ED 482 108 IR 058 796

AUTHOR	Calef, Chris; Vilbrandt, Turlif; Vilbrandt, Carl; Goodwin, Janet; Goodwin, James
TITLE	Making It Realtime: Exploring the Use of Optimized Realtime Environments for Historical Simulation and Education.
PUB DATE	2002-04-00
NOTE	13p.; In: Museums and the Web 2002: Selected Papers from an International Conference (6th, Boston, MA, April 17-20, 2002); see IR 058 778.
AVAILABLE FROM	For full text: http://www.archimuse.com/mw2002/papers/calef/calef.html/.
PUB TYPE	Reports - Research (143) -- Speeches/Meeting Papers (150)
EDRS PRICE	EDRS Price MF01/PC01 Plus Postage.
DESCRIPTORS	*Computer Games; *Computer Simulation; Foreign Countries; Higher Education; Material Development; Models; Museums; World Wide Web
IDENTIFIERS	*Computer Models; Japan; Three Dimensional Design; University of Aizu (Japan); Usability; *Virtual Museums

ABSTRACT

As museums and educators struggle with the challenges of presenting their material in a digital format, many overlook the application that has spearheaded the development of virtual reality for the average consumer: 3D realtime game engines. These 3D game engines offer greater versatility, usability, maturity, simulation, and codebase than most current 3D realtime frameworks. At the University of Aizu (Japan), the authors are using the Quake engine in conjunction with the Povray raytracing engine to attack the problem of visualization and simulation from two sides. A temple from northern Japan that users can experience in realtime has been modeled. However, to deal with the limitations of simulation in realtime, the ability for users to select a view for greater detail has been added. The selected view is rendered in the background as the users continue to travel through the temple and is delivered in a separate window when finished. This paper describes relevant game paradigms, their usefulness, and the research in detail, including problems and solutions discovered along the way. It concludes with suggestions on how this work could assist museums and educators in simulation and modeling. (Contains 11 references.) (Author/MES)

PAPERS
Museums and the Web 2002

Making It Realtime: Exploring The Use Of Optimized Realtime Environments For Historical Simulation And Education.

Chris Calef, Turlif Vilbrandt, Mythworks, USA/Japan, Carl Vilbrandt, University of Aizu, Japan, Janet Goodwin, Aizu History Project, USA/Japan, James Goodwin, University of California, USA

http://www.myth-works.com/simtools and http://www.u-aizu.ac.jp/~vilb/aizu_history

A&MI

Archives & Museum Informatics
2008 Murray Ave.
Suite D
Pittsburgh, PA
15217 USA
info@archimuse.com
www.archimuse.com

Search A&MI

Join our Mailing List.
Privacy.

Abstract

As museums and educators struggle with the challenges of presenting their material in a digital format, many overlook the application that has spearheaded the development of virtual reality for the average consumer: 3D realtime game engines. These 3D game engines offer greater versatility, usability, maturity, simulation and codebase than most current 3D realtime frameworks. At the University of Aizu, we are using the Quake engine in conjunction with the Povray raytracing engine to attack the problem of visualization and simulation from two sides. We have modeled a temple from northern Japan that users can experience in realtime. However, to deal with the limitations of simulation in realtime, we have added the ability for users to select a view for greater detail. The selected view is rendered in the background as the users continue to travel through the temple, and is delivered in a separate window when finished. Our paper will describe relevant game paradigms, their usefulness, and our work in detail, including problems and solutions we have discovered along the way, and conclude with suggestions on how this work could assist museums and educators in simulation and modeling.

Keywords: Simulated Environments, Virtual Reality, Realtime, 3D Games, Cultural Heritage, Historical Restoration, Virtual Museum, Quake, Id Software, Povray

Introduction

The Web offers unprecedented opportunities for museums to escape the physical confines of their buildings and reach a vast new audience. Many institutions have begun to take advantage of this by using VR and 3D technologies, but this use does not take full advantage of the virtual information world we are entering. One factor to be aware of is the power of realtime modeling and simulation. This is useful both for creating engaging on-site exhibits, like the popular "virtual fish tanks" (MIT, 1999), and for reaching across the internet with immersive, educational simulation programs. There are widely varied approaches to realtime 3D modeling and simulation, but this paper will focus on game engines and solutions. Typical 3D environments used for academic and historical purposes are often hard to navigate, obtuse and short on interactivity. But 3D games, on the other hand, have been designed from the ground up to be usable, enjoyable and very interactive. The 3D gaming environments typically go further than just modeling a static object or environment, but instead try to simulate some additional properties and interactions. People in the academic community should embrace and extend gaming concepts and technology as a means of simulating, storing, testing, and transmitting their ideas.

