Thirteenth Eurographics Workshop on Rendering (2002)
P. Debevec and S. Gibson (Editors)

Video Flashlights – Real Time Rendering of Multiple Videos for Immersive Model Visualization

H. S. Sawhney, A. Arpa, R. Kumar, S. Samarasekera, M. Aggarwal, S. Hsu, D. Nister, K. Hanna

Vision Technologies Laboratory
Sarnoff Corp., Princeton, NJ, USA
Email:hsawhney@sarnoff.com

Abstract

Videos and 3D models have traditionally existed in separate worlds and as distinct representations. Although texture maps for 3D models have been traditionally derived from multiple still images, real-time mapping of live videos as textures on 3D models has not been attempted. This paper presents a system for rendering multiple live videos in real-time over a 3D model as a novel and demonstrative application of the power of commodity graphics hardware. The system, metaphorically called the Video Flashlight *system, "illuminates" a static 3D model with live video textures from static and moving cameras in the same way as a flashlight (torch) illuminates an environment. The Video Flashlight system is also an augmented reality solution for security and monitoring systems that deploy numerous cameras to monitor a large scale campus or an urban site. Current video monitoring systems are highly limited in providing global awareness since they typically display numerous camera videos on a grid of 2D displays. In contrast, the Video Flashlight system exploits the real-time rendering capabilities of current graphics hardware and renders live videos from various parts of an environment co-registered with the model. The user gets a global view of the model and is also able to visualize the dynamic videos simultaneously in the context of the model. In particular, the location of pixels and objects seen in the videos are precisely overlaid on the model while the user navigates through the model. The paper presents an overview of the system, details of the real-time rendering and demonstrates the efficacy of the augmented reality application.*

1. Introduction

In this paper, we present an immersive model-based video visualization system, called the *Video Flashlight* system, that provides an augmented reality solution for video surveillance and monitoring applications. The system enables active browsing and visualization of a 3D model of a large-scale site by rendering multiple videos from a blanket of ground and aerial video cameras over the model. By seamlessly rendering dynamic video data from multiple cameras on top of a 3D model of a site, the system allows the users to view the dynamic action in the context of a global 3D model while actively viewing the integrated model and videos from the viewpoint of a virtual camera. Thus, the system supports seamless viewpoint change for a sky-to-street level browsing mode with integrated views of the model and the videos.

Since any single video camera provides only a "soda-straw" view of the world, traditional video surveillance and monitoring systems attempt to create a global picture by simply stringing together multiple camera videos on a grid of 2D displays as shown in Fig. 1. Clearly, such a display does not allow a user to interpret the video images and actions in the context of a global 3D model. In addition, the relative locations of activities across multiple cameras is not made explicit in the display and is not easily derivable from the videos directly. In contrast, the proposed Video Flashlight system uses a 3D model of a site to provide global context for objects and activities seen in multiple videos. The Video Flashlight browser renders multiple video streams in real-time overlaid on the 3D model and allows the user to navigate through the model while the videos are continuously rendered. A snapshot of the system's display showing rendering of multiple videos on top of a 3D model is shown in Fig. 5. It is obvious from the figure that any object seen in any video can be easily referenced to the model.

A key component of the Video Flashlight system is the real-time rendering of multiple live videos overlaid precisely on a 3D model. Videos and 3D models have traditionally ex-

Figure 1: *Traditional multi-video "visualization" showing a grid of* 2×3 *displays. The context of the videos with respect to the environment and with respect to each other is not at all clear in this visualization.*

isted in separate worlds and as distinct representations. Although texture maps for 3D models have been traditionally derived from multiple still images, real-time mapping of live videos as textures on 3D models has not been attempted. The Video Flashlight system's rendering of multiple live videos in real-time over a 3D model is a novel and demonstrative application of the power of commodity graphics hardware. The rendering algorithms exploit known shadow mapping and projective texturing techniques from the graphics literature. These algorithms are implemented using multi-texturing modes available in the current PC graphics hardware like nVidia's boards. Our system is able to render up to sixteen half (SIF) resolution (360×240) videos at a time on a model of moderate complexity at 30Hz on a 1GHz PC with nVidia's GeForce4 Ti4600 board, while allowing the user to change the viewpoint of a virtual camera interactively. By switching between multiple sixteen camera videos, the system can provide live rendering from numerous cameras at interactive rates.

This paper presents an overview of the Flashlight system architecture and describes the online rendering algorithm. In order to enable seamless rendering of the videos, the system also allows solving for the 3D location and orientation (3D pose) of a camera either interactively or automatically. The real-time rendering can deal with both static and moving video cameras. However, for moving cameras, the pose estimation is not implemented in real-time. So with off-line camera pose estimation, moving camera videos also can be rendered in real-time. Since moving objects in videos are not part of the 3D site model, video textures corresponding to such objects are simply pasted on the 3D model as a part of rendering. The system is able to detect, track and reconstruct in 3D the moving objects using computer vision techniques. However, a full description of the aspects of the system dealing with moving objects is outside the scope of this workshop. For completeness sake, a brief description of how moving objects are handled is included in the paper.

2. Overview of the Video Flashlight System

Fig. 2 shows a functional diagram of the Video Flashlight System. In order to situate dynamic video data in the context of a model, first, a 3D model needs to be built for the static environment. The 3D-site model can be rapidly built using nadir and oblique aerial imagery, and interactive and automatic image processing tools [13, 14, 9]. A network of cameras is situated to provide dynamic coverage of the site. Each camera is located with respect to the model using a camera-to-model pose estimation module. For fixed cameras, the pose is estimated once, as a calibration step. For moving cameras (e.g. Pan/Tilt/Zoom and aerial cameras), pose needs to be estimated every frame.

The base level Video Flashlight Browser uses the 3D site model, and renders multiple videos from the cameras overlaid on the model in real time while the user navigates around the model. The rendering algorithms used in this browser are described in detail in the next section.

