December 21, 1998
RT0291
Computer Science 13 pages

Research Report

QoS Management for a Reactive Virtual Environment Browser

Tatsuo Miyazawa, Masaaki Taniguchi, Ryo Yoshida, Masaaki Murao

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

QoS Management for a Reactive Virtual Environment Browser

Tatsuo Miyazawa, Masaaki Taniguchi, Ryo Yoshida, Masaaki Murao

IBM Research, Tokyo Research Laboratory, IBM Japan, Ltd.

1623-14, Shimotsuruma, Yamato-shi, Kanagawa-ken, 242-8502 Japan

E-mail: {miyazawt, taniguti, yoshidar, murao}@trl.ibm.com

Abstract

We describe a method for managing the Quality of Service (QoS) of the Virtual Reality Modeling Language (VRML) browser. We implemented a QoS management capability in a VRML 2.0 browser that we developed, called the Reactive Virtual Environment (RVE) browser. The QoS management consists of two layers: (1) a resource allocation layer, which manages the CPU allocation and execution control by means of a scheduling algorithm, and (2) the service adaptation layer, which provides best-effort services under the allocated CPU resource by automatically selecting the level of detail (LOD) of 3D scenes and adjusting the frame rate. The QoS management guarantees the real-time nature of the system response and causes graceful degradation of the QoS when system resources are limited. We demonstrate experimentally that the two-layer approach is very effective for dynamic 3D and multimedia applications.

Keywords

VRML, Quality of Service, Scheduling Algorithm, Load Management, Level of Detail

1. Introduction

The Virtual Reality Modeling Language (VRML) is a standard language for describing interactive and animated 3D objects and worlds to be used on the Internet, intranets, and local client systems. One of the most important features of VRML 2.0, and one of the major advances over VRML 1.0, is the capability it provides for dynamically changing the state of a scene graph by means of event routing, scripting, and interpolation. In December 1997, VRML 2.0 was approved by the ISO/IEC as an international standard file format, called VRML 97 [13], for 3D multimedia on the Internet.

VRML browsers enable users to download dynamic 3D objects and worlds via the Internet or an intranet, and to browse those worlds on their own workstations or PCs. One of the problems of VRML is the difficulty of achieving consistent simulation performance; in other words, realistic motions in the 3D world and real-time interaction with the world on clients with different performance capabilities, from low-end PCs to high-end workstations. The problem becomes more serious when multiple users share the same 3D worlds and interact with them collaboratively. In addition, dynamic 3D and multimedia applications require real-time processing, since such applications convey appropriate meanings only when the contents are presented continuously in time. On the other hand, in practice, users often want to be able to run a variety of applications, such as a 3D scene viewer, a video player, and other desktop applications, on the clients at the same time, and to visualize more than two 3D worlds at the same time. In such cases, it is important to coordinate the allocation of the limited CPU resource and the service level to each application according to users' interests.

To solve these problems, we incorporated the concept of quality of service (QoS) management into an

experimental VRML system called the Reactive Virtual Environment (RVE) system, which consists of a VRML browser and a multi-user shared-state server. The goal of QoS management is to enable the system to guarantee a certain level of QoS in terms of throughput and response time, by providing a CPU allocation framework suitable for the VRML system and a mechanism for dynamically adapting the frame rate and LOD (level of detail) of the scene according to the available system resources and the system load. The QoS approach can guarantee the real-time nature of system response and cause graceful degradation of the QoS when system resources are limited.

In this paper, we describe a method that provides a capability of the QoS management to a VRML browser. In section 2, we review related work. In section 3, we give an overview of the RVE system. We describe the QoS management capability in the RVE browser in section 4, and the implementation and results in section 5. Section 6 concludes the paper.

2. Related work

There are several approaches to achieving an interactive frame rate during visualization of complex 3D worlds. View-volume culling methods [8] discard objects in the worlds that are positioned completely outside the frustum of the view volume in the current frame. Occlusion culling methods [1] cull away large portions of models that are occluded from the observer's viewpoint in the current frame and improve frame rates significantly. However, these visibility algorithms are not sufficient to guarantee an interactive frame rate, because, in very detailed models, more polygons are often visible from certain observer viewpoints than can be rendered in real time. Detail elision techniques are used to reduce the number of polygons rendered in each frame. Each object must be represented at multiple levels of detail (LODs). To choose a level of detail at which to render each visible object, these techniques use static thresholds regarding the size or distance of an object from an observer [9] or the number of pixels covered by an average polygon [11]. Although the techniques improve frame rates in many cases, they do not generally produce a uniform or bounded frame rate. Adaptive detail elision is an adaptive algorithm that adjusts the size of the threshold for LOD selection on the basis of feedback regarding the time required to render previous frames. This adaptive technique [8] works reasonably well for applications in which there is a large amount of coherence in scene complexity from frame to frame. A predictive algorithm [2] estimates the time required to render every potentially visible object at every level of detail, and then computes the largest threshold that allows the current frame to be rendered within the target frame time, to guarantee a bounded frame rate during visualization of discontinuous 3D worlds.

