Eurographics Symposium on Parallel Graphics and Visualization (2009)
J. Comba, K. Debattista, and D. Weiskopf (Editors)

A Flexible Adaptation Service for Distributed Rendering

Michael Repplinger[1] and Alexander Löffler[1] and Martin Thielen[1] and Philipp Slusallek[1]

[1]Lehrstuhl für ComputerGraphik, Saarland University, Germany

Abstract

Even though high-performance real-time rendering showed significant improvements through implementing its algorithms on top of many-core technologies, achieving interactivity in large scenes still requires a networked cluster for distributing the workload. Available frameworks assume a high-bandwidth networking between nodes of a cluster and ignore remote rendering scenarios where adaptation to limited resources (e.g., low bandwidth) is required.

In this paper, we present an extension to the flexible URay framework for distributed rendering that allows to react to unfavorable and changing network conditions. We show how adaptation strategies are applied to streams of rendered images, and how to realize application scenarios that are even able to use the Internet as a communication network, which suffers from unpredictable conditions in terms of latency and bandwidth.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.2]: Graphics Systems—Distributed/network graphics; Computer-Communication Networks [I.3.2]: Network Architecture and Design—Remote Systems

1. Introduction

The performance of real-time rendering has improved significantly through the last years. A large part of this development is due to the rise of parallelization, enhanced through many-core hardware like General Purpose GPU (GPGPU) or Cell processors, and the possibility to easily implement rendering algorithms on top of them. However, the goal to interactively render large scenes in high visual quality, for example by using high-resolution ray tracing, still requires a networked cluster of machines for distributing the accumulating workload. In-between hardware for fast image creation and fast image display, the network turns out to be the bottleneck in a processing pipeline for distributed rendering. Adaptation techniques for reducing the required network bandwidth are necessary for creating an efficient and performant rendering setup.

The URay framework [RLRS08] introduces a flexible approach to distribute rendering tasks transparently in the network. It uses the Network-Integrated Multimedia Middleware (NMM) [LWRS08] to embed the task of interactive rendering in a multimedia processing pipeline. Application design is based on the *flow graph* concept of NMM, which enables a fine-grained control of distributed data processing and data flow through the network [Mar02]. For the scenario of distributed rendering, URay enables postprocessing steps in image space like brightness and color adjustment or data encoding.

In this paper, we present an extension to the URay framework, which allows to use remote rendering also across networks of varying bandwidth and latency. We show how new optimization and adaptation techniques are incorporated into the framework, and how adaptation of bit rate and resolution allows to use remote rendering also across heterogeneous networks. Through permanent monitoring of the quality of service (QoS) achieved, we are able to exploit a maximum of the available processing and networking capabilities for a resulting high-quality scene display. Moreover, the user is able to explicitly specify a *QoS level*, to ensure that certain adjustment parameters stay within desired limits.

The structure of this paper is as follows: In Section 2, we will show work in the related fields of distributed rendering and adaptation of multimedia streams. Section 3 will introduce the general architecture of the URay framework, whereas Sections 4 and 5 show concepts and implementation of the new URay adaptation service, respectively. Section 6 shows the performance of URay adaptation strategies in a real-world scenario. Section 7 concludes the paper, and gives an outlook on extensions of this work.

2. Related Work

Molnar et al. [MCEF94] presented a classification scheme for distributed rendering. The authors subdivide techniques that distribute geometry according to screen-space tiles (*sort-first*), distribute geometry arbitrarily while doing a final z-compositing (*sort-last*), or distribute primitives arbitrarily, but do per-fragment processing in screen-space after sorting them during rasterization (*sort-middle*). It is difficult to apply the scheme for a generic rendering architecture supporting other techniques besides rasterization, for example ray tracing. Here, Molnar's classification approach is no longer applicable, as geometry processing and screen-space projection are combined in the single operation of sampling the scene with rays. Also, recursive ray tracing requires global and on-demand access to the scene data.