The base level system can be enhanced to render and visualize unmodeled moving objects that are not modeled as a part of the 3D site model. There are two ways in which the system handles moving objects. The first method uses 2D processing only to detect moving objects in videos. The detected and tracked moving objects can be overlaid as colored blobs on the rendered video flashlight textures. The 2D motion detection and rendering can be done in real-time too. The "Motion Detector Flashlight Browser" shows the moving object blobs. However, in the 2D processing mode, moving objects, such as people and vehicles, can appear distorted especially when viewed from a virtual viewpoint that is very different from the viewpoint of a flashlight camera. This is because un-modeled objects are simply rendered as textures on surfaces that are behind the objects in the flashlight camera viewpoint. In the second method for moving object processing, stereo cameras are required as flashlight cameras. Using stereo processing, moving objects are accurately delineated from the background. The delineated objects are either represented as 3D icons or stereo depth information is used to create simplified models. These 3D icons or models can then be rendered along with the site model.

3. Related Work

The focus of this paper is on model-centric live video visualization and the associated real-time hardware assisted rendering techniques. To the best of our knowledge, real-time rendering of multiple live video streams overlaid on a 3D model rendered from arbitrary virtual viewpoints has not been attempted in the past. The closest work that we came across is that of the MIT AI Lab. and Brigham and Women's Surgical Planning Lab. [8] on Image Guided Surgery. A live video of a patient undergoing surgery is manually registered using fiducials to an MRI and 3D model and the composite images are displayed.

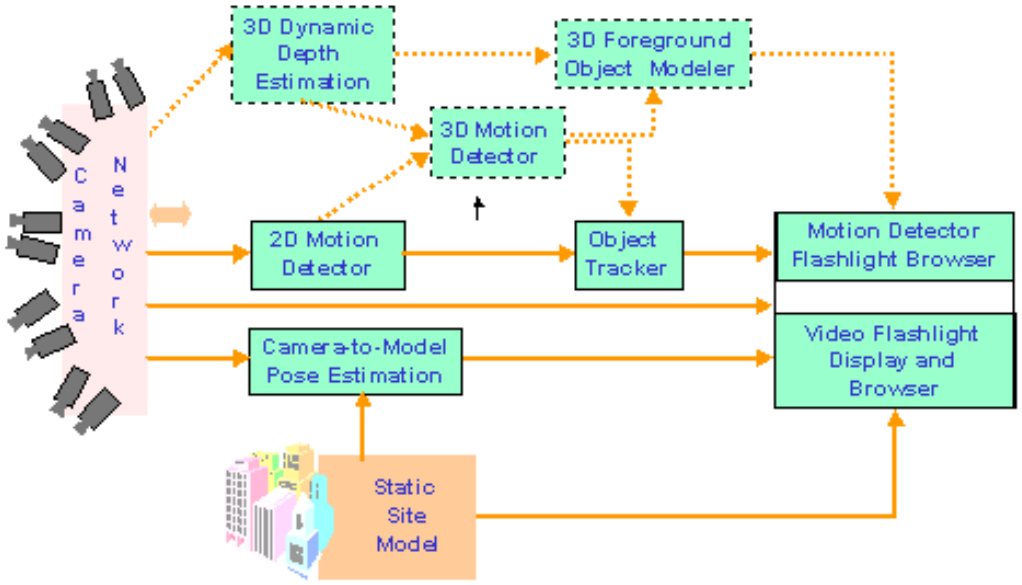

Figure 2: *A functional diagram of the Video Flashlight System.*

Shadow mapping techniques with hardware acceleration have been well known in the graphics rendering community [12, 4]. We have adapted the shadow mapping techniques to perform real-time rendering of live videos. In addition, the system captures multiple live videos and interfaces these with rendering and navigation to enable interactive browsing by users.

Image-based modeling and texture mapping have been used in the past few years to create realistic geometry and appearances of complex scenes. For buildings and site models, a number of systems have been developed [13, 14, 9]. Debevec et al. [2] use multiple images of a scene to perform view-dependent texture mapping with hardware acceleration and projective textures. However, the texture mapping is for off-line, static texturing purposes.

Moving object detection and 3D reconstruction from video and imagery is an active area of research in computer vision [10]. Matusik et al. [7] present a technique for real-time image based rendering of dynamic objects using visual hulls for implicit reconstruction. Our work employs known methods of object detection and reconstruction based on stereo cameras to enable model based moving object localization and visualization.

4. Video Flashlight Rendering Algorithm & Implementation

The Video Flashlight rendering algorithm generates an image at each time instant from a viewpoint, the virtual camera viewpoint, specified by the user through the browser interface. The image consists of pixels that show real video textures in regions that are illuminated by any of the real cameras, and pixels that contain background model textures for regions that are not illuminated by any camera. The background model textures are static and are represented as standard 3D graphics model representations along with the 3D vertices of the model triangles and their texture coordinates. The textures from the live video frames change at the video frame rate. The composite image created for any virtual viewpoint combines the two in the rendering implementation. A functional diagram of rendering using one video flashlight camera and the model is shown in Fig. 3. Multiple video streams are handled in an identical manner.

The projection of the video flashlight textures onto a virtual view cannot be achieved simply by projective texturing coupled with automatic texture coordinate generation because of two problems. First, regions that are hidden from the flashlight camera but are visible in the virtual view need to be detected so that flashlight texture projections in those regions can be eliminated. Second, since automatic texture coordinate generation provides two solutions for each projection ray (one each for the front and back of the camera), the projected textures behind a flashlight camera should be ignored for the virtual view. Our approach addresses the first problem using shadow mapping, and the second using clipping planes or special texture culling operations depending on the hardware support and performance.

