Spatially Coherent Visualization of Image Detection Results using Video Textures

Volker Settgast[1], Marcel Lancelle[1], Sven Havemann[1], Dieter Fellner[1,2]

[1] Institute of Computer Graphics and Knowledge Visualization
Graz University of Technology, Austria
{*v.settgast,m.lancelle,s.havemann,d.fellner*}*@cgv.tugraz.at*

[2] Fraunhofer Institute for Computer Graphics Research and
Darmstadt Technical University, Germany

Abstract
Camera-based object detection and tracking are image processing tasks that typically do not take 3D information into account. Spatial relations, however, are sometimes crucial to judge the correctness or importance of detection and tracking results. Especially in applications with a large number of image processing tasks running in parallel, traditional methods of presenting detection results do not scale. In such cases it can be very useful to transform the detection results back into their common 3D space. We present a computer graphics system that is capable of showing a large number of detection results in real-time, using different levels of abstraction, on various hardware configurations. As example application we demonstrate our system with a surveillance task involving eight cameras.

1 Introduction

In the image processing research community the common way of presenting the results of, for example, a new object detection algorithm is to show grayscale images with colored rectangles around the detected objects. This approach permits to compare different algorithms by running them on the same set of benchmark images or videos. Both false positives and false negatives are easy to spot and to compare using this approach.

However, in real-world applications this might not be the best way to visualize the results of detection or tracking algorithms. While researchers naturally focus on the methods and algorithms, users are typically only concerned with the meaning of a detection event. In some cases they are interested in specific events or locations, while at other times a more ambient impression may be sufficient. The visualization must be scalable, from individuals to groups and to flows, as well as from concrete video pixels to rectangles to more abstract motion patterns.

A detailed 3D model overlaid with full video and detection information from multiple camera sources can still be remarkably irritating for a human operator. To realize and manage different modes of information presentation are a challenge not only from a computer graphics point of view, as image synthesis problem, but also a comfortable user interface is a not easy to realize – let alone the problem of intuitive 3D navigation. In principle different options exist to obtain a coherent integrated

information space, as outlined in the next section. We propose to use a scriptable scene graph engine that already has much of the required functionality built in. In this paper we describe the basic system, as well as the enhancements we needed to develop in order to accommodate video textures and to integrate the image processing information. Finally we demonstrate what needs to be done to realize an example application, a surveillance task of a building complex using eight cameras.

2 Related Work

In principle a whole number of different options exist to realize the vision of an integrated 3D information space as an infrastructure for computer vision researchers to visualize the results of image processing algorithms, such as object detection, object recognition, and object tracking.

The first, most straightforward option is to use a standard low-level 3D API such as OpenGL or DirectX to directly code video billboards, to simply replace the colored 2D rectangles by textured 3D quads. Although absolutely feasible, this approach has little sustainability as soon as more and more standard geometry (modeled buildings, materials, interaction components) are to be integrated to achieve a more realistic result. So this approach is not likely to result in a flexible, sustainable, easy-to-use infrastructure.

The next option is to use a standard game engine. This is absolutely feasible as well. However, the drawback may be that game engines are typically highly optimized to guarantee a fluent game play. An open architecture for easy extensibility is often less of a concern, which impedes the flexibility.

Consequently, the resulting option is to use a scene graph engine. Our solution is based on the open source scene graph system OpenSG by Reiners et al. [10] in combination with the Generative Modeling Language (GML) by Havemann [5]. Our visualization framework is similar to a presentation tool for cultural heritage proposed in [6].

The general problem of visualizing image recognition results in 3D is not new, of course. Hall et al. present an integrated visualization of outdoor scenarios and live video images [4]. Although they have a good integration of multiple remote video cameras their system is restricted to flat outdoor scenes.

Impressive approaches that are very similar to our surveillance application were presented by Neumann et al. [7], [13] and Sawhney et al. [12]. Neumann presents a system that implements projecting images of cameras. Sawhney describes a system called Video Flashlight for projecting videos of static and moving cameras onto a 3D model. It also uses a shadow mechanism on the graphics hardware and is able to use smooth blending between two images that partially overlap. However, these systems are aiming at the specific surveillance setting. The goal is not so much to provide a general infrastructure.