In contrast to this, the URay framework [RLRS08] shows that realizing a rendering framework on top of NMM and the *Real-Time Scene Graph* (RTSG) [GRHS08] provides much more flexibility and can be applied to rendering technologies such as rasterization and ray tracing. RTSG provides a strict separation of the scene graph and a specific implementation of a renderer, which allows to integrate a new rendering back end just by implementing the simple interface as well as node renderers for the X3D nodes the renderer should support.

Due the usage of NMM and RTSG, URay supports several application scenarios that are not supported by other frameworks. The first application scenario (AS1), called *single-screen rendering*, comprises presenting rendered images on a single screen but using multiple systems for rendering.

The second application scenario (AS2) called *multi-screen rendering*, extends (AS1) by splitting the 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.

The third application scenario (AS3), called *multi-view rendering*, comprises presenting of multiple views of the same scene at the same point in time. For example, this is required for stereo imagery or rendering for Virtual Reality installations like a CAVE [CNSD*92].

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 in a controlled and secure area. While the previous application scenarios assume high-bandwidth networks, (AS4) requires an encoding and decoding of the streams of rendered images to reduce the data rate when transmitting for example over the Internet.

The last application scenario (AS5) is *collaborative rendering*, an arbitrary number of combinations of the previously described application scenarios. An ideal system scenario should allow both a large control center with tiled display walls, and simultaneously remote 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.

In the area of distributed multimedia processing, several adaptation frameworks have been proposed to adapt to the changing quality and bandwidth of a network: The work presented in [KWcF03] and [KD05] focuses on achieving end-to-end QoS when streaming video data over an Internet connection while the work presented in [LG05] focuses on transmitting video data over wireless connections. In addition to this, the adaptive streaming architecture described in [BCCV05] is used for transparent management of mobility. The emphasis of this work lies on heterogeneous networks and the support for adaptation during vertical hand offs (i.e., switching from WLAN to UMTS) and is limited to the networking aspect. In multimedia streaming often scalable video coding is used. For example the streaming system presented in [NO06] combines several such techniques. Since their applicability is limited to the encoding process itself, they can not be used to adjust the rendering process. Together, there is no flexible adaptation framework for distributed rendering.

3. The URay Framework

The following section introduces the basic concepts of the URay framework, explains components and typical usage scenarios.

3.1. Components

The URay framework builds on top of an NMM flow graph consisting of custom processing nodes supplemented by existing nodes of core NMM. The following are the specific NMM nodes used within the URay framework:

`RenderNode` A *render node* performs the actual rendering of a scene description to a 2D image. Key principle of the URay framework is rendering a single frame distributed on multiple render nodes in the flow graph.

`ManagerNode` The single source node of the URay flow graph is called *manager node*. The manager node distributes the workload between the render nodes by splitting the image into many *image tiles* and assigns them dynamically to render nodes on demand.

`DisplayNode` A *display node* constitutes a sink of the flow graph, and simply presents any incoming image buffer synchronized according to its timestamp.

`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.

`EncoderNode/DecoderNode` The basic functionality to compress image data is represented by an *encoder node* and a *decoder node*, which are inserted between the tile assembly node and display node. Core NMM provides multiple specializations of encoder and decoder nodes for video streams, for example using an MPEG-4, H.263, H.264 or AVC codec.

`ScalerNode` To accomplish scenarios with multiple displays having different resolutions, but avoid having to render the scene multiple times, the setup described so far is augmented by a *scaler node*, which allows to scale down the resolution of a passing image stream. If both are present at the same time, the scaler node precedes an encoder node to avoid having to encode larger frames than actually needed.

Figure 1 shows the presented the most important nodes of the URay framework assembled to a simple, single-display flow graph.

3.2. Single-Screen Rendering (AS1)

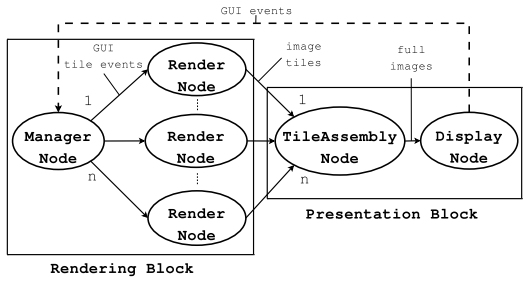

Figure 1: *URay encapsulates the flow graph in different basic processing blocks. The rendering block includes the managing node as well as all rendering nodes. The presentation block includes all remaining nodes required to present rendered images.*

The primary processing block, which occurs in every URay 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.