Shadow mapping requires texturing, depth buffering, and arithmetic/logical operations at the texture and fragment lev-

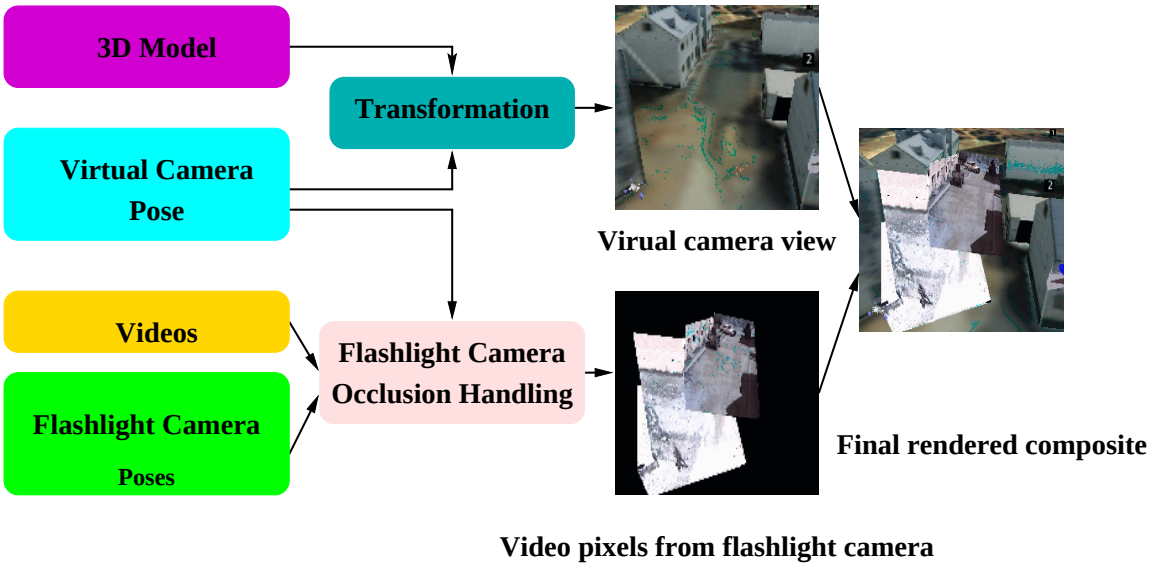

Figure 3: *A functional diagram of the flashlight rendering algorithm with one video camera.*

els. Current hardware enables shadow mapping in multiple ways. A generic approach, supported in most common platforms, employs projective textures, texture compositing and fragment testing without the need to rely on any restrictive hardware extension. Alternatively, dedicated extensions like SGI extensions, that are supported by various platforms, can be used. We have implemented the system using both the approaches.

Segal et al. [12] and Heidrich [4] describe a hardware assisted shadow mapping and rendering technique. We have adapted the shadow mapping technique for the purposes of Video Flashlight rendering. The light source used in the shadow map algorithm is replaced by a real video flashlight camera. The video texture for every frame is considered the "illuminant". Our generic approach implements shadow mapping by taking advantage of projective texturing, depth buffering, automatic texture generation, texture compositing and alpha testing. On platforms that support multi-texturing, like nVidia GeForce2-4 or ATI Radeon graphics cards, this method only takes one rendering pass for shadow map generation. It is equivalent to a dedicated extension of OpenGL.

The key idea behind the algorithm is shown schematically in Fig. 4. The algorithm can be subdivided into three stages:

1. Create the z-buffer for each camera by rendering the scene with color masking.
2. Render the scene from the virtual camera coordinate system.
3. In the virtual camera view, determine the pixels that are illuminated by the flashlight camera and replace them with the image data from the flashlight camera.

The third step outlined above resolves visibility of objects between the flashlight cameras and the virtual view. Fig. 4 schematically shows the typical relationship of scene objects with respect to the positions of one real flashlight camera and the viewpoint of the user-specified virtual camera. For illustration, two scene objects/surfaces, *O1* and *O2*, are shown in the figure. The virtual camera sees both objects *O1* and *O2*. However, the flashlight camera, sees object *O1* only; *O2* is occluded by *O1* from the viewpoint of the flashlight camera. Only *O1* is illuminated by textures from the flashlight camera whereas *O2* has the static model textures only. Hence, when rendering the virtual camera view, only the pixels corresponding to object *O1* that are illuminated by the flashlight camera and seen by the virtual camera need to be projected with the dynamic video textures. The occlusion handling between the flashlight cameras and the virtual viewpoint, and the rendering of the static model need to be done at video frame rate.

The key idea behind handling occlusions for video flashlights is to associate two depth values for each point in the scene. The first depth value corresponds to the z-buffer depth for the flashlight camera. Each scene point along a view ray from the flashlight camera is assigned a depth value corresponding to that ray's z-buffer value, like a projective texture. The second depth value is the true depth value for each scene point with respect to the flashlight camera. By comparing these two values for every point in the virtual view, it can be established whether the point is seen by the flashlight camera or is occluded. Points with the two depth values identical are seen by the flashlight camera and should be rendered with the flashlight video texture. Fig. 4 schematically depicts this technique.

