The Visual Computer

Resolving occlusion in image sequence made easy

Kiem Ching Ong, Hung Chuan Teh, Tiow Seng Tan

Department of Information,
Systems and Computer Science,
National University of Singapore,
Lower Kent Ridge Road, Singapore 119260
E-mail: ong-kc@hotmail.com,
{tehhc|tants}@comp.nus.edu.sg

While the task of seamlessly merging computer-generated 3D objects into an image sequence can be done manually, such effort often lacks consistency across the images. It is also time consuming and prone to error. This paper proposes a framework that solves the occlusion problem without assuming a priori computer models from the input scene. It includes a new algorithm to derive approximate 3D models about the real scene based on recovered geometry information and user-supplied segmentation results. The framework has been implemented, and it works for amateur home videos. The result is an easy-to-use system for applications like the visualization of new architectures in a real environment.

Key words: Augmented reality – Occlusion – Structure-from-motion – Segmentation

Correspondence to: H.C. Teh or T.S. Tan

1 Introduction

The deployment of computer graphics to insert virtual objects into real scene has many applications in medicine, manufacturing, visualization and entertainment (Azuma 1997). To enhance the illusion that the computer objects are actually present in the real scene, three problems need to be solved:

1. Common viewing parameters must be found when the virtual camera used to render the inserted computer objects is to correspond as closely as possible to the actual camera.
2. The problem of occlusion must be solved when the computer objects are to be properly occluded by objects in the real scene in the event that they are placed behind them.
3. Illumination must be attended to when the computer objects are to be subject to the same lighting conditions as the real objects and must have appropriate shadows cast to and from the real objects.

This paper focuses on achieving correct occlusion and does not deal with the illumination issue. The problem of occlusion occurs when the inserted computer objects are placed behind real objects in the scene. Usually, the region occupied by the real objects is masked so that no computer objects can be drawn within. This mask essentially captures the silhouettes of the blocking real objects, and the mask edges dictate how seamless the merging will be. The masking effect can be achieved in two dimensions by designating a 2D pixel region in the real image manually (Nakamae et al. 1986; Ertl et al. 1991) or semiautomatically (Berger 1997). This method is simple but tedious because each frame requires its own mask. Furthermore, flexibility suffers as the 2D mask will in effect occlude all inserted computer objects regardless of whether they are in front or behind the real objects. The masking effect can also be achieved in three dimensions with known computer models or simply depth values. When a computer model of the blocking real object is registered in the image, i.e. its 2D projection matches the real object in the image, correct occlusion is achieved if the inserted computer object is placed behind the real object (Breen et al. 1996). This approach is general, but unfortunately, computer models of real scene are usually unavailable, and it is often time consuming to create one by hand. Due to this, some researchers use depth information derived from the real scene for occlusion resolution (Koch 1993; Wloka and Anderson 1995). Typically, ste-

The Visual Computer (1998) 14:153–165

reoscopic image pairs are used for feature correspondence to recover the depth value at each pixel. Since this approach has no notion of the geometrical aspect of the real objects, silhouettes of the real objects are not explicitly delineated. Due to errors in feature correspondence, the recovered depth values often cause an unsatisfactory merging effect, especially at the occluding boundary of the real objects.

In view of the flexibility of the 3D mask and the importance of accurate masking at the edges of real objects, we propose the following framework. Some real object silhouettes are first segmented by hand in selected frames called *key frames*. The result is then used to construct automatically approximate 3D computer models, which serve as 3D masks to the real objects. The 3D models are approximate because their geometry may not correspond exactly to that of the physical real objects they represent. However, they are constructed in such a way that, when projected onto the images, they cover exactly the silhouettes of their respective real objects. In general, we offer a solution to the following problem: given a monocular image sequence with neither a priori knowledge about the scene nor the availability of any real scene models, resolve the occlusion problem that occurs when computer objects are inserted into the scene. The framework is not intended for real-time applications; rather, an image sequence of the real scene is captured, digitized and then processed. Nonetheless, the whole framework is designed to be fully automated with little intervention from users, except for object-silhouette extraction, when the user outperforms the computer. Based on the framework, an architecture visualization application has been implemented. Such a system assists an architect to evaluate the impact of his or her design by merging a computer model of the architecture in a video of the location where it is to be built. The augmented video is most useful in multimedia presentations since it conveys the architectural design more realistically. Furthermore, the framework requires only a monocular image sequence that can easily be obtained with a video camera. Virtual objects are merged into real images interactively through a GUI.

In Sect. 2, we outline the proposed framework. Sections 3–7 zoom into the respective modules within the framework and examine the implementation issues. Results from our implemented system are shown in Sect. 8, and we conclude this paper in Sect. 9.

2 Proposed framework

Given an input image sequence, the proposed framework (Fig. 1) derives sparse 3D feature points from the scene using a structure-from-motion algorithm from computer vision. We use the user-segmented object silhouettes in the key frames to cluster the 3D points of the real objects to which they belong. We then use each of the clustered point sets to define and subsequently enlarge 3D bounding boxes of the real objects so that their 2D projections onto the key frames will completely cover their respective object silhouettes. These bounding boxes form the initial models, which we call *clones* to the real objects.

