Interactively Modeling with Photogrammetry

Pierre Poulin Mathieu Ouimet Marie-Claude Frasson
Département d'informatique et de recherche opérationnelle
Université de Montréal

Abstract. We describe an *interactive* system to reconstruct 3D geometry and extract textures from a set of photographs taken with arbitrary camera parameters. The basic idea is to let the user draw 2D geometry on the images and set constraints using these drawings. Because the input comes directly from the user, he can more easily resolve most of the ambiguities and difficulties traditional computer vision algorithms must deal with.
A set of geometrical linear constraints formulated as a weighted least-squares problem is efficiently solved for the camera parameters, and then for the 3D geometry. Iterations between these two steps lead to improvements on both results. Once a satisfying 3D model is reconstructed, its color textures are extracted by sampling the projected texels in the corresponding images. All the textures associated with a polygon are then fitted to one another, and the corresponding colors are combined according to a set of criteria in order to form a unique texture.
The system produces 3D models and environments more suitable for realistic image synthesis and computer augmented reality.

1 Introduction

Realism in computer graphics has greatly evolved over the past decade. However very few synthetic images simulating real environments can fool an observer. A major difficulty lies with the 3D models; creating realistic models is an expensive and tedious process. Unfortunately the growing need for this level of accuracy is essential for realistic image synthesis, movie special effects, and computer augmented reality.

One attractive direction is to extract these models from real photographs. Although two decades of computer vision research has led to important fundamental results, a fully automated and reliable reconstruction algorithm in general situations has not yet been presented, at least for 3D models satisfying computer graphics general requirements. Misinformation in computer vision algorithms resulting from false correspondences, missed edge detections, noise, etc. can create severe difficulties in the extracted 3D models. We base our premise on the fact that the user *knows* what he wants to model, and within which accuracy. He can decide what must be modeled by geometry, and what could be simulated by a simpler geometry with a texture applied on it. To provide this functionality, we developed a fully interactive reconstruction system.

Getting an accurate 3D model requires the solution of several problems, which are all interrelated. We must first compute correct camera parameters, and then use the cameras and constraints to reconstruct the 3D geometry. After discussing some related work, we outline our geometry reconstruction system. A set of correspondences and incidences result in simple and efficient linear constraints. Although these constraints are not new, the improvements obtained in accuracy and speed demonstrate the importance of considering all of them together. User intervention at every step of this process, results in more satisfying general reconstructed 3D models.

Simple 3D geometry will be effective only with good quality textures. We focus

in Section 3 on a more complete, view-independent, treatment of textures. Textures are extracted for each 3D geometry from all images it projects to. The *best* texels (2D texture elements) are then combined into a single texture according to various criteria including visibility, projected areas, color differences, and image quality.

By solving accurately each problem, we will better understand the robustness, stability, and precision of our techniques. It should become easier later on to extend our interactions with more automatic computer vision and image processing techniques in order to alleviate some of the more cumbersome and tedious tasks, while keeping user intervention where required. The results of our system should help us create more precise textured synthetic models from real 3D objects in less time than current 3D modelers, and more robustly than fully automated geometry extraction algorithms.

2 Extracting 3D Geometry

Twenty years of active research on 3D reconstruction from 2D images in computer vision and robotics have left a considerable legacy of important results [12, 6, 5].

The first problem to address concerns camera calibration, i.e. computing camera parameters. This is a difficult and unstable process often improved by the use of specific targets. By putting in correspondence points or lines between images, it becomes possible to calibrate cameras. Similarly with known camera parameters, one can reconstruct a 3D scene up to a scale factor [4]. In these classical approaches, segmentations and correspondences are automatically determined. One typical example of the results obtained by these approaches was recently presented by Sato *et al.* [19].

A few recent projects such as REALISE [3, 10], *Façade* [1], *PhotoModeler* [16], and AIDA [21] propose to integrate more user intervention into the reconstruction process. They are derived from projective geometry [13], and are applied to the reconstruction of man-made scenes from a set of photographs and correspondences.

REALISE [3, 10] integrates user intervention early in the correspondence process, letting the user specify the first correspondences, and then returning to a more classical approach to identify automatically most of the other correspondences. The more stable initial solution greatly helps to reduce the errors of subsequent iterations. Nevertheless the same errors of fully automatic systems can still occur, and the user must then detect and correct the origin of the errors, which is not a simple task as the number of automatic correspondences increases.

