



Kraków, June 17, 2005

AUDIOVISUAL SCENE RENDERING AND INTERACTION

Abstract: The paper aims to present work of partners within the scope of group of problems under the common title “Audiovisual scene rendering and interaction”. First section covers the Rendering Engine software with capabilities of rendering complex hybrid scenes for immersive environments. The client-server architecture of the engine is described and then a number of standard plug-ins extending the functions of the system are presented. Section 2 describes an interactive system for navigation through high-resolution cylindrical panoramas. Topics such as acquisition of panoramas, camera calibration and occluding objects removal, are covered in detail. Next section provides a short description of the specialized VRML viewer – a piece of software to visualize large environments. And finally last section is devoted to a novel algorithm for spatialisation of sound sources in 3D space. Unlike known techniques e.g. finite elements or beam tracing and radiosity, the proposed algorithm is based on a geometric pre-computation of a tree-like topological representation of the reflections in the environment (beam tree) through spatial subdivision techniques. The beam tree is then used for real-time auralization through a simple look-up of which beams pass through the auditory points.

1. RENDERING ENGINE OF HYBRID AUDIOVISUAL SCENE

1.1. Overview

Immersive environments are gradually becoming an important paradigm used in advanced multimedia applications such as distant surgery, distant learning, virtual museum tours, 3D games, etc. Fully immersive environments not only integrate synthetic data, i.e. 3D graphics, with natural audio-video (AV) streams but deal as well with interactive aspects in both local and remote. The provision of interaction, especially remotely, in synthetic natural hybrid (SNH) scenes in which users are immersed is very challenging and imposes a considerable complexity on the design of rendering software.

The rendering complexity can be significantly reduced by using Open-GL graphics cards with advanced extensions. However, if the immersive scene is built using media of multiple modalities, the use of such advanced cards is not able to overcome the extra-incurred complexity. Indeed, combining different types of rendering operations such as image based with classical 3D graphics based on the triangle meshes and associated textures, is not a trivial operation. When the SNH scene consists of different parts with such different rendering modalities, integration of them at the rendering engine creates several technical problems, especially if different

rendering parts are developed by different programming teams.

In section 1.2 we show how the concept of synthetic-natural hybrid (SNH) camera and microphone is used to design a software platform for fully immersive networked environments. The client-server architecture designed for the platform is accordingly presented in this section. Section 1.3 describes functionalities for selected software modules (plug-ins) that were developed for the system. Sections 1.4 and 1.5 illustrate the approach both through a set of simple demo applications using the proposed API and through the Virtual Classroom, an example of a more sophisticated application. Section 1.4 provides indications on how to install and run the software, whereas section 1.5 describes how to use it to create an immersive application. The software developed is written in C++ using MS Visual Studio 6.0.

1.2. Software architecture

SNH camera and microphone integrate three algorithmic aspects of software design for immersive environments:

- SNH AV scene view generation;
- view rendering;
- view controlling by a user.

The software platform for immersive environments that we have created presents architecture as illustrated in figure 1.1. In this architecture, the SNH camera and microphone are implemented by the RendGen (**render**er and **generator**) application at the user terminal. The RendGen is a client of the specific immersive application server SNHgen (Hybrid Generator), synchronizing views and events in the whole system. The RendGen application is actually a collection of plug-ins that could be either general or specific for the particular location.

The main element of our software immersive platform is the Hybrid Generator, which is implemented by HybridGenerator C++ class. The HybridGenerator is responsible for the actual generation of the SNH scene and:

- provides clients with a scene description and animation;
- manages events within the application;
- manages the interaction within the scene, namely between-users, between-objects, and users between objects;
- provides the programmer with network communication methods.

The HybridGenerator acts as the server for the RendGen client application, which is a universal browser for the end user. The RendGen consists of two modules:

1. *renderer* that displays the scene in a viewport. The renderer asks HybridGenerator to send the scene geometry as a list of OpenGL commands to be executed in a client application. It then combines the scene geometry together with other signals, e.g. live video coming from other locations;
2. *generator* that manages all data sources available at the client side (e.g. video cameras) and sends the signals to different locations like other RendGens, therefore participating in the generation of the whole scene by the HybridGenerator.

The *knowledge* of how to combine the scene geometry together with live AV streams from other locations is managed by the HybridGenerator.

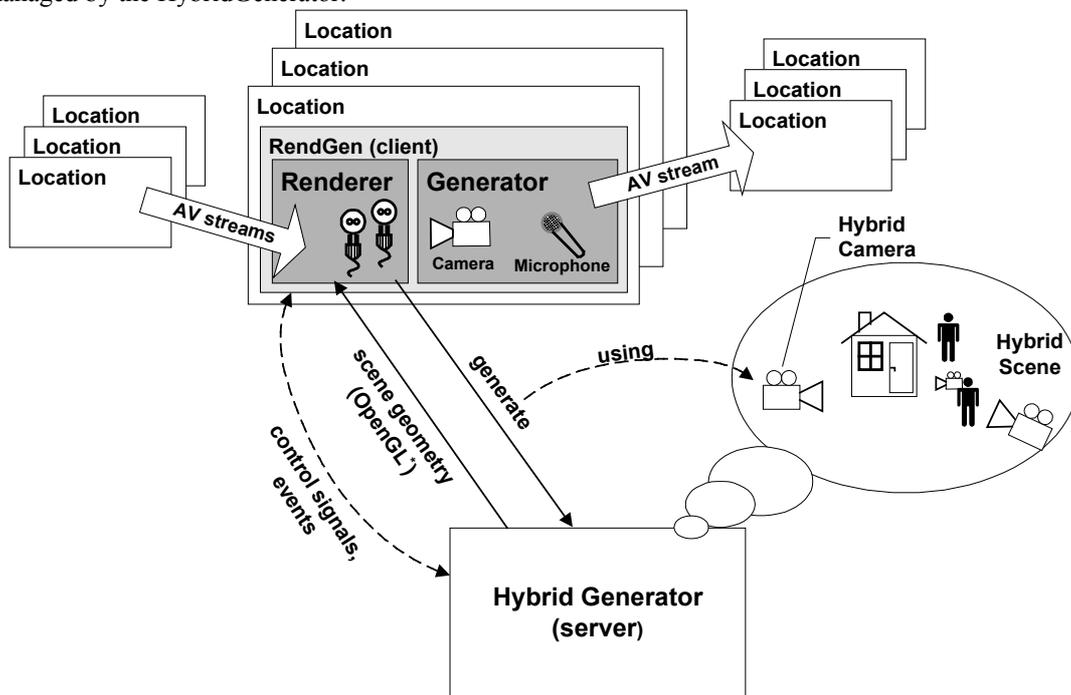


Figure 1.1 Architecture of distributed Rendering Engine.

