Virtual View Generation for 3D Digital Video

Saied Moezzi, Li-Cheng Tai, and Philippe Gerard
University of California, San Diego

Virtual reality systems use digital models to provide interactive viewing. We present a 3D digital video system that attempts to provide the same capabilities for actual performances such as dancing. Recreating the original dynamic scene in 3D, the system allows photorealistic interactive playback from arbitrary viewpoints using video streams of a given scene from multiple perspectives.

Virtual reality can become a medium of great usefulness in entertainment and education if it can incorporate recordings of actual events. While much work has gone into creating synthetic environments that correspond to counterparts in the real world, few have attempted to incorporate real people and events into such environments. To achieve this, an event of interest must be truthfully captured by real-time sensors such as video. From this recording, faithful digital replications must then be created such that the original performances can be presented using standard computer graphics methods and viewed from arbitrary perspectives.

We present a system for generating and replaying photorealistic 3D digital video sequences of real events and performances. 3D video embodies the truthfulness of video recordings and the interactivity of 3D graphics (see Figure 1). This system employs Multiple Perspective Interactive Video (MPI-Video),[1] an infrastructure for the analysis and management of, and interactive access to, multiple video cameras monitoring a dynamically evolving scene such as a football game.

Related work

The problem of virtual view creation, or view synthesis or interpolation of real scenes, has received increasing attention in recent years. Current approaches divide into two classes: image-based and model-based. Image-domain methods employ warping or morphing techniques to interpolate intermediate views from real images. Model-based methods first recover the geometry of the real scene; the resulting 3D model can then be rendered from desired viewpoints. Our method belongs to the latter class.

Image-based methods

The best-known image-domain method is Apple's QuickTime VR.[2] By capturing the 360-degree views (cylindrical panoramic images) of an environment from a fixed position, you can interactively adjust view orientation by rendering the corresponding portion of the panorama. Other approaches use image warping. Chen and Williams[3] determined camera transformations with pixel correspondences, then used morphing to generate intermediate views. Skerjanc and Liu[4] used known camera positions to obtain depth information and generate virtual views. Chang and Zakhor[5] obtained depth information by using an uncalibrated camera that "scans" a stationary scene and transforms points on camera image planes onto the plane of the virtual view. Seitz and Dyer[6] proposed exploiting monotonicity along epipolar lines to compose physically valid intermediate views without the need for full correspondence information.

Several recent developments employ the plenoptic function, which describes light rays visible at any point in space. McMillan and Bishop[7] developed an image-based rendering system using a 5D representation and cylindrical projections. Levoy and Hanrahan[8] and Gortler et al.[9] used 4D formulations of the plenoptic function for virtual view synthesis. In general, image-domain approaches need fewer computational resources than 3D model-based approaches, but they often limit supported virtual views to a narrow range.

Model-based methods

At a high level, a model-based approach to 3D digital video creation involves three processes. First, an event or a scene must be recorded by multiple strategically located cameras. Our experiments used 17 cameras surrounding a stage area to record various performances. In a similar way, Virtualized Reality (Kanade et al.[10]) used six to eight cameras, placed around a hemispherical dome five meters in diameter, to record an actor in motion. Fuchs et al.[11] used image data acquired by many cameras installed around a small environment such as a conference room. By contrast, Tseng and Anastassiou[12] used images captured simultaneously by a set of equidistant cameras with parallel axes, in vertical and horizontal lineups.

The next step extracts a 3D model of the environment using computer vision techniques. Existing methods use depth maps as 2.5D repre-