Façade [1] develops a series of parameterized block primitives. Each block encodes efficiently and hierarchically several constraints frequently present in architectural design. The user must first place the blocks with a 3D modeler, and then set correspondences between the images and these blocks. Non-linear optimization of an objective function is then used to solve for all these constraints. The system has proven to be quite efficient and provides precise 3D models with little effort. However it requires the user to build with the blocks the model he wants to reconstruct. We believe this might be more difficult when general 3D models cannot be as easily created with these blocks.

PhotoModeler [16] is a commercial software for performing photogrammetric measurements on models built from photographs. Once the camera is calibrated, the user has to indicate features and correspondences on the images, and the system computes the 3D scene. The models obtained appear quite good, although it seems to be a lengthy process (they reported a week for a model of 200 3D points) which uses images of very high resolution (around 15 MB each). We also noticed many long thin triangles and gaps in some of their models. The system can apply textures coming from the photographs but does not seem to perform any particular treatment since the shadows,

highlights, etc. are still present. No details are provided on the algorithms used.

The AIDA system [21] is a fully automatic reconstruction system that combines surface reconstruction techniques with object recognition for the generation of 3D models for computer graphics applications. The system possesses a knowledge database of constraints, and selects the constraints to apply to the surface under reconstruction after performing a scene interpretation phase. We believe it might be safer, less cumbersome, and more general to let the user choose which constraints he wants to apply to its 3D primitives rather than letting the system pick some constraints from a knowledge database created specifically for the type of scene to reconstruct.

For these reasons, we introduce a system essentially based on user interaction. The user is responsible for (almost) *everything*, but also has the control on (almost) *everything*. This should provide a comprehensive tool to improve on the modeling from real 3D objects and on the computer graphics quality of these 3D models, while offering the opportunity to focus on the details important to the designer.

2.1 Our Reconstruction System

System Overview. We have developed an *interactive* reconstruction system from images. The images define the canvas on which all interaction is based. They can come from any type of cameras (even a virtual synthetic camera) with any settings and position. The user *draws* points, lines, and polygons on the images which form our basic 2D primitives. The user interactively specifies correspondences between the 2D primitives on different images. He can also assign other constraints between reconstructed 3D primitives simply by clicking on one of their respective 2D primitives. These additional constraints include parallelism, perpendicularity, planarity, and co-planarity.

At any time, the user can ask the system to reconstruct all *computable* cameras and 3D primitives. The reconstructed 3D primitives can be reprojected on the images to estimate the quality of each recovered camera and the 3D primitives. The user then has the choice to iterate a few times to improve on the mathematical solution, or to add new 2D primitives, correspondences, and constraints to refine the 3D models. This process, illustrated in Fig. 1, demonstrates the flexibility and power of our technique. The 3D model is reconstructed incrementally, refined where and when necessary. Each error from the user can also be immediately detected using reprojection. Contrarily to Debevec *et al.* [1], the user does not create a synthetic model of the geometry he wants to recover, although the reconstructed 3D model can as easily be used to establish new constraints between 2D and reconstructed 3D primitives.

Each image thus contains a set of 2D primitives drawn on it, and a camera computed when the set of resolved constraints is sufficient. To bootstrap the reconstruction process, the user assigns a sufficient number of 3D coordinates to 3D primitives via one of their corresponding 2D primitives. For instance, six 3D points in one image allow the computation of the corresponding camera. Once two cameras are computed, all 3D geometry that can be computed by resolving the constraints is reconstructed. With the assigned and the newly computed 3D values, the constraints are resolved again to improve the reconstructed cameras. This process iterates until no more constraints can be resolved, and the 3D geometry and cameras are computed to a satisfactory precision. Typically, a convergence iteration solving the equation systems for computing all the cameras and 3D positions takes between 0.05 and 2 seconds,[1] depending on the complexity of the scene (50 to 200 3D points) and the constraints used.