Since our purpose in building the 3D clones is to deal with occlusion, we want to construct them so that they project exactly to their respective object regions in the real images. Therefore, for each of the initial clones, their 2D object silhouettes in the key frames are back-projected to the 3D space to trim away unwanted 3D regions in the clone. This, however, results in overtrimming due to errors in the estimated virtual camera parameters. A set of 3D patch voxels that can dynamically be turned on or off is used to compensate for regions on clones that are to be retained in certain key frames, but have been trimmed in others. Geometrically, the resultant clone is a solid 3D object with a set of dynamic 3D patch voxels that are turned on or off, depending on which key frame is active. With the clones, we can then establish a common reference system between the real objects in the images and the inserted virtual objects. The subsequent merging process is simply depth comparison with which proper occlusion is realized.

This framework does not set out to recover detail 3D models of real objects. We exploit the fact that the clone is intended for occlusion and not for visual rendering. As a result, in terms of geometry, only approximate 3D clones are constructed. There are cases when exact geometry of the clone is desirable. This happens when there is physical contact between real and virtual objects; for example, in making a virtual vase sit on top of a real table. Our framework is more concerned with exhibiting correct visual occlusion for assessment purposes.

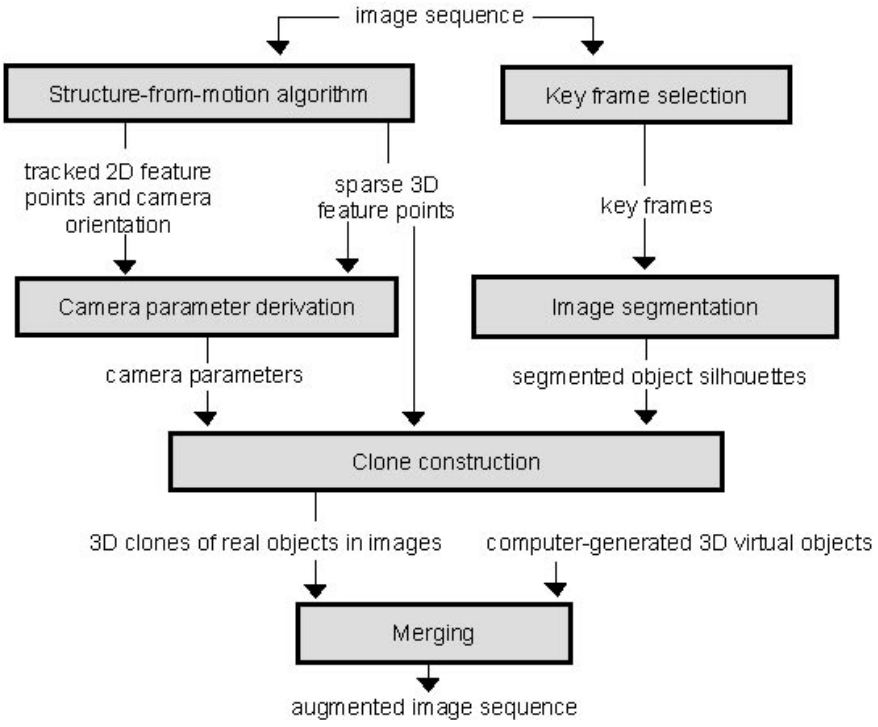


Fig. 1. Framework for resolving occlusion in image sequence

Also, since our approach constructs clones from the silhouettes, the registration problem associated with the model-based method is nonexistent.

3 Recovering structure from the image stream

A recent trend in solving structure-from-motion problem has focused on using a long image stream, which has a twofold advantage. Firstly, the feature correspondence problem between successive frames can more easily be solved due to a small change in the image content. Secondly, data redundancy can be exploited to counteract noise in images.

The structure-from-motion problem with perspective projection is inherently a nonlinear minimization problem. This class of problems is susceptible to local minima in the parameter space, and a good formulation is essential to better manage its complexity during minimization. Hu and Ahuja (1993) and Szeliski and Kang (1994) have used nonlinear optimization techniques to extract both 3D structure and camera motion from image streams. However, there are researchers who relax the requirement of perspective projection to its approximations like orthography, scaled orthography and paraperspective in order to convert the nonlinear problem into a linear one. In particular, the factorization method by Tomasi and Kanade (1992), which assumes orthographic projection, has shown good results. In fact, Yu et al. (1996) have managed to use the Taylor series expansion to generalize their factorization method to higher-order approximations to perspective projection. Apart from using point features to recover scene structure as in these papers, Taylor and Kriegman (1995) and Vieville and Faugeras (1990) have used straight-line features to solve the structure-from-motion problem. Quan and Kanade (1997) have recently extended the original factorization method to line features.