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. A presentation block 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 renderer to the corresponding assembly node that are to be displayed by the succeeding display node.

NMM incorporates a unified messaging system, which allows to send control events together with multimedia data within the same instream channels. Of key importance is the strict order of message handling: any node always receives messages from its preceding node in the exact same order they were sent [Mic05]. This allows, for example, to use events to switch the internal state of a node at a well-defined position within a multimedia stream.

In our case, we use the manager node of the rendering block to send information about the next tile to be rendered as events to its successive render nodes. A simple approach would be to use a round-robin distribution scheme. NMM however also has access to the internals of the network connection between two nodes. By choosing a suitable size of the network, it can control the distribution of tiles to render nodes, and assign additional tiles only if the previous tile has finished rendering and the buffer is not full anymore. This is achieved by treating a network connection as an additional queue. In case of a TCP connection, for example, the URay framework configures the underlying network connection so that sending and receiving side can each store exactly a single tile to be rendered. This allows to reuse the flow control mechanism of TCP to be informed if a new rendering task can be sent because new rendering tasks can be written to the sending queue. All this is only possible due to the scalable transparency of NMM.

This very simple scheduling approach leads to an efficient dynamic load balancing between the render nodes, because render nodes that finish rendering tiles earlier, also do receive new rendering tasks earlier. Differences in rendering time can be caused by different scene complexity, different network capabilities, or different processing power of different rendering machines.

3.2.1. User Interaction

URay should also allow *user interaction* with the rendered scene. Because all render nodes render the scene, each one has to be informed about user events, such as changes to the viewpoint. In our setup, interaction events, key presses or mouse movements, first are sent as out-of-band events from the application to the manager node. The manager node in turn forwards all incoming events to all connected renderers. The key point is that input events are propagated only between tiles of *different* frames to avoid changes of viewpoint before the processing of a frame is fully completed.

3.3. Multi-Screen Rendering (AS2)

The general idea to support applications that need to present rendered images on multiple screens can be seen in Figure 3.3. In contrast to (AS1), the application specifies more than one presentation block. All these presentation blocks are then connected to the same rendering block by the framework. This is possible because `RenderNode` supports an

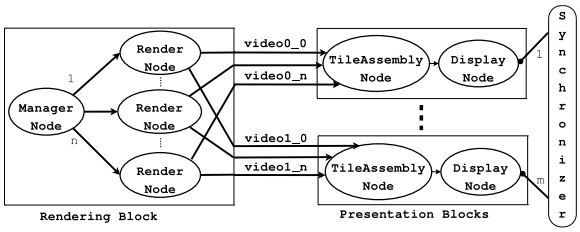

Figure 2: *URay allows defining multiple independent presentation blocks (e.g., for realizing video walls). The synchronized presentation of rendered images is achieved by connecting a synchronizer to the corresponding presentation blocks.*

arbitrary number of output connections. Nevertheless, presentation blocks have to be configured independently in order to achieve the desired distribution of rendered frames to available display nodes and devices. Here, any rendered frame can be presented on any of the screens simultaneously; either in full or in part for realizing a video wall setup.

In addition to (AS1), another important additional component for realizing (AS2) is the *synchronizer* component. The synchronizer realized in the URay framework is described in [RLRS08] and allows for synchronizing the presentation of either partial or full frames in multiple display configurations. This is especially required for video-wall setups, where skews in synchronization severely deteriorate the user experience.

3.4. Multi-View Rendering (AS3)

The general approach of URay 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.

The implementation of our manager node allows rendering an arbitrary number of viewpoints which is required for example for virtual reality installations. Subsequently, the manager node initiates the rendering of all viewpoints in the same way. Afterwards, the manager node sends an event in-stream to inform the downstream flow graph that the frames can now be presented.

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.