[1] on a 195 MHz R10000 SGI Impact with unoptimized code

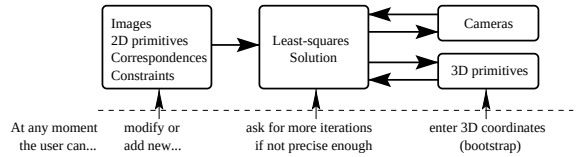

Fig. 1. Geometry reconstruction process

All our constraints are expressed as linear equations, typically forming an over-determined set of equations. A least-squares solution to this system is computed by singular value decomposition [17]. We use this solution for the unknown camera parameters, and the unknown 3D coordinates of points and lines. As an *exact* correspondence is hardly achievable by drawing 2D points on the image plane, we compute the *best* camera parameters in the least-squares sense. Hartley [7] demonstrate simple conditions under which linear systems of equations used to determine the camera parameters are as precise as their non-linear counterparts. Moreover this technique is simple to implement, efficient, general, always provides a solution, and we observed that it is more robust than the non-linear systems.

Geometrical Considerations. We express the geometrical constraints in our system as a set of incidences onto 3D planes. We model the perspective projection associated with each image with a 4×4 transformation matrix $\mathbf{T}$ [18]. This matrix is sufficient for our purposes because we do not intend to extract real camera parameters. A 3D point P expressed in homogeneous coordinates is transformed by $\mathbf{T}$ into a normalized homogeneous 2D point p on the image plane as

$$\begin{bmatrix} p_u \\ p_v \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} T_0 & T_1 & T_2 & T_3 \\ T_4 & T_5 & T_6 & T_7 \\ 0 & 0 & 0 & 0 \\ T_8 & T_9 & T_{10} & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix}.$$

In projective geometry, a 2D point p on an image defines a line in 3D. Any point P on this 3D line projects onto p. A 3D line can be defined as the intersection of two 3D planes. We define two orthogonal planes A_h and A_v such that they respectively project as horizontal and vertical 2D lines l_h and l_v intersecting at point p on the image plane. This is illustrated in Fig. 2 (left). We compute the plane A that contains the 2D line (a, b, c) and its 3D corresponding line as

$$A = \begin{bmatrix} a & b & \star & c \end{bmatrix} \mathbf{T} = \begin{bmatrix} aT_0 + bT_4 + cT_8 \\ aT_1 + bT_5 + cT_9 \\ aT_2 + bT_6 + cT_{10} \\ aT_3 + bT_7 + c \end{bmatrix}^T.$$

A 3D point P lies on a plane $A = \begin{bmatrix} a & b & c & d \end{bmatrix}^T$ when $A \cdot P = 0$. For the plane A_h defined by the 2D line l_h as $v = p_v$ in Fig. 2 (left), the constraint such that the 3D point P lies on this plane is satisfied when

$$A_h \cdot P = \begin{bmatrix} 0 & 1 & \star & -p_v \end{bmatrix} \mathbf{T} \cdot P = 0$$

that can be rewritten as

$$P_x(T_4 - p_v T_8) + P_y(T_5 - p_v T_9) + P_z(T_6 - p_v T_{10}) = p_v - T_7. \quad (1)$$

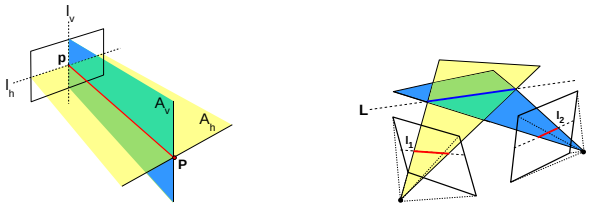

Fig. 2. Constraints for point position (left) and line correspondences (right)

Similarly the 3D point P lies on the vertical plane A_v defined by the 2D line $u = p_u$ if

$$P_x(T_0 - p_u T_8) + P_y(T_1 - p_u T_9) + P_z(T_2 - p_u T_{10}) = p_u - T_3. \quad (2)$$

Each 2D point with a known 3D position thus provides two equations. If we want to solve for the eleven unknowns T_i of our projection matrix, we need a minimum of six such points. Although less correspondences could resolve a non-linear system of equations [11], we consider finding six points in one image into which we want to reconstruct geometry not to be an overwhelming request. Once the cameras for two images are recovered in this way, it becomes possible to compute the position of a 3D point given its two projected 2D points in the images using Eq. (1) and (2).