In our implementation, we employ the factorization method under orthography by Tomasi and Kanade. This method encodes an image stream

as a $2F\times P$ measurement matrix of the image coordinates (u, v) of P points tracked through F frames. Under orthography, the registered $2F\times P$ matrix (W) has rank 3 and can be decomposed into camera orientation (R) and the object shape (S).

$$\underset{2F\times P}{W} = \underset{2F\times 3}{R}\ \underset{3\times P}{S}$$

Their method is robust due to the use of singular value decomposition, which is a numerically stable factorization method. Furthermore, shape information is derived without the use of intermediate camera-centered depth values, which are known to be noise-sensitive for distant objects.

Even though the orthographic projection assumption is tolerable in our intended application (architecture visualization where the camera is far away from the real scene), it does limit the kind of camera motion allowed. For example, the camera should not move towards or away from the scene in order to minimize perspective distortion. Also, when the camera is purely translated across the scene with a constant line-of-sight vector, the factorization method will fail due to violation of the basic principle of Ullman's (1979) results, which say that at least three distinct views are required to determine structure and motion under orthography. This is because the images generated by this kind of camera motion effectively only give a 2D view of the real objects.

The output from Tomasi and Kanade's factorization algorithm is the matrices R and S. R contains the relative orientation of the camera in each frame, whereas S contains the recovered 3D feature point coordinates. The origin of the coordinate system is at the centroid of the recovered 3D points and its orientation is arbitrarily aligned to the first frame's camera orientation.

4 Camera parameter derivation

The recovered camera orientation alone is not sufficient to specify a virtual camera completely. We also require the camera position and zoom factor in each frame. In our implementation, the camera zoom factor has been assumed to be constant throughout the image sequence. As a result, there are only $3F+1$ unknown camera parameters to recover (F is the number of frames). To this end, we make use of the recovered 3D feature points, together with their corresponding tracked 2D feature points, to fit for the $3F+1$ unknowns using Levenberg-Marquardt minimization (Press et al. 1992). The error function to minimize is

$$\chi^2(\mathbf{a}) = \sum_{f=1}^{F}\sum_{p=1}^{P}\left\| \begin{pmatrix} u_{fp} \\ v_{fp} \end{pmatrix}_{\text{tracked}} - \begin{pmatrix} u_{fp} \\ v_{fp} \end{pmatrix}_{\text{computed}} \right\|$$

where:

F: number of frames
P: number of feature points
$\mathbf{a}$: vector of $3F+1$ camera parameters
$\begin{pmatrix} u_{fp} \\ v_{fp} \end{pmatrix}_{\text{tracked}}$: tracked 2D feature point p in frame f
$\begin{pmatrix} u_{fp} \\ v_{fp} \end{pmatrix}_{\text{computed}}$: computed 2D projection of 3D feature point p in frame f given the estimate of $\mathbf{a}$.

Experimentally, we find that when orthography is still used as the underlying projection model, the solution converges faster (less than ten iterations) and is more accurate. When perspective projection is used, the fitting process sometimes fails because the solution does not converge.

5 Image segmentation and key frame selection

To date, a robust and automatic image segmentation algorithm is still elusive. *Snakes* (Kass et al. 1988), *Intelligent Scissors* (Mortensen and Barrett 1995) and the ubiquitous magic wand available in most image processing packages, though speed up the segmentation task, still require human guidance to achieve correct segmentation. Boundary tracking across multiple frames for image segmentation has been explored by Ueda et al. (1992) using *Snakes* and by Mitsunaga et al. (1995) using alpha values. Nevertheless, these methods are not perfect at all times, and user correction is needed when the segmentation is unsatisfactory.

The proposed framework relies on correct object-silhouette segmentation in the key frames to guide the construction of 3D clones of real objects. The definition of the key frames depends largely on the geometrical changes to the real objects in the

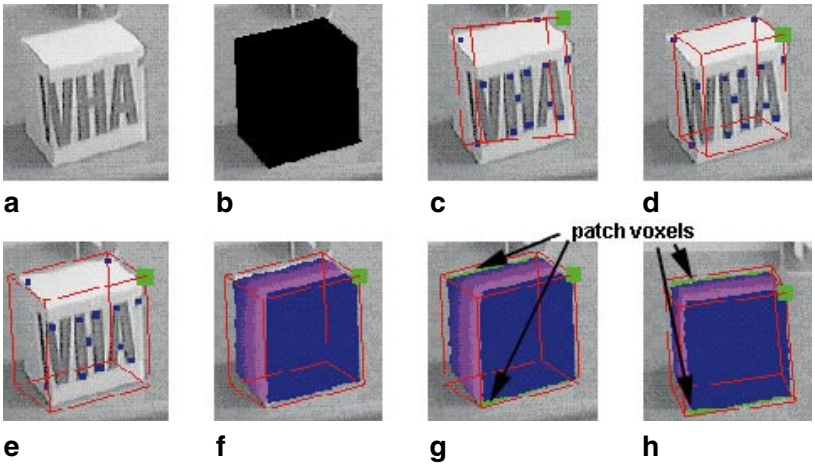


Fig. 2. Constructing a 3D clone for the white box (panel **a**)

images. A frame should be made a key frame when there is large geometrical change, such as the appearance or disappearance of facets on real objects. Generally, the more key frames the user defines, the more accurate the constructed clones will be.
In the implementation, we use *Intelligent Scissors* as a tool to help users define the object silhouette. We find that using *Intelligent Scissors* is intuitive, and its interactive snapping live-wire feature is highly desirable in practice. However, the live wire sometimes snaps to a nearby wrong edge that exhibits a stronger edge characteristic. To accommodate this, a simple point-and-click straight-line tool (which does not automatically snap to edges) has been incorporated into *Intelligent Scissors*, and we alternate between the two at our convenience. We have, by default, made the first and the last frames key frames, and users have to decide which of the other intermediate ones are to be key frames.