3.5. Remote Rendering (AS4)

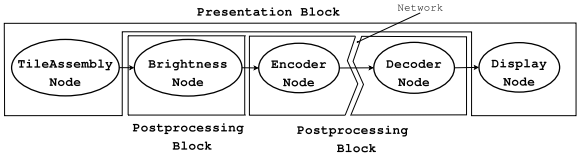

Figure 3: *Extended presentation block: URay allows to add an arbitrary number of postprocessing blocks. In this example we first adjust brightness and then encode rendered images before sending them through an Internet connection.*

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 as described in Section 3.2, 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, like brightness adaptation or tone mapping.

To enable all these scenarios, we allow the insertion of one or more *postprocessing blocks* into a presentation block. Figure 3 shows a presentation block enhanced by two postprocessing blocks: one for brightness adjustment, and one for encoding and decoding the stream. The implementation of the block consists of a common interface that uses raw image input and output format, and arbitrary NMM video processing nodes in between. A postprocessing block with all its internal nodes is plugged in between the tile assembly or scaler node and the display node of any presentation block.

3.6. Collaborative Rendering (AS5)

The final application scenario to be covered by the URay framework is the situation of multiple parties working on and interacting with one and the same rendering block, realizing a collaborative environment. This includes for example industrial collaborations in which 3D models are synchronously displayed to engineers in distinct offices around the globe. In terms of the URay 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. The flexible architecture of URay allows, for example, to realize different encoded streams for

each one of the presentation blocks, and arbitrary display setups for the participating parties. In addition, multiple views into the same scene are possible as well as different resolutions realized by the available scaler nodes.

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 this framework.

4. Adaptation Service

On top of URay, a flexible network adaptation service is realized, which hast to support different adaptation techniques due to the different requirements of supported application scenarios.

4.1. Requirements

In (AS1), (AS2) and (AS3), the components of the rendering block are typically connected using high speed wired networking technologies. Here, the available bandwidth together with the available processing power is the main issue for distributed rendering applications. The implementation of our manager node already performs load balancing in respect to the duration of rendering different tile frames as described in Section 3.2. However, if a different adaptation mechanism for load balancing should be required, it could be integrated by implementing a new manager node. The render block offers the possibility to specify a different manager node by a single method call. So an application can easily configure a new mechanism for load balancing.

Especially for (AS4), adaptation of remote rendering application scenarios to changing bandwidth is essential, because when using Internet connections for data transmission no guarantee about the connection quality can be done. Depending on the utilized network protocol lost packet are either retransmitted (e.g., when using TCP) which can cause unacceptable latencies due to interactivity of all application scenarios, or will be discarded (e.g., when using UDP) which will cause unacceptable artifacts.

To restore an acceptable video quality, the amount of transmitted data has to be reduced greatly. This can be achieved, for example, by reducing the bit rate of a stream. When network conditions worsen, a higher bitrate causes more artifacts when using the same resolution. So it is desirable to adjust the resolution of the rendered video as well. Moreover, it is desired to reverse the previously applied reductions when the available bandwidth is increasing again.

For (AS1), (AS2) and (AS3) this is not possible because in general it should be assumed that a corresponding postprocessing block is available. In this case an adaptation mechanism has to insert a new postprocessing block during runtime which is possible because our framework supports runtime reconfiguration. In other cases, it can happen that no further reduction of quality is possible with the current stream. Therefore, a reconfiguration of a postprocessing block and the utilization of a different codec can be required.

In summary, the main aspect of an adaptation service for URay is to be easily extendable. This includes adaptation techniques as well as general support for supervising arbitrary components of the URay framework, like different network protocols or processing elements. Again, this shows the great advantage of using a flexible rendering framework.