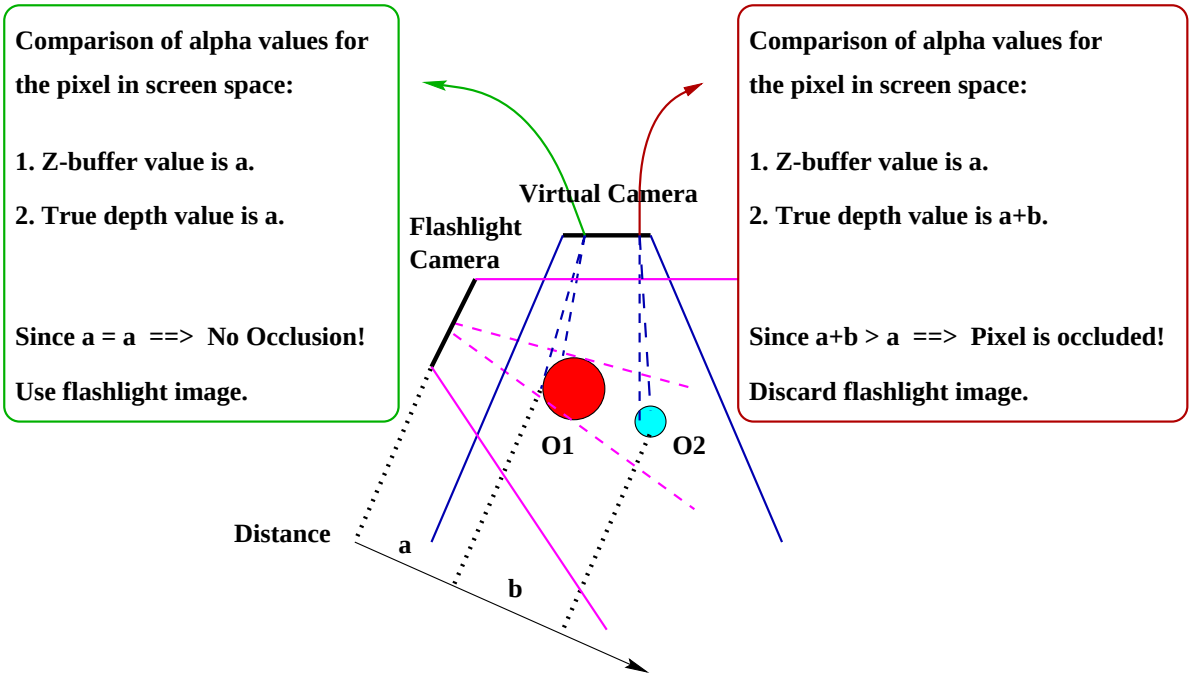

Figure 4: *A schematic depicting occlusion handling for rendering a video flashlight camera and the model from a virtual viewpoint.*

The occlusion handling technique in our generic approach is implemented as a multi-pass rendering algorithm. The scene is first rendered in the flashlight camera coordinate system to obtain the z-buffer. The z-buffer is stored in the alpha channel of the texture used for flashlight video projection. This is the first alpha value. In other words, the first alpha value for each scene point represents the depth of the first hit for a view ray from that point to the flashlight camera center (the depth *a* in Fig. 4). The second rendering uses automatic texture coordinate generation to set the texture coordinate for each vertex to the true depth value for the vertex with respect to the flashlight camera. A 1–dimensional ramp-texture is used to define a mapping of this depth to alpha values. This is the second alpha value. In other words, the second alpha value represents the true depth for each scene vertex in the flashlight camera coordinates. Therefore, for all points visible from the virtual camera, we have two alpha values that need to be compared in order to do determine which points are illuminated by the flashlight camera. Pixels that satisfy the equality test for the alpha values are illuminated by the flashlight camera and are rendered with the flashlight camera texture while the rest are rendered with the model texture.

A pseudo-code for the rendering algorithm follows:

```
Display {
  for all visible flashlight cameras {
   if (play)
     UpdateVideoContent(Video Source, Frame Number);
   if (moving)
     UpdateDepthMap();
   else
     UpdateDepthMapOnce();
   }
  SetupViewport(ScreenResolution);
  ClearScreenColorBuffer();
  ClearScreenDepthBuffer();
  MultMatrix(Inverse(Virtual Camera Pose));
  RenderScene(Geometry+Textures);
  for all visible flashlight cameras
   ExtractVisiblePixels();
}

UpdateDepthMap {
  SetupViewport(DepthMapResolution);
  ClearDepthBuffer();
  MultMatrix(Inverse(Camera Pose));
  // Decrease precision error by offseting the geometry
  SetPolygonOffset;
  MaskColors; // Only need z-buffer
  RenderScene(Geometry);
  ReadDepthBuffer();
  TransferDepthToAlphaChannel(VideoTextureRGBA);
}

ExtractVisiblePixels {
  SetupTexture1();
```

```
  BindTexture(RampTexture);
  // ZToTextureCoordinateMatrix extracts
  // Z-component
  SetupEyeLinearTextureCoordinateGeneration(
    ZToTextureCoordinateMatrix*Inverse(Camera Pose));
  SetupTextureEnvironment(UseTextureAlpha);
  SetupTexture2();
  BindTexture(VideoTextureRGBA);
  SetupEyeLinearTextureCoordinateGeneration(
    Inverse(Camera Pose));
  SetupTextureEnvironment(
    SubtractPreviousTextureAlpha,UseTextureRGB);
  EnableAlphaTesting();
  SetAlphaFunc();
  // Avoid back projection
  SetupCullingPlane(CameraImagePlane);
  RenderScene(Geometry);
}
```

In the algorithm, first the video content and depth maps are updated on an as–needed basis. Depth map textures are obtained by rendering the geometry in either the frame–buffer or in the p–buffer. During this process, polygon offsetting is required in order to avoid re-sampling and precision errors. Once all the textures are updated, the scene is rendered with all the static model (background) textures. Then for all the visible flashlight cameras, two textures are projected using the corresponding camera pose matrices. The first texture uses an extra matrix operation (*ZToTextureCoordinateMatrix*) to map the Z values of all the scene points in the frustum of a flashlight camera in the camera coordinates to the s coordinate in the texture coordinates. Since we do the occlusion calculation in the texture domain, this value is mapped to texture values using a 1–D ramp texture. The Z-values corresponding to the scene points behind the camera (hence not visible in the video) are culled by specifying the camera image plane as the culling plane. The second texture has the z-buffer depth map in the alpha channel and the video content in the RGB channel. The texture environment is set such that the alpha values are subtracted from the previous texture's alpha, which effectively implements the occlusion test depicted in Fig. 4. This test is actually done on a per-pixel basis by alpha testing while rendering.

Shadow map extensions can be thought of as special texture filters that enable some of the calculations mentioned above in the generic approach. In order to use these extensions, the z-buffer created in the first step is transferred to a shadow map texture. The output values, usually 0 or 1, generated by this texture, flag the pixels as shadowed or illuminated, and can be used for rendering the final pixels. When the automatic texture generation in the flashlight camera coordinates is turned on, for a computed texture coordinate, (s,t,r,q), $(s/q,t/q)$ points to the z-buffer value, and r/q represents the true depth. Implicit comparison of the z-buffer value with the true depth enables occlusion handling in the shadow map extensions. Recall that r/q corresponds to the 1–D ramp texture, and the comparison operation corresponds to the alpha test in the generic approach presented above.