These approaches mostly work well when the visualization of such 3D worlds is done in a nearly dedicated manner on PCs or workstations. In practice, however, the computer is often required to support a variety of applications, such as 3D contents browsing, video playback, audio and video conferencing, and other Web-based applications and desktop applications, at the same time. In this case, multiple applications may issue computation requirements that exceed the available CPU resource. To meet these requirements, we implemented a CPU allocation-scheduling algorithm suitable for VRML systems, and combined it with a reactive service adaptation algorithm.

3. Overview of RVE system

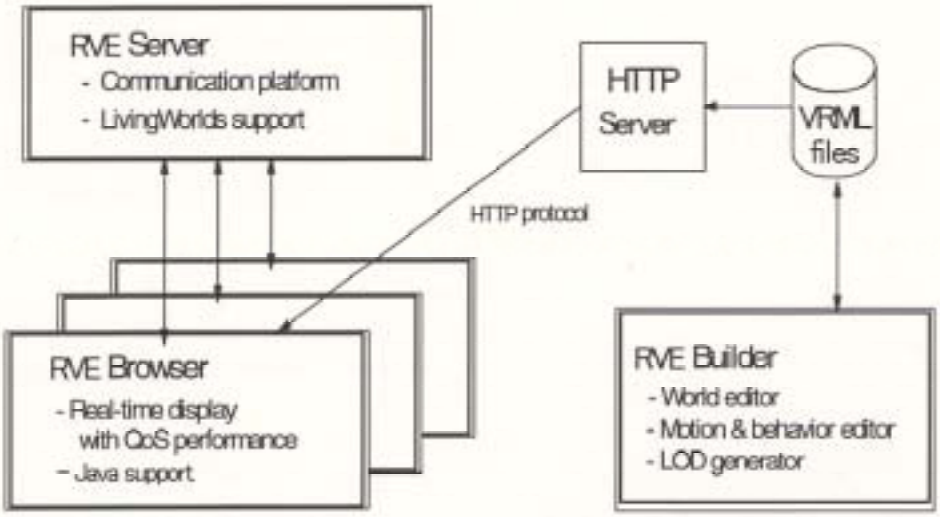


Figure 1: Overview of RVE system

Figure 1 shows an overview of the RVE system. The RVE System [14, 15] that we developed is a client-server-based multi-user VRML system. Our primary objective is to develop a multi-user VRML system that can realize real-time 3D Web collaboration, thus enabling distant users to do collaborative work using 3D contents over the Internet or an intranet.

The RVE browser is a VRML 2.0 browser with QoS performance. The browser provides the Java in Script Nodes Authoring Interface (JSAI) coded in Java version 1.1, but dose not support any External Authoring Interface (EAI), because the EAI has not yet been approved as an international standard. We developed the RVE browser on a Windows 95 platform with Microsoft DirectX (Direct3D, DirectDraw, and DirectSound). One of the features of the RVE browser is multiple view support. It can display multiple 3D worlds in the multiple sub-windows, or display scenes of a 3D world from different viewpoints in the multiple sub-windows. Another feature is robustness to event processing for complicated route connections, including multiple eventIns and cyclic dependencies [12]. The browser first determines the evaluation order of nodes on the event cascades and then propagates the events accordingly, to handle the complicated event routing. A third feature is QoS management described in this paper, which provides the RVE browser with a capability for CPU resource allocation and service adaptation. For the QoS management, multi-resolution geometry models generated by the RVE builder are used effectively as candidates for LOD selection.

We developed the RVE server (a Java-based multi-user shared-state server) and LivingWorlds support module. LivingWorlds [6] is a specification for multi-user VRML worlds, and defines interfaces between a VRML browser and multi-user-enabling software. We used the LivingWorlds specification to develop the components for basic functions such as sharing of a dynamic 3D world. All the nodes defined in the specification are coded as prototyped nodes by using PROTO/EXTERNPROTO and several VRML standard nodes, including Script nodes working with Java programs. Some of the Java programs run asynchronously as a separate process of the VRML browser to communicate with the RVE server. The server maintains the information on the positions and orientations of shared objects, as well as the

specified field values of the shared objects. We are also developing additional functions for various types of collaborative work. Our multi-user VRML system makes it possible for multiple users to share a 3D world and, for example, to watch animated demonstrations of merchandise with operator-assisted view control and object manipulation.

4. QoS Control

Figure 2 shows the architecture of the QoS control of the RVE browser. There are two layers: a resource allocation layer and a service adaptation layer.