6 Clone construction

The most important function that a 3D clone of a real object serves is to occlude the inserted virtual objects. When a virtual object occludes a real object in the image, simply overlaying the virtual object on top of the image will do the trick. This is not so straightforward when a real object in the image is to occlude the inserted virtual object. We need a reference 3D clone of the real object in order to determine which part of the virtual object is to be occluded. Since the virtual objects should only be occluded at the boundary of real objects, the 3D clone should be constructed in such a way that its 2D projection onto the image plane covers exactly the real object's silhouette in every frame.
Due to the use of orthographic projection in the camera fitting process, we also assume orthography as the underlying projection model in clone construction. The whole process is accomplished in five steps.

Step 1. Initial model definition

Based on the user-supplied object silhouettes (Fig. 2b) in the key frames, we cluster the recovered 3D feature points from the factorization method (Sect. 3) to their respective real objects. For each such object, we then define a rough model using the bounding box of the 3D feature points. Figure 2c shows the world-coordinate-axis-aligned bounding box of the 3D feature points derived from the box.

Step 2. Model orientation determination

Due to the fact that our centroid-based coordinate system has an arbitrary orientation, the ground truth information is unknown. This means that we cannot assume that the ground is on the x–z plane, and thus our initial rough model may not have the correct orientation. Having a correct orientation, though not critical, speeds up subsequent processing. As a result, we provide an interactive mechanism for users to change the orientation of the initial model. Figure 2d shows that the user has interactively adjusted the orientation of the

bounding box so that it is closer to that of the white box.

Step 3. Model enlargement

Since a good clone is one that projects to its 2D object silhouette exactly, we need to ascertain that the initial model at least covers its object silhouettes in all the key frames, albeit overcover. We systematically enlarge the model with the following algorithm. Figure 2e shows that the enlarged model projects to completely cover the user-defined object silhouette.

The algorithm works as follows. Assume that the initial model is bounded by bottom-left-far point (x_1, y_1, z_1) and top-right-near point (x_2, y_2, z_2) where $x_1<x_2$, $y_1<y_2$, and $z_1<z_2$, the algorithm goes through all the key frames in turn to progressively enlarge the model.

For each key frame, the algorithm computes the 2D (axis parallel) bounding box of the object silhouette and locates two points (possibly one point in degenerate cases), each on the object silhouette, that are farthest apart on each boundary edge of the bounding box. For each of these eight (or less) points, the algorithm checks whether any enlargement is necessary, as in the next paragraph.

For each of the eight (or less) points, the algorithm casts a ray from the point on the image plane to the 3D space, using the computed camera parameters. If the ray intersects the current model, then no enlargement is necessary. Otherwise, the ray is outside the current model, indicating that enlargement is necessary because the model does not project to cover this point. In this case, the algorithm finds the corner, say C, of the current model closest to the ray. Let 3D point (x, y, z) be the point on the ray closest to C. Then, the algorithm enlarges the current model by setting: $x_1=\min(x_1, x)$; $y_1=\min(y_1, y)$; $z_1=\min(z_1, z)$; $x_2=\max(x_2, x)$; $y_2=\max(y_2, y)$ and $z_2=\max(z_2, z)$.

Step 4. Model trimming

This step is crucial because it trims away the 3D region on the model that does not project to the 2D object silhouette. To facilitate the trimming process, we use a voxel representation for the model. Each model is subdivided into voxels of similar size and all these voxels are initially assigned ”IN” to indicate that they belong to the model. The dimension of the voxel is computed so that each voxel can only potentially project to cover a single pixel. We subsequently employ a ray-casting algorithm that casts rays from the image plane to the 3D space to trim the model. When a ray originating from a pixel outside the object silhouette intersects with the model, we assign ”OUT” to voxels that fall on the path of the ray. The whole process can be implemented efficiently by casting rays along the boundary of the object silhouette and discarding the 3D region that does not project to the silhouette.

Another way to look at the whole trimming process is by solid object intersection. First we define a valid 3D region (model defined after Step 3) where intersection is allowed. The resultant model is essentially the intersection of the object-frustum back-projected from the object silhouettes in the key frames to the 3D space. Figure 2f shows the constructed model, which is represented by voxels. Notice that the model has missed out boundaries at the top and bottom of the box.