In the case of 3D lines, any pair of 3D points P_1 and P_2 ($P_1 \neq P_2$) satisfying the condition $A \cdot P = 0$ defines a 3D line on A, the plane going through the 3D line and its projection l. We represent a 3D line L by using the *point-form* matrix $\mathbf{L}$, a 4×4 antisymmetric matrix containing the six *Plücker coordinates* [20] of the line. With this representation, $\mathbf{L}$ lies on A when

$$A\mathbf{L} = 0 \Rightarrow \begin{bmatrix} aT_0 + bT_4 + cT_8 \\ aT_1 + bT_5 + cT_9 \\ aT_2 + bT_6 + cT_{10} \\ aT_3 + bT_7 + c \end{bmatrix}^T \begin{bmatrix} 0 & P_{xy} & P_{xz} & P_x \\ -P_{xy} & 0 & P_{yz} & P_y \\ -P_{xz} & -P_{yz} & 0 & P_z \\ -P_x & -P_y & -P_z & 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

where for any two points P_1 and P_2, we have $P_{xy} = x_1 y_2 - x_2 y_1$, $P_x = x_1 - x_2$, etc.

By isolating the six Plücker coordinates, we obtain four constraints for each plane going through the 3D line. With more than one plane, we can determine the line in a least-squares sense as illustrated in Fig. 2 (right).

Additional 3D Constraints. Most man-made scenes exhibit some form of planarity, parallelism, perpendicularity, symmetry, etc. Using correspondences only, reconstructed geometry often does not respect these properties, which can lead to objectionable artifacts in the reconstructed 3D models. It is therefore very important to integrate this type of constraint in a reconstruction system. They are unfortunately difficult to detect automatically, as perspective projection does not preserve them in the image. The user can however very easily indicate each such constraint directly onto the images. They are integrated into our convergence process by adding equations to the system of linear equations used to compute 3D coordinates. Although they are not strictly enforced because they are simply part of a least-squares solution, they often result in more satisfying 3D models especially with respect to the needs of computer graphics models.

Coplanarity: A planar polygon with more than three vertices should have all its vertices on the same plane A. For each polygon with 3D vertices P_i, we add a *planarity*

constraint of the form $A \cdot P_i = 0$ that will be used during the computation of each P_i. Polygons, points, and lines can also be constrained to lie on the same supporting plane. To compute this plane $A = [\ a \quad b \quad c \quad d\ ]^T$, the user can specify a normal direction $N = [\ a \quad b \quad c\ ]^T$. We compute the *best* value for d by using the known 3D points. If there is no information about the plane orientation, we compute the *best* plane in the least-squares sense, that passes through at least three known 3D points of the coplanar primitives, and apply the same constraint.

Parallelism: Similarly, several polygons and lines can be parallel to each other, providing additional constraints. For each polygon, we get its orientation N_i from its plane equation, if available. These orientations allow us to calculate an average orientation N_{avg} that will be attributed to all the parallel polygons, even those for which no orientation could be first calculated. A *planarity constraint* is added to the computation of the polygons 3D points. For parallel lines, we compute their average direction.

Perpendicularity: If the normals N_{perp} to perpendicular polygons are known, we can add other constraints for the computation of N_{avg}. Two perpendicular polygons have perpendicular normals, thus for every polygon orthogonal to a set of parallel polygons, we have $N_{perp} \cdot N_{avg} = 0$.

Many other constraints should be exploited. *Symmetry* could constrain characteristics such as lengths or angles. *Similarity* between models could specify two identical elements at different positions. *Incidence* of points and lines can be extended to different primitives. These are only a few of the constraints we observe in 3D scenes.

Each basic constraint described above can be used as a building block for more elaborate primitives. A cube for instance becomes a set of planar faces, with perpendicularity and parallelism between its faces and segments, and constrained length between its 3D vertices. Rather than letting the user specify all these constraints, a new primitive for which all of these are already handled represents a much more efficient tool for the user. These new primitives can be described in a library of primitives organized hierarchically. Debevec *et al.* [1] shows how this representation can also reduce significantly the number of constraints to resolve.

We can also weight the contributions of the constraints depending on their importance in the current reconstruction. The default weights assigned to each type of constraint can be edited by the user. The resolution of our equation systems is simply extended to a weighted least-squares.