1.3. RendGen's plug-ins

In order to obtain a flexible and extensible platform, where new functionality could be added easily as needed, we have decided to create a plug-in oriented architecture. Furthermore, the use of plug-ins enables different programmers' teams to integrate their software. Twelve general plug-ins have been developed to use within the RendGen. These are addressed in the following paragraphs.

Viewport. Hybrid Generators use this plug-in to display 2D/3D graphics, animations of the whole scene that are based on OpenGL. The Viewport plug-in creates an OpenGL MDI child window in the client area of the RendGen application window. To cover the most typical OpenGL operations for 2D/3D graphics, 66 OpenGL commands were chosen. These commands were mapped onto functions from RendGen API. In addition to these, there are also other OpenGL-like commands using the RendGen application architecture

The RendGen has a modular architecture and is based on plug-ins. It is responsible for making the connection with the HybridGenerator and for managing resources, e.g. plug-ins. It is a Multiple Document Interface (MDI) application allowing the user to open many views of a scene. The RendGen is not directly responsible for displaying the scene; this responsibility is assigned to appropriate plug-in.

A plug-in is any application compiled as the Dynamic Linked Library (DLL). It uses the developed API to communicate with the HybridGenerator through the RendGen that acts as a broker for such a communication. The next section describes all the generic plug-ins developed as well as specialized plug-ins developed for particular applications.

and shared resources like images or memory buffers. In the original OpenGL functions, one of the arguments is a pointer to a memory buffer that contains, for instance, texture or image data. As the HybridGenerator operates remotely on them, the argument had to be changed to resource ID.

AudioCapture plug-in is used to capture audio from an external device such as microphone. It creates an audio resource that is regularly updated with the data captured from an external audio device. This plug-in uses any audio device available in the OS and settings of standard audio mixer.

OpenALPlayer plug-in is used to generate 3D audio using OpenAL library. OpenALPlayer works similarly to ViewPort plug-in, it executes OpenAL commands received from HybridGenerator. It can play live audio streams as well as WAV and OGG-Vorbis files. There are 6 typical OpenAL commands implemented as well as 5 OpenAL-like commands that re-

place some original OpenAL counterparts or extend functionality of OpenAL by streaming capabilities. These commands use pointers to some memory buffers that had to be changed to resource IDs as the Hybrid-Generator manages them remotely.

AVPlayer plug-in provides the application with an easy programming interface to use AVI movie files in a composed scene and to display them within the Viewport plug-in window, e.g. as the “texture” of 3D object or as the background of the scene. This plug-in creates an ImageResource and fills this resource with current data from the movie file. The resource is available for any other plug-ins running within the RendGen. The Hybrid Generator can control the playback by sending appropriate events.

AVPlayer is based on MS AVIFile library (vfw32.lib) and some problems were observed under Windows 2000 when playing AVI files compressed with particular codecs, e.g. Xvid, DivX codecs. The preferred codec is Intel Indeo, which runs fine on most versions of Windows.

Capture provides the application with an ability to obtain video streams from any capture device registered at the operating system level. Such a video stream can be used as a “texture” for objects or a background for the scene. The captured data, i.e. the created resource, is available both for any other plug-in running within the RendGen and the Hybrid Generator.

ImageLoader plug-in loads a given file resource of type IMAGE into memory and creates ImageResource, which is then available for other plug-ins and the Hybrid Generator. Only Windows BMP image file format is supported so far.

MemBufManager. The function of MemBuf-Manager plug-in is to manage shared memory. Memory can be shared between plug-ins and remotely between plug-ins and HybridGenerator. This plug-in is mainly provided for transferring large OpenGL data sets like vertices, indices, colors, normals, buffers, and any other data the application deals with.

MulticastSender. This plug-in provides the application with a mechanism of transferring data, usually audio/video streams, between RendGen applications running within the same Local Area Network using multicast. MulticastSender works in pairs with MulticastReceiver.

MulticastReceiver. This plug-in provides the application with a mechanism of receiving data, usually audio/video streams, from other RendGen applications in the same Local Area Network using multicast transmission. MulticastReceiver works in pairs with MulticastSender.

TextTerminal. The plug-in provides an easy and flexible bi-directional text communication mechanism between user and HybridGenerator. It may also be used to provide users with text chat functions. This plug-in creates MDI child window in the client area of the RendGen.

ScreenCapture provides the application with an ability to obtain screenshots or a video stream from the user’s desktop or any other application running on the

same machine. It creates an ImageResource and fills its buffer with current screenshot of the desktop. If a video is needed, then all that is necessary is to start capturing with the respective frame rate. The captured data, i.e. the created resource is available for any other plug-in running within the RendGen and for the Hybrid Generator as well.

VideoCompressor plug-in compresses video data or still images using any codec available within the operating system. Each codec is configured and controlled by its native functions and native user interface. The plug-in works in pairs with VideoDecompressor.

VideoDecompressor plug-in decompresses video data or still images using any codec available within operating system. Each codec is configured and controlled by its native functions and native user interface. The plug-in works in pairs with VideoCompressor.

Model3DLoader loads 3D model of any object from a file and creates Model3DResource available for other plug-ins. This model can be rendered in a View-Port. The purpose of this plug-in is to support 3D models of heads used in Virtual Classroom application – the test application for the engine. The plug-in supports the very basic transformations of the model.

1.4. Interaction control engine for use in the immersive environment

The process of developing a rendering engine of hybrid audiovisual scene provided us with the platform for communication between elements of networked immersive environment system. Having this we developed a communication protocol based on events. There was a variety of events developed for different types of interaction within an immersive environment. A typical interaction within immersive environment exists between user and application graphical user interface, e.g. selecting objects or entering text etc. Another type of interaction occurs between users of the system. Finally the system interacts with the user, e.g. by sending an AV or textual message to the user. These types of interaction are handled by events at the low level.

The Rendering Engine is plug-in oriented and the events, more precisely, objects that represent and encapsulate them, circulate between different components of the system. There are two kinds of events:

- Standard events sent and received by core elements of the engine, e.g. EvtConnect used to establish the connection, EvtShow used when the user requests the plug-in to show its window if any, etc.
- Events exchanged between plug-ins and the rest of the system; these events are recognised by particular plug-ins, e.g. EvtGenerate for Viewport plug-in, used to make the plug-in to generate the scene for the requester, etc.

In order to define more sophisticated way of controlling the interaction within the scene we defined an API of Interaction Control Engine. The API combines animation and interaction since the user interacting with object of the scene changes the animation as well as object may interact one with another so that the result

of this process changes the animation, too. One of the key element of the API is C++ class HybridScene. Objects of this class are threads. The instance of the HybridScene plays the role of a container for objects from the scene.