3D Gaming = Usability = Learning

Taking advantage of realtime 3D game paradigms yields several advantages: increased user enjoyment, increased use of the application, and transparent learning. Games have been designed from the ground up for usability and fun. The more hours a user spends in a game environment, the better it tends to do in the market place, and the more money it makes. As a result, the primary focus for most game companies is on making 3D environments that are highly functional, easy to learn, and enjoyable to use.

By embracing game code and techniques, a planned application can instantly come up to speed with a usable, sophisticated interface, in an environment that has proven staying power. Users come back to their favorite games again and again. Game techniques in conjunction with modeling and simulation can yield a very interesting potential byproduct for academic applications: transparent learning. If users are constantly interacting with a program for enjoyment, they will pick up a variety of skills and knowledge without approaching it as a learning experience, and in some cases learn without even realizing it. Currently, Mythworks and the Oregon Center for Applied Science are working under a grant for the National Institute of Health on a project that involves teaching children how to navigate and cross streets safely. One of the goals of the program is to make children feel as though they are playing a game, allowing the skills to be learned through modeling and simulation. The more a child, or any user, comes back to such an environment, the more the modeling is reinforced and the more the skills become "second nature". Making sure an environment is "playable" goes a long way toward this goal. This has also been demonstrated in flight-simulator-based games in which users who fly fighter jets in air combat and other such engaging simulations demonstrate a highly accelerated learning curve when learning to fly a real plane (Hampton, 1994). There is at times a balance that must be struck for an application between accurate modeling and playability, but there is no doubt that the tools and techniques developed by the game industry hold great promise for academics and educators.

A Tested Interface for Free

One of the most important elements to be gleaned from the gaming community is the set of user interfaces that have evolved for control of player movement and view direction. In contrast to other virtual environments, the typical 3D game has evolved as an arena for competition between players. This means there is no room for anything but the most efficient interface between the player and the computer. In VRML, on the other hand, the single worst feature is its viewer interface. Most VRML players are difficult at best to navigate in; usually only the mouse is used, often with a very counterintuitive set of controls. In Cosmo, for instance, one must grab the center of the screen and then stretch a line out from it to move in a certain direction. Compare this to the very natural and efficient command standard used in Quake (ID, 1996) gaming, in which the mouse determines "look" direction and buttons or keys apply forward, backward, and sideways motion. The rest of the interface choices in Cosmo and other

VRML clients just get worse, and there is usually no easy way to modify them. Also, Quake and other game engines offer support for additional human interfaces like joysticks. Although in some VRML players it is possible to program a new interface, this can be time-consuming and difficult.

Perhaps the most important lesson to be learned from gaming interfaces is that many of these choices have been made not by the gaming companies themselves but by the users, over long periods of trial and error. Game companies learned long ago to leave interface choices up to the user, and as a result the users have found the best combinations for different types of games, goals, hardware interfaces, and handicaps. This natural evolution of 3D navigation should not be ignored, and as the gaming companies learned, it should always be easy to change.

Making It a Sim

Yet another aspect of many 3D games is representation and simulation of the natural world. Nearly all such games implement basic Newtonian forces like gravity. Other games go much further. In Black & White, by Lionhead Studios Ltd., the user plays the role of a god who rules over the population and resources of a small island. The game simulates population growth and decline, natural disasters, disease, and social interaction. Weather in the game has an impact on the growth of vegetation and crops. Going even further, if a user is connected to the Internet, the game can actually check the player's online local weather report and simulate these conditions within the game environment.

While most museums have yet to discover 3D gaming and simulation techniques, in the past few years many have discovered the utility of "raytracing" programs for visualization. Applications like 3D Studio Max, Maya, Poser (for character animation), and many others have provided the capability for modeling virtual environments, enabling 3D "walk through" animations of historical sites, graphic representation of scanned artifacts, and so forth. Although there are too many examples to begin to list them all, one of the most impressive of these "walk throughs" was made by the "Virtual Olympia" project (Ogleby, 1999) directed by Cliff Ogleby of the University of Melbourne.