Fleck et al. present a distributed network of smart cameras for real-time tracking [2]. Similar to our solution, they integrate 2D detection results into a 3D scene as billboards. 3D models of the observed area are acquired as point clouds. Their system allows large scale scenarios with hundreds of cameras. However they only show the detected objects as billboards but do not project the whole video image. In areas of low point density their visualization exhibits disturbing holes.

The implementation of our video texture projection is based on a combination of projective texture mapping [14] and depth map shadows [9].

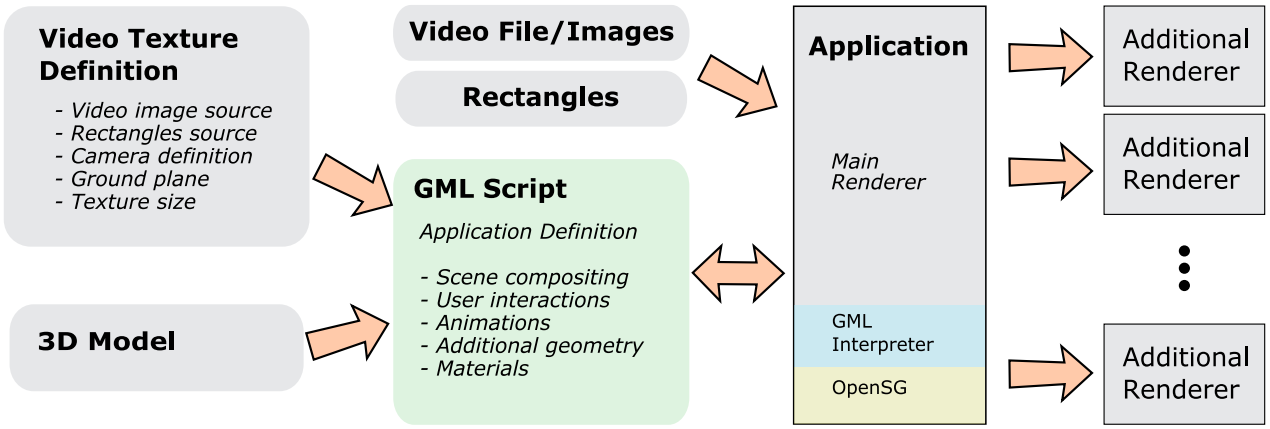

Figure 1: Overview of the visualization system. The Generate Modeling Language (GML) script defines all of the main aspects of the application. It takes care of the scene initialization, animations and user interactions. The definition of the video input data is stored in a separate location for each video texture.

For our example application we tried different techniques for the visualization of buildings. Among others Wang et al. show methods to visualize multiple floors simultaneously, e.g. floors of a building can slide apart so that they do not occlude each other [16].

Automatic detection and classification of moving objects with multiple surveillance cameras was presented by Collins et al. [1] We use the output of the person detector presented by Roth et al. [11].

3 System Overview and Input Data

Our proposed visualization system is a combination of a general OpenSG application and a script defining the scenario specific parameters as proposed by Ousterhout in [8]. The input data consists of a set of videos, detection results and a 3D model of the observed area. Figure 1 shows a schematic overview of the system. The main application can connect to multiple OpenSG render servers to form a visualization cluster. In this way it is possible to realize a tiled display or a CAVE like immersive screen setup.

3.1 Video Input Data

The definition of a video source is encapsulated in a simple XML configuration file. Each camera is defined in a separate file. A typical configuration file is shown in Figure 2.

Video images can be stored in video files or delivered via network from one or multiple custom video servers. A network connection is defined by the server name, a port number and the server ID. Local video files are defined by a file name with a path relative to the configuration file location.