2.2 Results of Geometry Reconstruction

To evaluate the precision and the convergence of our iterative process, we constructed a simple synthetic scene made of seven boxes. Five images of resolution 500×400 where rendered from camera positions indicated by the grey cones in Fig. 3 (left). 2D polygons were manually drawn and put in correspondences within 60 minutes on a 195 MHz R10000 SGI Impact. The 3D coordinates of six points of the central cube on the floor were entered to bootstrap the system. The three curves in Fig. 3 (right) represent the distance in world coordinates between the real 3D position of three points in the scene ((-2,3,0),(0,2,-2),(1,2,-2)) and their reconstructed correspondents.

200 iterations without any constraints other than the point correspondences took about 5 minutes. We then applied successively the constraints of planarity, coplanarity, and parallelism between all the 3D polygons. Calculating all these constraints typically adds a few tenths of a second per iteration depending on the complexity of the 3D scene.

The three curves reach a plateau after a certain number of iterations. This does not mean that the 3D model is then perfectly reconstructed, but rather that the solution is

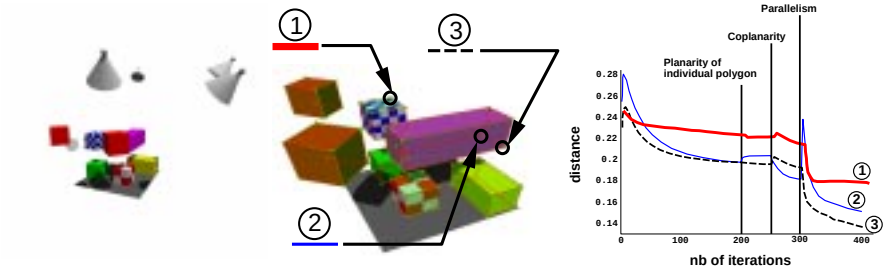

Fig. 3. Reconstruction of a synthetic scene

stable and should not change significantly with more iterations. When we introduce the planarity and then the coplanarity constraints for individual polygons, the points move slightly. In this scene where parallelism is preponderant, the addition of this last constraint improves significantly the reconstruction for all three points, which is shown by the drop of all three curves after iteration 300. The introduction of a new constraint can sometimes perturb the whole system, affecting more the less-constrained elements as demonstrated by the sudden spike in curve 2. In most observed cases, the system quickly returns to an improved and more stable state.

In Fig. 3 (center), we reproject in wireframe mode the reconstructed model using the computed camera from one of the original images. Distances between the 2D drawn points and the reprojected reconstructed points all lie within less than one pixel from each other. When 3D constraints are used to improve the model, this distance can reach up to two pixels. The 3D scene then corresponds more to reality but does not fit exactly the drawn primitives when reprojected with the computed projection matrix. The constraints thus compensate for the inaccuracy introduced by the user interaction or by the primitives far from the entered coordinates. Because the user draws 2D primitives at the resolution of the image, getting a maximum of two pixels is considered satisfactory. Sub-pixels accuracy is obtained if these primitives are drawn at sub-pixel precision, but this lengthen the user interaction time.

3 Extracting Texture

Textures have been introduced in computer graphics to increase the realism of synthetic surfaces. They encode via a surface parameterization the color for each point on the surface. While the contribution of textures to realism is obvious, it is not always easy to extract a texture from real images. One must correct for perspective foreshortening, surface curvature, hidden portions of the texture, reflections, shading, etc. All these limitations have restricted the type of extracted real textures. However our reconstructed geometry and cameras provide a great context within which we can extract these textures.

Most current approaches are based on view-dependent textures. Havaldar *et al.* [8] use the projection of the 3D primitive in all the images to determine the *best* source image for the texture. Then we apply to the texture the 2D transformation from the projected polygon in this best image, to the projected polygon in the image from a new viewpoint. Unfortunately, this 2D deformation of the texture is invalid for a perspective projection, and prone to visibility errors.

Debevec *et al.* [1] reprojects each extracted texture for a given primitive as a weighted

function based on the viewing angle of the new camera position. The technique provides better results with view-dependent information. However, neglecting the distance factor in the weights can introduce important errors, and aliasing can appear from the use of occlusion maps. Moreover all the textures must be kept in memory as potentially all of them might be reprojected for any new viewpoint.

Niem and Broszio [14] identifies the best image for an entire polygon (according to angle and distance criteria), and samples the texture from this image. Because adjacent polygons can have different best images, they then proceed to smooth out the adjacent texels, possibly altering the textures.