4.2. Components

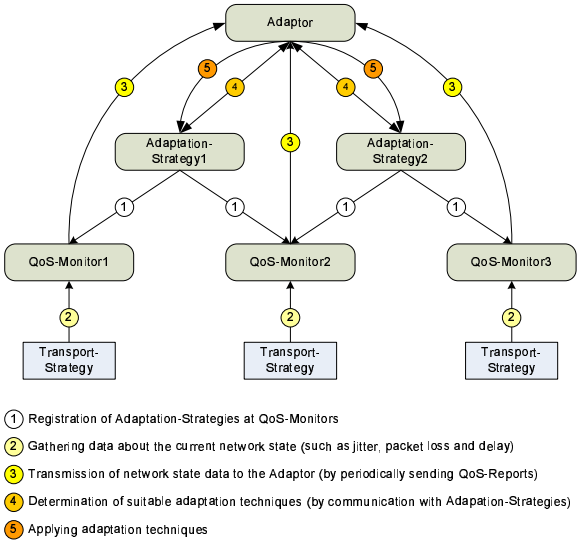

Figure 4: *Components of the adaptation service and single steps of communication flow between components.*

To achieve a high degree of flexibility, the adaptation service is split into the following components:

QoS-Monitor : A *QoS monitor* gathers information about a specific URay or NMM component. Furthermore it filters and forwards the so called *QoS reports* to the Adaptor. The information included into a QoS report depend on the implementation of a specific QoS Monitor that can include arbitrary information like packet loss, or used processing power.

Adaptation Strategy : Instead of a static adaptation logic, a modular approach is taken for performing adaptation techniques. An *adaptation strategy* performs the interpretation of specific types of QoS reports and determines an appropriate reaction, like reducing the bitrate. Each adaptation strategy describes this reaction in terms of changes on the corresponding connection format between two components of the URay framework. Furthermore, an upper and lower bound can be set for each of these changing parameters which enables to specify a specific *QoS*

level, e.g., changing resolution in a certain interval. As can be seen in Figure 4 each adaptation strategy registers itself by all QoS monitors that produce supported types of QoS reports to be informed about variations in media processing. Finally, different adaptation strategies can be combined to build more complex adaptation strategies, e.g., adapting resolution together with bitrate.

Adaptor : The *adaptor* is the link between the application and several different QoS monitors as can be seen in Figure 4. The application can use the adaptor to configure a certain QoS level that has to be achieved during runtime. For this purpose, the application informs the adaptor about the desired connection format between URay components together with parameters of the connection format that can be changed. Allowed values for a specific parameter can be specified as a list of single values and as a range of values, e.g., an upper or lower bound for framerate or used bandwidth. Furthermore, a priority can be assigned to each parameter to specify the order of changed parameters. Since the adaptor knows available adaptation strategies it first checks if specified QoS levels are supported and determines which strategies should be used for adaptation. If multiple adaptation strategies support the same QoS report and QoS level the adaptor selects a single adaptation strategy which is important to forward a QoS report only to a single adaptation strategy. Otherwise, different adaptation strategies would try to adjust the same bottleneck. However, if no adaption is possible without violating the QoS level specified by the user, the adaptor throws an exception to the application.

5. Implementation

5.1. QoS-Monitor

Together with this adaptation service, we extended the transport strategies of NMM for UDP and RTP by a QoS monitor as can bee seen in Figure 5. Furthermore, we extended the transport strategy for UDP and RTP to create transmission statistics that are stored in a QoS report and include the following information.

- **Fraction lost**: Specifies the ratio of lost packets since sending the last report.
- **Packets lost**: The number of packets lost since start of reception.
- **Statistics lost**: The number of transmission statistics lost since start of reception.

5.2. Adaptation Strategies

To evaluate the adaptation framework we realized two adaptation strategies, one for adjusting the bitrate of an video encoder, and one for adapting the resolution of rendered scene. Those two strategies show the flexibility of the adaptation approach itself as well as the benefits of a flexible rendering framework.