Step 5. Derivation of dynamic patch voxel

Up to this point, we have had 3D clones that we have trimmed using their respective object silhouettes in the key frames. Due to inaccuracy in the computed camera parameters (camera position, orientation and zoom factor), the resultant clone may be overtrimmed at certain places, especially near the boundary. This implies that the clone will no longer project to cover exactly the object silhouette in some key frames, and visual error will occur at the occluding boundary between real and virtual objects during merging. To remedy this, we make use of a set of dynamic 3D patch voxels that have the same voxel dimension as the clone. For each key frame, we locate the missing region on the clone that will make the clone project perfectly to the object silhouette. We later encode this missing region using patch voxels.

The challenge of this problem is, given that the clone does not project exactly onto the silhouette, how to reinstate the missing 3D region on the clone. We know that the missing region is still within the bounding box of the clone and must be geometrically close to the overtrimmed model,

so we sort of extrapolate the model with the following algorithm.

The patch voxels needed for each key frame are computed with two passes through all pixels within the object silhouette.

Let us discuss the first pass. For each pixel, the algorithm casts a ray from the pixel on image plane to the 3D space (using camera parameters computed earlier). If the ray does not intersect the clone, then overtrimming of the clone has occurred for this pixel and patching of voxels is needed for this pixel in the second pass. Otherwise, the algorithm computes d_1 and d_2, the nearest and farthest z-coordinates, respectively, of the clone intersected by the ray.

In the second pass, we create patch voxels for pixels identified by the first pass. For each such pixel, the algorithm estimates d_1 (and d_2) of the pixel as the average of the d_1 values (and d_2 values, respectively) of the surrounding eight (or less) pixels. Note that the estimation has to be done in some order of pixels closest to pixels with known d_1 and d_2 values and so on. Next, the algorithm casts a ray from the pixel so as to encode the ray segment between d_1 and d_2 as patch voxels.

Note that the last statement on casting a ray is replaced in practice by casting more than one ray per pixel. This is to ensure that any potential patch voxel that may belong to the missing region is not overlooked. Additionally, instead of casting rays to the 3D space as in the first pass of the algorithm, we can also project the voxels of the clone to the image and then perform the second pass for pixels that fall outside the projected region, but within the user-defined object silhouette.

Now, each key frame has a set of 3D patch voxels that will be employed in the following manner. During merging of virtual objects into real images, when the frame is a key frame, we use the overtrimmed model, together with the key frame's patch voxels, as a clone to the real object. This ensures that the object silhouette is exactly covered by the clone. For intermediate frames that are not key frames, we define the clone to be the overtrimmed model together with the patch voxels of the *nearest* key frame. Thus, when we step through the image sequence, the clone, which is the overtrimmed model, is dynamically patched up with patch voxels belonging to different key frames. Figure 2g, h shows two views of the clone with patch voxels drawn.

The clone construction process is somewhat similar to Szeliski's (1993) work, which uses the octree to construct 3D object models from a set of images. However, we wish to point out that his work does not cater to the overtrimming case, which often occurs due to errors in the estimated camera parameters.

7 Merging with virtual objects

To achieve occlusion between real and virtual objects, we make use of the hardware *Z*-buffer in the graphics workstation. The *Z*-buffer is used primarily for visible surface determination. It keeps track of the nearest z-coordinate (in camera coordinate) at each pixel. A pixel colour is only updated if the incoming z value is smaller than that stored in the *Z*-buffer, i.e. is closer to the camera.

To merge the virtual objects with the real images, the following rendering pipeline is used. Note that no specific texture-mapping algorithm is required.

Step 1. Display the image in the frame buffer.
Step 2. Render clones (both overtrimmed model and patch voxels) in the *Z*-buffer only.
Step 3. Render virtual objects in both the frame buffer and the *Z*-buffer. A virtual object appears occluded in the merged image if it is behind real objects with clones rendered in Step 2.

In order to ensure that the clones still conform to the way they are built, we use orthographic projection to render the clones to the *Z* buffer. The inserted virtual objects are rendered with perspective projection model.

8 Results

Figure 3a shows a sample of eight frames from a 99-frame video sequence. Figure 3b shows the augmented video after we add in a virtual dome and two trees. The dome is made up of polygons and is rendered with a single, fixed light source. The trees are texture mapped from images onto some 3D planes. To illustrate the correct occlusion between the inserted virtual objects and the real object in the images, we position the dome and one of the trees behind the front building and let the other tree block the front building. Notice that

