CAA 2009 Williamsburg

ISReal: Advanced Computer Graphics Methods for Archeology

Michael Repplinger[12], Alexander Löffler[1], and Philipp Slusallek[12]

[1] Saarland University, Computer Graphics Lab, Germany
[2] German Research Center for Artificial Intelligence (DFKI), Agents & Simulated Reality, Germany

Abstract

The computer has become an indispensable tool in modern archeology for measuring, recording, organizing, searching, discussing, and presenting results. Particularly important is the use of realistic and interactive 3D graphics technology: It allows to accurately capture and present the archaeological evidence within its current 3D environment as well as to create 3D models of the historic location itself. Different possible reconstructions can be discussed and compared to the existing evidence within the same model, thus allowing to better evaluate scientific hypotheses. Given such interactive 3D models, we can then apply a multitude of other simulation techniques from computer graphics and other fields (e.g., illumination, acoustic and architectural simulations) to try to recreate the historic reality. Using the Internet, we can interconnect research groups on different locations but also make this data widely available and more accessible to non-experts.
In this paper we present the ISReal project together with its first results which allows for dynamically create, large-scale, and physically realistic simulated realities with systematic semantic annotations.

Key words: *Interactive 3D archeological models, Realtime Ray Tracing, Collaborative Reviews, Server-based rendering*

1 Introduction

Even though very promising, current computer graphics technology has also been a severely limiting factor for visualizations in the archeological context. The realism of the virtual artifacts was often minimal, the striving for realtime performance greatly limited the size of the historic models, and – not the least – significant graphics expertise has often been required to use existing tools. In this paper we present ISReal (Intelligent Simulated Reality), a joint research project between the German Research Center for Artificial Intelligence (DFKI) Saarbrücken and the Computer Graphics Lab of Saarland University. Using standards wherever possible (e.g., X3D) and extending them where necessary, ISReal allows for interactively handling massive 3D models with very fine detail and a high degree of realism, including physically accurate materials and illumination. Our models are extended to include pieces of semantic information accessible in the scene and multi-agent technology to describe the functionality of devices and behavior of characters.

The main goal of the ISReal project is to develop the base technology as well as the tools to create dynamic, large-scale, and physically realistic simulated realities with systematic semantic annotations, supporting intelligent behavior of entities driven by multi-agent technology. To create scenarios that are hard to distinguish from reality, convincing modeling, simulation, and presentation of virtual environments are required. This may be application-dependent but typically includes realistic visualization and rendering,

accurate algorithmic simulation of behavior, functionality, and materials. Most importantly, simulated realities need to be robust and reliable. Decision makers must be able to verify key properties of the model and need to trust and intuitively understand presented results.

The use of simulated reality in industry has been continuously increasing over the years and will see a tremendous acceleration as it becomes computationally much more capable, covers more aspects of reality, and will be readily available through upcoming highly-parallel hardware and 3D interaction devices. In many areas of research, virtual prototyping speeds up release cycles drastically by enabling reliable early decision making before or during the construction process. Costly error correction later in the process is thus avoided.

Virtual technology enables the training of people in yet non-existing environments and allows to communicate and collaborate about ideas as if they were already real. For relevant industrial scenarios virtual prototyping has to cover a number of heterogeneous aspects, which require specialized treatment by known modeling, simulation, and analysis techniques but on the other hand have to be integrated into a common model. For example, a comprehensive world model for scientific use and research has to fulfill the following requirements:

1. Provide accurate geometric and optical information for high quality immersive rendering that supports interaction and decision making.
2. Include parts that exhibit a precisely defined mechanical behavior enforced by the laws of physics.
3. Use of agent-based techniques to model the logical behavior of objects (including virtual humans and their ergonomics), which may interact with those controls and the rest of their virtual environment.
4. Provide interfaces to the virtual world for a variety of potential simulation algorithms, enabling services described by semantic service descriptions.

Most of these aspects alone can be covered by existing techniques that have been successfully investigated at DFKI and elsewhere. The main conceptual challenge of the ISReal is therefore not so much the continuation of this disciplinary research but primarily to come up with a design for an integrated conceptual framework that integrates these techniques and an implementation platform for future industry projects and interdisciplinary research.

Particularly, we will concentrate on the interactive rendering of highly complex and highly realistic models using realtime ray tracing. Therefore, we demonstrate a collaborative, server-based approach that allows multiple users to explore and interact with large and realistic 3D models from different scales of client systems, ranging from mobile devices up to multi-wall Virtual Reality installations. A distributed group of users should be able to share a common view onto the model as well as an audio connection for collaborative sessions. Multiple groups or individuals can independently explore models offered on large-scale servers, be that independently or in a collaborative effort. In addition to this, multimedia information can be seamlessly integrated into the world model which is required to suit the needs for the ever-increasing realism of those worlds.

This paper is structured as follows: In Section 2, we give an overview of the ISReal architecture and identify key components. Section 3 explains the foundations at the graphics end, which enable us to realistically visualize even very large models and presents related work. Section 4 continues by presenting our approach to server-based rendering, i.e. to spread the simulation workload across multiple machines. We show how the presented components interact in a multi-user setup, and how the distinct interactions are fused together to a collaborative rendering session. In Section 5 explains the media convergence inside ISReal, and how we unify 3D and multimedia processing in the very same software architecture. Finally, Section 6 concludes the paper and gives an outlook to future research challenges.

2 ARCHITECTURE OF ISREAL

The fundamental structure of the ISReal architecture is depicted in Figure 1. The ISReal architecture consists of three fundamental components.

1) World Model: The central part of ISReal is the *world model*, which maintains the representation of the entire 3D scene. The pure geometrical scene description is enriched by semantic metadata (e.g., physical properties or textual descriptions of buildings), which is attached directly to the geometry it belongs to. The software component that for covering the central world model is the Real-Time Scene Graph (RTSG)[1], developed at Saarland University. RTSG implements the ISO standard X3D[2] as model for 3D geometry and metadata. In particular X3D has been designed with extensibility in mind and thus will form the basis of our work. Furthermore, it is supported by most modeling tools.

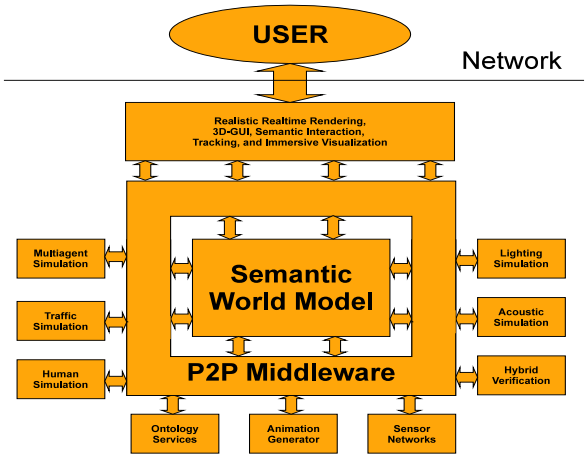

Figure 1. The architecture of ISReal consisits of three essential components. The *semantic world model* represents the entire 3D scene enriched by semantic meta data. *Services,* e.g., for rendering the scene, operate on the world model and communicate with the world model through a *P2P middleware layer*.

2) Services: *Services* are build around the world model and perform a sepcific functionality, e.g., visualization, interaction, or simulation, on the world model or interact with other services. The important aspect is that a service is strictly separated from the world model itself. Especially the visualization components inside ISReal are conceptually separated from the world model. In contrast to other scene graphs, which are built closely around the actual rendering technique, RTSG supports arbitrary rendering plugins that may operate on the scene it holds. This way, RTSG is able to combine several visualization techniques, e.g., *rasterization*, or the highly realsistic *ray tracing*, but also interaction, and simulation techniques on the same scene.

Another important aspect of the architecture is that services can depend on each other. For example, the tasks a multi-agent system might perform the simulation of the behavior of virtual characters traveling the scene: here, the path and motion planning of the characters might be done by AI engines like *Xaitment Map and Move*[3], while the rendering engine RTfact[4] takes care of visibility and rendering. In turn, our new Monte-Carlo-based lighting and acoustic simulation may again use RTfact functions for determining visibilty inbetween locations of auditory interest.