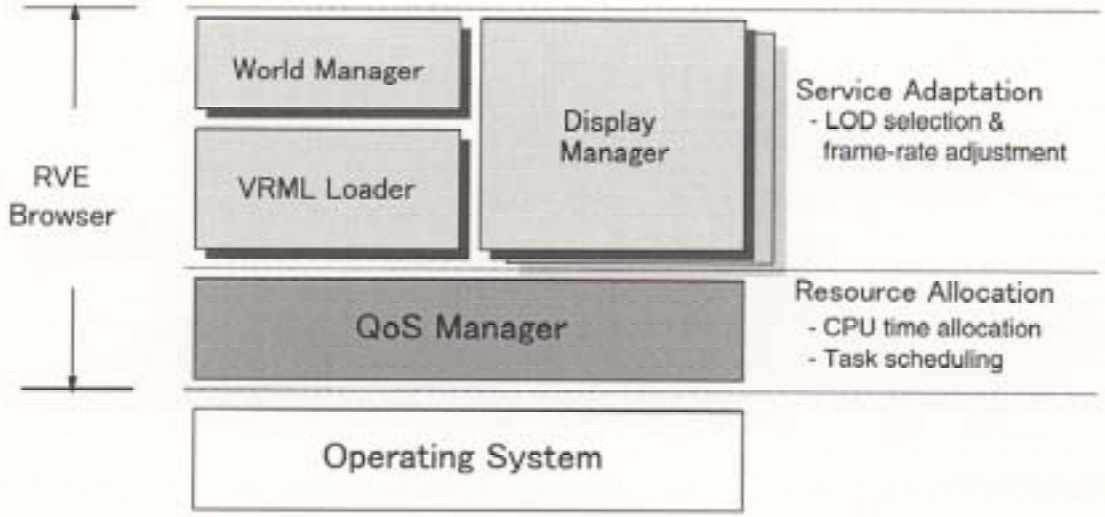


Figure 2: Architecture of the QoS control of the RVE browser

The QoS manager in the resource allocation layer controls the allocation of the CPU resource to multiple modules such as the display manager and VRML loader, and the execution of the associated tasks. The display manager in the service adaptation layer provides the best-effort QoS by selecting the LOD of the scene and adjusting the frame update rate to complete the process within the allocated CPU time.

4.1 QoS Manager

Although predictive QoS control requires an operating system to have capabilities for resource reservation and adaptation, generally available operating systems such as Microsoft Windows 95 do not provide such capabilities. Since the RVE browser runs on Windows 95, a scheduler for the QoS control has to be realized on the upper layer of Windows 95.

The CPU bandwidth required for rendering a 3D scene varies from frame to frame (i.e., on a time scale of 50-100 milliseconds), and the variations are unpredictable. Furthermore, owing to restrictions of the operating system and graphics library, the calling of the rendering APIs by the rendering process occupies the CPU resource, and as a result the process context switching from an active rendering process to

another waiting rendering process makes the system unstable. This means that once the process for rendering a scene of a frame starts, no other process can start until the running rendering process has finished. These features lead to the following requirements for a scheduling algorithm for VRML applications:

1. A scheduling algorithm must not require a *priori* knowledge of computation requirements of tasks, because it is difficult to predict the computation requirements of the rendering process.
2. The algorithm must provide some QoS guarantees even in the case of CPU overload, because the accumulated requirement from the multiple tasks may exceed the processing capacity of the CPU.

We used the Start-time Fair Queuing (SFQ) algorithm [3,4], which is a resource allocation algorithm that can be used for achieving fair CPU allocation and that meets the above requirements. Let us describe fair allocation. If w_a *is* the weight of task a and $E_a(t_1,t_2)$ is the total length of service carried out in the interval $[t_1,t_2]$ by the CPU for task a (i.e., the total CPU execution time), a CPU allocation is considered to be fair if the normalized services received by two tasks a and b are equal (i.e., $\frac{E_a(t_1,t_2)}{w_a}-\frac{E_b(t_1,t_2)}{w_b}=0$), for all intervals $[t_1,t_2]$ in which the two tasks are runnable. Therefore the objective of a fair scheduling algorithm is to minimize the difference of the normalized services received by two tasks (i.e., to ensure that $\left|\frac{E_a(t_1,t_2)}{w_a}-\frac{E_b(t_1,t_2)}{w_b}\right|$ is as close to zero as possible.).

To achieve this objective, SFQ assigns two tags, a start tag and a finish tag, to each task, and schedules tasks in the increasing order of their start tags. The SFQ algorithm is defined as follows:

1. When quota q_a^i is requested by task a, it is stamped with the start tag S_a^i of task a, computed as:

$$S_a^i = \max\{v(R(q_a^i)), F_a^{i-1}\} \qquad i \geq 1, \qquad (1)$$

where $v(t)$ is the virtual time at time t, $R(q_a^i)$ is the time at which the i th quota is requested by task a, and F_a^i is the finish tag of task a. F_a is initially 0, and when the i th quota finishes execution, it is incremented as:

$$F_a^i = S_a^i + \frac{E_a^i}{w_a}, \qquad (2)$$

where w_a is the weight of task a and E_a^i is the measured service received by task a (i.e., the CPU execution time of task a).

2. Initially the virtual time $v(0)$ is 0. When the CPU is busy, the virtual time at time t, $v(t)$, is defined to be equal to the start tag of the task in service in time t. On the other hand, when the CPU is idle, $v(t)$ is set to the maximum of the finish tag assigned to any task.