1070-986X/97/$10.00 © 1997 IEEE

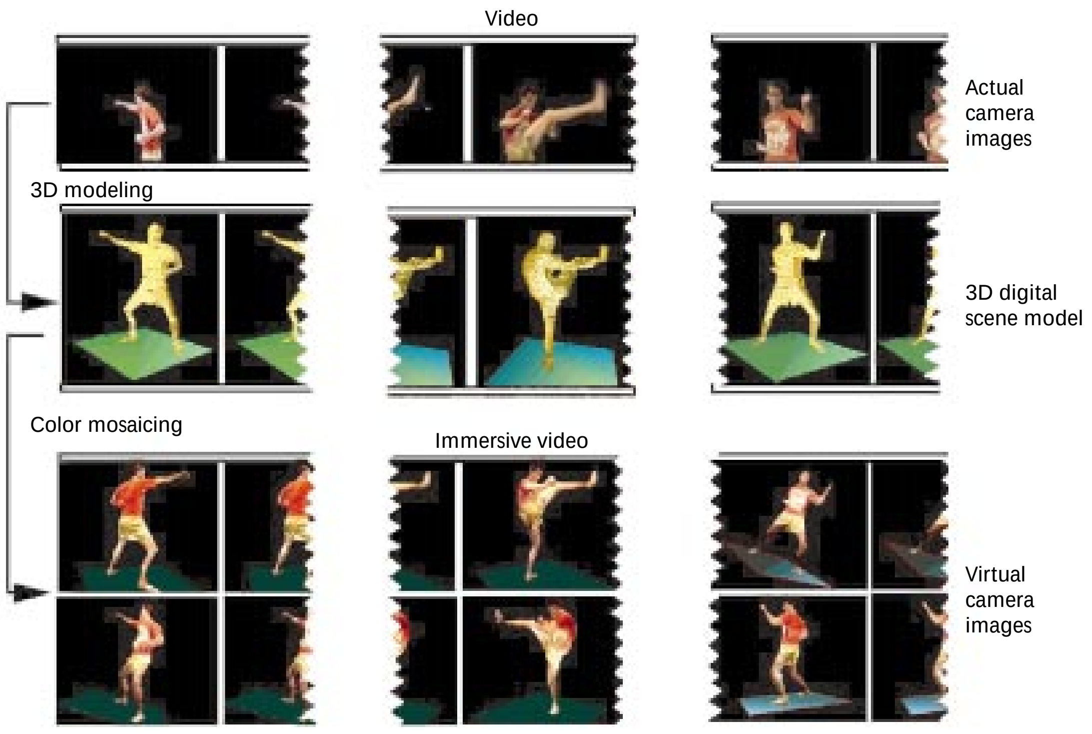

Figure 1. Reality rendering. Given multiple camera images of the same scene (top), the 3D shapes of the scene object are first reconstructed (middle), then the surface colors are determined, resulting in a photorealistic 3D video sequence (bottom).

sentations of the scene's geometry. Some researchers[10-12] have used stereo methods to compute time-varying depth maps. Our system concentrates on accurate recovery of the 3D shapes of dynamic or foreground objects with a volume occupancy method.

The final step in generating 3D digital video is to render the obtained model from the view of a virtual camera. In systems described by Kanade et al.[10] and Fuchs et al.,[11] stereo-based depths are already aligned with the pixels of their corresponding images. The former uses the depth map from the closest camera, based on the viewer's position, to render the scene. The latter updates, maintains, and displays the acquired depth map for the viewer's current position and orientation. Similarly, Tseng and Anastassiou[12] generated virtual images by interpolating real views scanline by scanline based on disparity information.

By comparison, our approach creates true 3D models with fine polygons, each separately colored (thus requiring no texture-rendering support), and the viewing position plays no role in the modeling process. Our 3D models can use standard object formats such as VRML (Virtual Reality Modeling Language) delivered through the Internet and viewed with VRML browsers (see Figure 2 and Figure 3). Our approach is more suitable in the client-server scenario because, unlike

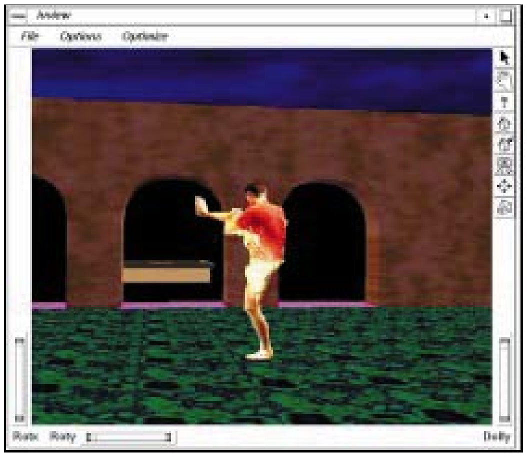

Figure 2. 3D video sequence placed in a synthetic environment.

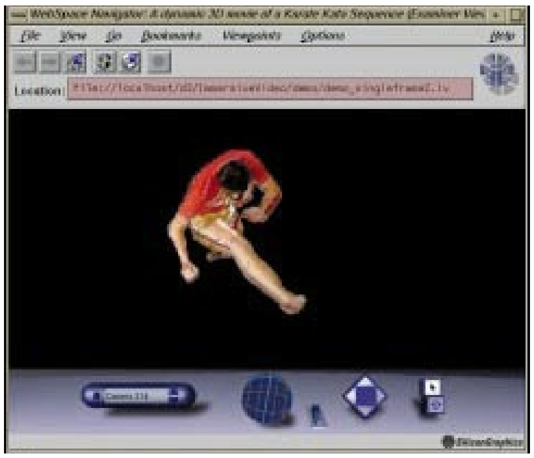
WebSpace Navigator: A dynamic 3D movie of a Karate Kata Sequence (Examiner View)
File
View
Go
Bookmarks
Viewpoints
Options
Help
Location:
file:/localhost/d3/ImmersiveVideo/demo/demo_singleframe2.iv
SiliconGraphics

Figure 3. 3D video sequence viewed with an Open Inventor browser.