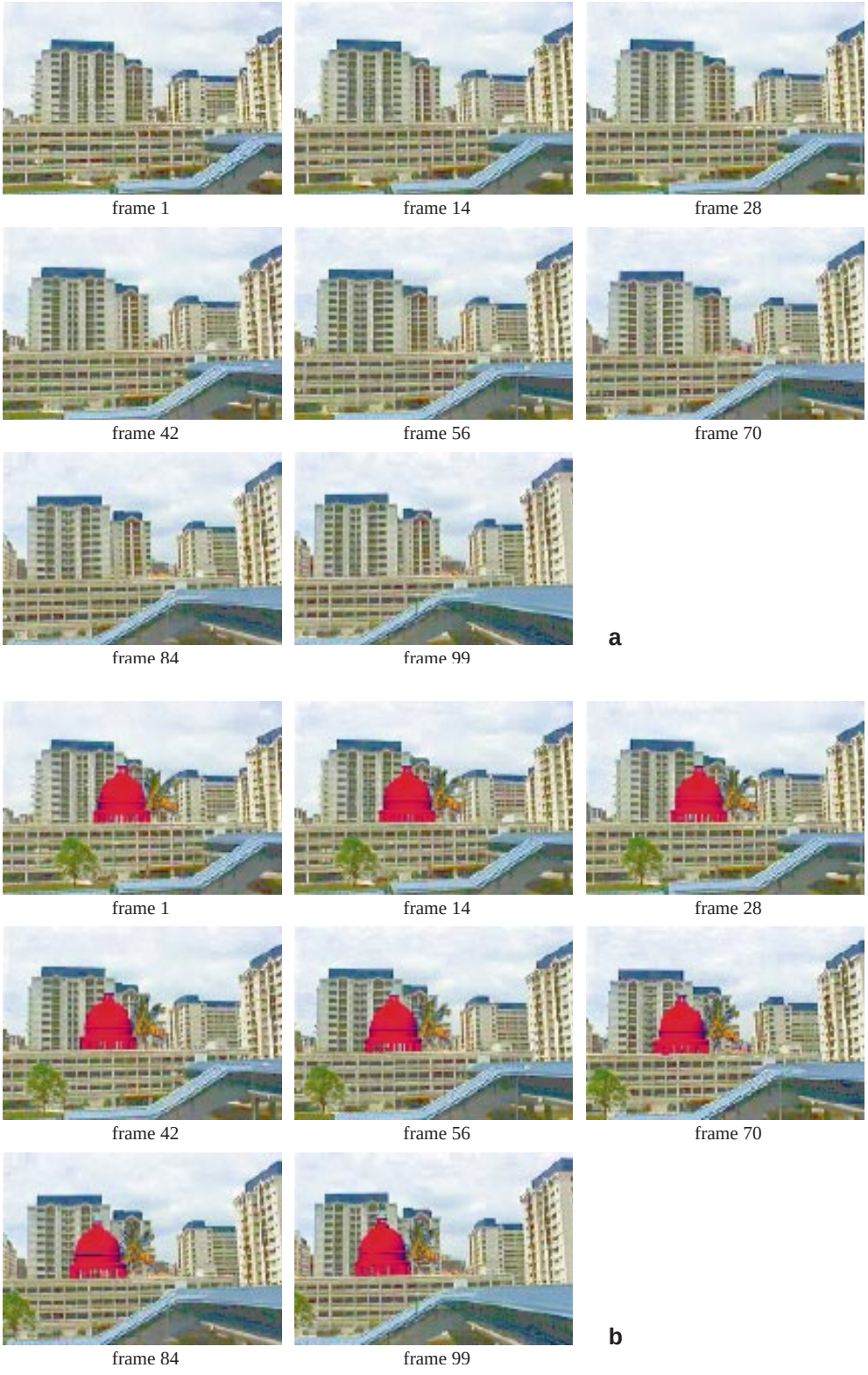


Fig. 3a, b. **a** Original image sequence; **b** augmented image sequence;