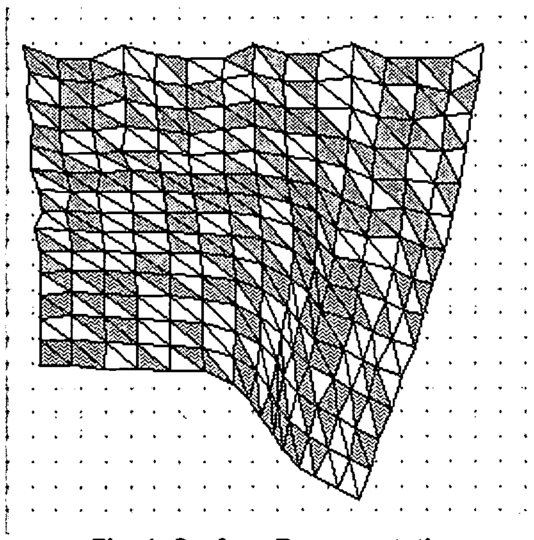

Fig. 1: Surface Representation

While raytracing software can be very useful for 3D visualization, we find that on their own, these applications lack many features that would be necessary for us to use them for modeling and simulation. Most commercial and open-source raytracing environments, and 3D games for that matter, focus exclusively on representation of surfaces. While surface representation can be adequate for creating relatively static scenes, more tools and better data sets are necessary in order to accurately portray a dynamic world. A good example is an animator who wishes to model a scene involving the eruption of a mountain, with rocks flying into the air. Using old-style raytracing software, the animator would have been required to "fake" the paths of the rocks, by defining arbitrary curves for them to follow. This was because the rocks were surface representations only, and as such even if the environment included gravity the rocks would have no mass for it to affect. Most packages nowadays do better than this, implementing some procedural tricks with the surfaces to give the impression of a gravity algorithm. However, these tricks are extending surface data and math beyond its natural and practical limits. Because the code is based on mathematical "tricks" it can be implemented in many different ways. This allows the primary commercial packages to each come up with their own method of handling and storing object or material properties, such as density, tensile strength, and mass distribution. This makes it impossible to share this kind of data between applications, or even within a single software package. Without this "real" mathematical data, accurate simulation is impossible, and without a shared format, the user will find it impossible to adequately combine models from more than one application. Most 3D games share these portability and accuracy issues.

As an example, consider an animator who is using Poser to do cloth simulation on a walking human figure, and then needs that human figure to brush against a tree created in PlantStudio. She will likely find that while the cloth has a collision detection algorithm it uses to respond accurately to the human figure, there is no such algorithm to enable it to interact with a model from another program. Furthermore, the cloth algorithm probably does not include the possibility of the fabric catching on a branch and tearing, because this situation does not happen in a software package focused only on human figures. In order to have an interaction between models, there is a need for an open object library that includes simulation logic, so that cloth, humans, and trees can be handled within the same procedural framework.

SEDRIS (Sedris, 2001) seems to be the best hope for an implementation of a common format for simulation, and we are excited to see how it develops. Many vendors readily support OpenFlight and SEDRIS formats. We have questions regarding the depth and breadth of this support, however. To the best of our knowledge, there has been no attempt to apply SEDRIS logic to any game framework except OpenFlight, and as part of our research we hope to make some contribution toward linking SEDRIS to the Quake engine.

Simulation logic can help an animator with tasks far beyond simple physical simulations like cloth behavior and falling rocks. Artificial intelligence for moving "actors" in the scene is one prime example. The value of introducing AI into a raytracing environment was proven dramatically in the recent film "Lord of the Rings: Fellowship of the Ring" with the use of the MASSIVE simulator program, which automated much of the combat AI for the movie. This enabled the director to create gigantic battle scenes that would have otherwise required a prohibitive number of animator hours. The same concept could be applied to more peaceful purposes, enabling animators to automate the life of an entire village, for instance, based on a few simple behavioral rules for each actor.