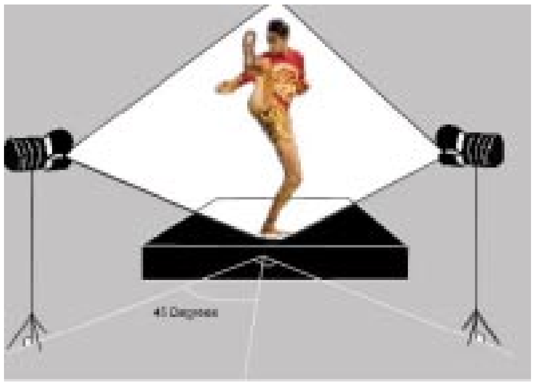

Figure 4. Standard studio lighting.

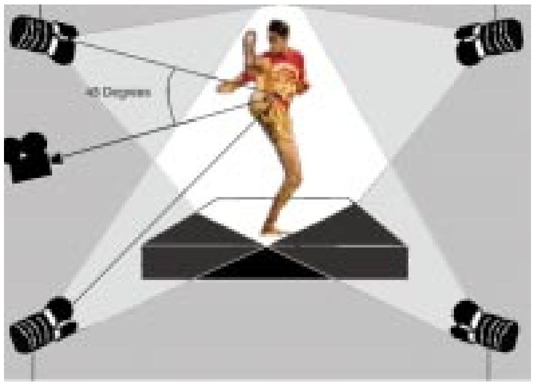

Figure 5. Lighting for 3D video data acquisition.

earlier approaches, real views need not be transferred to the client, significantly reducing the bandwidth required.

3D digital video creation and playback

Our system, Immersive Video,[13] converts multiple video streams of a given scene into a 3D realistic dynamic model sequence for interactive viewing (as shown in Figure 1). Stages involved include data acquisition, camera calibration, object segmentation, 3D model creation, color mosaicing, and interactive playback.

In data acquisition, multiple perspective cameras record the scene. Camera calibration determines the positions and orientations of all cameras in a global coordinate system, which then facilitates assimilating object information from all camera views. The object segmentation process computes the projection of dynamic objects in every frame, in each camera. For every set of synchronized video frames, model creation reconstructs the 3D shapes of all dynamic objects in the scene. Color mosaicing "paints" these time-varying 3D geometric models into realistic replications of actual objects. The resulting 3D video sequence is then rendered from a virtual camera that the viewer can control interactively during playback. Detailed explanations of each stage follow.

Data acquisition

Data acquisition involves recording an event from appropriate perspectives. Important considerations include camera placement, video synchronization, scene illumination, and camera calibration. Camera placement influences shape determination. Our experience indicates the following considerations:

- Cameras should be arranged to give, as closely as possible, complete coverage of the scene such that any part of the scene is visible by two or more cameras.
- Cameras should be evenly distributed across some imaginary sphere covering the center of interesting objects to maximize the resolution power for shape determination, as discussed later.

Accurately synchronizing frames from all cameras plays a crucial role in the quality of the final results. We found it easy to achieve video synchronization by providing a common time reference in the recordings using an event visible to all cameras (such as turning lights off and on).

In earlier experiments,[13] we recorded outdoor and indoor events without consideration for illumination. Problems in separating foreground dynamic objects from the background led us to try a studio setup (Figure 4 and Figure 5), where we can control illumination to facilitate good separation of objects (in our case, the performers) from the background. One possible way to simplify foreground-background separation is to intensely illuminate the performers in the scene (as shown in Figure 5). Intense lighting allows a smaller camera aperture, hence less-illuminated background objects such as walls, cameras, tripods, and so forth will not appear in the recordings. Cameras must also be white-balanced; this calibrates the color parameters of all cameras, an important issue in the accuracy of the final colored 3D frames.

During data acquisition, for calibration purposes, an object of known geometry must be placed in the scene. Obtained from each camera view, images of this object can provide the basis for determining each camera's position and orientation.

Camera calibration

External calibration determines a camera's position and orientation with respect to some world coordinate system. Accurate results are critical for assimilating information from multiple cameras and hence accurately determining objects' shapes and colors. Camera calibration requires a set of image points whose world coordinates are known. In our approach, cameras placed at the desired locations remain stationary and need to be calibrated only once.

We use Tsai's coplanar calibration algorithm,[14] which requires a minimum of five coplanar points and their 3D world coordinates. Grid points on a calibration object (such as a cubic box with unit markers), placed in the scene and recorded by all cameras, can be used for this purpose. Our interactive calibration software allows us to manually select corresponding image and model points for each camera and obtain their positions, axes of rotation, and pitch angles with respect to a reference world coordinate system.

The accuracy of the results for each camera can be examined by rendering the calibration object from the computed position and orientation and overlaying the rendering on top of the actual image. Significant errors often occur during the first few tries, so several iterations of the point selection and verification processes are needed to obtain satisfactory results.

Object segmentation

During the object segmentation phase, each frame of the digitized sequences from all available cameras is processed and image regions belonging to foreground objects are computed. Foreground objects are detected by computing the difference between the current frame and a background frame. For a given camera, the background frame represents the scene captured by that camera when no dynamic object was present in the environment. This process results in a set of binary images indicating object versus background.