Together, the world model communicates with a variety of different services, all of which operate directly on the scene graph though the standard

[1] Dmitri Rubinstein et al.: "RTSG: Ray Tracing for X3D via a Flexible Rendering Framework". In: *Proceedings of the 14th International Conference on 3D Web Technology (Web3D Symposium '09)* Darmstadt, Germany, June 16-17, 2009.

[2] X3D Homepage, "X3D. Extensible 3D Graphics for Web Authors", Don Brutzman, and Leonard Daly, and Morgan Kaufmann, http://x3dgraphics.com/

[3] Xaitment GmbH Homepage. "Xaitment Map and Move", Xaitment GmbH, http://www.xaitment.com

[4] Georgiev, Iliyan and Philipp Slusallek: "RTfact: Generic Concepts for Flexible and High Performance Ray Tracing". In: Proceedings of the IEEE/EG Symposium on Interactive Ray Tracing 2008, pp. 115-122, Los Angeles, USA, August 9-10, 2008.

X3D *Scene Access Interface* (SAI)[5], and standard-conformant extensions thereof. The seemingly static world model thus becomes dynamic through the simulations operating on the scene graph and inbetween each other in order to realistically compute the real-time dynamics of the world.

3) P2P Middleware: To enable the communication of all these components in a generic fashion, and to distribute all services within the network, we introduce a *middleware layer* separating the simulation modules from the world model core, enabling not only communication with RTSG, but also inbetween each other. The middleware layer binds simulation modules together and provides semantic peer-to-peer services for selection, configuration, coordination of, and communication between many, often distributed components. The communication architecture is based on the *Network-Integrated Multimedia Middleware* (NMM)[6], which provides a *unified messaging system* to exchange arbitrary data between different components.

The important aspect of using a unified messaging system here is to enable communication between different and partial incompatible components. Thus, integrating new components into ISReal mainly refers on integrating the messages of the communication model of the component into the unified messaging of NMM. This in turns automatically allows for exchanging information between other integrated components, either running locally or distributed in the network.

Additonally, NMM allows synchronized communication and either central or peer-to-peer service selection and coordination. The multimedia capabilities of NMM will furthermore be used for unifying multimedia content delivery and 3D computer graphics, all of which will be detailed on in Section 4.

Within the context of the ISReal project, *intelligent simulation* in the form of networked simulation modules with components for *realistic visualization* are combined that were mainly developed at Saarland University and the new working group "Agents and Simulated Reality" at the DFKI. In general, ISReal enables an unprecedented degree of realism in both simulation and visualization algorithms, all of which integrated in a performant and highly modular platform. The rest of this paper describes the service for interactive and realistic rendering and visualization in more detail.

3 Related Work

Since this paper concentrates on the interactive and realistic rendering and visualization services of ISReal, we also limit the related work to this topic. In general, there are two principal ways to render images of 3D environments: *rasterization* and *ray tracing*. Rasterization has been the strongly dominant technique for interactive application (likely more then 99.9% of the market), receiving constant hardware and software improvements. However, its use of the primitive operation to project isolated triangles imposes some severe limitations that prohibit this approach from accurately computing even simple optical effects like shadows, reflection, and refraction. Developers and content creators have to use many "tricks" to achieve the desired effects.

As advanced visual effects, reliability, and ease of use (to enable user generated content) become more and more important, ray tracing has received a lot of attention recently. Ray tracing is a physically based algorithms that follows the path of photons backwards from the virtual camera towards the light sources. While ray tracing has been able to generate more realistic images from simple description of geometry and material properties, its slow performance (often hours per frame) has limited adoption to the realm of off-line rendering.

The idea of realtime ray tracing, which is required

[5] Web3D Homepage, "X3D Scene Access Interface (SAI)", http://www.web3d.org/x3d/specifications

[6] Marco Lohse et al.: "Network-Integrated Multimedia Middleware (NMM)". In: *Proceedings of ACM Multimedia 2008 (ACM MM 2008)*, Vancouver, Canada, October 27-31, 2008

to achieve interactivity within a 3D simulations, came up initially on large supercomputers[7,8]. New hardware architectures such as the Intel Larrabee processor and the quickly evolving GPUs widen the degrees of freedom for future designs by allowing optimizing not only the algorithms and their implementation, but also suggesting new hardware features and extensions.

However, the development of realtime and scalable lighting simulation solutions that robustly cover at least the major illumination effects as well as interactive rendering of large scenes, requires developing highly optimized and specialized ray tracing engines.

Despite this tremendous hardware progress in rendering power, there are and will always be applications that require distributed configuration. This can be due to the need for high-resolution output via many tiled displays connected via a number of different computers.

There has also been a long history of scene graph libraries in computer graphics to describe 3D environments in an application independent way. Most of them have been proprietary designs but some of them have eventually been made open, e.g. Performer[9], or OpenInventor[10]. VRML and later its successor X3D mainly described a file format but also specify a runtime environment and can thus already be considered world models, while the popular Collada[11] is exclusively an data exchange file format.

RTSG supports the Object-Oriented Graphics Rendering Engine (OGRE)[12] as a rasterization back-end supporting both platform-independent OpenGL[13] graphics and the Windows-only Direct3D[14]. On the ray tracing side, RTSG supports the legacy OpenRT[15] backend, as well as the state-of-the-art RTfact engine for realtime ray tracing[16]. RTfact is currently under active development at Saarland University and provides a modular system combining adaptability to different hardware architectures with realtime rendering performance.

Related X3D projects are the commercial BS Contact by the German company Bitmanagement, the instantreality toolkit by IGD, and a small selection of OSS tools. There has been a considerable amount of research on tone mapping operators but very few address our main concern in accurately mapping images according to perceptional issues. The main exception is the first

[7] Michael J. Muuss: "Towards Real-Time Ray-Tracing of Combinatorial, Solid Geometric Models". In: *Proceedings of BRL-CAD Symposium '95*, June 1995.

[8] Steven Parker et al.: "Interactive Ray Tracing for Isosurface Rendering". In: *IEEE Visualization 1998*, pp. 233–238, October 1998.

[9] SGI Performer Homepage. "Performer", SGI, http://www.vis-sim.com/performer/

[10] Wikipedia. "OpenInventor", Wikipedia, http://en.wikipedia.org/wiki/Open_Inventor

[11] Khronos Group Homepage. "Collada", Khronos Group, www.collada.org

[12] OGRE Homepage, "Object-Oriented Graphics Rendering Engine (OGRE)", http://www.ogre3d.org

[13] Khronos Group Homepage. "OpenGL", Khronos Group, http://www.opengl.org

[14] Microsoft Homepage. "DirectX", Microsoft Inc., http://www.microsoft.com/windows/directx

[15] Andreas Dietrich, Ingo Wald and Philipp Slusallek: "The OpenRT Application Programming Interface: Towards a Common API for Interactive Ray Tracing". In: *Proceedings of the 2003 OpenSG Symposium,* pp. 23-31, 2003.

[16] Iliyan Georgiev, and Philipp Slusallek: "RTfact: Generic Concepts for Flexible and High Performance Ray Tracing". In: *Proceedings of the IEEE/EG Symposium on Interactive Ray Tracing 2008*, pp. 115-122, Los Angeles, USA, August 9-10, 2008

such paper in graphics[17], which describes the basic setup for perception based tone-mapping and will form the basis for our work. We expect to extend the model of human perception and also include the steering of the ambient illumination in order to better control adaptation and related effects.

4 RENDERING AND VISUALIZATION

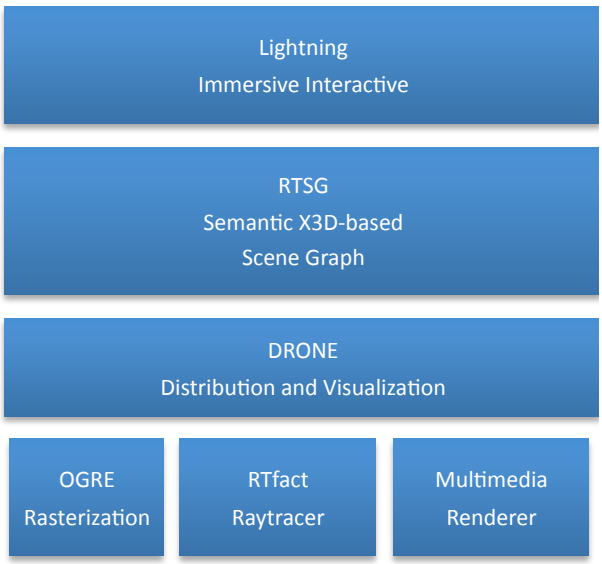

Figure 2. The general structure of the rendering and visualization service consists of the VR system Lightning that allows immersive interaction with the world model represented by an RTSG scene. The DRONE component is responsible for distributed rendering and visualization using rendering engines integrated into RTSG.