The camera frustum is described by *projectionDistance* and *aspectRatio*. The projection distance refers to the distance of the camera while capturing a one meter wide object filling the whole width of the image. Lens distortions are ignored in the current implementation. The position and direction of the camera as well as the up vector are defined as float triples. A ground plane is defined in the same way by position and normal. The plane will be used to place the detections in 3D which is described later on. All of the position definitions refer to the coordinate system of the 3D model.

```
...
  <!-- the_camera -->
  <camera position = "24.0,32.3,6.1" direction = "0.79,0.28,-0.54"
          up = "0.4573,0.3078,0.8343"
          projectionDistance = "0.86" aspectRatio = "1.333"/>
  <!-- the_geometry -->
  <geometry groundp = "0, 0, 0.36" groundn = "0, 0, 1"
            humanheight = "1.85" rectangles = "detections/085.log" />
  <!-- video source -->
  <server videofile = "085.avi"
          serverAdress = "localhost" serverPort = "5678" serverID = "1" />
  <texture width = "512" height = "256" />
```

Figure 2: Example configuration file for a video source. The position of the camera, the ground plane and the human height correspond to the coordinate system of the 3D model.

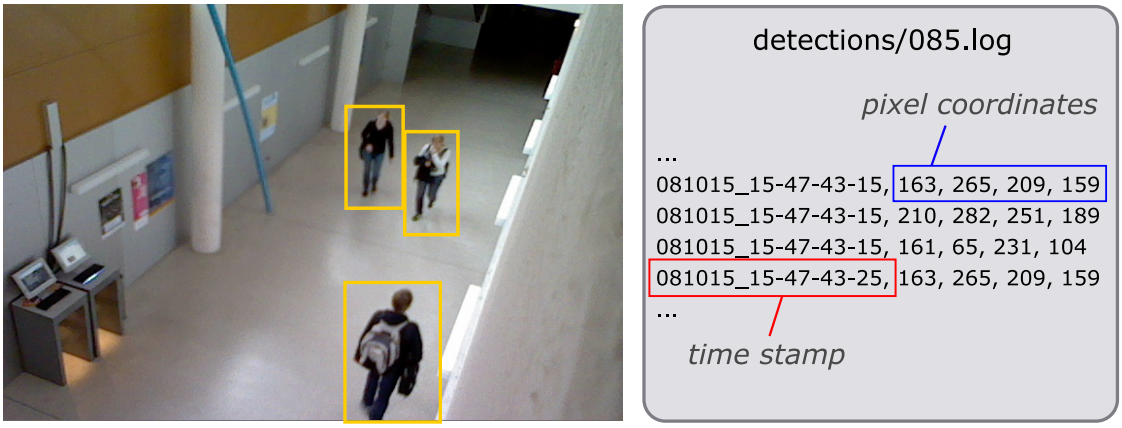

Figure 3: A typical area definition of detections in a video frame. In this example, three persons are detected. The rectangles are stored in a simple text file together with a time stamp as a reference to the video frame.

The resolution of the input video is arbitrary. In the configuration file a different resolution can be defined to deal with hardware restrictions or counteract performance issues. Depending on the graphics hardware it may be necessary to choose a power-of-two width and height.

3.2 Detection Data

2D detections are defined as regions in the video frame. For now our solution supports image aligned rectangular areas defined by two 2D points. This is the most common definition of regions in an image. In principle our system allows to define an arbitrary number of detections for one video frame.

Similar to the video frames, the rectangle coordinates can be read from a file or they are sent via network. The file format for the detection and tracking results is very simple. It basically is a text file with pixel coordinates of rectangles and a corresponding time stamp, as shown in Figure 3. Time stamps are used to connect the rectangles to their video frame. We also support additional data like IDs and certainty values for each rectangle.