The next important class is HybridObject that is a base class for all objects added to the HybridScene. The instances of objects derived from HybridObject usually can be visualised in a 3D space. One of classes derived from HybridObject is HybridCamera. Its main purpose is to provide the system with the view of the hybrid scene for users. The hybrid camera similarly to other hybrid objects encapsulates coordinates in 3D space. It can be tied down to other hybrid object so that it will move within the scene according to the object.

To detect collisions within the scene CollisionDetector class was defined. It defines a method that returns a collection of objects that are currently colliding with a given object. Instances of CollisionDetector keep its own representation of the scene or a list of references to those objects within the scene that will be checked for a collision. There are many different ways and algorithms to find collision of objects which demand different input data e.g. position of object, orientation, radius of a bounding sphere or size of a bounding box etc.

Despite the fact that there can be one common hybrid scene for all users the visualisation of the scene and objects relies on attributes that are bound to the specific virtual desktops or OpenGL window. For instance identifiers of the textures generated in OpenGL can differ between contexts even if they represent the same image. This concept is implemented in RenderingContext class which instances keep the OpenGL

context of rendering. The RenderingContext instances are ready to store additional client-specific information.

2. IMAGE BASED RENDERING

2.1. Overview

Work has also been conducted towards the development of a system based on MPEG-4 for rendering of high-resolution panoramic views. The use of MPEG-4 technology provides many new features compared to conventional 360° panoramas. Video objects, dynamic 3-D computer models [5, 7], or spatial audio as illustrated in Figure 2.1, 2.2 can be embedded in order to vitalize the scene. Pressing interactive buttons gives additional information about objects or modifies the current location. The MPEG-4 system also ensures that only visible data is transmitted avoiding long downloads of the entire scene. In this way it is possible to create large high quality environments that enable an enhanced immersive experience to the user into the virtual world.

2.2. General Introduction

In image-based rendering [1], cylindrical panoramas have received particular interest in current applications due to their simple acquisition setup. Only a couple of pictures need to be captured on a tripod or freely by hand [2]. The images are stitched together forming one panoramic image as shown in Figure 2.4. From the 360° scene information, new views can be rendered, enabling the user to turn the viewing direction and interactively decide the point of interest. One well known example of such a system is QuicktimeVR [3].



Figure 2.1 Multiple panoramas from the Ethnological Museum in Berlin. Interactive scene elements allow the user to jump between the rooms. Dynamic objects are added to vitalize the scene.

In contrast to light fields [4] or concentric mosaics [5], the viewing position for panoramic rendering is restricted to a single point. Only rotation and zoom are permitted for navigation. This restriction can somewhat be relaxed by allowing to jump between different pano-

ramas as shown in Figure 2.1. However, for many applications this is sufficient and panoramic views can be found more and more often on web sites creating virtual tours for city exploration, tourism, sightseeing, and e-commerce.

In this work package we have been working on the development of a system for rendering high-resolution panoramic views, based on MPEG-4. The use of MPEG-4 technology provides many new features compared to conventional 360° panoramas. Video objects, dynamic 3-D computer models [6, 7], or spatial audio as illustrated in Figure 2.2 can be embedded in order to vitalize the scene. Pressing interactive buttons gives additional information about objects or modifies the current location. The MPEG-4 system also ensures that only visible data is transmitted avoiding long downloads of the entire scene. Thus, large high quality environments can be created that enable the user to immerse into the virtual world.

Although the acquisition of large panoramas is quite simple in principle, in practice, the situation is often much more complex. For example, people, objects, or clouds in the scene may move while capturing the single images. As a result, the pictures do not fit to each other properly and ghost images appear. Moreover, capturing 360 degrees of a scene may impose high demands on the dynamic range of the camera. Especially in indoor scenes, extreme changes in intensity may occur between windows and the interior. We have therefore investigated algorithms for the removal of moving people and objects in order to simplify the stitching. Multiple views are captured at different time instants and the covered regions are warped from the same areas in other views. Capturing the scene with different shutter times enables a spatial adaptive adjustment of the dynamic range and to create panoramas also for scenes with extreme brightness changes.



Figure 2.2 Scene elements of our MPEG-4 player. Besides the panorama, dynamic video objects, interactive buttons, 3-D models or spatial sound can be added to the environment.

This part of the paper is organized as follows. First, the MPEG-4 framework, which is responsible for view-dependent rendering and streaming of panoramas, videos and 3-D objects is described. In section 2.5 “Camera Calibration” the determination of focal length and lens distortion, which supports the accuracy of the stitching, is described. The algorithm for the removal of objects is illustrated in section 2.6 “Object Removal”. Section 2.7 “Dynamic Range Adaptation” finally de-

scribes the local adjustment of dynamic range and provides examples from real panoramas.

2.3. MPEG-4 System for Panorama Streaming and Rendering

The system for panorama rendering uses MPEG-4 technology which allows local display or interactive streaming of the virtual world over the internet. The scene is represented very efficiently using MPEG-4 BIFS [8] and rendered at the client using our MPEG-4 player [9]. The basic scene consists of a 3-D cylinder textured with a high resolution panoramic image as shown in Figure 2.4. Other scene elements like 2-D images, video sequences, 3-D audio as well as interactive scene elements, like buttons or menus can easily be added. Since alpha masks can be provided to create arbitrarily shaped video objects, moving people or objects in motion can be added to the static scene creating more lively environments. Buttons allow to walk from one room to the next (Figure 2.1) by requesting new BIFS descriptions, or to allow displaying additional information.

For a particular panoramic scene with video sequences, 2-D images and 3-D audio objects, the movements of the pointer device are evaluated and the appropriate data for the desired viewing direction is requested from the server. In order to avoid the necessity of streaming initially the entire panorama, which would add severe delays, the high-resolution image is subdivided into several small patches. To each patch, a visibility sensor is added. If the current patch is visible the sensor is active and it becomes inactive if the patch disappears again. Only active parts need to be streamed to the client and only if they are not already available there. The partitioning into patches and the visibility sensors are illustrated in Figure 2.3.

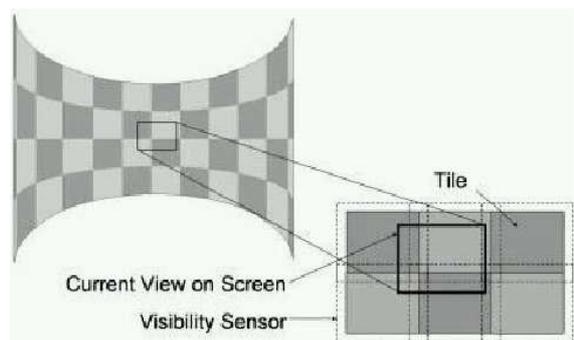


Figure 2.3 Subdivision of the panorama into small patches and visibility sensors.