4.1. Flashlight Rendering Results

Fig. 5 shows virtual views with flashlight camera rendering. The top panel in the figure shows a rendering of the textured model of a site from a virtual viewpoint. The location and orientation of seven video cameras located at the site is shown using red camera icons. The bottom panel shows a virtual view with the rendered textures from the seven flashlight cameras overlaid on the textured model rendering. Figs. 6 and 7 show close-up views of the rendering from two flashlight cameras overlaid on the model, the former with no blending between the rendered video frames and the latter with blending. Note that the background texture is not blended with the video frames since one purpose of the flashlight display is to distinguish regions of the scene illuminated by the cameras from the rest of the scene. Furthermore, the display of blended versus non-blended video frames may also be provided as a user selectable option since a user might want to visualize the boundaries between the projected video frames. The blending is currently done using a pyramid based technique [1, 11]. We are investigating how mip-mapping can be combined with selection and fusion operations to enable the pyramid based blending in hardware.

4.2. Camera Pose Estimation

Rendering the video flashlight cameras over the model requires the knowledge of the location, orientation and intrinsic parameters of the video cameras. We have developed an interactive tool in which a user specifies point and line correspondences between a video frame and a 3D rendering of the model. The system automatically uses photogrammetry [3] to solve for all the camera parameters.

The video flashlight system can also handle moving cameras. In the case of moving cameras, the pose estimation needs to be done continuously over the video to obtain the camera pose at every frame instant. We employ the algorithm developed by Hsu et al. [5] to compute moving camera poses. The system is initialized manually in one frame of the video. Subsequently, the pose estimation continuously tracks the cameras using the previous frame's pose as initialization and by refining the pose estimate for every new frame. The refinement is done by iteratively aligning strong linear features in a video frame to lines defined by intersection of planes in the model. The existing site model is considered to be a collection of untextured polygonal faces. Face edges in the given model imply discontinuities in surface normal and/or material properties in the actual 3D scene, which generally induce brightness edges in the image. The alignment method selects 3D line segments from the model and projects them onto the video frame using the current pose estimate. The local edge strength in the image itself is represented as an

Figure 5: *Video Flashlight Rendering:* ***Top****: A rendering of a textured model of a site with the location and orientation of video flashlight cameras shown in red.* ***Bottom****: Rendered view of the textured model and flashlight camera textures for 7 cameras rendered from a virtual viewpoint. The rendered view shows the effect of "illuminating" the scene with live cameras by "switching" on the cameras. The switched on cameras are shown in blue.*

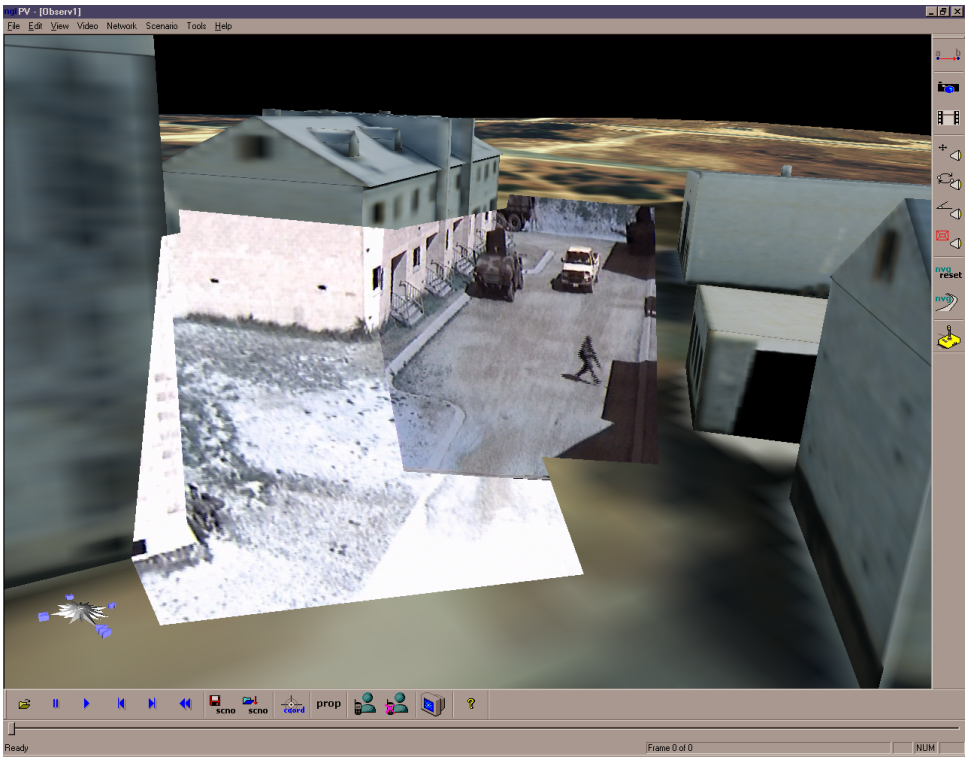

Figure 6: *A close-up virtual view from ground level in which two flashlight cameras are rendered* ***without blending****.*

oriented energy field. Model lines are projected to the orientation image which is nearest their orientation. Fig. 8 shows the video to model alignment inputs and outputs.

Fig. 9 shows flashlight rendering of two frames from a video captured using a camera mounted on an autonomous helicopter. The pose estimation of the video frames was done off-line. Using the estimated poses, the flashlight rendering algorithm was used to render the video frames over the model. The figure also shows a moving object (running person) circled in orange color. This illustrates the idea that even from a moving camera, using flashlight pose estimation and rendering, a moving object can be localized and visualized in the context of a 3D model.

5. Moving Object Processing & Visualization

Moving object detection and tracking is used to enhance the visualization of moving objects in the flashlight browser. The flashlight rendering algorithm described above will render unmodeled moving objects as textures on background surfaces. By detecting the moving objects, the user can be alerted to their presence and they can be visualized as colored moving blobs or icons.

