); the third step requires some technique to merge the textures when they overlap (for example, with view-dependent textures as in [5], which unfortunately would require an ad-hoc renderer).

In order to increase the geometrical detail, a good technique is to synthesize bump-maps rather then increasing the number of faces used in the mesh. As in [5], displacement maps could be computed with *model based stereo*, where the second photograph is compared, rather than directly to the first photograph (which is shot from a different point of view), to an image obtained by rendering the textured model obtained by first photograph, so that the viewpoint of the two images is the same.

As an alternative, one could obtain a normal-map, using multiple photos shot form the *same* viewpoint at different time of the day. For each pixel, comparing at least three different shading corresponding at different sun positions, it is possible to compute the normal and the base color, similarly to [2].

Finally, in the current implementation, sometimes, zones of the reconstructed texture contain an occluding object (like other parts of the same building). Those parts should be identified and filled otherwise.

References

[1] P. Beardsley, P. Torr, and A. Zisserman. 3D model acquisition from extended image sequences. *Lecture Notes in Computer Science*, 1065:683, 1996.

[2] Fausto Bernardini, Jack Wasserman, Joshua Mittleman, Holly Rushmeier, and Gabriel Taubin. Studying sculpture with a digital model: understanding Michelangelo's Pietá of the cathedral. In *Conference abstracts and applications: SIGGRAPH 98, July 14-21, 1998, Orlando, FL*, Computer Graphics, pages 281–281, New York, NY 10036, USA, 1998. ACM Press.

[3] A. Criminisi, I. Reid, and A. Zisserman. Single view metrology. In *Proc. 7th International Conference on Computer Vision, Kerkyra, Greece*, pages 434–442, September 1999.

[4] A. Criminisi, A. Zisserman, L. Van Gool, Bramble S., and D. Compton. A new approach to obtain height measurements from video. In *Proc. of SPIE, Boston, Massachussets, USA*, volume 3576, 1-6 November 1998.

[5] P.E. Debevec, C.J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 11–20. ACM SIGGRAPH, Addison Wesley, August 1996.

[6] M. Gangnet, D. Perny, and P. Coueignoux. Perspective mapping of planar textures. In D. S. Greenaway and E. A. Warman, editors, *Eurographics '82*, pages 57–70. North-Holland, 1982.

[7] David Liebowitz, Antonio Criminisi, and Andrew Zisserman. Creating architectural models from images. In Hans-Peter Seidel and Sabine Coquillart, editors, *Eurographics '99*, volume 18, pages C39–C50, Computer Graphics Forum, 1999. Eurographics Association, Eurographics Association and Blackwell Publishers Ltd 1999.

[8] Eric N. Mortensen and William A. Barrett. Intelligent scissors for image composition. *Computer Graphics*, 29(Annual Conference Series):191–198, November 1995.

[9] C. Tomasi and T. Kanade. The factorization method for the recovery of shape and motion from image streams. In *Image Understanding Workshop*, pages 459–472. Defense Advanced Research Projects Agency, Software and Intelligent Systems Office, January 1992.

[10] Ling-Ling Wang and Wen-Hsiang Tsai. Camera calibration by vanishing lines for 3-D computer vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-13(4):370–376, April 1991.