A key service of the ISReal platform is the realtime realistic rendering and visualization service. Through it, even very large models with highly detailed material properties and lighting effects provide a very high level of realism. Despite the tremendous progress in rendering power, these kind of applications still require distributed configurations employing several machines for rendering and display. The general structure of this service can be seen in Figure 2. This service consists of the existing VR system *Lightning*[18] that allows immersive interaction with the world model represented by an RTSG scene. The DRONE component, which is described in detail int the rest of this section, is responsible for distributed rendering. DRONE uses the rendering engines integrated into RTSG for rendering the world model. The rendering engines consist of OGRE for rasterization and RTfact for ray tracing. The *Multimedia Renderer* is responsible for all kind of multimedia processing, e.g., audio and video processing, within the world model. The Multimedia Renderer together with required X3D extensions are described in detail in Section 5.

4.1 APPLICATION SCENARIOS

Before we handle details of our rendering architecture, we will discuss several typical application scenarios the rendering and visualization service supports as well as available solution strategies. The first application scenario (AS1), called *single-screen rendering*, comprises presenting rendered images on a single screen while using multiple systems for rendering. The major demand of flexibility for (AS1) is the possibility of using available systems in the network both for rendering and displaying a scene, while being independent of the network infrastructure connecting them. A distributed middleware like NMM provides network transparency, which in turn allows transparent access to distributed objects, and aids in achieving this high degree of flexibility.

Another required aspect of (AS1) is the possibility to use different rendering techniques such as ray tracing or rasterization, all working on the same scenes. Available distributed rendering frameworks like WireGL[19], Chromium[20], and

[17] Tumblin, Jack and Holly Rushmeier: "Tone Reproduction for Realistic Images", *IEEE Computer Graphics and Applications*, 13(6), pp. 42-48,1993.

[18] Bues, Matthias, Tim Gleue, and Roland Blach: "Lightning: Dataflow in motion". In: Latoschik, Marc Erich (Ed.) u.a.: Software engineering and architectures for realtime interactive systems (SEARIS): IEEE VR 2008 Workshop, March 9, 2008, Reno, Nevada, USA. Aachen: Shaker, 2008.

[19] Greg Humphreys et al.: "WireGL: A scalable graphics system for clusters". In: *SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (2001)*, pp. 129–140, 2001.

Equalizer[21] are limited to the rasterization approach.

The second application scenario (AS2) called *multi-screen rendering*, extends (AS1) by splitting the resulting frame and presenting it on multiple displays simultaneously and fully synchronized. This is required for display walls, for example for large-scale terrain or industrial visualization. These kinds of applications need high resolutions beyond that of a single monitor or projector, as the display of details is a vital factor in these domains. The major desired aspect of (AS2) is the flexibility of combining multiple displays as if they were a single one. For (AS2), we thus need distributed synchronization to present an image simultaneously on all displays: hardware-based solutions, e.g., using the genlock signal of the video output of special graphics cards allow for exact frame synchronization, while software-based solutions like NTP, are able to synchronize PCs over the Internet with a few milliseconds of variance. We believe a flexible rendering framework should be able to support arbitrary synchronization mechanisms of both kinds.

The third application scenario (AS3), called *multi-view rendering*, comprises the presenting of multiple views of the same scene at the same point in time. For example, this is required for stereo imagery required for Virtual Reality installations like the CAVE. The main requirement for a framework is the flexibility to add an arbitrary number of different views and display surfaces. In this case, we need a unified way to process, transmit and synchronize multiple video streams from multiple viewpoints.

The fourth application scenario (AS4), called *remote rendering*, covers situations where rendered images have to be transmitted through a network connection with limited bandwidth, often because the original data sets have to stay at a controlled and secure location. The main demand for of (AS4) is the ability to add different post processing steps, e.g., the encoding of rendered images before a network transmission. Here, the application of a *distributed flow graph* within our rendering shows its full potential by providing the means to transparently insert new processing elements in the data processing pipeline.

The last application scenario (AS5) is *collaborative rendering*, an arbitrary number of combinations of the previously described scenarios. An ideal system scenario should allow both, for example, a large control center with tiled display walls, and simultaneously thin clients only receiving some important aspects of large rendered images. This is especially interesting for collaborative work where people on different locations have to work with the same view of a scene. This requires that the application is able to share the same rendered images between multiple users while each user may be able to interact with the scene.

4.2 Overview of DRONE

DRONE builds on the Network-Integrated Multimedia Middleware (NMM) and is responsible for distributed processing. Even though NMM is especially designed for multimedia processing and does not explicitly consider rendering, its generic approach for distributed media processing perfectly fits the requirement of flexibility the framework should provide. The DRONE framework builds on top of an NMM flow graph consisting of custom processing nodes supplemented by existing nodes of core NMM.

NMM uses the concept of a *distributed flow graph* for distributed media processing, which again perfectly fits the requirement of flexibility we defined for the framework. This approach provides a strict separation between media processing and media transmission as well as a transparent access to local and remote components. The *nodes* of a distributed flow graph represent specific operations (e.g., rendering, or compressing images), whereas *edges* represent the transmission between those nodes (e.g., pointer forwarding for local connections, or TCP for a network connection). Nodes can be connected to each other via their

[20] Greg Humphreys et al.: "Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters. In: *SIGGRAPH '02: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques (2002)*, pp. 693–702, 2002.

[21] Stefan Eileman, and Renato Pajarola: "The Equalizer Parallel Rendering Framework". Tech. Rep. IFI 2007.06, Department of Informatics, University of Zurich, 2007.

input jacks and *output jacks*; depending on the type of operation a node implements, their numbers may vary. Prerequisite for the successful connection of two nodes is a common *format*, which must be identical for the output jack of the predecessor node and the input jack of the successor node to be connected. NMM incorporates a unified messaging system, which allows sending control events together with multimedia data from sources to sinks, being processed by each node in-between. The important aspect of NMM is, that nodes and edges are represented as first-class objects to the application, which allows the application to configure and control media processing and transmission transparently, for instance by choosing a certain transport protocol from the application layer.

The ability of NMM to distribute nodes arbitrarily in the network, but still access them transparently from within an application allows the placement of application sub-tasks on arbitrary hosts, enabling high flexibility and efficient use of a cluster. Below, we will explain the basic nodes of DRONE in more detail; Figure 3 shows them assembled to a simple, single-display flow graph.

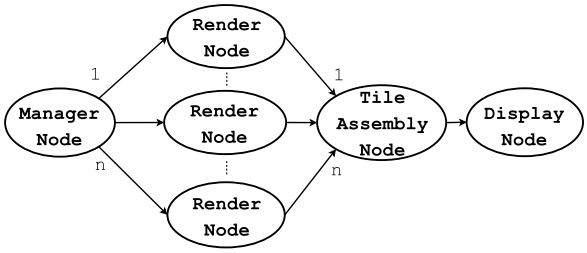

Figure 3. This flow graph shows the general idea of distributed and parallel rendering in DRONE using *n* rendering nodes to render to a single output device.

- **`RenderNode`:** A *render node* performs the actual rendering of a scene description to a 2D image. Key principle of DRONE is rendering a single frame distributed on multiple render nodes in the flow graph. All render nodes have access to an identical copy of the scene graph. This node renders the assigned scene to a memory buffer – in this case an NMM buffer – for further processing. Within our implementation, RTSG is reused for this node because it specifies a unified interface to select and configure arbitrary rendering back ends and can be extended by a new rendering back end just by implementing the simple interface. Furthermore, it already includes the rendering back-ends RTfact, and OpenRT for ray tracing, as well as the rasterization-based OGRE.

- **`ManagerNode`**: The single source node of the DRONE flow graph is called *manager node*; its job is to distribute the workload of rendering an image to the available render nodes, which are attached downstream. The manager node distributes the workload between render nodes by splitting the frame to be rendered into many *frame tiles* and assigning them dynamically to render nodes.