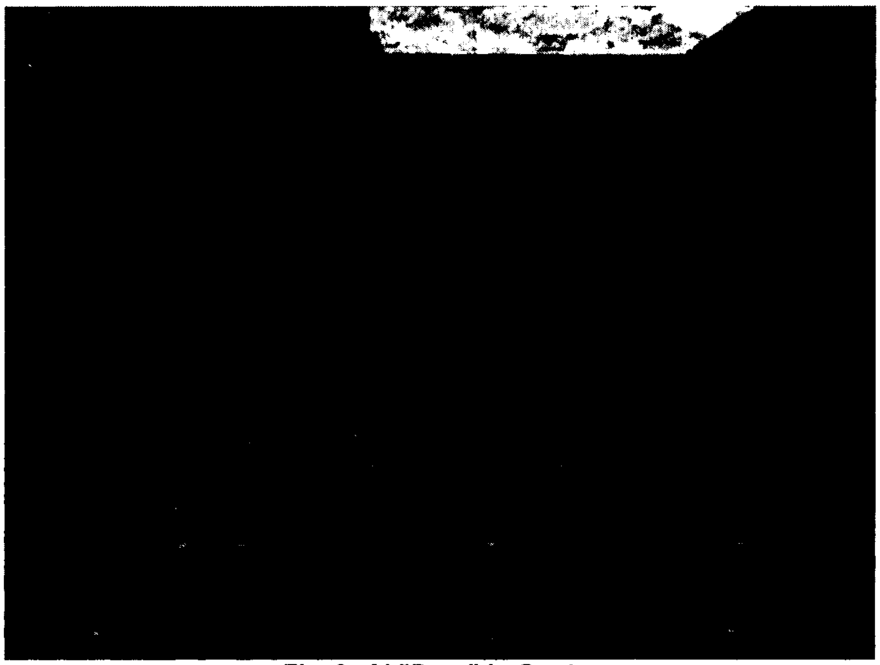

Fig. 2: AI "Bots" in Quake

For this kind of simulation to be really effective, a cross-disciplinary approach is necessary. For example, with a historical simulation, only part of the "rule set" will fall within the bounds of what we would ordinarily call "history", and the rest will fall into other academic fields, such as physics, architecture, geography, and botany. In order to provide a useful three-dimensional simulation of a Japanese village, for example, there would first need to be historical knowledge about what kind of people lived there, what crops they raised, what kind of buildings they lived in, and so forth. To make the simulation work, there would also need to be a set of rules describing how fast an object falls if someone drops it, what happens when it hits the ground, and so forth -- clearly the realm of the physicist. To animate the people of the village is also outside the traditional domain of the history department and belongs, rather, to the field of biomechanics. Meanwhile, the land area around the village should not be empty space; it should reflect the topography and ecosystems one would expect at this particular time and place. This calls for expertise from geographers and botanists. Meteorologists could contribute a working weather model, which would have a direct impact on the agricultural cycle.

The possibilities for simulation are as infinite as the complexity of the natural world. The choice for educators, then, becomes one of deciding, for a particular application, what form the simulation should take and what aspects of reality should be included.

Realtime versus Rendertime

In addition to their enormous utility for the animator wishing to create realistic still scenes and animations, simulation tools also make possible an entirely different kind of application. In realtime simulation using gaming methods, a user can interact with the program, as well as with other users, in an immersive and entertaining environment. This ability does not come without cost, however. In offline simulation, the only limitations to detail are the accuracy of the algorithms employed, and to a lesser extent the computing time necessary to reach the desired degree of accuracy. If necessary, the simulation can be allowed to run for days or weeks to attain this accuracy. In contrast, the restrictions imposed on a realtime environment are significant. Even at a relatively slow frame rate of ten frames per second (barely adequate for games), all computation for each frame must be finished within 100 milliseconds. Even on the most powerful gaming systems, this hardly allows for unlimited complexity in the simulation model. Many CPU-intensive algorithms are simply impossible to model in this environment. In many cases, processes that are too difficult and time-consuming for true realtime computation can be precalculated and rendered into prerecorded animations or stills, which are then sent to the user as a movie or used inside the realtime game environment, transparent to the user. Using a server-client model we can have real-time interactivity and simulation on a low power client machine, backed up by super computers or distributed computing to produce high detail simulations, images and animations.

Currently at the University of Aizu, we have modeled such a system to run on a single computer using Quake as the front end "client" with a thin bridge to Povray as the backend "server". Our test case is a model of Enichiji temple, from the Aizu region of northern Japan. (See (Vilbrandt, 2001) for an early version of the Enichiji model.) In order to provide an immersive environment, we have created a model of this temple which runs in Quake, and allows the "player" to climb the stairs, inspect the internal architecture, and move under, over, through, or around the temple in full realtime.