3. Tasks are serviced in the increasing order of the start tags

4.2 Service Adaptation

Once the QoS manager allocates the CPU resource to a task for the visualization of a dynamic 3D world, the task tries to provide best-effort QoS using the allocated resource. To finish the rendering process of the task within a allocated CPU bandwidth (i.e., within the specified target frame time), we used an adaptive algorithm that adjusts the criteria of the LOD selection based on feedback regarding the time required to render previous frames. If the previous frames took longer than the target frame time, the criteria for LOD selection is increased so that coarser than normal models are chosen and future frames can be rendered models quickly.

This adaptive approach works reasonably well for the visualization of a dynamic 3D world in which there is a large amount of coherence in scene complexity from frame to frame. However, in visualization of more discontinuous 3D worlds, scene complexity can vary radically between successive frames. We first used a predictive algorithm similar to [2] in which the LOD selection is determined predictively, on the basis of the estimated rendering time of multiple models including LOD models, rather than reactively, solely on the basis of the time required to render previous frames [8]. We implemented the predictive algorithm on an IBM RS/6000 43P/GXT-1000-2 with IBM AIX and OpenGL [7] and the estimation of the rendering time is reasonable in the system environment. However, it was quite difficult to estimate the time required to render the world in the system environment of Microsoft Windows 95 and Direct 3D [5, 10], since the required rendering time varies considerably from frame to frame. Consequently, we adapted the reactive algorithm for the RVE browser on Windows 95.

In the reactive algorithm, a closed-loop control system is constructed to adjust the LOD selection criteria according to the "stress" imposed by the system load, to maintain a user-specified frame rate. The algorithm compares the desired and actual frame times and calculates the system load. On the basis of the user-supplied stress parameters, the algorithm adjusts the global LOD scale factor by increasing it when the system is overloaded and decreasing it when the system is underloaded. In this way, the system load is monitored and adjusted before each frame is generated. The reactive algorithm is as follows:

```
/* Current load */
load = actualFrameTime * desiredFrameRate

/* Decrease stress level when below low load level */
if(load < lowLoad)
        stressLevel -= stressCoef * stressLevel
else
/* Increase stress level when above high load level */
if(load > highLoad)
        stressLevel += stressCoef * stressLevel

/* Bound the stress level */
if(stressLevel < stressMin)
        stressLevel = stressMin
else
if(stressLevel > stressMax)
        stressLevel = stressMax
```

The parameters *lowLoad* and *highLoad* define a hysteresis band for system load, *load*. The first if-test demonstrates that when the instantaneous system load is within the hysteresis band, there is no change in the system stress level, *stressLevel*. When *load* is below *lowLoad* or above *highLoad*, *stresslevel* is reduced or increased, respectively, until the system load is stabilized within the hysteresis band. If the size of the hysteresis band is too small, oscillatory distress is the probable result. A *stresslevel* > 1.0 increases the LOD ranges, with the result that coarser than normal models are drawn and the system load is reduced. On the other hand, a *stresslevel* < 1.0 (indicating that the system is underloaded for frames) decreases the LOD ranges so that finer-than-normal models are drawn.

5. Implementation and Results

We implemented the RVE browser on the Microsoft Windows 95 platform with Microsoft DirectX (including DirectDraw, Direct3D, and DirectSound) and the Sun Microsystems Java Development Kit, mostly in C/C++ and partly in Java for the interface to JSAI implementation. We implemented the QoS control module described in the previous section, and included the module in the RVE browser. In the RVE browser, the view in which a 3D world is displayed corresponds to a task. As a policy for determining the weight of each task, we use the states of rendering windows associated with the tasks, such as focussed, minimized, and maximized. When the rendering window of a task obtains the window focus or is maximized, the weight of the task is increased. On the other hand, when the window loses the window focus or is minimized, the weight is reduced. This policy satisfies users' desire to manipulate 3D models and scenes in a sub-window of interest and examine the scene in the window in detail. Figure 3 shows an overview of task scheduling in the QoS manager of the RVE browser. The QoS manager enqueues all tasks ready to run in a task queue, and sorts entries in the task queue in increasing order of their calculated start tags. Then it selects the top task in the queue and executes the task. Once the task has been scheduled, the QoS manager pauses until the visualization process of the scheduled task in a frame is completed. When the task is finished, the QoS manager receives the measured CPU execution time of the task and the computed finish tag of the task. It also adjusts the weight of each task according to the state of the rendering windows associated with the tasks obtained by the window event handler.