The controlled environment of the studio setup can be exploited to acquire video sequences that produce highly accurate binary images. Ideally, the background appears uniformly black, while foreground objects have high-intensity colors and can be unambiguously determined. This ideal condition results from careful control of the lighting and background materials.

Model creation

Given the position and orientation of all cameras and the synchronized binary frames computed by the above process, we can now recover the object shape at each time instance. We use a volume intersection method to reconstruct an object's 3D shape. We divide the volume enclosing the scene into small elements (voxels), and then for each frame iterate through all voxels to determine if at that instant the voxel is occupied by the object (Figure 6).

This algorithm, given in Figure 7, assumes all voxels are occupied initially. For each voxel, we calculate the corresponding pixel location (*x*, *y*) in each camera's image plane. Checking the binary images, we determine whether (*x*, *y*) is part of the objects or the background. In our current algorithm, if one voxel maps to a background (empty) pixel in any binary image, the voxel is declared "empty." Assuming perfect camera cali-

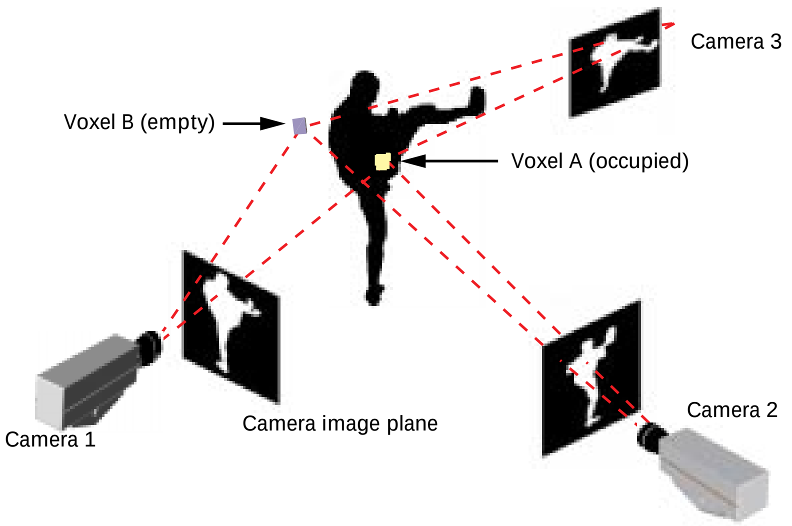

Figure 6. Voxel occupancy determination. The scene is filmed by three cameras. The camera image planes show the respective binary images, with white corresponding to dynamic objects and black to empty space. Two sample voxels are shown. Voxel A projects to "object-occupied" regions in all cameras and is determined occupied; Voxel B projects to "empty" regions in cameras 1 and 2 and is declared empty.

```
for each frame f do
        Initialize all voxels to be "occupied"
        for each voxel v do
            for each camera c do
                project the center of the voxel back into
                the camera view plane and determine
                the coordinates (x,y of the corresponding
                pixel p
                if in the segmented image of frame f and
                camera c, p is empty
                    then
                        set v to be empty
                        goto ENDINNERLOOP
                fi
            end
            ENDINNERLOOP:
        end
end
```

Figure 7. Algorithm to determine voxel occupancy.