- **`DisplayNode`**: A *display node* constitutes a sink of the flow graph, and simply presents any incoming image buffer synchronized according to its timestamp. Display nodes are part of core NMM and usually platform-dependent: for example, an `XDisplayNode` would be used on a Unix platform running the X-Window system. A display node can also be a window in a web browser that then presents the rendered image as part of a web page.

- **`TileAssemblyNode:`** A tile *assembly node* in general can receive frame tiles from all rendering nodes, and assembles them to a composite image buffer. As there is one dedicated tile assembly node for each downstream display node, the nodes receive only those tiles of the rendered image stream that are relevant for the particular display node they precede. The information about the frame area to be presented is part of the connection format between a render node and tile assembly node. Thus, a render node knows the rendered tiles that have to be forwarded to a specific tile assembly node. This entails no communication overhead for parts of the image that would not be displayed afterwards.

4.3 Architecture of DRONE

Based on the NMM flow graph components presented in Section 4.2, DRONE provides its functionality to the application in the form of *processing blocks*, which bundle their underlying modules and provide high-level access to an application developer. Furthermore, *composite blocks* allow the application to group different

processing blocks that can be treated in the same way as a single processing block afterwards. Below, we will use the application scenarios presented in Section 4.1 as a guide through the specific design decisions of the framework, and explain how the different processing and composite blocks fit together.

Single-Screen Rendering (AS1): The primary processing block, which occurs in every DRONE application, is the *rendering block*. It contains those NMM components that are responsible for rendering a two-dimensional image from a 3D scene description: In particular, it consists of a single manager node and at least one render node. NMM enables these nodes to be transparently distributed across physical hosts in the network to spread the workload as uniformly as possible. Together, all render nodes take care of rendering a frame. The distribution among nodes is done by tiling the frame, and assigning single frame tiles to separate render nodes. The rendering block is connected to at least one *presentation block*, which combines the tiles and displays the frame on an actual physical display device.

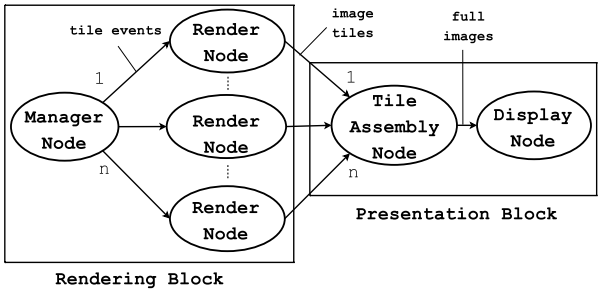

Figure 4. DRONE encapsulates the flow graph in different basic processing and composite blocks. The rendering block includes the manager node as well as all rendering nodes. The presentation block includes all remaining nodes required to present rendered images.

A presentation block is a composite block that can be extended by additional nodes for post processing but contains at least two NMM nodes: a tile assembly node, and a display node. All those tiles of the rendered frame that are sent from the render nodes to the corresponding assembly node have to be displayed by the succeeding display node. The type of a display node can be explicetly configured because it strongly depends on the application scenario. For example a display node can represent a graphic board of a PC that is connected to a projector, or a window within a web browser. Since information about the specific view is part of the connection format between a render and tile assembly node, each render node knows which frame tiles have to be sent to which tile assembly node. In the trivial case of having just one display (depicted in Figure 4), the tile assembly node receives all tiles of each frame.

The manager node of the rendering block is responsible for load balancing because it sends information about the next tile to be rendered as so called *tile events* to its successive render nodes. A simple approach would be to use a round-robin distribution scheme. NMM however also has transparent access to all local and remote nodes. Therefore, the manager node can control the distribution of tiles to render nodes, and assigns a new tile event only if the previous tile event has been rendered.

This very simple scheduling approach leads to an efficient dynamic load balancing between the render nodes, because render nodes that finish rendering tiles earlier, do receive new rendering event earlier as well. This approach automatically considers differences in rendering time that can be caused by different scene complexity, or different processing power of different rendering machines. Moreover, NMM informs the manager node about a failed network connection to a render node, so that the manager node no longer tries to send tile events to this node.

Multi-Screen Rendering (AS2): The general idea to support applications that need to present rendered images on multiple screens can be seen in Figure 5. The application specifies multiple presentation blocks as well as the partial frame configuration to be displayed by each block. All these presentation blocks are then connected to the same rendering block by the framework. Synchronized presentation of rendered images is achieved by adding presentation blocks to a specialized composite block, called *synchronization block*. This block connects a *synchronizer* component to all display nodes of child presentation blocks. The synchronization block is then connected to the rendering block, while the framework automatically connects all presentation blocks to the rendering block, and in

doing so adds the information about the partial frame to be presented as part of the connection format between render nodes and tile assembly nodes. In summary, any rendered frame can be presented on any of the screens simultaneously; either in full or in part for realizing a video wall setup. For scenarios where blending between adjacent projectors is required, the overlap between presentation blocks can be freely adjusted.

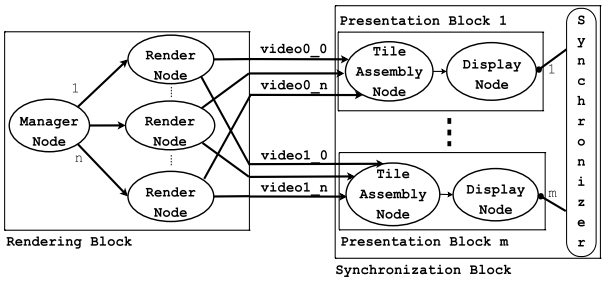

Figure 5. DRONE allows combining multiple independent presentation blocks (e.g., for realizing video walls). The synchronized presentation of rendered images is achieved by adding these presentation blocks into a synchronization block which then connects a synchronizer to these presentation blocks.

The most important aspect in this context is that the synchronization itself is encapsulated into a specific implementation of a composite block and thus hidden from the application or presentation blocks. This allows to use different synchronization mechanisms, e.g., a global network time, or the genlock signal of used graphic cards, simply by providing a different implementation of a synchronization block. The synchronizer realized in the DRONE framework, which is used by default, calculates and propagates the latency for sending rendered images to all presentation blocks and does not depend on a specific hardware. The used communication protocol and algorithms are described in detail in the respective technical report of our group[22].

[22] Michael Repplinger et al.:"URay: A Flexible Framework for Distibuted Rendering and Display". Tech. Rep. TR-2008-01, Computer Graphics Group, Department of Computer Science, Saarland University, Germany, 2008.

Multi-View Rendering (AS3): The general approach of DRONE to support rendering multi-view images for stereo or Virtual Reality scenarios, is to treat the multi-view setup as a special case of (AS2) in which each eye is conceptually represented as a separate presentation block. Of course, hardware components that are represented by display nodes (e.g., a video projector), have to be adjusted correctly to generate a correct stereo image on a screen – this is, however, out of the scope of this paper.

From a technical point of view, a stereo image consists of two independent images to be generated, showing the same scene from slightly different viewpoints for each eye. Those two independent images have to be treated as a single frame and to be presented at the identical point in time, which has to be considered by the synchronization mechanism of the framework. Our synchronization mechanism is completely independent of the actual source of a frame, the viewpoint that creates it, and the time needed for rendering it. Therefore, the presented synchronization is also well-suited for stereo images or even multi-wall Virtual Reality installations.

The implementation of our manager node allows rendering an arbitrary number of viewpoints which is required for example for virtual reality installations. Notable in this context is that renderers integrated into our framework are automatically extended to support rendering multi-view stereo images, even if the original implementations did not consider this functionality at all. Again our framework greatly simplifies supporting different specific application scenarios as well as developing new rendering engines, as developers can focus on their specific implementation.

Remote Rendering (AS4): To enable sending a stream of rendered images across a high-latency network like the Internet and still enable an interactive manipulation of the rendered scene, the bandwidth of the rendered raw video stream has to be reduced drastically. The necessary reduction of the data rate is typically done by means of encoding the image stream before sending; for example using an MPEG-4 or H.263 video codec.

Besides encoding of the stream, one can imagine many more potential operations to be performed on the rendered images. For instance, a color correction of neighboring projections of a video wall setup, tone mapping or arbitrary other operations in pixel space.