The visibility sensors are slightly bigger than the associated patch. This allows to pre-fetch the image patch before it becomes visible. The size of the sensors is a trade-off between pre-fetching time and the space necessary to store them locally this way, a standard and efficient compliant streaming system for panoramas with additional moving and interactive scene elements is realized.



Figure 2.4 Cylindrical panorama captured at the Adlon hotel, Berlin, Germany.

2.4. Acquisition of Panoramas

The images for the panoramas are captured with a digital camera mounted on a tripod. For indoor environments, a wide angle lens converter is used to increase the viewing range. The camera on the tripod is rotated around the focal point by 15 to 30 degrees (depending on the viewing angle) between the individual shots. The resulting images are then stitched together into a single panorama using a commercially available tool (e.g., PanoramaFactory, www.panoramafactory.com). The output is a panoramic image as shown in Figure 2.4, which is then subdivided into small patches of size 256x256 pixels for view-dependent streaming with the MPEG-4 system. With the current configuration, the resolution of the entire panorama is about 14000 by 2100 pixels which allows viewing also small details by changing the zoom of the virtual camera. Figure 2.5 shows a magnification of the white box in the panorama of Figure 2.4.



Figure 2.5 Closeup of the Adlon pool panorama. The content corresponds to the interior of the white box in Figure 2.4.

2.5. Camera Calibration

In tests with several stitching tools, it became evident that the accuracy of the results could be improved by determining focal length and lens distortions of the camera in advance rather than optimising these param-

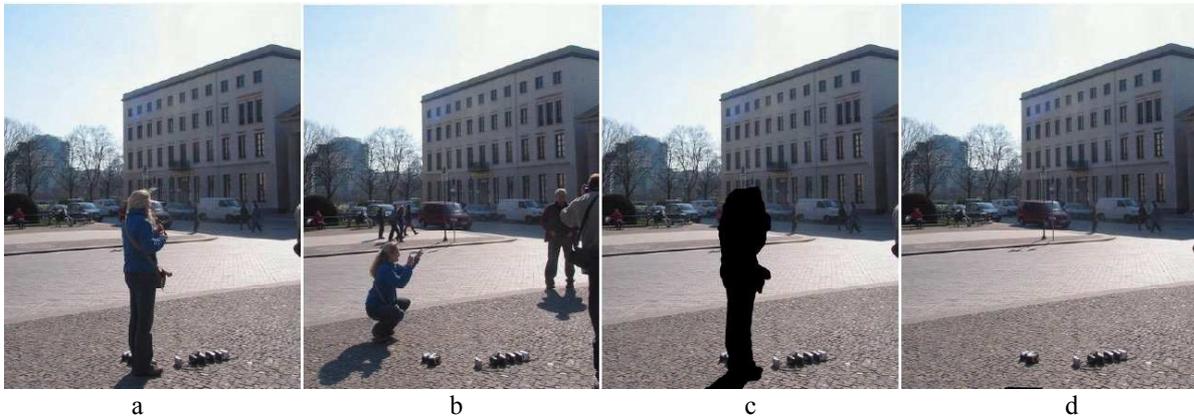


Figure 2.6 a, b: two pictures with a person that moves between the shots, c: manually selected image mask, d: final composed image ready for stitching.



Figure 2.7 Part of the panorama of the Brandenburger Tor with people removed.

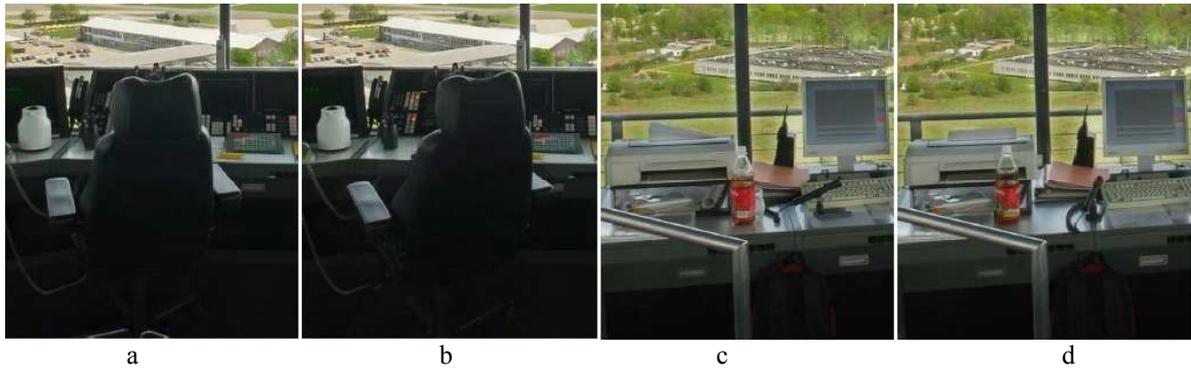


Figure 2.8 *a, c*: original images from an airport tower. *b, d*: same images after warping several objects from other views. The chair is rotated, printer, microphone, and bottle are moved appropriately.

ters during stitching. We have therefore calibrated the camera with a model-based camera calibration technique [10]. The resulting intrinsic parameters, like viewing angle and aspect ratio, are passed to the stitching tool while the lens distortion parameters are used to correct the radial distortions in the images. Especially for the wide-angle lenses, severe distortions occur which have to be removed. Since the used camera can be controlled quite reproducibly, it is sufficient to calibrate the camera once for various settings.

2.6. Object Removal

The stitching of multiple views to a single panorama requires the pictures to overlap in order to align them and to compensate for the distortions (due to projection, camera position, lenses, vignetting, etc.). After alignment, the images are blended to obtain a smooth transition from one image to the next. If a single camera is used and the images are captured one after the other, ghost images can occur in the blending area if objects or people move during capturing as, e.g., in the upper row images of Figure 2.6. These mismatches have to be removed prior to stitching. For the warping, we use the eight-parameter motion model that can describe the motion of a plane under perspective projection as the first approximation which is next refined using a gradient-based motion estimator [11].

This warping is also done if multiple images are captured from the same viewing position. Wind or vibrations can easily change the camera orientation slightly so that shifts of one or two pixels occur. The rightmost image of Figure 2.6 shows the result of the warping and filling. The person in the left two images has been removed.

In the same way, many people can be removed. Figure 2.7 shows how the Brandenburger Tor in Berlin looks like without a crowd of people, which can rarely be observed in reality.

The different acquisition time of the images can also lead to photometric changes, especially if clouds are moving. We therefore estimate also changes in color and brightness between the shots using a polynomial of second order.