In our system, a texture is extracted upon user request for a given primitive projecting in a number of images. The texture is the result of recombining the estimated *best* colors for each point of the surface projected in each image. Even though extracting a single texture is prone to errors as will be discussed later, it is more suitable for general image synthesis applications such as applying it to different primitives, filtering, and use of graphics hardware.

3.1 Sampling Colors from Images

Assume a one-to-one mapping between every point on a primitive and its surface parameterization. We reconstruct the texture as a regular grid of 2D texels from a sampling of this parameterization.

Each texture point (s, t) on the surface maps to a 3D point. For each image with a computed projection matrix, we can easily find the 2D point corresponding to the projection of the 3D point. Therefore each 3D point on the surface goes through the extraction pipeline of Fig. 4 to determine its color as viewed from each computed camera. These colors are then recombined according to a set of criteria to determine the final color associated with this point.

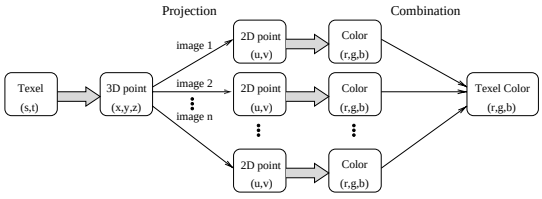

Fig. 4. Texture-to-color extraction pipeline

3D primitives and cameras are reconstructed as least-squares solutions from drawn 2D primitives. The precision is considered acceptable for 3D models, as demonstrated by the results of the previous section. However the small variations from one image to the other, or assuming a surface is perfectly planar while it is not can slightly affect the alignment of the corresponding extracted texture in each image. Matching directly each texel in the set of extracted textures thus tends to produce aliasing artifacts, or blurry textures if a wider filter is used. User intervention again allows the correction of these errors. For each extracted texture, the user selects (draw) a few (three to six) 2D feature points in all the corresponding extracted textures. They create a set of 2D-to-2D correspondences. A linear transformation of the form

$$\begin{bmatrix} M_0 & M_1 & M_2 \\ M_3 & M_4 & M_5 \\ 0 & 0 & 1 \end{bmatrix}$$

is solved again using a least-squares method in order to find a transformation to better fit two textures together.

We reconstruct the color of a 3D point by applying a filter kernel on the neighboring pixel colors at the 2D point it projects to. We can also improve on the texture definition by supersampling each texel, i.e. by using more samples to represent the texel color. Each such sample is weighted by the filter kernel on the texel. As the texture space can be very different than the 3D surface space, we adaptively supersample the texels for which adjacent samples project to more distant (> 2) pixels. This adaptive sampling scheme can also be used to determine the finest texture resolution required according to the projection of the 3D primitive in all the images.

3.2 Occlusion

For a given 3D point, its projection in one image will most likely be a visible point because the user drew its supporting 3D primitive as a corresponding 2D primitive on the image. However some portion of the 3D primitive might be blocked by another 3D primitive closer to the image plane, thus leading to an incoherent color.

A simple test determines the zone with an occlusion risk by intersecting the 2D primitive with all 2D primitives on this image. If there is intersection, we must determine the 3D intersection between the corresponding 2D point on the image (two planes) and the potentially occluding 3D polygon (a third plane). If there is intersection and the depth is smaller than the one of the 3D point, we simply mark this occluded color sample as invalid.

3.3 Weighting the Contributions for the Final Color

We extract a color for each texel in each image. The final color for this texel must be computed from these colors.

The size in pixels of the texel projected in the image is a good indication of the quality of the color extracted for this texel. The larger the projected area of a texel, the more precise the texture should be. Therefore, for all the valid colors of a given texel, we weight its color contribution as a relative function of the projected areas of the texel in all selected images.

Ofek *et al.* [15] stores each pixel in each image into a *mipmap* [22] pyramidal structure for the texture. Color information is propagated up and down the pyramid, with some indication of *certainty* according to color variations. The structure fits each texel to the image pixel resolution, thus adapting its processing accordingly. However unless very high resolution textures are required, we believe the extra cost of propagating the information in the pyramid and the inevitable loss of information due to the filtering between levels might not be worth the savings.

Our solution is simple, but might require sampling many texels projecting within a small fraction of a pixel area. However all information is kept at the user-specified texel resolution, and as such, we get much flexibility in ways of interpreting and filtering the information. It is also fairly simple to integrate various new criteria to improve on the process of combining the extracted texels.