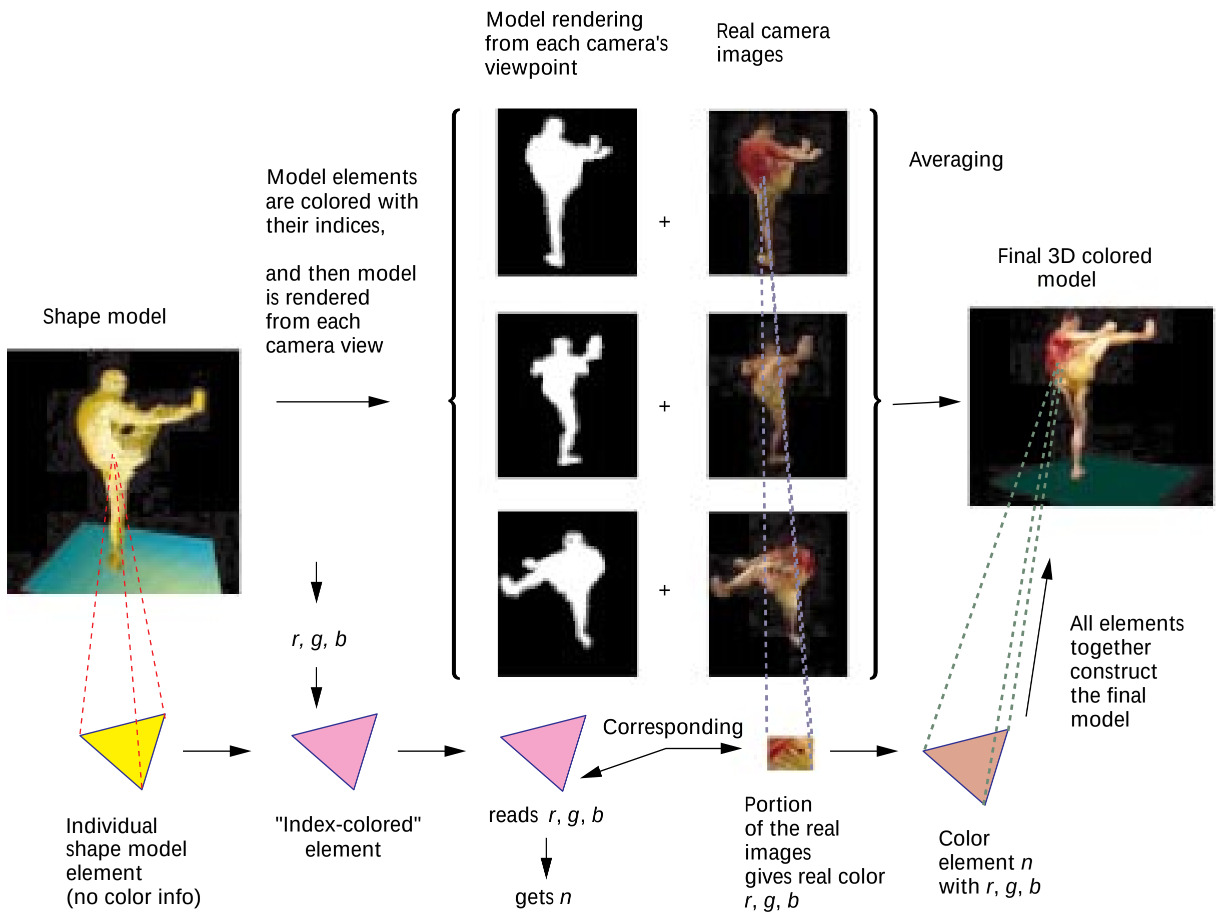
Model rendering from each camera's viewpoint
Real camera images
Averaging
Model elements are colored with their indices,
and then model is rendered from each camera view
Final 3D colored model
Shape model
+
+
+
r, g, b
All elements together construct the final model
Corresponding
Individual shape model element (no color info)
"Index-colored" element
reads r, g, b
gets n
Portion of the real images gives real color r, g, b
Color element n with r, g, b

Figure 8. Color mosaicing "paints" the object model with color information from real camera views. The model is first painted with colors that are actually mappings of element indices. Then rendering from real camera views establishes the correspondences between model elements and pixels in real camera images. The corresponding pixels in turn yield true color values for each element on the object surface.

bration, this approach accurately detects all occupied voxels, but not all empty ones. Thus our model can contain more volume than that actually occupied. We can eliminate such "extra" voxels using depth maps computed via stereo methods.

This shape determination algorithm is very sensitive to calibration errors. Small deviations in camera parameters can translate to "losses" of many actually occupied voxels. In the future, we plan to develop a more robust approach using statistical methods. Another shortcoming of this approach is that it cannot eliminate voxels inside a concave portion of an object. Since such regions may be occluded in all available camera views, they are still marked "occupied" after volume intersection. Good camera distribution and coverage will minimize this problem and maximize the ability to resolve voxel occupancy correctly.

This algorithm can be tuned for speed or quality of the model by adjusting the voxel size. We can quickly visualize the object shape using large voxels or build accurate models using small voxels and longer computation time. This customizability allows us to construct models suitable for various applications from low-resolution video games to high-resolution 3D movies. For example, for a 1m × 1m × 2m volume covered by 17 cameras, using a high-powered graphics workstation (such as an SGI Onyx Reality Engine 2), we can compute 5-cm voxel models in less than 30 seconds. A high-quality 1-cm model takes about 15 minutes.

After determining all occupancies, we have a set of filled voxels. Because we are interested in the shape only, we can ignore the information about the voxels internal to an object to reduce the data size. One method for internal voxel elimination is the Marching Cubes[15] algorithm, which also converts the voxel-based model into a triangle-based surface representation. We then paint the shape models, either voxel- or surface-based, using color mosaicing.

Color mosaicing

To paint the 3D model, one approach uses the real camera images as textures and maps them onto the model. Our earlier work[13] shows this method can generate realistic renderings but requires high-powered graphics hardware with powerful texture-mapping support. To make 3D video playback practical on general-purpose machines and eliminate the need to transfer original camera streams to every client (which requires significant bandwidth), we decided to encode color information with elements (voxels or triangular faces) of the 3D model, which can be rendered on systems with simple polygon-based 3D graphics. Using small triangles or voxels, we can fully capture the original textures in the real images.

To recover the shape model's colors, we need to establish the correspondence between shape model elements and pixels in the camera images. A naive approach will iterate through all elements to calculate their projections on each camera image plane, and in case of no occlusion, the pixels will yield the proper color values. Projection calculation and occlusion check are expensive and slow processes. To speed up color determination, we take advantage of the high-speed graphics hardware and turn the video memory into a data processing buffer.