Figure 2.8 shows some more examples for object removal by warping and illumination adjustment. The images are recorded in a tower at an airport where several people are working. During the capturing, sev-

eral objects were moved and chairs were turned. The left side of Figure 2.8 shows the images captured with the camera while the right images are corrected by warping from previous or succeeding views.

2.7. Dynamic Range Adaptation

The dynamic range in a scene can vary drastically which might lead to saturation effects in a camera capturing the scene. In 360° panoramas with a large number of possible viewing directions, the chance is high that there exist very bright and very dark regions. Especially in indoor scenes, drastic discontinuities can occur, e.g., at windows with a bright scene outside and a darker interior. Regular digital cameras are not able to capture such a dynamic range so that they often saturate at the lower or upper end.

These saturation effects can be avoided by combining multiple differently exposed images [12, 13]. In [14], it has been shown, that the simple summation of these images combines all their information due to the non-linear characteristic of the camera. In our experiments, the resulting panoramas, however, showed lower contrast, so we decided to use a locally adaptive summation similar to [12].

Figure 2.9 shows an example for the automatically computed mask and its corresponding image. Given the mask, a weighted sum of the images is computed, with the weights being locally determined by the image mask. Thus, the contrast remains high in bright as well as in dark image regions.



Figure 2.9 *Left*: one image from the Tower panorama. *Right*: Automatically computed mask to distinguish bright from dark image regions.

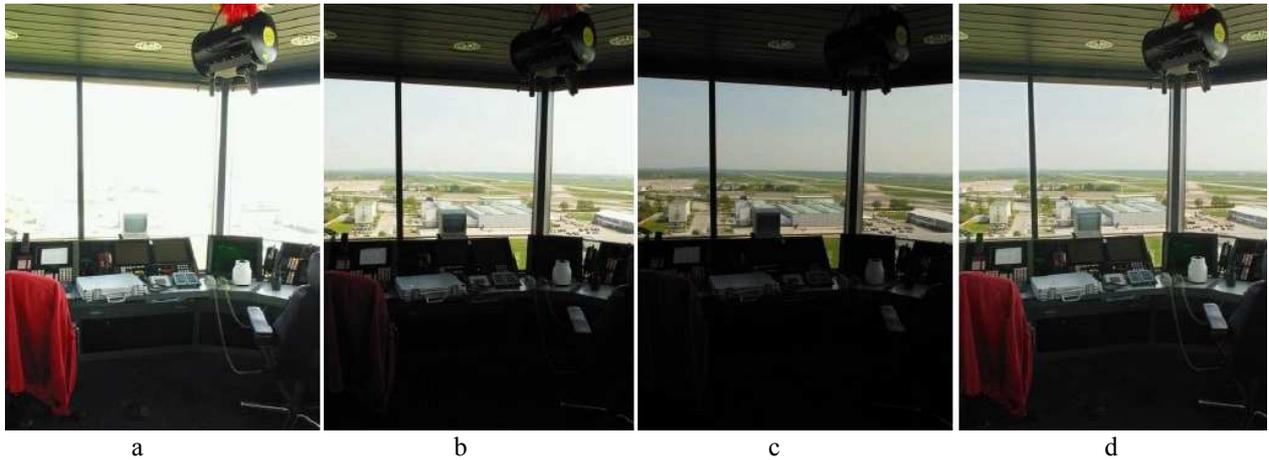


Figure 2.10 a, b, c: Differently exposed images from the interior of an airport tower. d: combined image with high contrast in dark as well as bright regions.

This is illustrated in Figure 2.10. The figure shows three differently exposed images from the interior of an airport tower with dark instruments in the foreground and a bright background. After image warping as described in Object Removal to account for moving objects, the images are adaptively combined into a new image shown on the lower right of Figure 2.10 that reproduces the entire scene with high contrast.

3. WEB3D VIEWER

The viewer developed is dedicated to high quality real-time visualisation of complex 3D environments such as virtual museums, virtual commercial galleries and historic site restoration. The viewer is not a complete, standard compliant, VRML browser. Instead it should be seen as a high performance viewer and experimental platform dedicated to real time visualization, and associated to well-defined and carefully prepared data bases. It requires a hardware accelerator for OpenGL on board. Its main functions cover:

- VRML97 parser
- GIS terrain data parser
- laser-scanner data file parser
- panorama visualisation
- object picking to obtain information from selected objects.

The 2D drawer is able to visualize pictures and 2D shapes, e.g. lines, rectangles and 2D or 3D splines. The viewer is capable of remote file operations, so that remote files containing 3D models can be loaded and visualized.

4. EFFICIENT SPATIALISATION OF SOUND SOURCES IN VIRTUAL ENVIRONMENTS

4.1. Introduction

Advanced sound rendering techniques are often based on the physical modelling of the acoustic reflections and scattering in the environment. This can be modelled using either numerical methods (finite elements, boundary elements and finite differences) or geometrical methods (image source, path tracing, beam

tracing and radiosity). With all these techniques, however, computational complexity may easily become an issue. One approach that enables an efficient auralization of all reflected paths is based on beam tracing. This method is based on a geometric pre-computation of a tree-like topological representation of the reflections in the environment (beam tree) through spatial subdivision techniques. The beam tree is then used for real-time auralization through a simple look-up of which beams pass through the auditory points. It is important to notice, however, that the tree-like structure of the beam reflections makes the approach suitable for early reverberations only, as it prevents us from looping beams and implementing the corresponding IIR structures. In order to overcome this difficulty, we recently proposed an approach that models both early and late reverberation by cascading a tapped delay line (a FIR) with a Waveguide Digital Network (WDN) (an IIR). In this approach, the beam tree is looked-up to generate the coefficients of a tapped delay line for the auralization of early reverberation. In order to account for late reverberation as well, we feed the outputs of the tapped delay lines into WDN, whose parameters are determined through path tracing.

One problem of the beam tracing approach is that it assumes that sound sources are fixed and only listeners are allowed to move around. In fact, every time the source moves, the beam tree needs to be re-computed. This may easily become a costly operation, particularly with environments of complex topology, as it is based on spatial subdivision algorithms.

We proposed and implemented a novel method that allows us to significantly speed-up the construction of the beam tree, by avoiding space subdivision. This allows us to dynamically re-compute the beam tree as the source moves. In order to speed-up the construction of the beam tree, we determine what portion of which reflectors the beam “illuminates” by performing visibility checks in the “dual” of the world space. In order to illustrate the approach, we will assume that the world space is 2-dimensional.