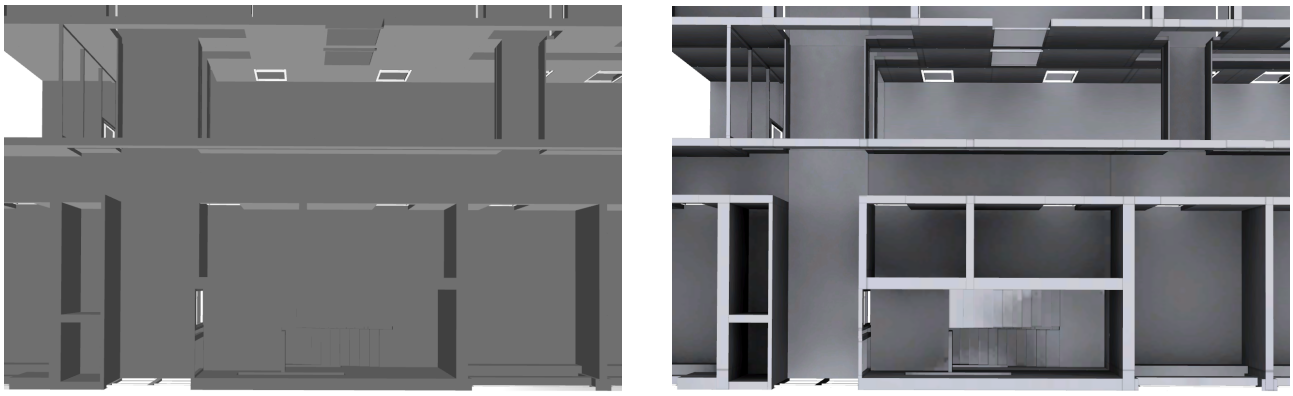

Figure 4: Renderning of the same 3D model with OpenGL lighting (left) and with advanced lighting calculations (right). The structure is recognized easier with advanced lights. Lighting can be precalculated (baked) to the texture.

3.3 3D Model Preparation

A 3D model of the scenario is needed to create an integrated visualization of the image detections. Geometric accuracy of the area observed by cameras is essential for a matching video back projection. OpenSG offers loaders for the most common 3D file formats including OBJ, VRML and 3DS. But more important, the GML is able to create generative models and geometry. For our example application we generated a building complex with GML based shape grammars. That way it was possible to add semantic information directly in the construction step.

It is necessary to address predefined parts of the model in the application script. For example a selective display of a single floor is only possible, if the geometry parts of that floor can be identified. This is done by giving unique names to nodes and sub graphs of the scene.

The straight forward way of rendering a 3D model using OpenGL lighting leads to artificial and sometimes irritating results. Buildings in particular have many planar surfaces, a fact that amplifies this effect (see Fig. 4). Advanced lighting techniques can create a more realistic and less confusing look but need a lot of performance if calculated in real time. We recommend using professional render software like Autodesk Maya^{TM}to bake the illumination. Baking in this context means to calculate the lighting effects in advance and store the data into a texture. OpenSG supports texture compression which should be used for high quality visualizations as the amount of textures is quite big.

4 Technical Details of the Video Textures

To display a video in the 3D scene we need all of the information in the configuration file described in Section 3.1. Since we use a single texture unit for each video, there is no upper limit for the number of projected images. As described the image of an input camera and the detection rectangles can be received via network or read from file. The image is converted into a texture with the target resolution which is then used in a special material. For the rectangles we prepare some simple geometry and apply the video texture material. The rectangles will be hidden until some of them are used to display detections.

Figure 5: Multiple video textures are added to the scene by duplicating the geometry. The left image shows the output of the video texture shaders which have discarded all pixels outside of the video image. In the right image the video textures are combined with the original scene geometry.

4.1 Projective Texturing

Our approach to project the video onto the 3D scene uses the GL shading language (GLSL) which is attached to a material. The material has to be applied to the scene, at least to the parts which are visible in the video. The GLSL shader gets the camera parameters and the video image in form of a texture as inputs. In the fragment shader we use the camera parameters to project the corresponding 3D world position of each pixel into the 2D video image. If the shader calculation discovers a position outside of the video image space, it discards the pixel. As the shaders knows the normal of the rendered pixel it can also discard back facing fragments.

By now this would result in a selective scene showing only geometry within the view frustum of the video camera (see Fig. 5). To compose the video texture with the original scene, a duplicate is generated and rendered in addition. This allows combining multiple video textures with the texture baked 3D scene even if the video areas are overlapping.