Our approach, illustrated in Figure 8, first assigns each element a unique index that is then mapped onto a point in the RGB color space (for the algorithm, see Figure 9). Each element is then uniquely colored with its RGB value. For each camera we render the shape model from that camera's viewpoint and save the resulting image in memory. Comparing each rendering with its corresponding real camera image pixel by pixel, we can determine what shape elements appear at certain pixels, which in turn gives the proper colors of the elements. For the voxel-based model we can also eliminate voxels not rendered in any camera views because they are not visible (that is, they are internal to objects).

One issue in the color mosaicing process concerns how to combine multiple color information when one element is visible in several cameras. A number of approaches are possible. For each element we can average colors or select cameras based on some measure of "closeness." For example, we can use the camera view with the smallest normal angle (the camera closest to directly facing the element) or the closest distance.

After trying different approaches, we obtained the best results with area-weighted averaging, or averaging color information from different cameras, weighted by the area. That is, if an element occupies a large area in a camera view plane, the color information from this camera should be more accurate and have more weight. This method has the benefit of blurring discontinuities in camera coverage boundaries due to color balancing errors.

Color mosaicing is a general algorithm and works well for both voxel- and surface-based models. Its only limitation is the color depth of the video memory. For 24-bit displays, it can handle up to 2^{24} elements, which we believe is more than necessary for most 3D scene models (in our experiment, a high-resolution model of a person contains just 70,000 elements).

```
for each shape element (voxel or triangle) $t$ with index $i_t$ do
    // assume a 24-bit RGB color space with 8 bits and $2^8 = 256$
      shades for each color channel
    $n = i_t$, $r = n/(256 \times 256)$ (integer division)
    $b = n - g \times 256$
    assign color $(r,g,b,)$ to $t$
end
for each camera $c$ do
    render the model from the view point of camera c with black
    background and save the resulting image as $V_c$
end
for each element $t$ do // initialize color accumulators and counter
                           of each element
    $R[t] = 0$, $G[t] = 0$, $B[t] = 0$, $k[t] = 0$
end
for each camera $c$ do
    for each pixel $p$ in $V_c$ do
        if $p$ is not black (i.e., has some color $(r_V, g_V, b_V)$)
          then
                $i = r_V \times 256 \times 256 + g_V \times 256 + b_V$
                get the element $t$ whose index is $i$
                let $(r_I, g_I, b_I)$ be the color of pixel $p$ in the real
                image $I_c$ from camera $c$
                // sum up colors corresponding to $t$
                $R[t] = R[t] + r_I$, $G[t] = G[t] + g_I$, $B[t] = B[t] + b_I$
                // accumulate number of pixels corresponding to $t$
                $k[t] = k[t] + 1$
        fi
    end
end
for each element $t$ do
    $r = R[t]/k[t]$, $g = G[t]/k[t]$, $b = B[t]/k[t]$
    assign color $(r,g,b,)$ to $t$
end
```

Figure 9. The color mosaicing algorithm determines the color of each shape element.

Interactive playback

Once the models for individual frames are available, they can be treated like ordinary 3D

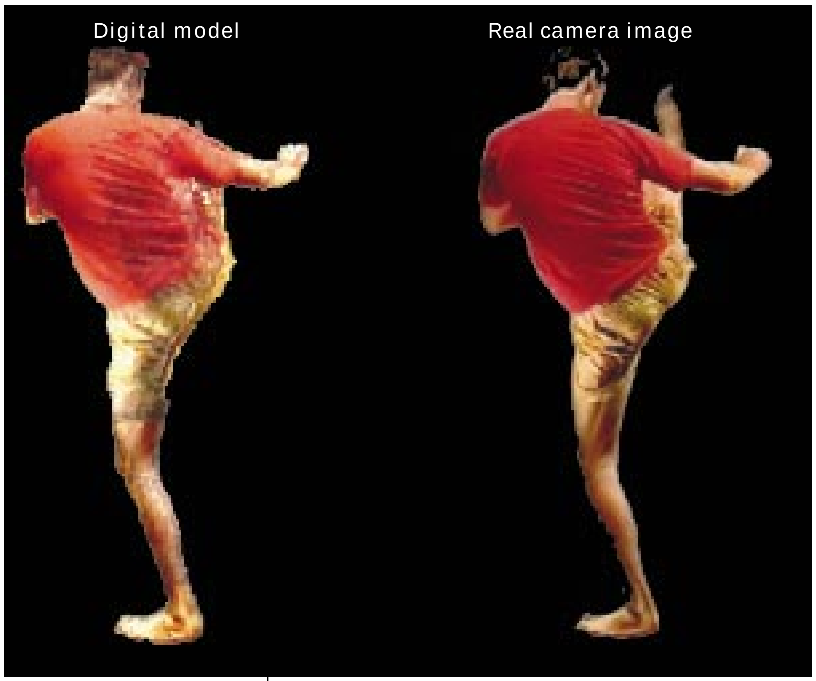

Figure 10. Comparsion between a real camera image (right) and the constructed 3D model rendered from the same viewpoint of the camera (left).

graphics objects and rendered with standard graphics libraries. A simple scheme for real-time playback involves loading them into memory and switching between frames at a controllable rate, a function supported by Open Inventor browsers. Many issues in 3D video playback remain unresolved; we discuss them later in this article.