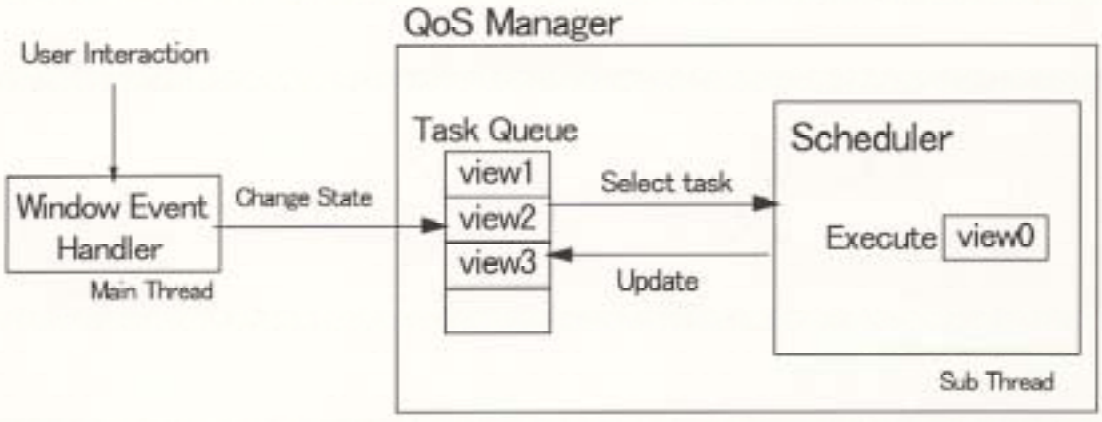


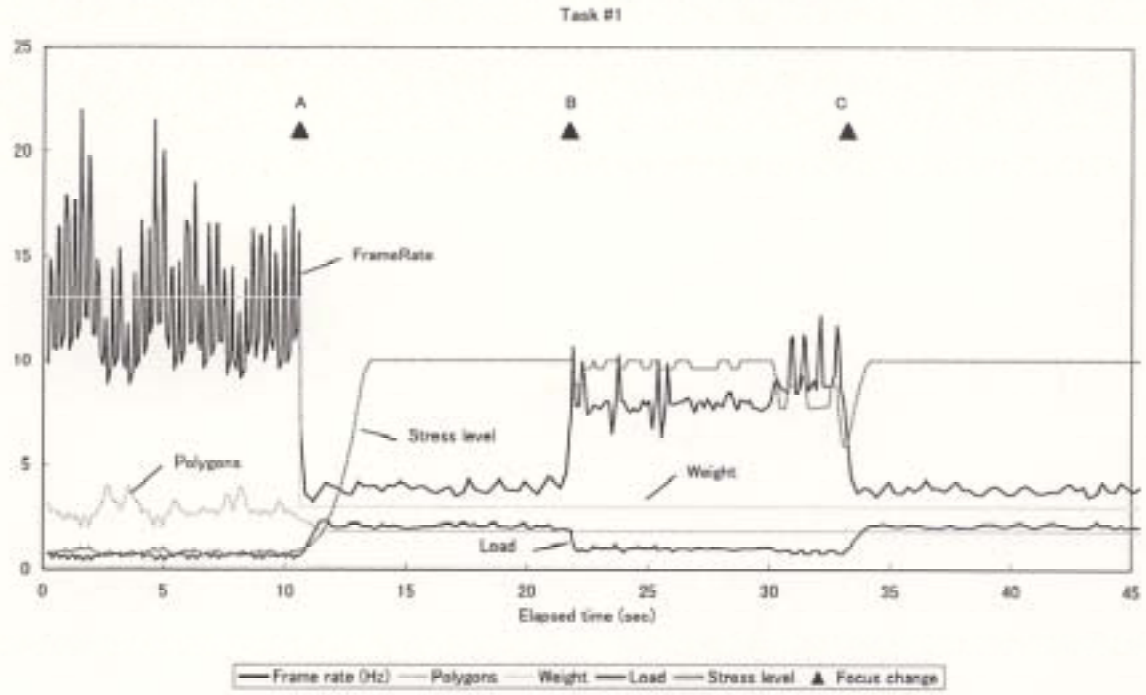
Figure 3: Task scheduling in the QoS manager of the RVE browser

We evaluated the performance of our implementation by using an IBM IntelliStation M Pro with an Intel Pentium II 300-MHz CPU, 64 MB of memory, and a Matrox Graphics Millennium II AGP. We used two kinds of VRML-based models with multiple LODs to run two tasks in separate view windows used to display the models simultaneously in the RVE browser, and an MPEG player as a dummy task to exhaust the CPU resource. Figure 4 shows two VRML-based models in the RVE browser. In this case, the upper window is focused, and therefore the details of the wheel parts of the car are not visible, since the LOD of the model displayed in the lower window is low.