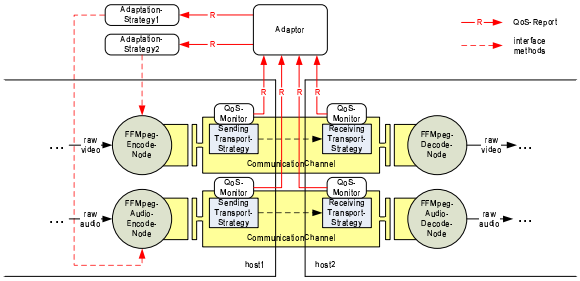

Figure 5: *Each transport strategy has its own QoS monitor to gather information about transmission statistics. This information is forwarded to the adaptor, which decides which adaptation strategy is used. The adaptation strategy in turn adjusts the bitrate of an encoder for example.*

5.2.1. Adaptation of Bitrate

For indicating a decreasing bandwidth, the number of lost packets of an incoming QoS report is considered. The general idea for adaptation is as follows: If the network is congested and packet loss occurs, the fast recovery of an acceptable quality is considered most important. Therefore, the bit rate of the corresponding encoding node is reduced depending on the packet loss ratio. This is only possible because URay provides access to the used components, in this case the respective encoding node.

If a number of reports without packet loss are received, the bit rate is increased again. Essentially, this is an AIMD scheme which is also used in the congestion control of TCP as described in [KR00]. In detail, there are four cases which are handled differently.

1. No packet loss: If no packet loss occurs and delay for next increase is not increased, the bit rate is increased by a moderate amount of 4 % of the current bitrate.
2. Packet loss after increasing bitrate: If the bit rate was just increased and packet loss occurs, the bit rate is only reduced by a small amount. Additionally, the number of reports to wait until the bit rate is increased the next time is depending on the previously conducted adaptation.
3. Packet loss level 1: If a relatively small amount of packets are lost $(0\% < \mathit{packet\ loss} < 1\%)$ the bit rate is reduced by a quarter. The next increase is at least delayed by 3 reports.
4. Packet loss level 2: For a larger percentage of packet loss $(\mathit{packet\ loss} > 1\%)$, the reaction is that the bit rate is halved.

5.2.2. Adaptation of Resolution

Adjusting the bitrate of a postprocessing block of an encoder will cause more encoding artifacts if the amount of images to be encoded is not reduced. A more desirable alternative for the user is to reduce the resolution of the rendered scene

together with the bitrate. If the rendered image should still be presented in the same resolution as originally specified in the presentation block, a scaler node is used. In addition, this will save processing power on the rendering side. In this case the adaptation strategy informs the manager node to reduce the resolution, which in turn informs all render nodes to reduce the resolution. This is required to ensure that resolution changes occur only between full frames.

If more processing power is available, however, the render nodes would create a higher framerate, which again would lead to a higher amount of data to be encoded. To avoid this, we extend our manager node not to exceed a certain framerate for this presentation block. The realized strategy for adapting resolution is grouped together with the adaptation strategy for bitrate and assumes a linear dependency between achieved picture quality using a specific bitrate and a specific resolution, that is if the bitrate is halved in value, the resolution is also halved in value.

Together, these two adaptation strategies allow an user to achieve still a good picture quality even though if the available network connection is reduced. Moreover, the user is able to configure the adaptation process as well as specific QoS level.

Although the used adaptation mechanisms are simple, they demonstrate that a developer can focus on the implementation of an adaptation strategy even though such a component reconfigures a distributed rendering application. This however requires a flexible rendering framework like URay, which allows transparent access to the underlying components as well as runtime reconfiguration of all components and a corresponding adaptation framework that automatically configures an adaptation strategy according to the specification of a user.

6. Measurements

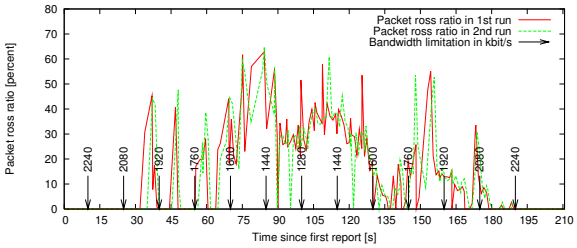

Figure 6: *This diagram shows the packet loss without the adaptation service presented in this paper. In this case up to 65 % of the encoded data will be lost.*

In [RLRS08], we already showed that the memory and performance overheads of URay are negligible, while applications greatly benefit when using URay due to the flexibility of the framework. Therefore we analyze the realized adaptation service of URay.