3.4 Color Differences

Unfortunately we must be aware that several situations might invalidate any such texture extraction algorithm. Any view-dependent feature that changes the aspect (color) of a 3D point as the camera moves might be a source of errors. These include specular

reflections (highlights), mirrors, transparencies, refractions, ignored surface deformations (sharp grooves and peaks), participating media, etc. Without user intervention, a combination of these *artifacts* can hardly be handled automatically.

When one color at one texel is very different than the others for different images, we simply reject its contribution, assuming it was caused by view-dependent features or noise in the image. When all the colors are very different from each other, we simply mark this texel as invalid. We will discuss in the conclusion how these differences could be used to extract such view-dependent information.

The color of a pixel at silhouettes and edges of polygons includes not only the texel color, but also background and other geometry colors. Camera registration errors also introduce slight misalignments between the reconstructed 3D primitive and each 2D primitive it should project to. We again simply mark such a texel as invalid.

We therefore end out with an extracted texture with some undefined texel colors. Fortunately, these typically represent a small portion of the entire texture. We currently fill in these texels by applying a simple filter, although some *filling* algorithms have proven to be quite efficient [9]. They should be even more effective for gaps as narrow as those observed so far in our tests.

3.5 Results of Extracted Textures

The quality of extracted textures is really important for computer graphics applications. To demonstrate the quality of our sampling approach, we reconstructed a simple scene from four photographs of a real scene displayed in Fig. 5 (left). Each photo is digitized at a 640×480 resolution. 2D polygons were drawn by the user for five faces of the Rubik cube, the cover of the magazine, and the top and one face of the book. The 3D positions of 8 corners of the cube were entered to *bootstrap* the projection reconstruction, and the cover and book 3D geometry have been reconstructed from point correspondences and polygons planarity. The entire process took less than 10 minutes.

The four corresponding extracted textures are displayed in Fig. 5 (center). Each texture was sampled at a higher 320×400 resolution, one sample per texel, without filtering or supersampling. The black cut to the right of the top right texture is due to the book occluding this part of the texture in the corresponding photograph.

Fig. 5. Four images and the extracted textures

A zoom on the **GR** in Fig. 5 (top right) shows the ghosting of the default reconstruction and the improvements after texture alignments by the user using four feature points. Although one texture was partly occluded, its valid texels were used to reconstruct the combined final texture. The final recombined texture with samples from all

extracted textures (except for occluded parts) is displayed as the larger front cover in Fig. 5. One can observe the quality of the reconstruction on the title of the magazine.

Color plates of two reconstructed scenes with geometry and textures appear in the Color Section. All photos have a 640×480 resolution. The tower took about 6 hours to reconstruct from a set of ten photos. 400 convergence iterations led to the model which contains over a hundred 3D polygons. The desk scene was made from a set of six photos in about 3 hours. It was greatly improved by the use of constraints which corrected the slanted surfaces due to the distance from the small reference primitive (entered coordinates of the Rubik cube corners). The space constraints and image reproduction limitations do not demonstrate well the quality and problems of reconstruction algorithms. We invite the reader to visit the site associated with this paper from `http://www.iro.umontreal.ca/labs/infographie/papers` to better appreciate additional images, 3D models, and textures.

4 Conclusion

We have presented a simple algorithm to register cameras and reconstruct 3D geometry from a set of 2D primitives drawn directly by a user on 2D images. Various correspondences and linear constraints are introduced to improve on the 3D model accuracy. Our system permits user interaction at every stage of the iterative process. Because the user specifies all geometry, correspondences, and constraints, the reconstructed 3D models are better adapted to general graphics requirements.

We presented an algorithm to extract the textures for a polygon and to combine their colors into a unified texture. When the color differences are too large, these colors are simply rejected. Missing information can be replaced by a filling algorithm.

While the results are very encouraging and already satisfying for many scenes, we have merely scratched the surface of all the possibilities. Our interactive system is rather basic, and it would greatly benefit from a better interface, standard 2D interactive editing tools, and efficient vision tools. These improvements combined with a tool to detect potential correspondence errors would dramatically reduce modeling time.