Fig. 10 shows a method of detecting foreground objects using 2D and 3D Moving Object Detection. The first step is to create a reference background image for each of the videos. One method of performing this is presented in Irani et.al. [6] where a background reference image is constructed by taking the median of a stack of images. The result is a reference image that does not contain any foreground objects. This is shown at the bottom left of Fig. 10. The reference image can then be subtracted from the current video image. The absolute value at each point can then be computed, and the result can be thresholded in order to highlight intensity or feature differences between the current video image and the reference image. The final result is shown in the image in the middle and to the top of Fig. 10. Differences are shown in yellow. Note that differences due to the vehicle and people have been correctly detected. These positions in the image can be reported to the browser and rendering module and displayed on the screen. The reference background image needs to be constantly updated during the day to reflect changing ambient illumination. Alternatively only moving objects may be detected by comparing current image with an image taken a few seconds before. The 2D moving object detection can be done in real time.

The advantage of 2D moving object detection is that only one camera is required. The disadvantage is that objects such as shadows are incorrectly detected as being foreground objects. These errors are corrected using 3D moving object

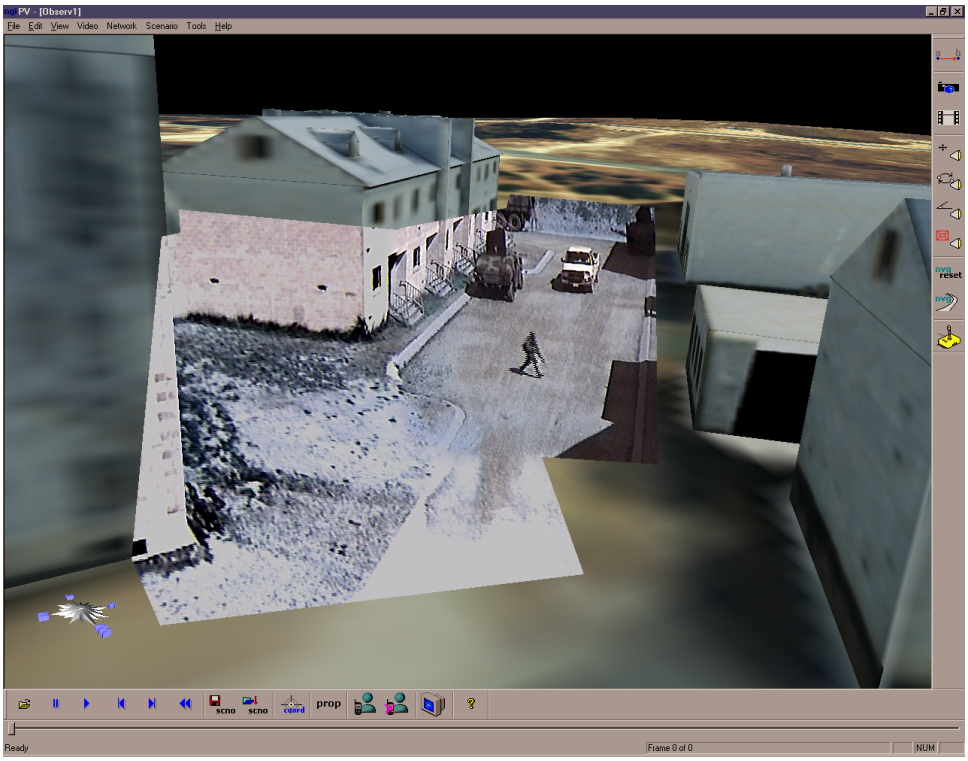

Figure 7: *A close-up virtual view from ground level in which two flashlight cameras are rendered* ***with blending****.*

detection as shown in the right of Fig. 10. 3D moving object detection begins by recovering a depth or shape map of the background scene where there are no foreground objects. This can be recovered either directly from the static site model, or it can be recovered from a range-finder device, or it can be recovered by performing depth estimation using two or more cameras when it is known that there are no foreground objects in the scene. The result is a background depth or shape map, shown at the bottom right of Fig. 10. The image is coded such that brighter intensities represent closer objects, and darker intensities represent further objects. Depth or shape recovery is then performed on the current video imagery using stereo cameras . The depth or shape of the current scene is then subtracted from the depth or shape of the background scene. The absolute value of the difference at each point can then be computed, and the result can be thresholded in order to localize areas in the scene that have a different shape from the background. Since shadows are projected onto the background scene, they are identified as belonging to the background shape or depth, and are not identified as foreground objects. A further advantage of 3D over 2D is that the depth or shape of the object is recovered. This can be used in 3D modeling of dynamic objects.

Once the foreground objects have been segmented, either in 2D or 3D, blobs or icons can be used to render the objects with the 3D model. Fig. 11 shows multiple moving objects detected in multiple videos rendered as blobs from a virtual viewpoint along with the video textures and the model. Fig. 12 shows a similar rendering but with 3D icons and object tracks representing people and vehicles.

6. Conclusions

We have presented the concept of Video Flashlights and described a system that integrates live video images with 3D models for augmented reality applications. Seamless integration and visualization of 3D models and live videos is achieved by adapting shadow mapping algorithms to video flashlight rendering using commodity PC graphics hardware. Current performance enables up to 16 half resolution videos to be rendered in real-time. Furthermore, we also presented methods that will allow detection, representation and visualization of moving objects along with the video and model textures. We are currently designing a scaleable system architecture that will extend the current system to handle numerous video inputs by a combination of multiple rendering systems like the one presented here, and view-dependent selection of flashlight cameras.