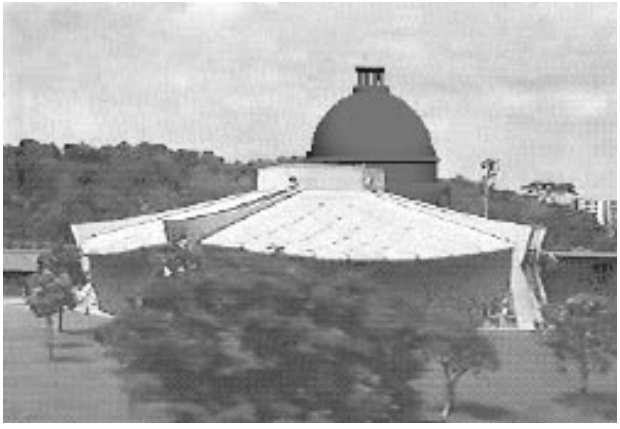
c

d

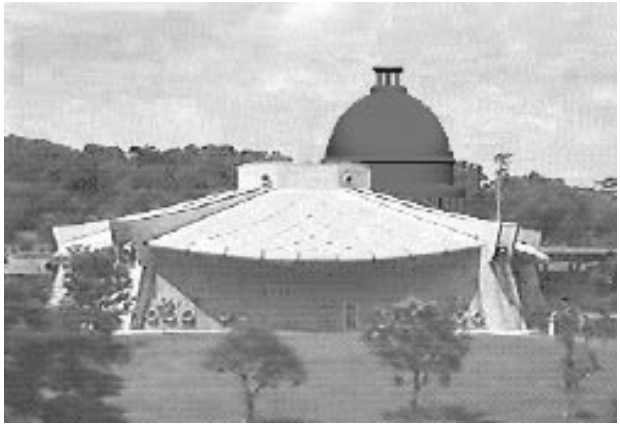
e

f

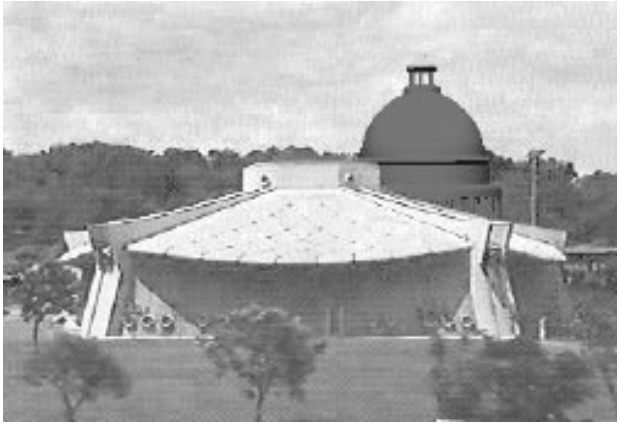
Fig. 3c–f. **c** user-defined object silhouettes; **d** recovered 3D feature points; **e** two views of the constructed clone (overtrimmed model and patch voxels); **f** two views of the objects represented in the computer: 3D feature points, clones, and virtual objects



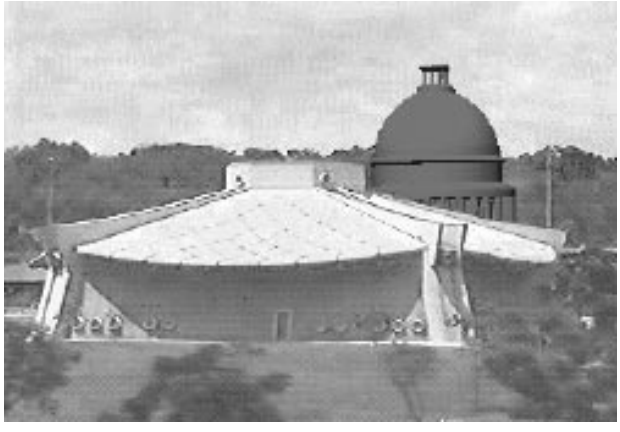
frame 1



frame 25



frame 45



frame 70

Fig. 4. Temple sequence

occlusion is properly shown at the occluding boundary in all the frames. Also, the estimated camera parameters used to render the computer objects are accurate because the virtual objects move together with the images.

Altogether, six segmented silhouettes of the front building (in frames 1, 21, 41, 61, 81 and 99) have been provided, and two of such segmentation results are shown in Fig. 3c. The input video sequence is fed into Tomasi and Kanade's factorization algorithm, and we manage to recover 204 feature points from the scene (Fig. 3d). Based on the object silhouette, the program then clusters the feature points belonging to the real building and constructs an initial model for the building using the 3D bounding box of these points. This is followed by model trimming, which eventually yields the clone (overtrimmed model with patch voxels for each key frame) as shown in Fig. 3e. For better understanding, we also provide two views of all the computer objects used in the merging process (Fig. 3f).

Figures 4 and 5 show some augmented frames from another two image sequences. In the first sequence, segmentation results from eight key frames (1, 10, 20, 30, 40, 50, 60 and 70) are used to construct the clone for the real temple before we insert the virtual dome behind the structure. In the second sequence, five key frames (1, 10, 20, 30 and 43) are used to construct clone for the exotic-looking mushroom-shaped real structure. Two similar looking 3D virtual structures and a 3D virtual statue are then placed behind and in front of the real structure, respectively.

frame 1

frame 15

frame 30

frame 43

Fig. 5. Hotel sequence

We have run the program for these examples on an SGI High Impact machine. A GUI has been provided to enable users to interactively manipulate virtual object positions and orientations. A PC version is also currently under development.

9 Conclusion

Exhibiting correct occlusion between virtual and real objects remains one of the most desirable features, when we insert virtual objects into a real environment. The proposed framework integrates a new clone construction algorithm with other best-of-tools from computer vision and image processing to solve the occlusion problem. A structure-from-motion algorithm from computer vision is employed to derive sparse 3D points from the scene. Using these points together with user-segmented object silhouettes in the key frames, we adopt a clone construction procedure to build approximate clones that exactly project to their respective real object silhouettes. The framework is easy to implement, and no expensive equipment is required.

In this paper, we have not addressed the issue of lighting, which is as critical as occlusion for a con-

vincing merge of virtual objects into a real environment. The 3D clones constructed in our work can be used as a basis to derive illumination interaction between virtual and real objects. Also, we see potential that automatic segmentation on the intermediate frames between two consecutive key frames can be performed. The dependency between the projection model assumed in the structure-from-motion algorithm and that used in clone construction is another area worth investigating.

Acknowledgements. This work is supported by the National University of Singapore under grants RP940641 and RP960618.