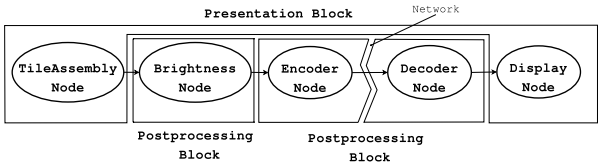

Figure 6. Extended presentation block: DRONE allows to add an arbitrary number of postprocessing blocks to a presentation block. In this example, we first adjust brightness and then encode rendered images before sending them through an Internet connection.

To enable all these scenarios, we allow the insertion of one or more *postprocessing blocks* into a presentation block. This is automatically supported by the framework because the presentation block is a composite block itself. Figure 6 shows a presentation block enhanced by two postprocessing blocks: one for brightness adjustment, and one for encoding and decoding of the stream. A postprocessing block with all its internal nodes is either inserted in front of the tile assembly node or between the tile assembly and the display node of any presentation block. Here, the application of a multimedia middleware like NMM, shows its full potential by providing the means to transparently insert new processing elements in the data processing pipeline.

Collaborative Rendering (AS5): The final application scenario to be covered by DRONE is the situation of multiple parties working on and interacting with one and the same rendering block, realizing a collaborative environment, as for example industrial collaborations in which 3D models are synchronously displayed to engineers in distinct offices around the globe. In terms of the DRONE framework, this scenario represents an arbitrary combination of (AS1) to (AS4) as presented above. As before, the framework configuration for (AS5) includes a single rendering block with potentially multiple presentation blocks attached.

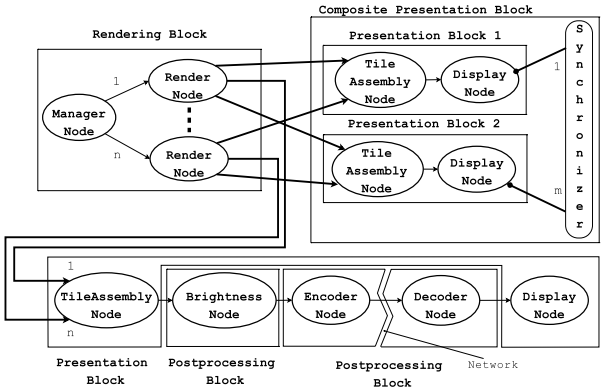

Figure 7. DRONE allows to connect several different presentation blocks to a single rendering block, even during runtime. Additionally, multiple presentation blocks can be combined into a composite presentation block, which then uses its own separate synchronizer.

The flexible architecture of DRONE allows, for example, to realize different encoded streams for each one of the presentation blocks, and arbitrary display setups for the participating parties. Moreover, different applications can access and share the same rendering block while adding their specific presentation blocks. For example, this could be used for application scenarios where users permanently enter or leave a collaborative virtual 3D environment.

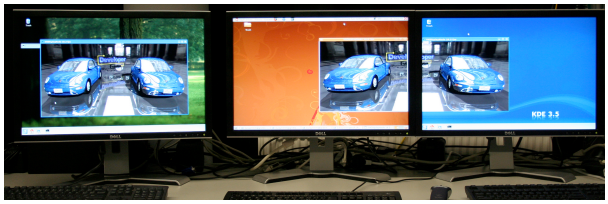

Figure 8. Real-time ray tracing simultaneously displayed on three presentation hosts, all of which are fully interactive and synchronized. The image is rendered on a cluster of six machines, and displayed in a single window on one computer (left) and a display-wall via two split video streams on two additional machines (right).

Moreover, DRONE allows to reconfigure a rendering block during runtime. This means that different applications can access and share the same rendering block while adding their specific presentation blocks. For example, this functioanlity is used for application scenarios where users

permanently enter or leave a collaborative virtual 3D environment. The possibility to realize this application scenario by combining and grouping previously presented results again shows the high degree of flexibility of our framework as well as the benefit for applications build on top of it.

4.4 User Interaction

Finally, we need to allow the user to interact with the world model in immersive and non-immersive visualization environments, modifying the world, steering simulation, and presenting and evaluating results. Since the scenarios described in Section 4 may incorporate an arbitrary number of rendering machines but also an arbitrary number users, we have to consider two issues: First, we potentially use multiple render nodes distributed in the network, and all of them render the scene of the world model. Thus each one has to be informed about interaction. Since we use multiple rendering nodes that render tile frames, we have to ensure that interaction is only performed between to *different* successive frames. Secondly, we have to ensure a correct handling of all kinds of interaction with the world model to avoid conflicts.

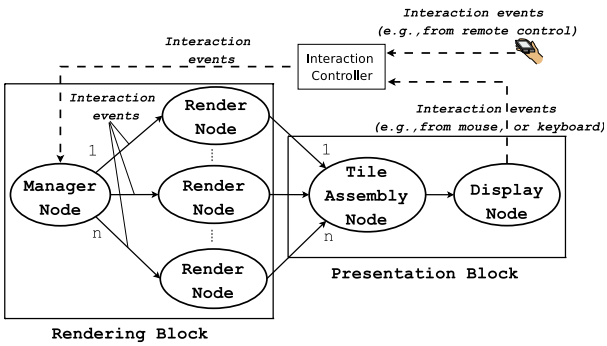

Figure 9. Interaction events are received from different components, e.g., the display node of a presentation block, or a remote control, and forwarded to the interaction controller. The interaction controller gains exclusive access to the manager node, forwards all interaction events to the manager node, which in turn sends them synchronized to all rendering nodes.

Within the context of user interaction, we differ between two kinds of interaction. The first one is *navigation* within the world model and refers to changing a specific *virtual camera* of the world model. A virtual camera specifies which parts of a world model is presented by a specific presentation block and is changed if a user navigates through the scene. The second one is *modification* of the world model and includes all kind of changes that affect the world model itself, e.g., changing the geometry, or adding and removing elements. The difference between these two kinds of interaction is that modifications on the world model need an exclusive access to the world model to avoid conflicts. In contrast to this, changes at least on different virtual cameras, e.g., by different users, allow the optimization to be done in parallel. However, if multiple users would like to change the same virtual camera, an exclusive access is also required to avoid conflicts.

The general idea in solving both issues can be seen in Figure 9. All interaction events are propagated to the manager node. The manager node in turn forwards all incoming events to all connected render nodes. The key point of this approach is that there exist only a single instance of this manager node, independent of the number of users and thus can be used to provide exclusive access for interacting with the world model. Furthermore, the manager node is responsible for sending tile events for rendering and knows when all frames for a apecific point in time have been rendered. Thus, it propagates interaction events only between tiles of *different* successive frames to avoid changes of viewpoint before the processing of frames for a specific point in time are fully completed.

To support both kinds of interaction, the manager node provides methods to gain either exclusive access to the world model or for a specific virtual camera. Each application provides a component we call *interaction controller,* which receives all navigation and modification events from the application, automatically requests the required access right, and forwards the events to the manager node. As can be seen in Figure 9, the interaction controller can receive control events from different components of the application, e.g., mouse and keyboard events from the display node of a presentation block, or via an infra-red remote control.

An important aspect of the implementation of user interaction and explicit access to the world model is to consider network failures, or failures within a client. Therefore, the manager node leases the

access right for a limited period of time. The interaction controller of an application has to explicitly refresh this lease by the manager node, if the application needs exclusive access for a longer period of time. If the leasing time is not refreshed within this time, the manager node automatically releases the access right so that other applications can gain access to send interaction events.

5 MULTIMEDIA EXTENSION FOR X3D

Applications based on ISReal are not separate entities anymore, but altogether form a larger collective. For the ever-increasing realism of those worlds, multimedia plays an important role to suit the needs of their users. This includes verbal communication between users but also accoustic simulation within construction and buildings. In order to realize virtual world scenarios based on the X3D standard, its capabilities to define and manipulate multimedia processing are insufficient and need to be extended. In this Section we describe the *Multimedia Renderer* that is part of the interactive rendering and visualization service of ISReal (see Figure 2), which we integrated into RTSG for an appropriate multimedia processing in X3D.