4.2. Tracing Beams

The world space that we consider is made of sources and reflectors. Sources are assumed as point-like, and reflectors are linear segments. We call “active” that region of a reflector that is directly “illuminated” by a source. The active portion of a reflector can be made of one or more active segments (connected sets), due to occlusions. Each one of the active segments generates a beam, which is defined as that bundle of acoustic rays that connect the source with points of the active segment. Each acoustic ray w can be described by an equation of the form $y = ax + b$, where a is the angular coefficient and b is the offset on the y axis, all referred to a world coordinate system (x, y) . The dual W of that ray is thus a point (a, b) in parameter space. The dual of a point p of coordinates (x, y) , on the other hand, is a line P in parameter space, given by all pairs (a, b) that correspond to rays passing through p . This means that the dual of a source is a ray.

Let us now consider a reflector, which is a segment in world space. In order to determine its dual, we can consider the duals P and Q of its two extremes p and q , respectively. Such points will correspond to two lines P and Q in the dual space. As the reflector r is made of all points in between p and q , its dual R will be made of all the lines in between P and Q . In other words, the dual of a reflector is a beam-like region. We recall, however, that an active segment of a reflector, together with a virtual source s , completely specifies a beam. The dual of this beam can be determined as the intersection between the dual R of the active segment (a beam-like region) and the dual S of the virtual source (a line). In conclusion, the dual of a beam is a segment, just like the dual of a segment (reflector) is a beam.

Pushing this formalism even further, we can define active reflecting segments at infinity (just like in projective geometry) and beams at infinity, which are those beams that are not reflected by any surface. If we have a collection of reflectors in world space, their duals will form a collection of beam-like regions (see Figure 4.1).

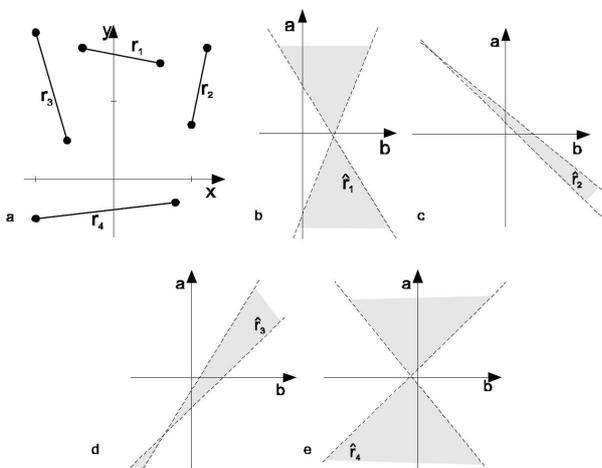


Figure 4.1 The duals of a set of physical reflector is a collection of strip-like regions in the dual space.

We would now be tempted to say that “the intersection between the dual of a source and the dual of the reflectors is the dual of the branching beams and can therefore be used to determine the beam tree”. This would be true if there were no occlusions between reflectors (i.e. if all points of the reflectors were active). In fact, mutual occlusions cause the duals of reflectors to overlap each other. Therefore, in order to determine the branching of the beams we need first to figure out the layering order of such regions, according to mutual occlusions (i.e. “which region eats which”).

The layering order is not the only problem, as we need to construct the tree of all possible reflections in the environment while keeping track of the occlusions. In order to do so, we propose an iterative approach that starts from the source (root) and tracks the branching of all beams in dual space. At each iteration we consider a beam at a time and we determine how this beam is split into various sub-beams as it encounters reflectors on its way. This splitting characterizes the branching of the beam tree.

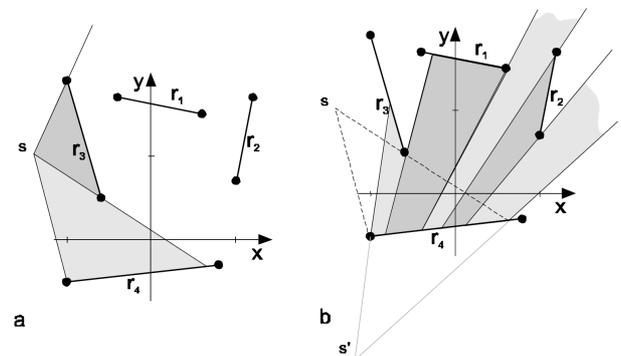


Figure 4.2 The visibility of r_{01} with respect of r_{00} . The visibility is limited to the right half-plane of the reference reflector.

At the first iteration we consider a source and the physical reflectors (Figure 4.2a). From this source a number of beams will depart, each corresponding to an active segment of the various reflectors. Such active segments can be determined quite straightforwardly by tracing all rays that depart from the source. At this point we consider all the beams one by one and figure out how each of them branches out as it encounters the reflectors (Figure 4.2b). Each beam will keep branching out until its cumulative attenuation or its aperture angle falls beneath some pre-assigned threshold. Let us consider a beam originating from a virtual source s . If we want to characterize how this beam branches out, we start from the active reflector segment r_0 (Figure 4.3a) that defines that beam (aperture) and we perform a change of reference frame in order for its extremes p_0 and q_0 to fall in the coordinates $(0, 1)$ and $(0, -1)$, respectively (Figure 4.3b).

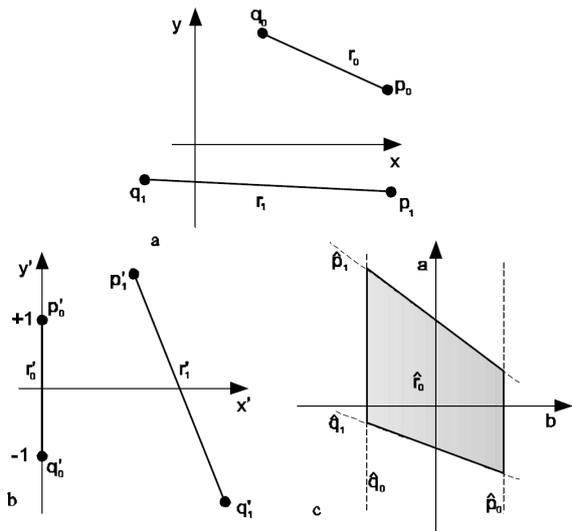


Figure 4.3 A reflective segment: a. in the world space, b. after normalization relative to r_0 , c. in the dual space.

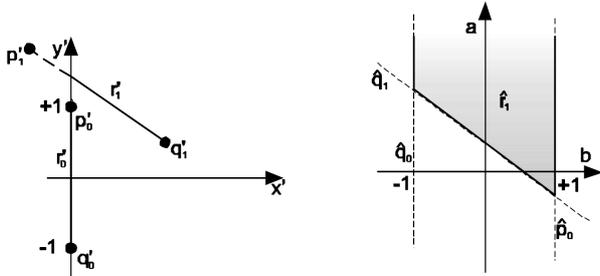


Figure 4.4 The visibility of r_{01} with respect of r_{00} . The visibility is limited to the right half-plane of the reference reflector.