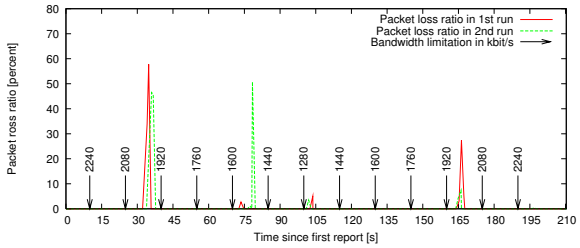

Figure 7: *This diagram shows the packet loss with enabled adaptation service presented in this paper. Only a view peaks of packet loss are contained in the diagrams. Afterwards, the packet loss ratio drops quite fast to 0.*

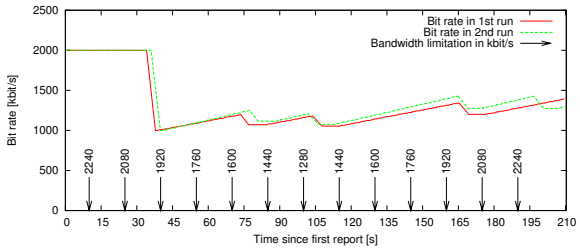

Figure 8: *This diagram shows the estimated bitrate of our adaptation strategy when first decreasing and then increasing available bandwidth.*

For the following tests we rendered a scene with a resolution of 720x480 using a rasterization back-end of an RTSG render node. The produced framerate is nearly constant at 30 fps and presented on a single presentation block on a remote PC. A postprocessing block is added to the presentation block to encode video data into H.263+ format before transmitting it to the presenting host. The encoder uses a bitrate of 2 MBit/s and RTP protocol is used for data transmission.

To simulate different network conditions as well as to reproduce results, we use the tool *NIST Net* [CS03] for evaluation purposes. NIST Net is a network emulator which is based on the IP layer and implemented for a Linux operating system. It is capable of emulating delay, bandwidth, packet loss and other parameters through a kernel module. The network tool is installed on PC that acts as router inbetween the renderer and presentation hosts. For each test, we present results of two iterations.

In our measurement, first the available bandwidth is reduced from 2.24 MBit/s slowly to about 1 MBit using our network simulator and then increased again. As can be seen in Figure 6, the bandwidth of 2.24 MBit/s is required to transmit the encoded video without data loss . This is required because the used H.263+ encoder uses a tolerance of 10 %. Thus the data stream could be 10 % larger then configured which is true in our case. However, since our adaptation

strategy reconfigures the bitrate by percent, it automatically configures a bitrate that is 10 % smaller.

Figure 6 shows the increasing packet loss when reducing the bandwidth and no adaptation is enabled. Due to a packet loss of up to 65 %, the user receives a completely damaged image, which makes it impossible to interact with a displayed 3D scene.

In contrast to this, Figure 7 shows the packet loss if adaptation is enabled. Here, only peaks of packet loss are observed. Afterwards, the packet loss ratio drops quickly down to zero again. As soon as the available bandwidth is increased again, the adaptation strategy detects this and starts increasing bitrate and resolution again which can be seen in Figure 8.

Peaks of packet loss are unavoidable as the adaptation applies only after packet loss occurs. Anyhow, the quality compared to a video without adaptation is much better. These results show that even though the adaptation logic of the adaptation strategies is simple, but the outcome is reasonably good.

7. Conclusion and Future Work

In this paper, we presented an adaptation to the URay system for distributed rendering and display. Using a flexible rendering system coupled with a system for distributed multimedia processing and streaming using a network of processing nodes connecting within a common flow graph. Using a middleware provides an unprecedented flexibility in parallelizing and distributing all aspects of a rendering system: user input, rendering, post-processing, display, and synchronization. Moreover, using a flexible framework greatly simplifies adding new services. We showed an adaptation service together with two adaptation strategies realized on top of URay, which allows to consider changes in network connections and automatically adapts the configuration of processing components to achieve a good rendering quality in an encoded distributed setup.