Experiment and results

For the experimental setup, we chose a TV studio because the environment can be precisely controlled. We arranged the lights to concentrate on a 2m × 2m region so that outside areas appeared dark, facilitating good segmentation. We arranged 17 cameras to give complete coverage around the region from the left, right, front, back, and top. A karate player served as the subject of our video and performed various actions within our designated area.

We digitized eight seconds of video, which we then processed for object segmentation. We manually corrected certain frames to clean up shadow and reflection effects. C++ programs using the Open Inventor library implemented the model creation and color mosaicing processes to generate 240 3D models. The resulting 3D sequence can be viewed with standard Open Inventor and VRML browsers such as Webspace (see Figure 3) and combined with synthetic environments, as in Figure 2 where a virtual "arena" provides the stage for the 3D performance.

Figure 10 compares the created model and a real camera view for a chosen frame. While some artifacts are apparent if you look closely, the model captures the fine texture details of the original image. These results demonstrate the success of our approach and its potential for 3D digital video.

To evaluate the quality of our result, we compare, for each camera, the real image and the corresponding virtual view. By calculating the percentage of area containing mismatches in the virtual image, we obtain a measure of errors in the 3D model.

This measurement ranged from 5 percent to 29 percent. The worst results are found in top and wide-angle views. Close-up viewpoints give an error measurement of less than 5 percent. Most errors consist of missing parts or shifts between the model and the real images and arise from problems in digitization (where horizontal shifts are introduced between frames), calibration, and segmentation. Thus close-up views work best for accurate 3D model construction. Using professional studios and equipment (such as the TBC, or time base corrector) can also reduce error sources.

Discussion

Making 3D video a practical medium requires reducing the complexity and amount of data. In particular, we must address the issues of model representation, rendering speed, and temporal compression. Significant trade-offs exist between voxel- and surface-based models. Graphics hardware tends to be optimized for rendering surfaces. In the Open Inventor formats that we use, voxel models result in more process overhead and storage space than surface models. One benefit of voxels, however, is their simplicity, an advantage when considering compression.

As in traditional digital video, data size presents a significant problem. At 1-cm voxel resolution, the model of a single person in any pose has about 17,000 voxels, or 70,000 triangles. Assuming 30 frames per second, storage needs quickly reach the gigabyte scale for several seconds of 3D video of this person. As neighboring frames are typically similar, temporal compression can produce significant savings in data size. There are several potential approaches to compressing 3D digital video, as follows.

Temporal differences. We can transmit the changes (additions or deletions) between consecutive frames. This method is easily applicable to voxel-based models.

2D slices. Slicing a 3D model along the *Y* axis gives us a 2D image for each *Y* value. Then we can simply compress these images using conventional methods like JPEG or even MPEG. Another possibility is to compute bounding rectangles (or polygons) for every slice. We can then transmit the information related to the object by a hierarchical representation inside the bounding rectangle. An advantage of 2D slices is that we can use temporal correlation (on the same slices of consecutive frames) or spatial correlation (on neighboring slices) for compression. This representation also allows us to create a mathematical function that approximates the contour and represents it as a mathematical curve.

Transforms in the 3D domain. By considering voxels the 3D equivalent of pixels, we could apply a domain transform to this 3D array to determine a quantizer and an encoder that would permit a scalable display. Then, if we decide to transmit only a coarse representation of our model to display it on a slow machine, we could transmit only the low-frequency parameters. JPEG and MPEG methods are good candidates, but may waste a lot of time transmitting coding of empty spaces because the bounding box in which the model is calculated itself contains a lot of empty space. It might be more effective to use a 3D wavelet transform and zero-tree coding[16] instead.

Efficient data organization. We can use index tables and store color indices instead of the full colors. Another approach is to use quantization to reduce bits required for color and coordinate values.

Geometric compression approaches. These include lossy compression by rounding coordinates and polygon mesh simplification.

In selecting a compression and representation scheme, one issue cannot be overlooked: The decoding and rendering speed of the chosen scheme is a function of the graphics hardware architecture. Voxels can be more easily compressed, but polygon surfaces have an advantage over voxels for rendering speed in graphics workstations accessible to us. The ideal scheme has to support quick decoding yet generate representations that can be passed quickly to the graphics pipeline and rendered for real-time video performance.

Conclusions and future work

Our 3D digital video system creates realistic time-varying 3D model sequences from multiple video streams. The 3D sequences allow full representation of the original dynamic objects and support new types of interaction for the viewer. This new medium has the potential to revolutionize the use of video in interactive television, video games, and the entertainment industry.