5.1 REQUIREMENTS

In a first step we define four requirements an integration of multimedia processing in X3D should fulfill:

1. **Explicit Specification of Media Processing:** From a high-level point of view, the drawbacks of current multimedia integration in X3D revolve around the inability to specify the actual media processing in more detail. Actions like post-processing of a video or audio stream, e.g. adjusting the brightness of a video, adding hall, delays or doppler effects to aduio data or routing the components of a multimedia stream to dedicated places of output in the scene, are currently impossible to realize completely within an X3D application. Applications involving multiple live and synchronized audio and video streams in a 3D world, for instance a virtual conference room including an archelogical model for discussing, or manipulating, should be possible to define inside a single X3D file.
2. **Streaming Media:** URL strings used to specify media items in X3D always describe *files*, even though the Internet is predestined for *streaming media*. The standard assumes that browsers completely download audio or video files specified by URLs before they start playing them. That is, they assume a *local* processing, even though URLs may very well specify a *remote* location. A scenario like listening to a live real-world audio stream, talking with different users inside a virtual-world is thus rendered impossible, because the media download will not ever be able to finish.
3. **Control of Components:** Continuing the mentioned "virtual conference room" scenario, users want to be able to control the cameras as well (e.g., zoom, or tilt). Back in the real world, this entails controlling the camera connected to a PC. X3D does not yet include a technique to route a click inside the virtual world to an outside event on an actual device. Even though X3D should indeed abstract actual hardware implementations to keep scenes portable, providing an interface to multimedia devices is essential for a model that combines aspects of a virtual and real world.
4. **Abstract Specification of Media Processing:** The last requirement is actually a result of the flexibility introduced by the former four. The theoretical ability to specify an entire multimedia processing pipeline within X3D should never entail that a user is *forced* to specify all the details every time she wants to include anything related to multimedia in her scene. Instead, we envision a principle of *scalable transparency* when specifying multimedia content: While it should be possible to dig deep into lower layers of processing, a basic multimedia usage should be intuitive and should involve only very few X3D nodes in the scene specification.

As a result of the four presented requirements, we will introduce extensions to the X3D specification, enable on the one hand full definition and control of a distributed multimedia flow inside an X3D application, and on the other hand the possibility to specify only very few necessary components.

5.2 MULTIMEDIA EXTENSIONS FOR X3D

In the following, we will use the requirements defined in Section 5.1 as a guideline to illustrate our integration of multimedia processing into the X3D language specification.

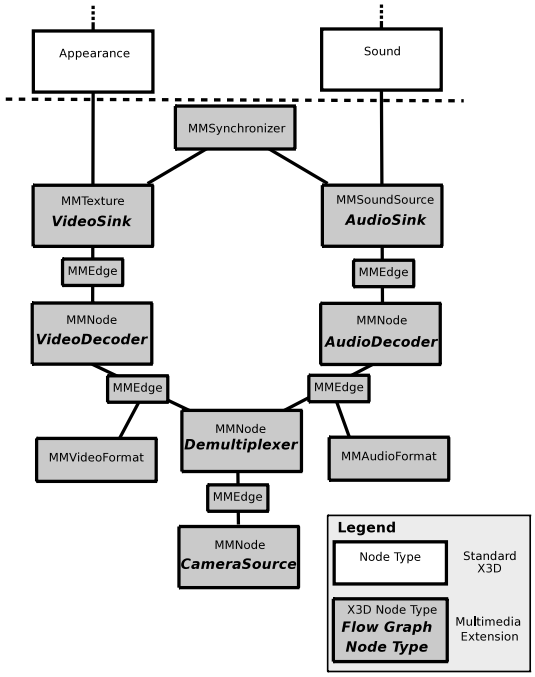

Figure 10. The general idea of integrating a multimedia flow graph into an X3D scene graph is to specify the flow graph including all components as a sub-tree within the X3D scene. Since the sink nodes of a flow graph represent the connection between produced media data and their representation within X3D, they are connected as children of standard X3D nodes. Thus, the source nodes of a flow graph represent leaf nodes of the full X3D scene graph.

To fulfill requirement (1), we first extend the X3D language specification by the concept of a multimedia flow graph. This approach provides the maximum flexibility to specify media processing as discussed in Section 5.1. The general idea of this multimedia subgraph in an X3D scene is depicted in Figure 10. To integrate a flow graph into an X3D scene graph we specify the entire flow graph including all components as sub-tree inside the X3D scene. The sink nodes of a flow graph represent the connection between produced media data and their representation within X3D. Therefore, the sink nodes of a flow graph are connected as child nodes of classic X3D nodes responsible for media specification. The source nodes of a flow graph represent leaf nodes within an X3D scene graph.

To clearly define multimedia processing within an X3D scene graph, we introduce the following new X3D nodes:

- **MMNode:** A *multimedia node* represents a single node of a multimedia flow graph within the X3D scene graph. For explicit specification of media processing it only includes a `nodeName` field to specify the name of the node within a flow graph.
- **MMEdge:** A *multimedia edge* represents an edge of the flow graph, connecting two multimedia nodes. Within the specification, a multimedia edge is always a child of an `MMNode`. The edge models the data flow from the node to its parent node. It includes all the required information to describe a flow connection between two multimedia nodes.
- **MMFormat:** A *multimedia format* describes the media format, which should be used on the connection between two multimedia nodes. Since there is a vast number of multimedia formats with very different properties that should be exposed to the scene, we defined an abstract node type `MMFormat` from which concrete format types inherit. To allow specifying only the major type of a format, we additionally introduce the `MMVideoFormat` and `MMAudioFormat` node types.
- **MMTexture:** A *multimedia texture* represents the video output of a flow graph and includes the video stream as a texture into the X3D scene graph. Therefore, this X3D node acts as a link between the flow graph and the X3D scene graph. To enable its usage as a texture within the scene, it inherits from `X3DTexture2DNode`, so it can be used within the standard X3D `Appearance` node.
- **MMSoundSource:** A *multimedia sound source* represents the link between audio output of media processing within the flow graph and a source of audio data within the X3D scene graph. Therefore, it inherits from `MMNode` as well as from the standard X3D type `X3DSoundSource`. It can thus be

seamlessly used in the `source` field of an standard X3D `Sound` node.

- **MMSynchronizer:** The multimedia synchronizer represents a synchronizer of the flow graph and has an arbitrary number of multimedia nodes as children. For specification, this node is required to define which multimedia nodes – or the corresponding nodes of the underlying flow graph, respectively – have to be synchronized by the utilized multimedia middleware.

To master requirement (2) as stated in Section 5.1, we extend our previously defined `MMNode` to be able to specify *distributed* multimedia nodes. Using a distributed multimedia middleware, the specification of remote components is limited to specify either a single location, or a set of locations from where to request a remote component. The benefits of our approach, using a distributed multimedia middleware are shown here: We integrate streaming functionality just by extending the specification of the `MMNode` type by a `location` field. The newly introduced field `location` stores an arbitrary number of locations which are used to request the corresponding flow graph node. If the field is not set, the underlying multimedia middleware searches all reachable systems to request the node. A single specific location can be set by adding exactly one entry to this field.

To meet requirement (3) as stated in Section 5.1, nodes of a flow graph should be controllable from within an X3D scene graph. Multimedia components are controlled by interfaces used from within a controlling application. In the case of X3D being the language of application definition, we need to introduce a concept of interfaces inside of X3D: We represent an interface by an X3D node derived from the abstract type `MMInterface`. A concrete implementation thereof then defines the fields necessary to work with the interface in an X3D context. For example, a derived `CameraInterface` would at least define one `zoom` field, which is readable and writable and thus can be used for routing within the X3D scene. Internally, the underlying multimedia technology has to provide the mapping to actual method calls to execute zoom command on the actual hardware. To assign the functionality of a multimedia interface to a multimedia node in X3D, we add the respective interface node to the `interfaces` field of a multimedia node, which may contain an arbitrary number of interfaces. Here, each interface represents a self-contained group of functionality exposed by its parent multimedia node. This enables multimedia nodes to be defined completely abstract, just as a collection of interfaces it supports. The resolution towards a concrete implementation is done by the multimedia system underneath.