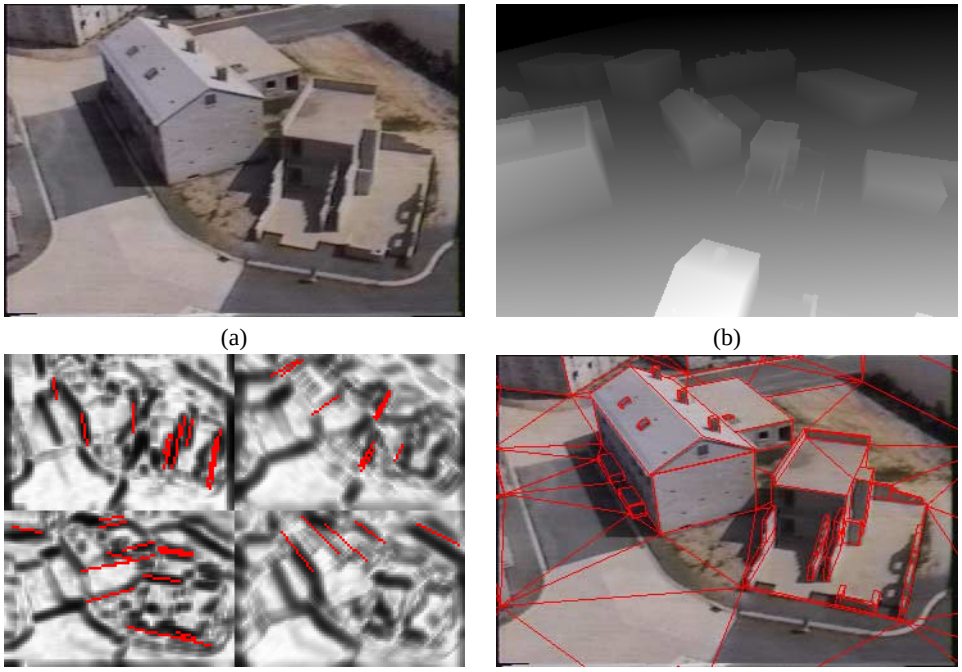

Figure 8: *Alignment of video to site models: (a) Original video frame, (b) 3D site model, (c) Oriented energy image with 4 major orientations: 0, 45, 90 & 135 degrees; Model lines used for alignment are overlaid on top in red. (d) Model lines are projected on to the image using estimated pose.*

Figure 9: *Flashlight rendering of two frames of a video captured from a moving aerial autonoumous helicopter.*

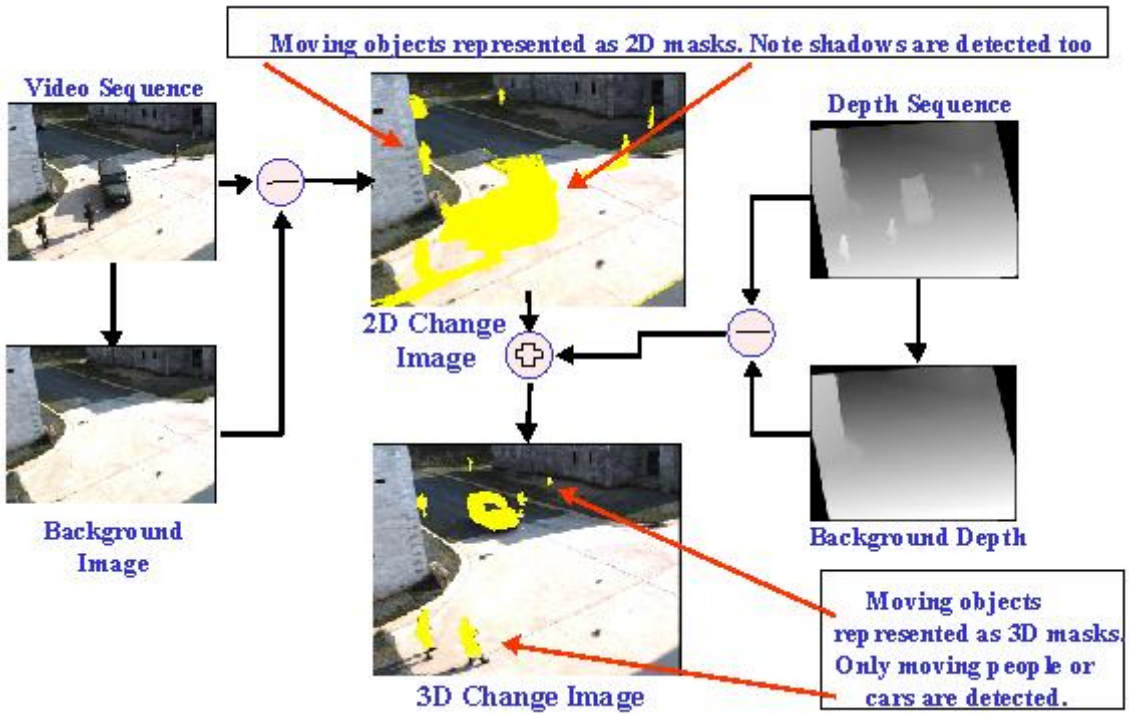

Figure 10: *A schematic depicting moving object detection using 2D and 3D video processing.*

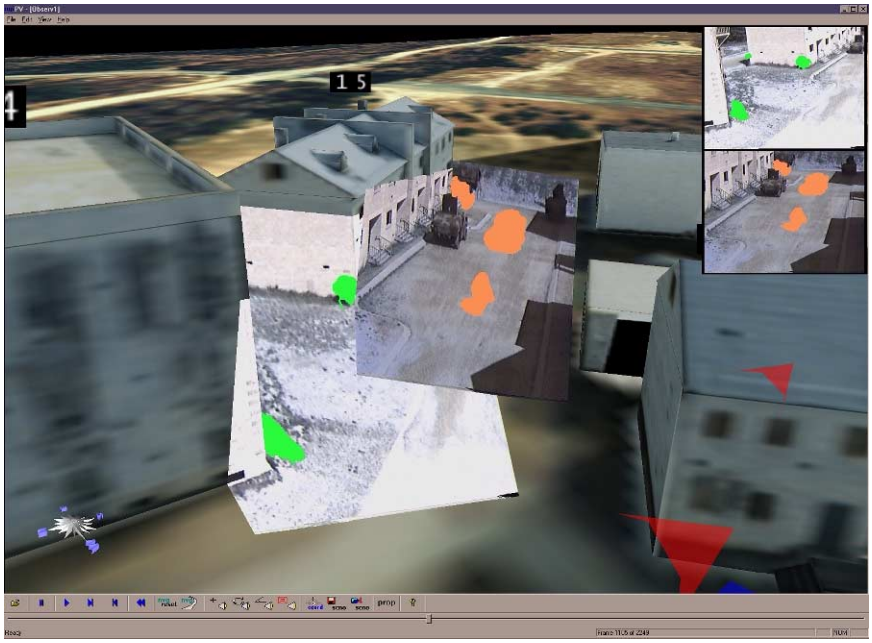

Figure 11: *Detected moving objects in two cameras represented as 2D blobs rendered with the flashlight browser. Color of each blob represents the camera in which it was detected.*