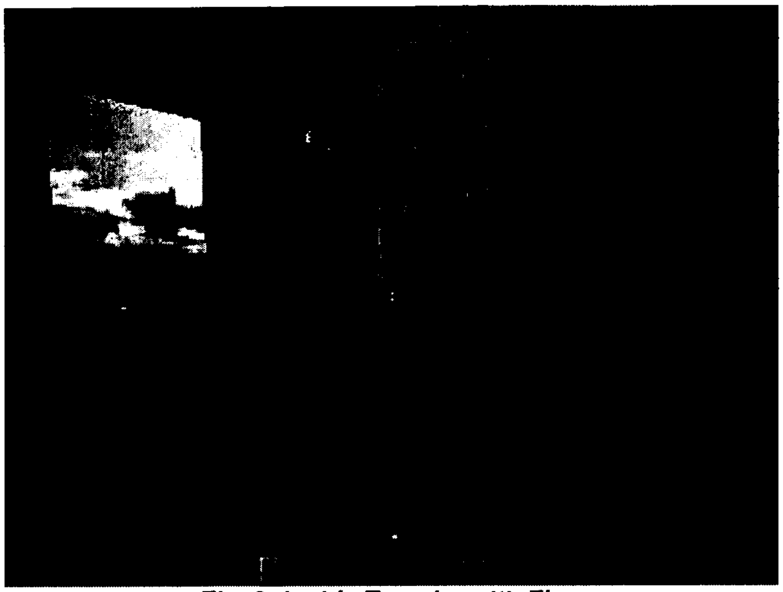
Fig. 3: Inside Temple, with Flag

However, to overcome the unavoidable limitations on complexity of a realtime application, the "player" may at any point choose a scene to be viewed in greater detail. Using Povray, a much more complex model of the chosen scene is rendered, creating still images or animation to be viewed in a separate window. We are working toward extending the complexity of this scene, so that in addition to the temple, there will be a section of terrain, flags blowing in the wind, various vegetation and rocks, an animated monk figure, and a flock of crows. This will give us a better test case for demonstration, because the interactions among the wind, the flags, the birds, and the monk's robes will be impossible to render in Quake at the same level of detail that would be possible in Povray.

Another way we have discovered to combine "rendertime" raytracing with "realtime" gaming is by precalculating special effects to be included in the realtime game. As a trivial example, imagine stirring cream into a cup of black coffee. The cloud shapes that swirl around the cup could be simulated using a particle-based fluid motion algorithm, but this is almost certainly too much computation to run in a realtime gaming environment for such a minor effect unless this action is for some reason critical to the game. Instead, the motion could be simulated in advance and stored as one of the animations possible for a "cup of coffee" object, and unless the user tried to pour the cream from the other side of the cup, he or she would likely never know the difference.

In our research, we found this technique of precalculating simulations to be mandatory for presenting any kind of adequate realtime simulation. Cloth animation is a good example; in our test model, a hanging cloth was draped in a doorway of the temple, and a wind effect simulation caused it to blow back and then settle down again. The cloth was first simulated using a particle algorithm in C++. It was then written out as a series of frames in Povray include files. When a suitable sequence of frames was found, after adjusting the parameters of the simulation to get the most realistic movement, then the coordinates of the cloth mesh vertices were imported to a Quake model format. After that, the same animation could be called from within Quake whenever game logic demanded.

Current Tools, Work and Web Interface

For an offline rendering application, we chose to use Povray (Hallam, 1995), an open source raytracing program with a number of features that we found useful. It has a fairly usable scripting language, but of even greater value was the ability to call an external application between frames. We use this option to call our executable program, written in C++, with the frame number and animation clock sent as arguments. The C++ code then handles all moving entities, physics, collision detection, etc., and after determining the new position and orientation for each visible entity in the current frame, writes out a POV script file, which is then rendered.

For our realtime game engine, we are using a modified version of the Quake game engine, released by Id Software under the GPL license. While a game engine has drawbacks in terms of supporting limited platforms and requiring users to download a piece of software, we feel that these limitations are more than balanced out by the speed, realism, overall versatility and extendibility offered by such a solution. Under GPL, any historical or educational game created with the Quake engine can be given away for free or sold for a profit, providing only that the source code is made available to the public. For academia, this should be a plus.