So far the video image would penetrate the whole 3D model within the camera frustum (see Fig. 6, left image). We only want to project the images onto the visible surfaces from the camera position (Fig. 6, right image). A slightly modified shadow algorithm is used to detect occluded areas. The visualization uses an additional preceding render pass generating a depth map for each surveillance camera. This depth map has to be updated only if the camera description changes. It represents the first hit of the projection rays from the camera. In the final render step the distance between the camera and the currently rendered pixel is compared to the first hit in the depth map. Pixels behind the first hit are also discarded. A small offset is used to avoid artifacts due to numerical errors.

An important aspect is the modulation of the transparency of pixels. We included a fade effect for pixels which are far away from the camera. To reduce the distraction from distorted textures the projected image is also faded to transparent when viewed from an angle very different to the viewing direction of the camera.

4.2 Visualization of Detected Objects

The detection areas of the video image are added into the 3D scene as planar rectangles. These billboards are oriented to the respective video camera position and not to the viewer. It is assumed that detected objects are always connected to the ground plane that was defined in the configuration

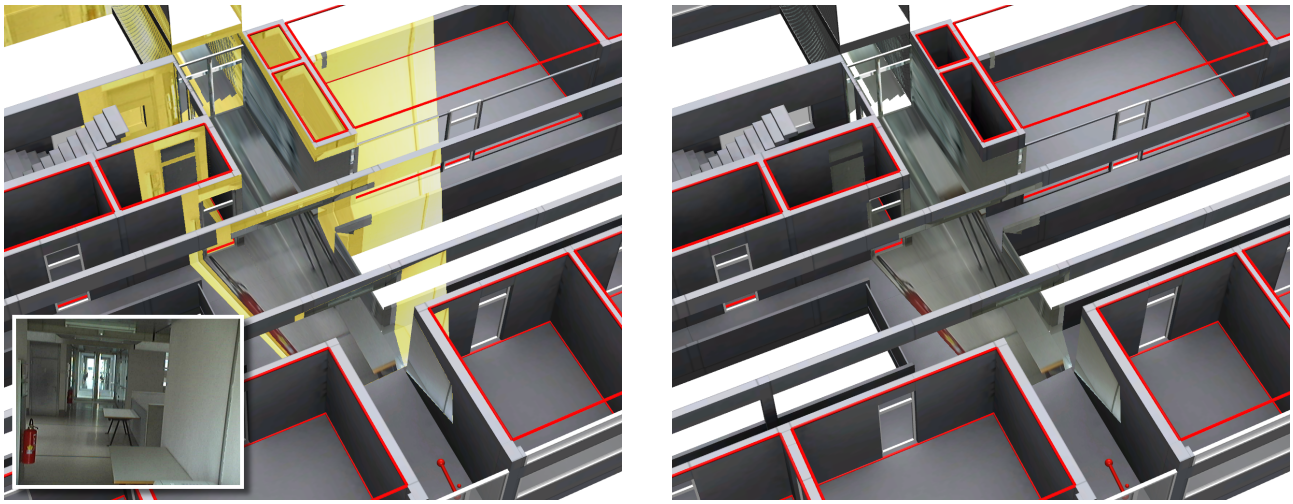

Figure 6: The video image is projected back into the 3D model. The yellow areas in the left Figure are in the camera frustum but occluded by closer objects. Thus they are not visible from the camera position. A modified shadow algorithm is used to remove the video image in these areas.