Many issues remain that we hope to address in future research. Of high priority is the development of appropriate temporal compression schemes to reduce data size. Encoding and decoding of 3D digital video will require more research and standardization, but we believe the benefits will justify the efforts. MM

References

1. P. Kelly et al., "An Architecture for Multiple Perspective Interactive Video," *Proc. ACM Multimedia 95*, ACM Press, New York, 1995, pp. 201-212.
2. S.E. Chen, "Quicktime VR—An Image-Based Approach to Virtual Environment Navigation," *Proc. Siggraph 95*, ACM Press, New York, 1995, pp. 29-38.
3. S.E. Chen and L. Wiliams, "View Interpolation for Image Synthesis," *Proc. Siggraph 93*, ACM Press, New York, 1993, pp. 279-288.
4. R. Skerjanc and J. Liu, "A Three-Camera Approach for Calculating Disparity and Synthesizing Intermediate Pictures," *Signal Processing: Image Comm.*, Vol. 4, No. 1, Nov. 1991, pp. 55-64.
5. N.L. Chang and A. Zakhor, "Arbitrary View Generation for Three-Dimensional Scenes from Uncalibrated Video Cameras," *Proc. IEEE Int'l Conf. on Acoustics, Speech, and Signal Processing*, IEEE Press, Piscataway, N.J., pp. 2455-2458.
6. S.M. Seitz and C.R. Dyer, "Physically Valid View Synthesis by Image Interpolation," *Proc. Workshop on Dynamic Representation of Visual Scenes*, IEEE Press, Piscataway, N.J., 1995, pp. 18-25.
7. L. McMillan and G. Bishop, "Plenoptic Modeling: An Image-Based Rendering System,'" *Proc. Siggraph 95*, ACM Press, New York, 1995, pp. 35-46.
8. M. Levoy and P. Hanrahan, "Light Field Rendering," *Proc. Siggraph 96*, ACM Press, New York, 1996, pp. 31-42.
9. S.J. Gortler et al., "The Lumigraph," *Proc. Siggraph 96*, ACM Press, New York, 1996, pp. 43-54.
10. T. Kanade, P.J. Narayanan, and P.W. Rander, "Virtualized Reality: Concepts and Early Results," *IEEE Workshop on Dynamic Representation of Visual Scenes*," IEEE Press, Piscataway, N.J., 1995, pp. 69-76.
11. H.Fuchs et al., "Virtual Space Teleconferencing Using a Sea of Cameras," *Proc. 1st Int'l Symp. on Medical Robotics and Computer Assisted Surgery*, 1994, pp. 161-167.
12. B.L. Tseng and D. Anastassiou, "Multi-Viewpoint

Video Coding with MPEG-2 Compatibility," *IEEE Trans. Circuits and Systems for Video Tech.*, Vol. 6, No. 4, Aug. 1996, pp. 414-419.
13. S. Moezzi et al., "Reality Modeling and Visualization from Multiple Video Sequences," *IEEE Computer Graphics and Applications*, Vol. 16, No. 6, Nov. 1996, pp. 58-63.
14. R.Y. Tsai and R.K. Lenz, "A New Technique for Fully Autonomous and Efficient 3D Robotics Hand/Eye Calibration," *IEEE Trans. Robotics and Automation*, Vol. 5, No. 3, June 1989, pp. 345-358.
15. W.E. Lorensen and H.E. Cline, "Marching Cubes: A High-Resolution 3D Surface Construction Algorithm," *Computer Graphics*, Vol. 21, No. 4, July 1987, pp. 163-169.
16. J.M. Shapiro, "Image Coding Using the Embedded Zerotree Wavelet Algorithm," *Mathematical Imaging: Wavelet Applications in Signal and Image Processing (Proc. SPIE 2034)*, SPIE, Bellingham, Wash., 1993, pp. 180-193.

Saied Moezzi is a project scientist in the Department of Electrical and Computer Engineering at the University of California, San Diego, where he leads several research efforts in immersive 3D video, telepresence, and interactive visual information systems for medical images. He received a BS in electrical engineering in 1980 and an MS degree in computer science with honors in 1983 from the University of Kansas, and a PhD in computer science and engineering in 1992 from the University of Michigan.

Li-Cheng Tai received a BS and an MS in computer engineering from the University of California, San Diego. Currently he is pursuing a PhD at UCSD. His research interests include open hypermedia systems, virtual view synthesis for 3D interactive video, and human-computer interaction in multimedia systems.

Philippe Gerard is an engineering graduate student at the Conservatoire National des Arts et Métiers, Paris. He earned an MBA from ISG-Paris in 1993 and an MS in audiovisual techniques from the University of Valenciennes in 1989. His industry work has included video engineering, production, and post-production. Research interests include digital video editing techniques and videoconferencing.

Contact Moezzi at the Visual Computing Laboratory, Department of Electrical and Computer Engineering, University of California, San Diego, 9500 Gilman Drive, La Jolla, CA 92093-0407, e-mail moezzi@ece.ucsd.edu.