We have specifically chosen the open-source Quake game engine because it is one of the most-used 3D game engines. The engine is fast and small. It was designed 5 years ago for pentium class machines and therefore has a broad base of systems on which it can run. However, due to development by the open user community, this version of Quake has evolved to take advantage of new hardware and software techniques and has begun to rival even the latest Quake 3 engine from Id Software.. If a project based on the Quake engine is managed and engineered properly, it can have both low-end compatibility and high-end glitz and hardware optimization in the same application. In addition, Quake runs on ALL the major desktop operating systems, including Linux. Most importantly, both Quake and Povray have Free Software licenses allowing easy modification by anyone, and solving some serious problems associated with data transparency. This means three things: one, that we have a proven 3D code base to work with; two, that our work and that of others after us can be preserved; and three, that persons or institutions will be able to make adjustments or modifications of our work in the future without need of our presence or permission.

In the process of our research, we have also developed a library of C++ code which is available from our Web site. Some of this code would be redundant to anyone already using a high-level simulation system such as SEDRIS, but some of it may have relevance to people wishing to experiment with procedural approaches to Povray and/or Quake. Among the possibly interesting functions are:

- Landscape generation -- using the simplest possible application of the diamond-square algorithm.
- Buildings -- procedural generation of some very simple building shapes. Not accurate enough for a detailed model, but perhaps useful for low-LOD background structures.
- Human skeletal animation system -- takes input from motion capture, .data files, and quake .mdl files.
- Basic physics code -- including gravity, drag, acceleration, collision detection.
- Cloth simulation -- with limited collision detection against a mesh.
- Webinterface tools -- Some of our code uses Postgres as a backend

database server for storing and sharing simulation data. We have a developed an advanced html interface to manipulate and manage this database over the Web.

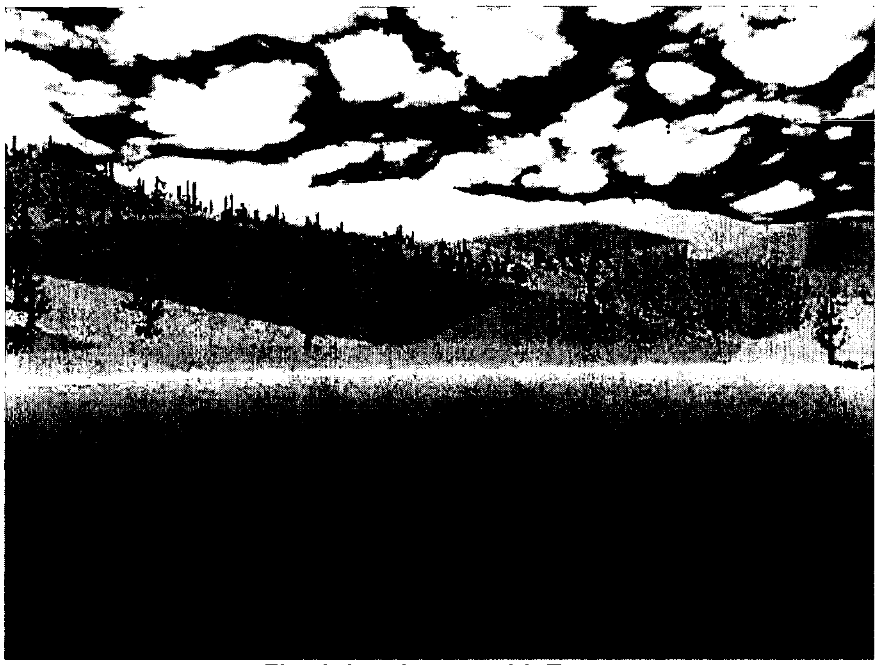
Fig. 4: Landscape with Trees

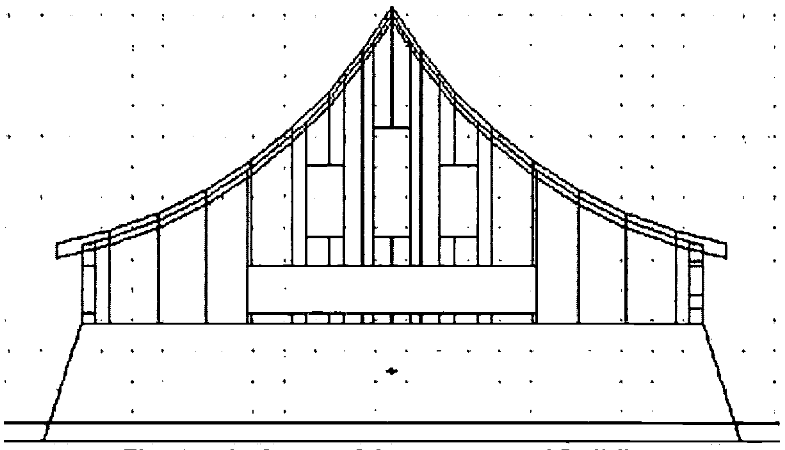
Fig. 5: Wireframe of Autogenerated Building