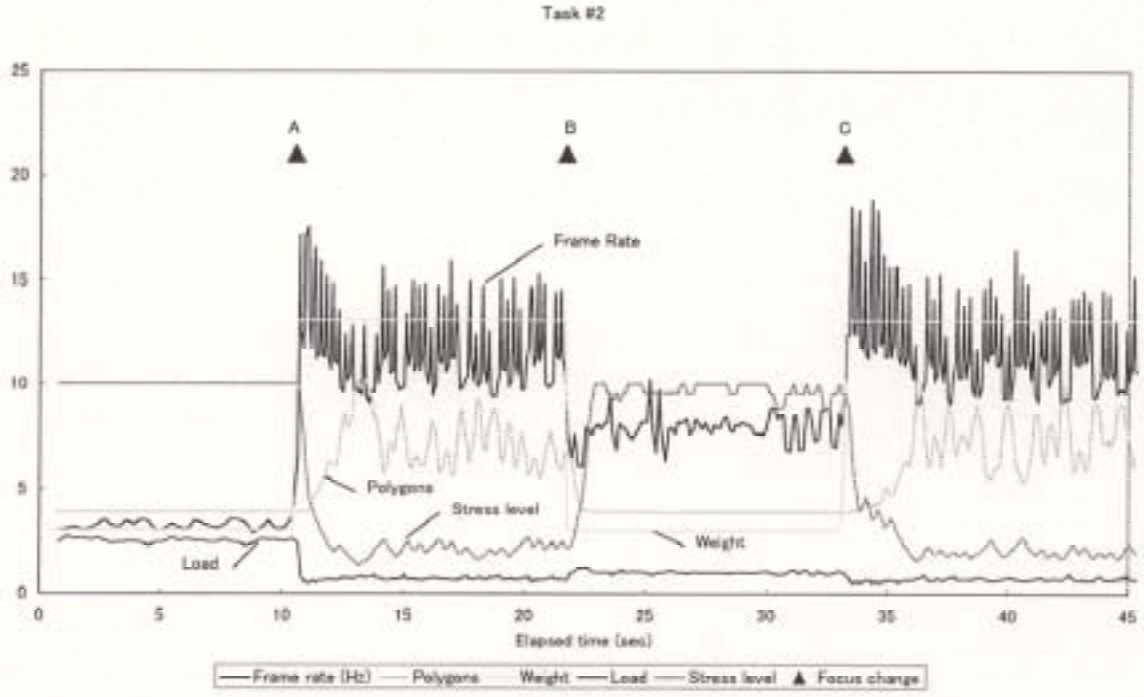
Figure 4: An image displaying two VRML models in multiple windows of the RVE browser.

Figure 5 shows an example of the variations in the frame rate, the number of polygons to be displayed, the weight for task scheduling, the load, and the stress level, of each of two views, to illustrate how well the RVE browser controls the QoS of multiple views. During the first ten seconds (until the triangle marker "A" was reached), task No. 1 (shown in the lower window in Figure 4) was focused. During the next twelve seconds (from the mark "A" to mark "B"), the window focus moved from task No. 1 to No. 2

(shown in the upper window in Figure 4). During the next twelve seconds, an MPEG player ran and played a short video clip, during which both the views in the RVE browser lost the window focus. During the last twelve seconds, the MPEG player stopped and task No. 2 grabbed the window focus again.



(1) Task No. 1



(2) Task No. 2

Figure 5: An example of QoS management

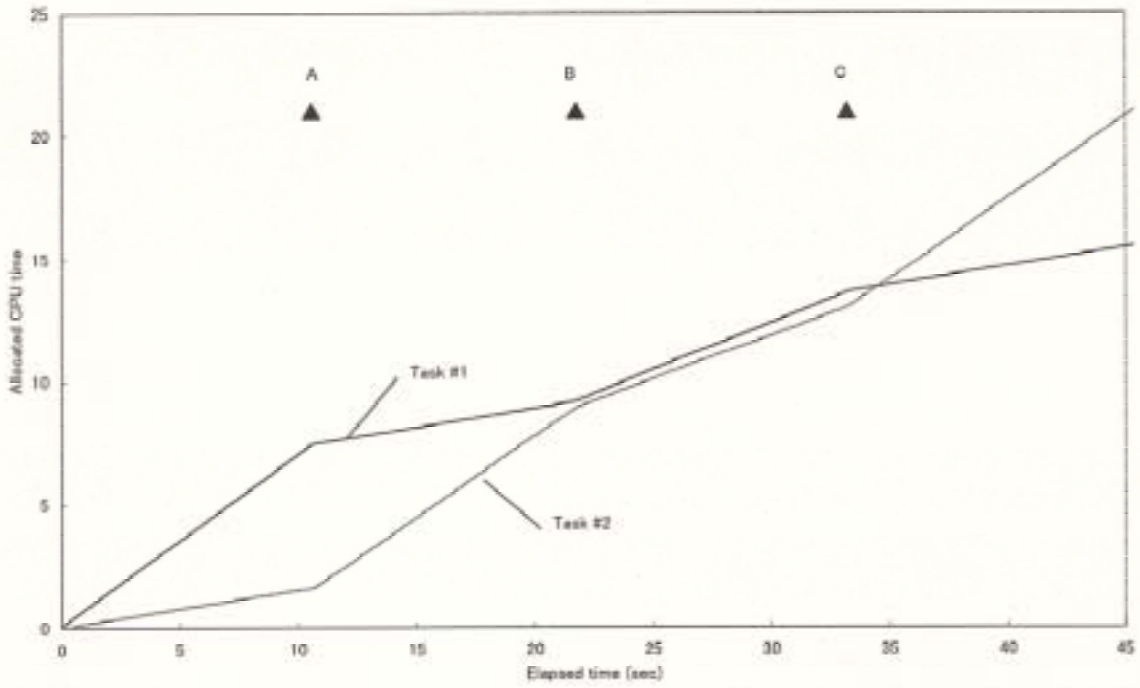


Figure 6: CPU time allocation result

According to the state of the window focus, the RVE browser adjusts the QoS levels of each view. Once the rendering window of task No. 1, for example, is focused, the QoS manager reduces the weight of task No. 2 and increases the weight of task No. 1, and then each task adjusts the LOD of the scene by using the service adaptation capability of the task. The LOD of the scene of task No. 2 is reduced so that the task can render the scene under the reduced CPU resource. On the other hand, the LOD of the scene of task No. 1 is increased, since the task is allocated more CPU resource than it requires to render the current LOD of the scene.