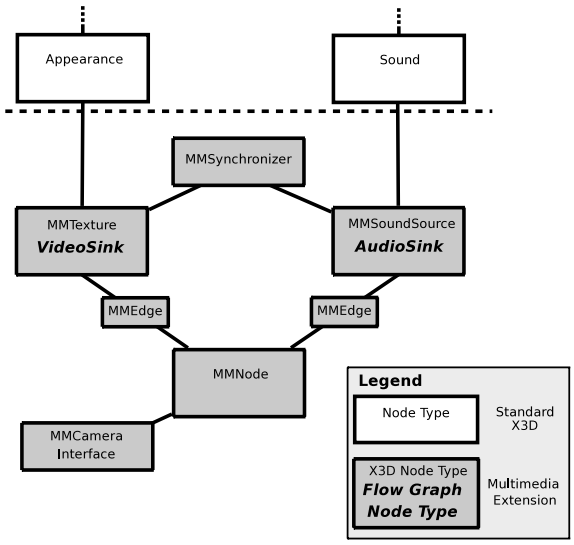

Figure 11. The abstract specification within an X3D scene enables an easy integration of distributed media processing and control. In this example for receiving data from a camera, only those aspects relevant for the scene graph developer are explicitly specified, the rest is configured automatically by the multimedia middleware.

To master the last requirement (4) as stated in Section 5.1, we integrate the concept of a *user graph* into X3D. A user graph is a high-level specification of a flow graph that only includes the essential nodes[23]. Here, only the key nodes, their

[23] Marco Lohse, and Philipp Slusallek: "Towards Automatic Setup of Distributed Multimedia Applications" In: *Proceedings of The 9th IASTED International Conference on Internet and Multimedia Systems and Applications (IMSA)*, pp. 359-364, 2005.

connections, and additional constraints are specified. The most important impact for the application developer is that a user graph can be independent of concrete nodes that are used. Instead, other aspects of nodes like interfaces are used to specify the required functionality of nodes. This user graph is then automatically mapped to the flow graph while the specified constraints, e.g., a specified format or transport protocol are considered. Another important aspect here is that a distributed multimedia middleware automatically searches the entire network or a set of specified locations to find nodes required to set-up a complete distributed flow graph.

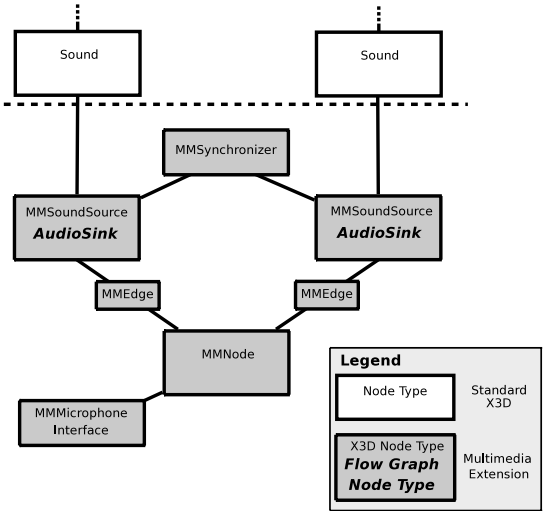

Figure 12. This abstract specification can be used within our system to assign the voice of the user to its 3D avatar and allows him to talk with other users of the scene. In this example, the stereo audio stream is also separated into an audio part for the left stereo channel, and an audio part for the right stereo channel to enable a correct playback of stereo audio data.

A media processing subgraph in X3D is always mapped onto the user graph of the distributed multimedia middleware because it allows to explicitly specify either a full flow graph or only components that are important for the application. Thus, this integration of media processing and control into X3D provides a scalable transparency for the scene graph developer when specifying multimedia content. The most basic description of a user graph within X3D consists of multimedia nodes that represent sources and sinks of the flow graph. Since the sink nodes of the flow graph are either represented by X3D nodes of type `MMTexture` or `MMSoundSource`, a scene graph developer first adds edges as children to these multimedia nodes and then the sources of a flow graph as leaf nodes.

The sources of a flow graph have to be specified either by setting the field `nodeName` or by adding a descriptive interface. Finally, the developer has to add a multimedia synchronizer as parent of those X3D nodes representing the multimedia sinks (`MMTexture` or `MMSoundSource`), if their media playback should be synchronized. An example of such an abstract description can be seen in Figure 11 which shows the abstract specification for receiving video from a camera. Here, the multimedia node representing a camera is specified by using a multimedia interface of type `CameraInterface`. Another example can be seen in Figure 12 which plays back audio recorded from a microphone. Within our system, we use this X3D description to assign the voice of the user to its 3D avatar. In this example the audio is also separated into an audio part for the left channel, and an audio part for the right channel to achieve a correct playback of stereo audio data. Even though the specification of both scenarios is quite simple, it highlights the benefits provided by our integration of media processing into X3D. Firstly, live streams can be used within X3D; secondly, remote resources can be transparently used; and thirdly, controlling the multimedia devices is seamlessly integrated into X3D by changing the values of the corresponding interface.

5.3 Presenting Audio and Video within Virtual Environments

Since we extended X3D by new nodes in the previous section, we have to realize a new engine for RTSG that is able to process these new X3D nodes correctly. Since all specified multimedia nodes of an X3D scene are directly mapped to nodes of a flow graph, this new engine is called *multimedia engine* and responsible for setting up a corresponding flow graph.

Furthermore, we needed to implement new NMM nodes for the special X3D nodes `MMTexture` and `MMSoundSource`, which realize the connection between X3D and multimedia

processing. An X3D node of type `MMSoundSource` is mapped to the NMM node `AL3DPlaybackNode`. This NMM node enables NMM to play 3D sound features by using the OpenAL library[24]. Using OpenAL enables some advanced features like Doppler effect or delayed playback based on the distance to the sound source in a 3D environment. Moreover, OpenAL allows to playback 3D audio through multi-speaker environments like 7.1 speaker setups. The corresponding flow graph for playing back 3D audio recorded from a microphone can be seen in Figure 13. In this example stereo audio is recorded from a microphone, splitted into two mono streams for post-processing before the two streams are played back synchronuously by two instances of an `AL3DPlaybackNode`. Since OpenAL includes a very limited number of effects that are required for a realistic audio simulation, our multimedia renderer allows to insert additional nodes for audio post-processing.

Using these new nodes, the multimedia renderer is able to set up the corresponding flow graph. Figure 13 shows the flow graph that is potentially generated from the X3D specification that can be seen in Figure 12. In addition to the abstract X3D specification, the multimedia renderer automatically adds required nodes. In case of an microphone, also a node for echo cancellation is required to avoid disturbing echo effects if multiple user talk to each other. If available, the multimedia renderer evaluates additional meta information of the world model. For example, it adds specific components for audio post-processing (e.g., an echo effect) that are required to achieve a realistic playback of audio data at the current position of the user.

After setting up the flow graph, the multimedia renderer sets the current viewer position and the position of the sound sources in the `AL3DPlaybackNode`, including all transformations. For this, it keeps an internal representation of the relevant parts of the transformation hierarchy of the world model and uses listeners to be notified about any changes in the X3D scene. These changes are propagated to the `AL3DPlaybackNode` which in turns forward this information to OpenAL. Furthermore, the multimedia renderer checks if the current nodes for audio post-processing are still correct or have to be changed due to movements of the user to a different place within the world model. In this case the multimedia adds new post-processing nodes during runtime, or removes nodes that are no longer required. Adding or removing nodes of the flow graph during runtime is directly supported by NMM.

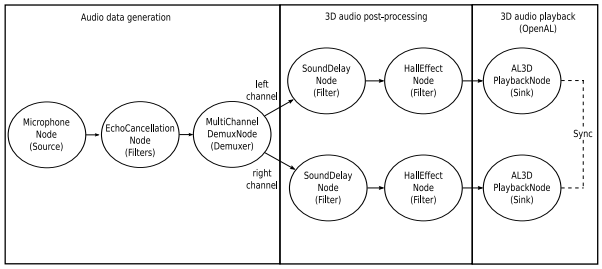

Figure 13 Flowgraph to record audio from a microphone and play back in a 3D environment. The flow graph includes a node for echo cancellation which is required to avoid disturbing echo effects, as well as nodes for post-processing, e.g., reverberation and delay, for a realistic playback in the 3D environment.