All of our code writes out to both Povray and Quake, in some form or another, unless it would be redundant in one of the applications. In Povray, most objects are written described as triangle meshes, whereas in Quake the format depends on the type of entity in question. Mobile, animated entities (humans, animals, etc.) are written as .mdl files, which contain a description of polygons, a set of animation frames, and a "skin" texture for the model. Larger, static entities such as buildings are written as BSP "brush" entities (see Feldman, 1997). Of course, for the same scene to function in both applications, the coordinate system must be identical. For our project, that meant adopting Quake coordinates to Povray.

Problems Unsolved

We have found that one of the major limitations in using Quake has been BSP (Binary Space Partition) representation of environments. The use of BSP trees enables the game engine to sort the entire scene into "potentially visible sets" of polygons. (For a more complete description of the BSP algorithm, see Feldman, 1997.) While this works just fine for the simple concave environments used in Quake (where the environment consists entirely of rooms linked by tunnels), it does not work so well for complicated and convex objects and environments. The worst-case example so far has been the compound convex shapes found in the roofs of Asian architecture. Some convex roof examples are acceptable while others produce such complicated BSP trees that they become unusable. Another issue with BSP trees is they are not easily modifiable in realtime, making them difficult or impossible for some aspects of simulation.

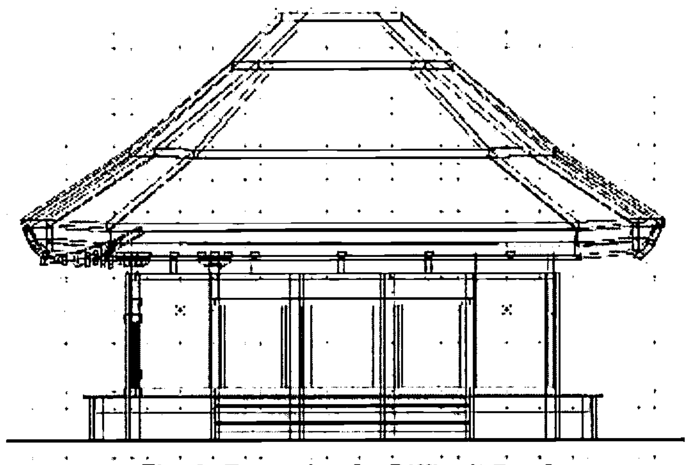

Fig. 6: Example of a Difficult Roof

Another difficulty with the Quake engine is its inability to support large open terrain areas, making scene design very challenging when one is attempting to simulate an outdoor environment. It is usually necessary to artificially "box in" a scene using terrain entities such as a "wall" of forest, a hedge, a rock wall, or some other device to limit the potentially visible area from any given point in the scene. When too much complexity is visible at any given time, the game can error by "graying out" whole sections of the scene, interfering dramatically with the overall sense of realism.

Finally, the game can handle only a limited number of moving entities in a scene, like human figures, animals, etc. The exact number is dependent on the hardware configuration and the amount of AI being run for each entity, but the limit seems to be something on the order of 40 to 60 entities visible at one time. This is enough for most simulation purposes, but it can pose significant limitations on scenes involving large crowds of people, herds of animals, and so forth.

Future Work

This work is in its infancy, and as such currently has very little simulation support outside of basic physics and collision detection. In addition to adding links to SEDRIS, we have plans to incorporate a much improved skeletal animation system utilizing a genetic algorithm to minimize energy expenditure. (See the work of Schmitt and Kondoh, in Schmitt, 2000). We also intend to increase our use of particle and voxel systems for solids representation, and to add links to the HyperFun library (Pascoe, 1996) for function representation of solids (F-Rep).

Our main focus in the near future will be to extend our current Quake-to-Povray framework. We intend to fully implement a realtime Quake client communicating with a backend Povray server across the Web. Users will be able to choose a scene or path through the realtime environment and have it rendered and returned to them in high detail. This will allow institutions to offer the equivalent of mainframe processing power to the average home computer user.