Figure 6 shows the accumulated CPU time allocated to each of the two tasks of the RVE browser. During the first ten seconds (until the triangle marker "A" was reached), the QoS manager allocated more CPU resource to task No. 1 than to task No. 2. During the next twelve seconds (from mark "A" to mark "B"), the manager allocated more CPU resource to task No. 1 than to task No. 2. During the next twelve seconds (from mark "B" to mark "C"), the manager allocated the resource almost equally to both the tasks, since their weights were equal in this period. Figure 7 shows how the tasks of the RVE browser are scheduled around the instant of a window focus change.

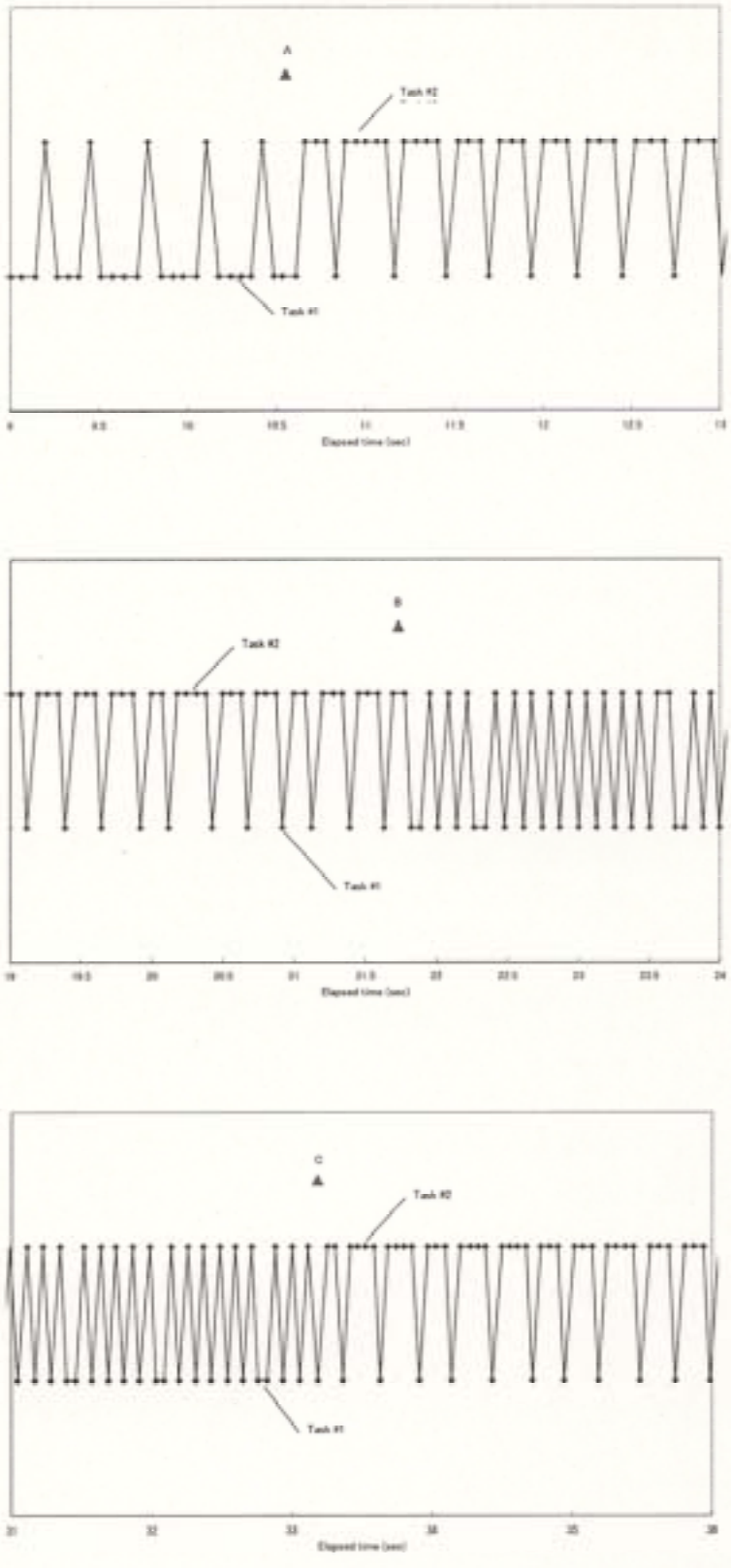


Figure 7: Task execution order

When the VRML browser and MPEG player run simultaneously, since both are CPU-intensive applications, an MPEG movie cannot be played smoothly. When the RVE browser loses its window focus, the QoS manager reduces the CPU resource allocation to the browser by using the latter's QoS control feature, so that the MPEG player can use more CPU resource. Consequently the camera motion of the movie becomes very smooth. In this way, according to the state of window focus, the QoS manager controls the amount of CPU resource to be allocated, and the browser adjusts the frame rate and level-of-detail of the scene to provide best-effort QoS for the available level of CPU resource.