Our future work will focus on developing further adaptation techniques. For example in different scenarios, different streams are more or less important. Often, this depends on the content. This is especially important for (AS5), where multiple users receive rendered images from the same rendering block. Here different presentation blocks could be treated with a different priority if the network bandwidth goes down, or presentation blocks with a lower priority are removed when the processing power is no longer sufficient.

Acknowledgements

We thank Dmitri Rubinstein for fruitful discussions, close collaboration on the URay framework, and for providing RTSG as a flexible scene graph system.

References

[BCCV05] BERNASCHI M., CACACE F., CLEMENTELLI R., VOLLERO L.: Adaptive Streaming on Heterogeneous Networks. In *WMuNeP '05: Proceedings of the 1st ACM Workshop on Wireless Multimedia Networking and Performance Modeling* (Montreal, Quebec, Canada, 2005), ACM Press, pp. 16–23.

[CNSD*92] CRUZ-NEIRA C., SANDIN D. J., DEFANTI T. A., KENYON R. V., HART J. C.: The CAVE: Audio visual experience automatic virtual environment. *Commun. ACM 35*, 6 (1992), 64–72.

[CS03] CARSON M., SANTAY D.: NIST Net: A Linux-based Network Emulation Tool. *SIGCOMM Computer Communication Review 33*, 3 (2003), 111–126.

[GRHS08] GEORGIEV I., RUBINSTEIN D., HOFFMANN H., SLUSALLEK P.: Real Time Ray Tracing on Many-Core-Hardware. In *Proceedings of the 5th INTUITION Conference on Virtual Reality* (Oct 2008).

[KD05] KUSMIEREK E., DU D. H. C.: Streaming video delivery over internet with adaptive end-to-end QoS. *J. Syst. Softw. 75*, 3 (2005), 237–252.

[KR00] KUROSE J. F., ROSS K. W.: *Computer Networking: A Top-Down Approach Featuring the Internet*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[KWcF03] KRASIC C., WALPOLE J., CHI FENG W.: Quality-Adaptive Media Streaming by Priority Drop. In *NOSSDAV '03: Proceedings of the 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video* (New York, NY, USA, 2003), ACM Press, pp. 112–121.

[LG05] LEI Z., GEORGANAS N. D.: Adaptive video transcoding and streaming over wireless channels. *J. Syst. Softw. 75*, 3 (2005), 253–270.

[LWRS08] LOHSE M., WINTER F., REPPLINGER M., SLUSALLEK P.: Network-Integrated Multimedia Middleware (NMM). In *MM '08: Proceedings of the 16th ACM international conference on Multimedia* (2008), pp. 1081–1084.

[Mar02] MARCO LOHSE AND MICHAEL REPPLINGER AND PHILIPP SLUSALLEK: An Open Middleware Architecture for Network-Integrated Multimedia. In *Protocols and Systems for Interactive Distributed Multimedia Systems, Joint International Workshops on Interactive Distributed Multimedia Systems and Protocols for Multimedia Systems, IDMS/PROMS 2002, Proceedings* (2002), vol. 2515 of *Lecture Notes in Computer Science*, Springer, pp. 327–338.

[MCEF94] MOLNAR S., COX M., ELLSWORTH D., FUCHS H.: A sorting classification of parallel rendering. *IEEE Computer Graphics & Applications 14*, 4 (1994), 23–32.

[Mic05] MICHAEL REPPLINGER AND FLORIAN WINTER AND MARCO LOHSE AND PHILIPP SLUSALLEK: Parallel Bindings in Distributed Multimedia Systems. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops (ICDCS 2005)* (2005), IEEE Computer Society, pp. 714–720.

[NO06] NGUYEN D. T., OSTERMANN J.: Streaming and Congestion Control using H.264/AVC Scalable Video Coding. *15th International Packet Video Workshop 7*, 5 (May 2006), 749–754.

[RLRS08] REPPLINGER M., LÖFFLER A., RUBINSTEIN D., SLUSALLEK P.: *URay: A Flexible Framework for Distibuted Rendering and Display*. Tech. Rep. 2008-01, Department of Computer Science, Saarland University, Germany, 2008.