Conclusion

We see the combination of simulation tools with realtime, potentially multiplayer gaming to provide museums and academics with any number of new ways to involve visitors in interactive learning experiences. Also it has been demonstrated that an effective application can create a community around it. This means larger and lasting participation in given fields, exhibitions or focus and for museums it means more visitors. Some applications might include the following:

- Users could participate in an industrial process, running an airplane or automobile factory, or being part of the operation of an early coal-fired electric plant.
- Museums could have "game rooms" with many computers networked together, allowing visitors to take part in multiplayer interactive simulations.
- A school class could become the population of a farming village, and spend the afternoon planting wheat, learning to fix sheds and houses using appropriate tools and resources, deciding what crops to plant, where to clear forests, where to trade and what to barter for. They could learn firsthand the need for pottery, because when the villagers stack the grain in open piles or in sheds, the water comes in and their next year's supply of food rots. Accurate simulation could require the players to build a kiln hot enough to fire the clay that the villagers dig up nearby, and that would help determine the amount of wood that the village harvests. This sort of game can teach constantly without the participants ever even becoming aware of the instruction.
- In an astronomy exhibit, visitors could view the orbits of the planets around the sun, or stars around the galactic core. Users could navigate a virtual spaceship or modify the masses of the stars and planets and observe the forces of gravity.

The content in these facilities could be updated regularly, for very little cost, by museum staff using an html interface to the simulation database. Constantly changing content could keep people interested and returning to the site or to the institution to see what is currently "going on" in the simulation. Any of these projects could also be made available over the general Internet. For museums, this means allowing people anywhere in the world to have access to the information stewarded by the museum, and as a byproduct potentially increasing membership.

As a final note, much of the logic and work done in the field of modeling and simulation seems to be related either to violent video games or military applications. We wish to be part of a move toward the exploration of more peaceful and educational subjects for simulation. Considering the dangers currently faced by heritage sites and natural resources as a result of human war, overpopulation, and over-consumption, it can only be a good thing for museums and public educators to have access to better tools for reaching out to the general public. Using realtime game engines can create immersive and entertaining environments for educating members of the public about the processes and forces that impact our lives, our history, and our future.

Acknowledgements

This work would not have been possible without the support of Drs. Naotoshi and Yuko Takeda of Shiokawa, Fukushima-ken, Japan; Mr. Kyoichi Okubo of the Aizu Digital Valley Promotion Association of Shiokawa, Fukushima-ken, Japan; Ms. Miyako Aotsu of Aizu-Wakamatsu, Fukushima-ken, Japan; Id Software and the online Quake community; the Povray team and online user group; and the Free / Open Source Software movement.

References

Feldman, M. (1997). Introduction to Binary Space Partition Trees. Last updated 25-Aug-1997. http://www.geocities.com/siliconvalley/2151/bsp.html

Free Software Foundation. (1985). Last updated 30-July-2001. http://www.gnu.org/fsf/fsf.html

Hallam Oaks Pty. Ltd. (1995). POVRay. Last accessed 20-Feb-2002. http://www.povray.org

Hampton, S., Moroney, W., Kirton, T. & Biers, D. (1994). The use of personal computer-based training devices in teaching instrument flying: A comparative study. Daytona Beach, US: Embry Riddle Aeronautical University

Id Software. (1996). Last updated 29-Oct-2001. http://www.idsoftware.com

MIT Media Lab & Nearlife. (1999). The Virtual Fish Tank. Last updated 03-Dec-200. http://www.mos.org/exhibits/current_exhibits/virtualfishtank

Ogleby, C. (1999). Virtual Olympia. Last updated 03-Nov-2001. http://www.phm.gov.au/ancient_greek_olympics

Pasko, A.(1996). HyperFun Project. Last updated 19 -Dec-2001. http://www.hyperfun.org

Schmitt, L.M. & Kondoh, T. (2000). Optimization of Mass Distribution in Articulated Figures with Genetic Algorithms. Aizu, Japan: University of Aizu

SEDRIS. (2001). Synthetic Environment Data Representation and Interchange Specification. Last updated 03-May-1998. http://www.sedris.org

Vilbrandt, C., Goodwin, J.M. & Goodwin, J.R. (2001). Digital Digging: Computer Models of Archeological Sites:: Enichiji in Aizu, Japan. *2001 PNC Pacific Neighborhood Consortium Annual Conference and Joint Meetings.* Taiwan: Computing Centre, Academia Sinica