6. Conclusion

We have described a method that provides a capability for Quality of Service (QoS) management to the Virtual Reality Modeling Language (VRML) browser. We implemented the QoS management capability in the Reactive Virtual Environment (RVE) browser, which is a VRML 2.0 browser that we developed. The QoS management consists of two layers: (1) the resource allocation layer, which manages the CPU allocation and execution control for multiple tasks by means of a scheduling algorithm, the SFQ algorithm, and (2) the service adaptation layer, which is a reactive load management mechanism that provides best-effort services under the allocated CPU resource by automatically selecting LOD of a 3D scene and adjusting the frame rate. QoS management by this two-layer approach guarantees a real-time system response and causes graceful degradation of the QoS when CPU system resource is limited. Since an RVE browser task also spontaneously releases part of the CPU resource when it loses the window focus, CPU-intensive applications other than the RVE browser, such as an MEPG player, can obtain more CPU resource to maintain their QoS levels. We demonstrated experimentally that the two-layer approach is very effective for dynamic 3D and multimedia applications.

Acknowledgements

This work was supported by grants from the Information Technology Promotion Agency in Japan (IPA). We would like to thank Akio Koide, Junk-Kook Hong, Kiyokumi Kawachiya, Masaki Aono, and Ryutaro Ohbuchi of IBM Tokyo Research Laboratory for their advice in this project. We would also like to thank Yoshiaki Sawano, Kaoru Hosokawa, Masayuki Ryuhtoh, Yoshio Horiuchi, and Carmine F. Greco of IBM for their contributions to the RVE system development.

References

[1] Airey, John M., John H. Rohlf, and Frederick P. Brooks, Jr., "Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments," ACM SIGGRAPH Special Issue on 1990 Symposium on Interactive 3D Graphics, Computer Graphics, Vol. 24, No. 2, pp. 41-50, 1990.

[2] Funkhouser, Thomas A., and Carlo H. Sequin, "Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments," Proceedings of ACM SIGGRAPH '93, In

Computer Graphics, Annual Conference Series, pp. 247-254, 1993.

[3] Goyal, Pawan, Harrick M. Vin, and Haichen Cheng, "Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks," Proceedings of ACM SIGCOMM '96, pp. 157-168, 1996.

[4] Goyal, Pawan, Xingang Guo, and Harrick M. Vin, "A Hierarchical CPU Scheduler for Multimedia Operating Systems," Proceedings of Second Symposium on Operating Systems Design and Implementation (OSDI '96), pp. 107-121, 1996.

[5] Microsoft Co., "Microsoft Direct3D Retained Mode SDK," http://www.microsoft.com/directx/dxm/help/d3drm/, 1998.

[6] Mitra et al., "Living Worlds," http://www.vrml.org/WorkingGroups/living-worlds/, 1998.

[7] Neider, Jackie, Tom Davis, and Mason Woo, "OpenGL Programming Guide: The Official Guide to Learn OpenGL," Addison-Wesley, 1993.

[8] Rohlf, John and James Helman, "IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics," Proceedings of ACM SIGGRAPH 94, In Computer Graphics, Annual Conference Series, pp. 381-394, 1994.

[9] Rossignac, Jarek, and Paul Borrel, "Multi-resolution 3D approximations for rendering complex scenes," IFIP TC 5. WG 5.10 II Conference on Geometric Modeling in Computer Graphics, Genova, Italy, 1993.

[10] Stein, Michael, Eric Bowman, and Gregory Pierce, "Direct3D: Professional Reference," New Riders, 1997.

[11] Strauss, Paul and Rikk Carey, "An Object-Oriented 3D Graphics Toolkit," Proceedings of ACM SIGGRAPH 93, In Computer Graphics, Annual Conference Series, pp. 341-349, 1993.

[12] Taniguchi, Masaaki, "Event Processing for Complicated Routes in VRML2.0," Proceedings of Third Symposium on the Virtual Reality Modeling Language (VRML98), pp. 83-88, 1998.

[13] VRML Consortium Inc., "The Virtual Reality Modeling Language ISO/IEC 14772-1:1997," http://www.vrml.org/Specifications/VRML97/, 1997

[14] Yoshida, Ryo, "Reactive Virtual Environment System: LivingWorlds server and VRML browser," IBM Research Report, RT0234, 1998.

[15] Yoshida, Ryo and Carmine F Greco, "Reactive Virtual Environment System: LivingWorlds multi-user world", Proceedings of Virtual Environments 98, pp. 55-1-55-12, 1998