An X3D node of type `MMTexture` is mapped to the NMM node `TextureSinkNode`. This node uses a texture buffer of the rendering engine that is currently responsible for rendering textures. The `TextureSinkNode` requests at least two memory blocks for a specific texture to enable double buffering. As soon as a new video frame arrives at the `TextureSinkNode`, it copies the video frame to a free memory block for the corresponding texture and informs the texture renderer to use the texture buffer including the latest video frame.

This approach automatically prevents any synchronization issues with the texture renderer because the renderer simply uses the latest texture buffer available. If the texture renderer achieves a higher frame rate as the assigned instance of the NMM `TextureSinkNode` delivers new frames, the same video frame is presented multiple times. If the texture renderer renders textures with a lower

[24] Creative Labs. "OpenAL", Creative Labs Inc., http://openal.org.

frame rate, some video frames can not be rendered, but the texture renderer always renders the latest video frame as texture.

6. Conclusions and Future Work

In this paper we presented the ISReal project, which provides a service oriented, flexible and extendable framework for intelligent simulation and realistic visualization. The framework consists of three main componts. (1) The world model, which is the central part of ISReal, maintains the representation of the entire 3D scene. In contrast to other approaches, the pure geometrical scene description is enriched by semantic metadata like physical properties. (2) Services are build around the world model and perform a specific functionality on it, like rendering and visualization. (3) The last component is the P2P middleware which interconnects all services with the world model and allows a distributed communication between all components.

One of the first and most important services we realized within is the ISReal project is the service for interactive and realistic rendering and visualization. Section 4 describes the flexible and distributed rendering and visualization system DRONE in detail which has been developed for this service. DRONE is based on a system for distributed multimedia processing and streaming using network (possibly distributed) of processing nodes connect into a common flow graph. Using this middleware provides an unprecedented flexibility in parallelizing and distributing all aspects of a rendering system: user input, rendering, post-processing, display, and synchronization. By designing a small set of modules that can be combined easily, the service can flexibly configure distributed rendering and display – it can even change the configuration and modify any parameter dynamically at runtime. This flexibility comes at a negligible cost over specialized and highly optimized implementations of the same functionality.

To achieve a realistic simulation that also includes accoustic simulation and to enrich the world model by multimedia content, we integrated a Multimedia Renderer into RTSG and extended X3D by a full inclusion of multimedia processing components that is described and discussed in Section 5. We identified four requirements of multimedia processing an X3D application definition should be able to include: (1) explicit specification of a multimedia flow graph inside X3D, (2) the ability to handle streaming media, (3) the possibility to control single processing elements via dedicated interfaces, and (4) the possibility to reduce the specification to only those parts that are absolutely necessary for disambiguation of the multimedia flow.

We introduced a minimal set of new X3D nodes to map a generic multimedia flow graph to an X3D scene graph and incorporated a policy of scalable transparency in our extensions, such that scene authors only have to specify a minimal multimedia flow to make it umambiguous. Finally we showed how to build up a multimedia flow from within X3D to extend e.g., an 3D avatar by the voice of the user. Finally, the playback of audio can be enriched by an arbitrary number of acoustic effects (e.g., echoes, or delay), to achieve the desired degree of realism. If the scene, or more precisely, specific parts of the scene, like a building, already include information about specific acoustic effects, the multimedia renderer automatically inserts the required components, and reconfigures the corresponding flow graph during runtime, if this kind of information changes due to movements of the user.

The main focus of our future work will concentrate on new services, especially in the context of agent-based software systems: We refer to “agents” as a special set of simulation modules that implement proactive (and reactive) behavior of virtual entities. These agents may interact among themselves, with their simulated environment, and with or on behalf of the user. Their functionality should reach from high-level intensional behavior (e.g. by the use of BDI-style agents), via simple reactive behavior, to the mapping of such behavior to the animation of its virtual physical representation.

Bibliography

Bues, Matthias, Tim Gleue, and Roland Blach: "Lightning: Dataflow in motion". In: *Latoschik, Marc Erich (Ed.) u.a.: Software engineering and architectures for realtime interactive systems (SEARIS): IEEE VR 2008 Workshop*, March 9, 2008, Reno, Nevada, USA. Aachen: Shaker, 2008.

Creative Labs. "OpenAL", Creative Labs Inc., http://openal.org, (accessed May, 2009).

Dietrich, Andreas, Ingo Wald and Philipp Slusallek: "The OpenRT Application Programming Interface: Towards a Common API for Interactive Ray Tracing". In: *Proceedings of the 2003 OpenSG Symposium,* pp. 23-31, 2003.

Eileman, Stefan and Renato Pajarola: "The Equalizer Parallel Rendering Framework". Tech. Rep. IFI 2007.06, Department of Informatics, University of Zurich, 2007.

Georgiev, Iliyan and Philipp Slusallek: "RTfact: Generic Concepts for Flexible and High Performance Ray Tracing". In: *Proceedings of the IEEE/EG Symposium on Interactive Ray Tracing 2008*, pp. 115-122, Los Angeles, USA, August 9-10, 2008.

Humphreys, Greg, Matthew Eldridge, Ian Buck, Gordan Stoll, Matthew Everett, and Pat Hanrahan: "WireGL: A scalable graphics system for clusters". In: *SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (2001)*, pp. 129–140, 2001.

Humphreys, Greg, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner and James T. Klosowski: "Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters. In: *SIGGRAPH '02: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques (2002)*, pp. 693–702, 2002.

Khronos Group Homepage. "Collada", Khronos Group, www.collada.org (accessed May, 2009)

Khronos Group Homepage. "OpenGL", Khronos Group, http://www.opengl.org (accessed May, 2009).

Muuss, Michael J.: "Towards Real-Time Ray-Tracing of Combinatorial, Solid Geometric Models". In: *Proceedings of BRL-CAD Symposium '95*, June 1995.

Lohse, Marco, Florian Winter, Michael Repplinger, and Philipp Slusallek: "Network-Integrated Multimedia Middleware (NMM)". In: *Proceedings of ACM Multimedia 2008 (ACM MM 2008)*, Vancouver, Canada, October 27-31, 2008.

Lohse, Marco and Philipp Slusallek: ”Towards Automatic Setup of Distributed Multimedia Applications” In: *Proceedings of The 9th IASTED International Conference on Internet and Multimedia Systems and Applications (IMSA),* pp. 359-364, 2005.

Microsoft Homepage. “DirectX”, Microsoft Inc., http://www.microsoft.com/windows/directx (accessed May, 2009).

OGRE Homepage, “Object-Oriented Graphics Rendering Engine (OGRE)”, http://www.ogre3d.org (accessed May, 2009).

Parker, Steven, Peter Shirley, Yarden Livnat, Charles Hansen, and Peter Pike Sloan: “Interactive Ray Tracing for Isosurface Rendering”. In: *IEEE Visualization 1998*, pp. 233–238, October 1998.

Repplinger, Michael, Alexander Löffler, Dmitri Rubinstein, and Philipp Slusallek: ”URay: A Flexible Framework for Distibuted Rendering and Display”. Tech. Rep. TR-2008-01, Computer Graphics Group, Department of Computer Science, Saarland University, Germany, 2008.

Rubinstein, Dmitri, Iliyan Georgiev, Benjamin Schug, and Philipp Slusallek: “RTSG: Ray Tracing for X3D via a Flexible Rendering Framework”. In: *Proceedings of the 14th International Conference on 3D Web Technology (Web3D Symposium '09)* Darmstadt, Germany, June 16-17, 2009.

SGI Performer Homepage. “Performer”, SGI, http://www.vis-sim.com/performer/ (accessed May, 2009).

Tumblin, Jack and Holly Rushmeier: “Tone Reproduction for Realistic Images”, *IEEE Computer Graphics and Applications*, 13(6), pp. 42-48,1993.

Web3D Homepage, “X3D Scene Access Interface (SAI)”, http://www.web3d.org/x3d/specifications (accessed March, 2009).

Wikipedia. “OpenInventor”, Wikipedia, http://en.wikipedia.org/wiki/Open_Inventor (accessed May, 2009).

X3D Homepage, “X3D. Extensible 3D Graphics for Web Authors”, Don Brutzman, and Leonard Daly, and Morgan Kaufmann, http://x3dgraphics.com/ (accessed May, 2009).

Xaitment GmbH Homepage. “Xaitment Map and Move”, Xaitment GmbH, http://www.xaitment.com (accessed May, 2009).