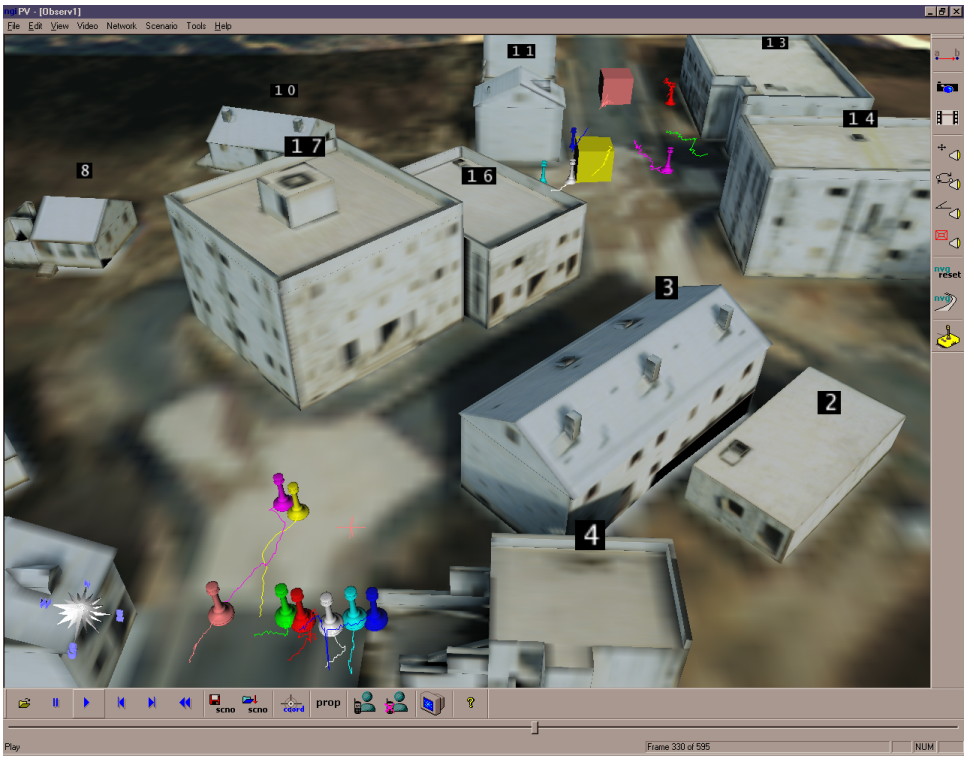

Figure 12: *Moving objects detected in 3D and rendered as icons representing people and vehicles, rendered with the flashlight browser. Each object is color-coded with a unique color and its track over time is also shown.*

Acknowledgments

We are grateful to the anonymous reviewers whose comments helped us greatly improve this presentation. This work has been supported in part by the Air Force Research Laboratory under Contract number F30602-00-C-0143.

References

[1] P. J. Burt and E. H. Adelson. A multiresolution spline with applications to image mosaics. *ACM transactions on Graphics*, 2(4):217–236, 1983.

[2] Paul E. Debevec, Yizhou Yu, and George D. Borshukov. Efficient view-dependent image-based rendering with projective texture-mapping. In *Eurographics Rendering Workshop 1998, Vienna, Austria, Edited by George Drettakis and Nelson Max*, pages 105–116, 1998.

[3] O. D. Faugeras. Three-dimensional computer vision: A geometric viewpoint. 1993.

[4] Wolfgang Heidrich. *High-quality Shading and Lighting for Hardware-accelerated Rendering*. PhD thesis, University of Erlangen, 1999.

[5] S. Hsu, S. Samarasekera, R. Kumar, and H.S. Sawhney. Pose estimation, model refinement, and enhanced visualization using video. In *CVPR00*, pages I:488–495, 2000.

[6] M. Irani, B. Rousso, and S. Peleg. Computing occluding and transparent motions. *International Journal of Computer Vision*, 12:5–16, 1994.

[7] Wojciech Matusik, Chris Buehler, and Leonard McMillan. Polyhedral visual hulls for real-time rendering. In *Proc. 12th Eurographics Workshop on Rendering*, pages 115–125, 2001.

[8] Project on Image Guided Surgery. http://www.ai.mit.edu/projects/medical-vision/surgery/surgical_navigation.html.

[9] PhotoModeler. http://www.photomodeler.com.

[10] Keith Price. Computer vision online bibliography. http://iris.usc.edu/Vision-Notes/bibliography/contents.html.

[11] H.S. Sawhney, R. Kumar, G. Gendel, J. Bergen, D. Dixon, and V. Paragano. Videobrush: Experiences with consumer video mosaicing. In *WACV98*, 1998.

[12] Marc Segal, Carl Korobkin, et al. Fast shadow and lighting effects using texture mapping. *SIGGRAPH*, pages 249–252, July 1992.

[13] General Dynamic Information Systems GLMX System, 1993. See also Rapid Generation and Use of 3D Site Models to Aid Imagery Analysts/Systems Performing Image Exploitation. Proc. of the SPIE, Vol. 1944, Conference on Integrating Photogrammetric Techniques with Scene Analysis and Machine Vision, April 1993.

[14] C.J. Taylor, P.E. Debevec, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. *SIGGGRAPH*, pages 11–20, August 1996.