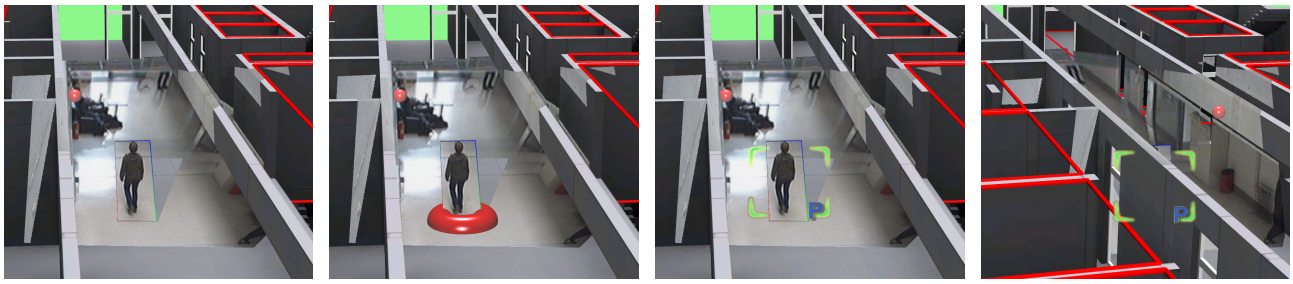

Figure 7: Different representations of a person detection in 3D: (a) Billboard with a thin colored surrounding line, (b) additional geometry, (c, d) overlay marker which is not occluded by the scene.

file for the video texture. From the position of the camera and with the parameters of the frustum we calculate a ray through the lower edge of the rectangle. By cutting the ground plane with this ray we get the position of the rectangle in 3D space. For some scenarios it may be necessary to use the upper edge of the rectangle because the lower part of the detection may be occluded in the video. In this case a default height of objects is used to estimate the position of the rectangle. The rectangles are placed right-angled to the ground plane.

The first implementation of the billboard technique revealed some drawbacks: 2D billboards are hardly visible from the side or from a top view position. Also in more complex 3D scenes the rectangles may be occluded by walls and other elements. To improve the visibility we added the possibility to connect GML generated geometry with the billboards. This includes shapes like spheres or boxes but also textual information. Overlaid renderings which are always visible help to signalize detections even if the rectangle is occluded by parts of the scene. See Figure 7 for some examples of enhanced billboard visualizations.

5 An Example Application

The presented rendering system was used to create a surveillance application for a small building complex. The application was installed on a tiled display with four render machines (see Fig. 9).

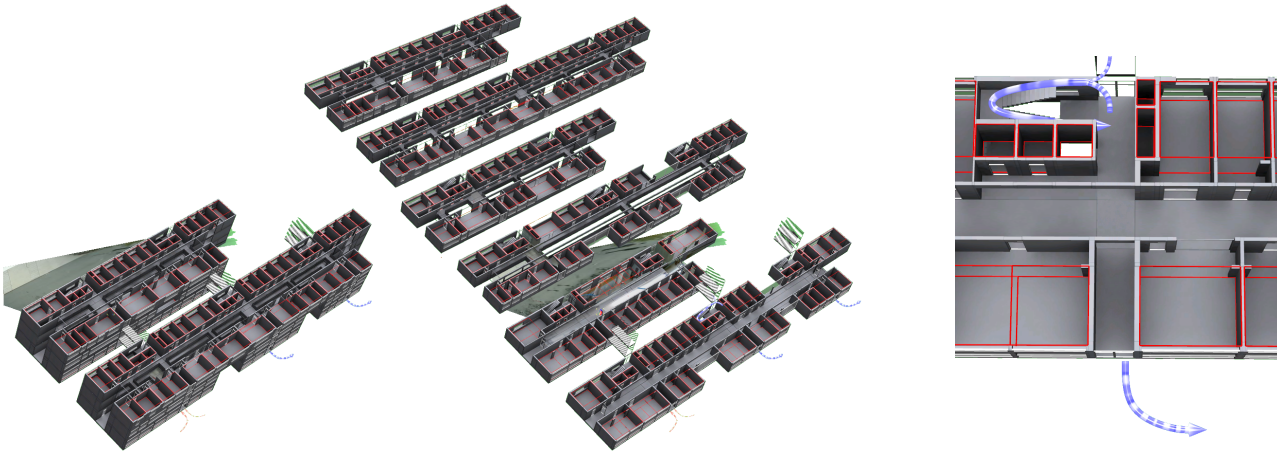

Figure 8: The example application with integrated video textures projected onto the 3D model. The floors can be hidden or slide apart like shown in the middle. Additional information can be added, for example 3D arrows visualizing main motion directions.