We are currently investigating the extraction of surface reflection properties from the radiance [2] differences at each texel. As the number of images necessary to extract a precise reflection model can be fairly large, we must extend our user-driven scheme to model-tracking in video sequences. With known geometry and emission of light sources, we can easily determine the regions in shadow from one light source. This direct illumination is very important to extract reflection properties. Once a first estimate of this information is computed, we will again use an iterative process to try to extend this solution to interreflections in the scene.

Our extracted geometrical models are fairly simple and mostly polygonal. We are also investigating a more automatic reconstruction of complex geometry in a form of displacement textures applied on simpler enclosing primitives.

We finally expect to develop bounds on the precision of the extracted 3D points, surface normals, colors, reflection coefficients, illumination, etc. These bounds would then provide a measure of the accuracy of the extracted information.

Acknowledgments. Pat Hanrahan provided very important ideas at the early stages of this project while the first author was a postdoc at Princeton. Craig Kolb also helped with the first implementation. We acknowledge financial support from NSERC. The third author also acknowledges a CFUW fellowship.

References

1. 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. August 1996.
2. P.E. Debevec and J. Malik. Recovering high dynamic range radiance maps from photographs. In *SIGGRAPH 97 Conference Proceedings*, pages 369–378. August 1997.
3. O. Faugeras, S. Laveau, L. Robert, G. Csurka, and C. Zeller. 3D reconstruction of urban scenes from sequences of images. Tech. Report 2572, INRIA Sophia-Antipolis, May 1995.
4. O.D. Faugeras. What can be seen in three dimensions with an uncalibrated stereo rig? In *Proceedings of Computer Vision (ECCV '92)*, pages 563–578, May 1992.
5. O. Faugeras. *Three-dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, 1993.
6. W.E.L. Grimson. *From Images to Surfaces: A Computational Study of the Human Early Vision System*. MIT Press, 1981.
7. R.I. Hartley. In defense of the eight-point algorithm. *IEEE Trans. on Pattern Analysis Machine Intelligence*, 19(6):580–593, June 1997.
8. P. Havaldar, M.-S. Lee, and G. Medioni. View synthesis from unregistered 2-D images. In *Proceedings of Graphics Interface '96*, pages 61–69, May 1996.
9. D.J. Heeger and J.R. Bergen. Pyramid-Based texture analysis/synthesis. In *SIGGRAPH 95 Conference Proceedings*, pages 229–238. August 1995.
10. F. Leymarie *et al.* Realise: Reconstruction of reality from image sequences. In *International Conference on Image Processing*, pages 651–654. September 1996. http://www.inria.fr/robotvis/projects/Realise
11. Y. Liu, T.S. Huang, and O. Faugeras. Determination of camera location from 2-D to 3-D line and point correspondences. *IEEE Trans. on Pattern Analysis Machine Intelligence*, 12(1):28–37, January 1990.
12. H.C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, 1981.
13. J. Mundy and A. Zisserman, editors. *Geometric Invariance in Computer Vision*. MIT Press, Cambridge, Massachusetts, 1992.
14. W. Niem and H. Broszio. Mapping texture from multiple camera views onto 3D-object models for computer animation. In *Proceedings of the International Workshop on Stereoscopic and Three Dimensional Imaging*, September 1995.
15. E. Ofek, E. Shilat, A. Rappoport, and M. Werman. Multiresolution textures from image sequences. *IEEE Computer Graphics and Applications*, 17(2):18–29, March 1997.
16. http://www.photomodeler.com
17. W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing (2nd ed.)*. Cambridge University Press, Cambridge, 1992.
18. D.F. Rogers and J.A. Adams. *Mathematical Elements for Computer Graphics*. McGraw-Hill, 1976.
19. Y. Sato, M.D. Wheeler, and K. Ikeuchi. Object shape and reflectance modeling from observation. In *SIGGRAPH 97 Conference Proceedings*, pages 379–387. August 1997.
20. J. Semple and G. Kneebone. *Algebraic projective geometry*. Oxford University Press, 1952.
21. S. Weik and O. Grau. Recovering 3-D object geometry using a generic constraint description. In *ISPRS96 - 18th Congress of the International Society for Photogrammetry and Remote Sensing*, July 1996.
22. L. Williams. Pyramidal parametrics. In *Computer Graphics (SIGGRAPH '83 Proceedings)*, pages 1–11, July 1983.