References

Azuma RT (1997) A survey of augmented reality. Presence: Teleoperators and Virtual Reality 6:355–385

Berger MO (1997) Resolving occlusion in augmented reality: a contour based approach without 3D reconstruction. Conference on Computer Vision and Pattern Recognition, pp 91–96

Breen DE, Whitaker RT, Rose E, Tuceryan M (1996) Interactive occlusion and automatic object placement for augmented reality. Eurographics 15:C11–C22

Ertl G, Mueller-Seelich H, Tabatabai B (1991) Move-x: a system for combining video films and computer animation. Proceeding of the International Conf. Eurographics '91 pp 305–314

Hu X, Ahuja N (1993) Motion and structure estimation using long sequence motion models. Image Vision Comput 11: 549–570

Kass M, Witikin A, Terzopoulos D (1988) Snakes: active contour models. International Journal of Comp Vision, Vol. 1, No. 3, pp 321–331

Koch R (1993) Automatic reconstruction of buildings from stereoscopic image sequences. Eurographics 12:C339–C350

Mitsunaga T, Yokoyama T, Totsuka T (1995) AutoKey: human assisted key extraction. Proceedings of SIGGRAPH 95 (Los Angeles, CA). In Computer Graphics Proceedings, Annual Conference Series, 1995, ACM Press pp 265–272

Mortensen EN, Barrett WA (1995) Intelligent Scissors for image composition. Proceedings of SIGGRAPH 95 (Los Angeles, CA). In Computer Graphics Proceedings, Annual Conference Series, 1995, ACM Press pp 191–198

Nakamae E, Harada K, Ishizaki T, Nishita T (1986) A montage method: the overlaying of the computer-generated images onto a background photograph. Siggraph, Dallas, ACM Press pp 207–214

Press WH, Teukolsky SA, Vetterling WT, Flannery BP (1992) Numerical recipes in C: the art of scientific computing, 2nd edn. Cambridge University Press, Cambridge, pp 683–688

Quan L, Kanade T (1997) Journal Version: Affine structure from line correspondences with Uncalibrated Affine Cameras. IEEE Transactions on Pattern Analysis & Machine Intelligence vol 19 No 8pp 834–845

Szeliski R (1993) Rapid octree construction from image sequences. CVGIP: Image Understanding 58:23–32

Szeliski R, Kang SB (1994) Recovering 3D shape and motion from image streams using non-linear least squares. J Visual Commun Image Rep 5:10–28

Taylor CJ, Kriegman DJ (1995) Structure and motion from line segments in multiple images. IEEE Trans Patt Anal Machine Intell 17:1021–1032

Tomasi C, Kanade T (1992) Shape and motion from image streams under orthography: a factorization method. Int J Comput Vision 9:137–154

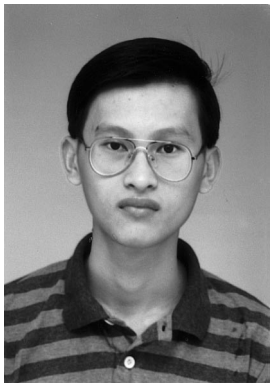
Ueda N, Mase K, Suenaga Y (1992) A contour tracking method using elastic contour model and energy minimization approach, IEICE Trans J75-D-II:111–120

Ullman S (1979) The interpretation of visual motion. MIT Press, Cambridge, Mass

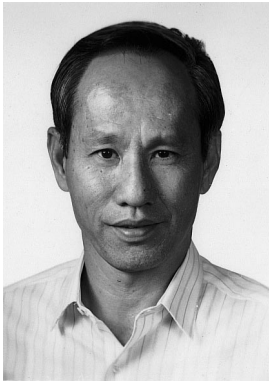
Vieville T, Faugeras O (1990) Feed-forward recovery of motion and structure from a sequence of 2D-line matches. IEEE International Conference on Computer Vision, pp 517

Wloka M, Anderson B (1995) Resolving occlusion in augmented reality. Symposium on Interactive 3D Graphics Proceedings, New York, ACM Press pp 5–12

Yu H, Chen Q, Xu G, Yachida M (1996) 3D shape and motion by SVD under higher-order approximation of perspective projection. Proceedings of 13th International Conference on Pattern Recognition, pp 456–460



KIEM-CHING ONG received his BSc (Hons) and MSc in Computer Science from the National University of Singapore in 1996 and 1998, respectively. He is now working as an R&D Engineer in a commercial simulation company. His research interests include computer graphics, computer animation and their applications.



HUNG-CHUAN TEH received his PhD (1972) in Physics from McMaster University, Canada. His previous research in physics includes studies of the thermal, magnetic and relaxation properties of matters using inelastic thermal neutron scattering and laser-light scattering. He is currently an Associate Professor at the Department of Information Systems and Computer Science at the National University of Singapore. His current research interests include geometric modeling, global illumination and image-based rendering. He is a member of the ACM and IEEE.

TIOW-SENG TAN received his BSc, majoring in Computer Science and Mathematics, from the National University of Singapore (NUS) in 1984. Having worked for a few years in the IT industry, he then returned to NUS to complete an MSc in 1988, and then a PhD in 1992 at the University of Illinois at Urbana-Champaign. He is currently a Senior Lecturer at the Department of Information Systems and Computer Science, NUS. His research interests include computational geometry, computer graphics and image processing.