This way, the dual of the considered aperture will be the reference strip $-1 \leq b \leq 1$ in parameter space (see Figure 4.3c). The intersection in the dual space between the reference strip and the representation of the other considered reflection is the area named R1 that represents the visibility of r_{01} from r_{00} .

In Figures 4.4 and 4.5 we can see some other examples of a reflector visibility from a reference segment.

Let us now consider again Figure 4.1, which has four physical reflectors. Let us assume that, as the iterations go along, r_4 is found to be an active reflector, and we want to show how to determine the branching of the beams. Therefore we will consider r_4 as an active aperture, and a coordinate change is used to normalize its representation in the space domain. We will name the new coordinate system (x_4, y_4) (superscript denotes which reflector is currently the reference for the normalization), and the new representation of the other reflectors will become r_{41} , r_{42} and r_{43} . In the dual space we obtain the set of regions of Figure 4.6. Notice that r_{43} partially occludes r_{41} . In the dual space, when the areas representing two reflectors overlap, there is an occlusion. By analyzing the slopes of the dual represen-

tation of the reflector endpoints, it is possible to define in the correct way the relative visibility.

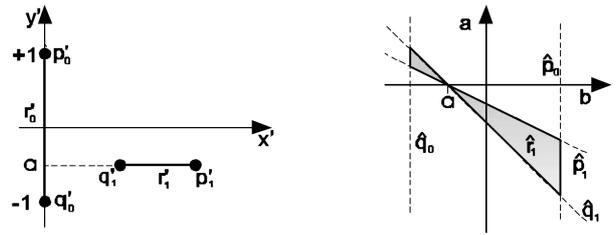


Figure 4.5: Another particular case of visibility. Varying the observation point on the reference segment, the two different sides of the reflector r_{01} are seen. This is clear from the observation of the dual space.

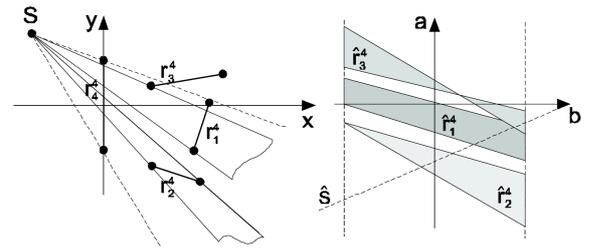


Figure 4.6: Beamtracing in the dual space. The intersection between S and the colored regions generates a collection of segments, which are duals of the branching beams.

A particular ordering situation between reflectors occurs when the sorting between two (or more) reflectors is ambiguous with respect to a third reflector (see Figure 4.7). This happens when the reflector's order changes with respect to a moving point on the reference reflector. This could be a problem when working in metric space, but it becomes easy to solve in the dual space. In fact, in the dual space the visibility information is obtained by partitioning the reference strip $(-1 \leq b \leq 1)$, by looking at the intersections of the segments that define the reflector's endpoints.

As a last foremark it is important to notice that, without loss of generality, the source can always be placed on the left of the normalized aperture, as we can always replace a (virtual) source with its specular with respect to the aperture. As we can see in Figure 4.6, the scan of the rays that define a source beam corresponds to the scan of the points of its dual along its extension.

4.3. Implementation

In order to test the method that we proposed, we implemented a software application that includes a 2D CAD for creating test environments, and an auralization system based on both a traditional beam tracer and a dual-space beam tracer. Auralization is based on tapped delay lines, and includes HRTF and cross-fading between beam configurations.

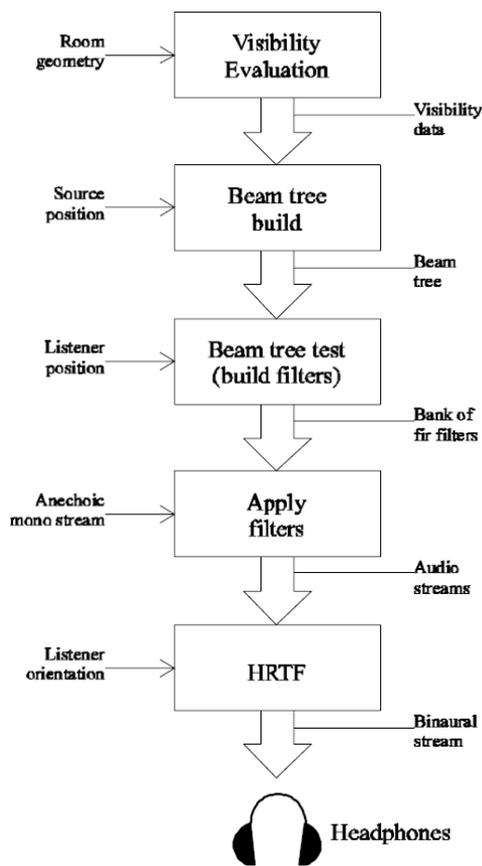


Figure 4.8 A schematic view of the auralization algorithm embedded in the beam tracing application.

The geometries considered by this application are only two dimensional, but the method can be extended to the 3D space by considering the reflections of a linear path on floor and ceiling. The basic structure of our auralization system is shown in Figure 4.8.

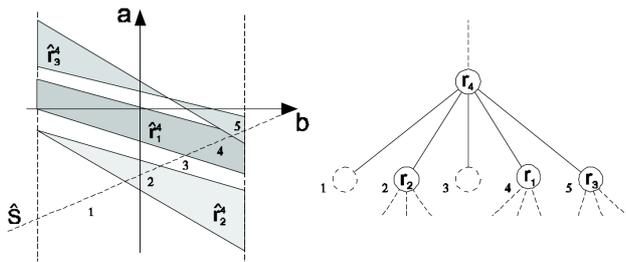


Figure 4.9 Building a tree level.

In a first step, the algorithm analyses the geometric description of the reflectors of the environment, and it extracts all the necessary information on visibility by working in the dual of the geometric space. As this step depends only on the geometry of the reflectors, as long as the environment is fixed, this operation can be performed beforehand, as a preprocessing step. The location of the sound source and the visibility information can now be used to construct the beam tree, or to update it every time the source moves.

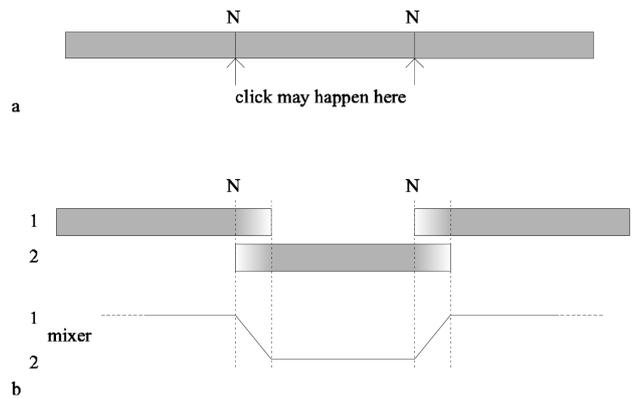


Figure 4.10 Mixing audio streams to avoid clicks.