We used the video recordings of eight static cameras and the results of the person detector by Roth et al. Seven of the cameras were standing inside the buildings and one was located outside pointing to one of the entrances. The scenario consists of four buildings with three floors and a basement. The camera registration to the 3D model was done manually.

On the basis of floor plans we generated a model of the four buildings using the GML shape grammar. Because of many similarities the creation was modularized and a quantitative model containing all the rooms could be written in a short time. Manual measurement results were added in the camera areas to reach accuracy in the range of a centimeter. With the shape grammar it was possible to add a hierarchy with a naming scheme and additional information like decorations for rooms and doors. Also segmentation into buildings, floors and smaller subparts is directly integrated into the shape grammar. Finally we used the VRML file format to export the model to Autodesk Maya. Global illumination was calculated and baked for each floor. For a nice appearance we added some additional buildings of the surrounding area as transparent boxes.

5.1 The Application Script

The GML script for the application declaration is organized in two main parts: the initialization and the user interactions part. In the initialization part the 3D scenario is generated. The external 3D models are loaded and added to the scene graph. A ground plane is created directly with GML code.

After the 3D scene is ready the virtual video cameras are initialized. A subset of the surrounding geometry is duplicated for each camera. In general it would be possible to make a copy of the whole scene but we only need the parts which are within the camera's field of view. As we segmented the buildings in the creation step, needed parts can be identified by their name. The setup of the video textures also includes the definition of geometry and its material for the enhanced billboard visualization. For the surveillance application we use an overlay image to highlight detected persons (see Fig. 7, right image). Billboards and extra geometry are then added to the scene graph as a sub child of the duplicated scene. As an example, in this way it is possible to translate a selected floor and the billboards will automatically stick to the floor.

In the next step we prepare the scene for user interactions. To look into lower floors of a building we define two functions: One is to simply hide selective floors, the other is to slide open the building floor by floor (see Fig. 8). For the sliding mechanism we add some additional transformation nodes. Again the node selection by name is used. To store the visibility status of a floor we initialize a map of floor names and Boolean values. Interaction functions can be toggled by the user via mouse or keyboard on the same machine which runs the application and also via a network connection on a remote machine.

5.2 Other 3D Information

The application script can integrate all kinds of additional information sources into the scene. As an example we employ curved 3D arrows for visualizing main motion directions of crowds (see Fig. 8, right image). To define such arrows a list of curve points can be read or directly used in the script. Animated materials add an additional cue for the direction.

6 Future Work

Figure 9: An operator watching our test location on the tiled display. Video data of multiple cameras and other input streams are combined into a single screen. Detected persons are highlighted by overlaid textures.

The transfer of large amounts of video data is still an issue, especially in a cluster setup. We currently work on a scalable system with video servers that scale and compress video into the needed resolution for the display. Size and frame rate of the videos should adapt according to the currently visible part of the scene and it is preferable to use image compression for the OpenSG cluster. Additionally we plan to update regions without motion less frequently to further reduce bandwidth usage. This could also be managed by a common video codec.

We currently only support a single ground plane per camera for computing the 3D position of a detection. In scenarios with multiple levels or stairs we plan to implement intersections with the

actual 3D model of the building. With a more accurate shape from the person detector we will also increase visual quality using transparent areas in the billboards. We plan to replace the image of detected areas with a background camera image to remove artifacts on the floor. Also individual colors for specific tracked persons may be defined via rectangle IDs. Moving objects that are not detected by the person detector will also be highlighted.

With usability experiments we plan to evaluate practical views of the 3D model, the suited types of navigation as well as input devices for the common tasks of an operator. We also plan to test different levels of abstraction like non-photorealistic rendering of the buildings.

7 Conclusion

We propose a real-time visualization system for integrating image based 2D detections into a 3D scene. For visualization we show the natural inverse process of capturing a camera image: the projection of the image into the 3D scene. Optimized shading and transparency effects enhance quality and usability.

The combination of detection results with a 3D scene is very useful for surveillance systems with many cameras. An informal user study shows that watching a moving person over multiple video camera images is much easier in a spatial coherent 3D environment. Other interesting events or detections recognized by different input sensors can also be added to the system and displayed in the same space.

With the scriptable scene graph engine it is possible to create a large variety of applications. Scripting of geometry, materials and user interactions offer a comfortable and fast way of manipulating the application. Using the OpenSG framework we also support large tiled displays and even immersive systems like CAVEs.

References

[1] Robert Collins, Alan Lipton, Takeo Kanade, Hironobu Fujiyoshi, David Duggins, Yanghai Tsin, David Tolliver, Nobuyoshi Enomoto, and Osamu Hasegawa. A system for video surveillance and monitoring. Technical Report CMU-RI-TR-00-12, Robotics Institute, Pittsburgh, PA, May 2000.

[2] Sven Fleck, Florian Busch, Peter Biber, and Wolfgang Straber. 3d surveillance a distributed network of smart cameras for real-time tracking and its visualization in 3d. In *CVPRW '06: Proceedings of the 2006 Conference on Computer Vision and Pattern Recognition Workshop*, page 118, Washington, DC, USA, 2006. IEEE Computer Society.

[3] A. Girgensohn, F. Shipman, A. Dunnigan, T. Turner, and L. Wilcox. Support for effective use of multiple video streams in security. In *VSSN 2006: 4th ACM International Workshop on Video Surveillance and Sensor Networks*, 2006.

[4] Brett Hall and Mohan Trivedi. A novel graphical interface and context aware map for incident detection and monitoring. 9th world congress on intelligent transport systems. In *9th World Congress on Intelligent Transport Systems*, 2002.

[5] Sven Havemann. *Generative Mesh Modeling*. PhD thesis, Institute of Computer Graphics, Faculty of Computer Science, Technical University Braunschweig, Germany, November 2005.

[6] Sven Havemann, Volker Settgast, Marcel Lancelle, and Dieter Fellner. 3d-powerpoint - towards a design tool for digital exhibitions of cultural artifacts. In Arnold, Niccolucci, and Chalmers, editors, *VAST 2007*, 8th Inter. Symp. on Virtual Reality, Archaeology and Cultural Heritage, pages 39–46, Brighton, UK, Nov 2007. Eurographics Association.

[7] Ulrich Neumann, Suya You, Jinhui Hu, Bolan Jiang, and JongWeon Lee. Augmented virtual environments (ave): Dynamic fusion of imagery and 3d models. In *VR '03: Proceedings of the IEEE Virtual Reality 2003*, page 61, Washington, DC, USA, 2003. IEEE Computer Society.

[8] John K. Ousterhout. Scripting: Higher Level Programming for the 21st Century. *IEEE Computer Magazine*, 31(3):23–30, 1998.

[9] William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 283–291, New York, NY, USA, 1987. ACM.

[10] Dirk Reiners. Special Issue on the Opensg Symposium and OpenSG Plus. *Computers & Graphics*, 28(1):59–61, 2004.

[11] Peter M. Roth and Horst Bischof. *Conservative Learning for Object Detectors*, chapter Conservative Learning for Object Detectors, pages 139–158. Springer, 2008.

[12] H. S. Sawhney, A. Arpa, R. Kumar, S. Samarasekera, M. Aggarwal, S. Hsu, D. Nister, and K. Hanna. Video flashlights: real time rendering of multiple videos for immersive model visualization. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 157–168, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[13] Ismail Oner Sebe, Jinhui Hu, Suya You, and Ulrich Neumann. 3d video surveillance with augmented virtual environments. In *IWVS '03: First ACM SIGMM international workshop on Video surveillance*, pages 107–112, New York, NY, USA, 2003. ACM.

[14] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. *SIGGRAPH Comput. Graph.*, 26(2):249–252, 1992.

[15] T. Tsuda, I. Kitahara, Y. Kameda, and Y. Ohta. Smooth video hopping for surveillance cameras. In *SIGGRAPH Sketches*, 2006.

[16] Yi Wang, David M. Krum, Enylton M. Coelho, and Doug A. Bowman. Contextualized videos: Combining videos with environment models to support situational understanding. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1568–1575, 2007.