Figure 4.9 refers to the construction of one level of the beam tree (a reflection on the r_4 reflector, as already shown in Figure 4.6). In this portion of the beam tree, we describe the splitting of the generating beam (the one that is incident to reflector r_4) into five sub-beams (numbered from 1 to 5). Sub-beams 1 and 3 are “unbounded” beams (they keep propagating undisturbed), while subbeams 2, 3 and 5 are “bounded”, as they end up onto reflectors r_2 , r_1 and r_3 , respectively. The process is repeated for each one of the “bounded” sub-beams, until the number of reflections reaches a pre-fixed maximum or until the energy associated to the beam becomes negligible.

The beam-tree lookup block of Figure. 4.8 generates the filter taps according to the beams that reach the listener’s location. This update operation is fast and can be performed frequently. Through this lookup process we obtain the direction of arrival (DOA), the path length and the list of the reflectors encountered so far. This phase requires all of the tree nodes and leaves to be visited, and the relative beam to be tested for listener’s inclusion. If the listener’s location falls within the beam area, the DOA and the path length are computed. From there we can go down the beam tree towards the root, in order to obtain the list of reflectors encountered along the propagation of the wavefront.

Since it is impossible to produce one stream for every DOA, we group together angular intervals in order to generate a limited number of audio streams, one per interval. Each one of these angular intervals will be attributed a separate delay line (a FIR filter). The taps are computed considering all path lengths and relative attenuations.

The filter bank whose parameters are generated at the previous step, constitutes the auralization algorithm, which generates the directional streams from an anechoic (dry) source. These streams are then mixed for stereo headphone auralization using a HRTF. In our implementation we used OpenAL to mix a user-defined number N virtual “equivalent sources” placed in circle (or sphere) around the listener (to simulate N discretized DOAs). The listener’s head orientation is accounted for, only in this last step. In the following table we summarise what kind of computation is required for source-listener geometry changes:

Type of Motion	Recomputation required
Source location	Beam Tree re-computation
Listener's location	Beam Tree lookup and Filter Bank update
Listener's orientation	No re-computation required

Every time a re-computation is required, the taps in the delay lines are changed, and this can cause some clicking sounds to be produced in the output. This can be avoided by smoothly interpolating parameters in place of just switching them. A simple method to avoid clicks is a linear interpolation in the audio streams generated by using the "old" and "new" filters (see Figure 4.10).

ACKNOWLEDGEMENT

The work presented was developed within VISNET, a European Network of Excellence (<http://www.visnet-noe.org>), funded under the European Commission IST FP6 programme.

BIBLIOGRAPHY

[1] H.-Y. Shum and L.-W. He, "A review of image-base rendering techniques," in Proc. Visual Computation and Image Processing (VCIP), Perth, Australia, June 2000, pp. 2-13.

[2] R. Szeliski and H.-Y. Shum, "Creating full view panoramic image mosaics and environment maps," in Proc. Computer Graphics (SIGGRAPH), 1997.

[3] S. E. Chen, "QuickTime VR - An Image-Based Approach to Virtual Environment Navigation," in Proc. Computer Graphics (SIGGRAPH), Los Angeles, USA, Aug. 1995, pp. 29-38.

[4] M. Levoy and P. Hanrahan, "Light Field Rendering," in Proc. Computer Graphics (SIGGRAPH), New Orleans, LA, USA, Aug. 1996, pp. 31-42.

[5] H.-Y. Shum and L.-W. He, "Rendering with Concentric Mosaics," in Proc. Computer Graphics (SIGGRAPH), Los Angeles, USA, Aug. 1999, pp. 299-306.

[6] I. Feldmann, P. Eisert and P. Kauff, "Extension of Epipolar Image Analysis to Circular Camera Movements," in Proc. International Conference on Image Processing (ICIP), Barcelona, Spain, Sep. 2003.

[7] P. Eisert, "3-D Geometry Enhancement by Contour Optimization in Turntable Sequences," in

Proc. International Conference on Image Processing (ICIP), Singapore, Oct. 2004.

[8] ISO/IEC 14496-1:2002, Coding of audio-visual objects: Part 1: Systems, Document N4848, Mar. 2002.

[9] C. Grünheit, A. Smolic and T. Wiegand, "Efficient Representation and Interactive Streaming of High-Resolution Panoramic Views," in Proc. International Conference on Image Processing (ICIP), Rochester, USA, Sep. 2002.

[10] P. Eisert, "Model-based Camera Calibration Using Analysis by Synthesis Techniques," in Proc. Vision, Modeling, and Visualization VMV'02, Erlangen, Germany, Nov. 2002, pp. 307-314.

[11] P. Eisert, E. Steinbach and B. Girod, "Automatic Reconstruction of Stationary 3-D Objects from Multiple Uncalibrated Camera Views," IEEE Transactions on Circuits and Systems for Video Technology, vol. 10, no. 2, pp. 261-277, Mar. 2000.

[12] S. Mann and R. Picard, "On being 'undigital' with digital cameras: Extending Dynamic Range by Combining Differently Exposed Pictures," in IS&T's 48th Annual Conference, Washington, May 1995, pp. 422-428.

[13] P. E. Debevec and J. Malik, "Recovering High Dynamic Range Radiance Maps from Photographs," in Proc. Computer Graphics (SIGGRAPH), 1997.

[14] M. D. Grossberg and S. K. Nayar, "High Dynamic Range from Multiple Images: Which Exposures to Combine," in Proc. ICCV Workshop on Color and Photometric Methods in Computer Vision (CPMCV), Oct. 2003.

[15] J. Royan, O. Aubault, C. Bouville, and P. Gioia, "Improved Geo-Visualization Methods," in Proc. Computer Graphics (SIGGRAPH), Los Angeles, Aug. 2004.

[16] M. Tomaszewski, K. Ignasiak, "A Tutorial on the Basics of Extensible MPEG-4 Textual Format (XMT)," in Proc. Special VISNET Session at Polish National Conference on Radiocommunications and Broadcasting (KKRRiT 2004), Warsaw, Poland, June 2004, pp. 29-42.

[17] R. Balter, P. Gioia, L. Morin, F. Galpin, "Scalable and efficient coding of 3D model extracted from a video," in Proc. Second International Symposium on 3D Data Processing, Visualization 3DPTV'04, Thessaloniki, Greece.