Submitted to IEEE CG&A

Making It Realtime: Exploring the use of optimized realtime frameworks for education and the web

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

Keywords: Simulated Environments, Virtual Reality, Realtime, 3D Games, 3D Engines, 3D Web Graphics, Quake, Povray

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

Abstract

The utility of 3D game engines for delivery of educational content is explored and developed. Emphasis is placed on the difference between polygonal surface representation and actual 3D simulation, taking into account object properties and physical characteristics. Through a combination of the open-source Quake engine and the Povray raytracing engine, a Japanese temple is simulated in two levels of detail and made available both through an interactive realtime simulation and a more detailed but slower raytraced environment. Tools are presented for web-based access to a simulation database, which can be used to design and modify 3D environments.

Making It Realtime: Exploring the use of optimized realtime frameworks for education and the web

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

The web offers unprecedented opportunities for dissemination of information of all kinds. From its humble beginnings in hypertext and 2D graphics, improvements in bandwidth and computer performance have allowed web servers to provide full 3D content. However, these very improvements in hardware are bringing to light the weaknesses of most current web 3D software. Specifically, reliance on polygonal surface representation as the industry standard for visualization of 3D environments imposes severe limitations on potential accuracy of a virtual environment. Additionally, most VRML and other web 3D players suffer from difficult and counter-intuitive user interfaces and inflexible coding environments.

In our research, we have focused on the use of optimized realtime game engines as an alternative method of viewing 3D content. These applications provide built-in logic to handle basic world properties like gravity and collision detection. In addition, 3D games provide highly intuitive and efficient user interfaces, advanced network code for multi-user distributed environments, and an open coding environment allowing unlimited customization.

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 will tend to do in the market place, and the more money it will make. 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 a user is constantly interacting with a program for enjoyment he or she will pick up a variety of skills and knowledge without approaching it as a teaching experience, and in some cases without even realizing it. Currently, we are working with the Oregon Center for Applied Science on a grant for the National Institute of Health 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.[1] 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, which 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

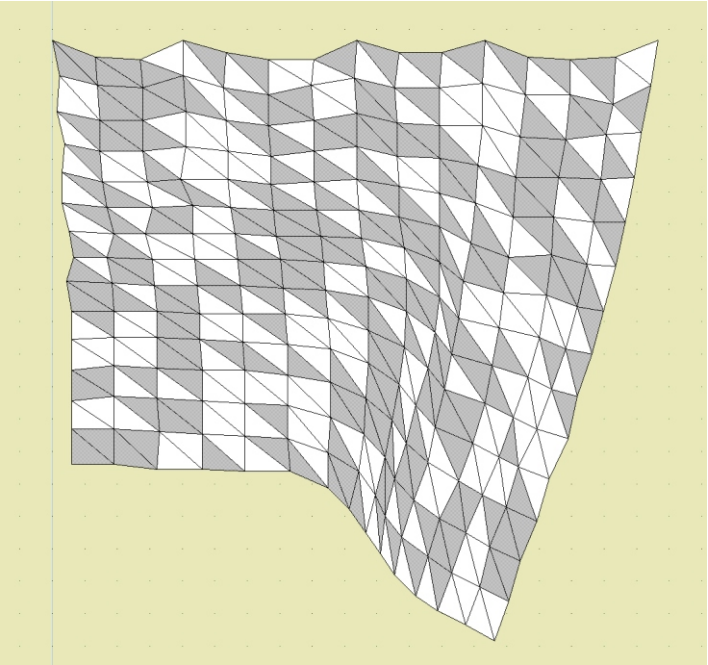
and then stretch a line out from it to move in a certain direction. Compare this to the very natural and efficient command standard used in Quake gaming, in which the mouse determines "look" direction and buttons or keys apply forward, backward, and sideways motion. The rest of the interface choices in Cosmo and other VRML clients just get worse, and there is usually no easy way to modify them. Quake and other game engines also 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 farther. 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.

Most commercial and open-source 3D games, and raytracing environments 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.



1 Polygonal meshes commonly used for surface representation are really only good for visualization and not accurate simulation.

3D modeling and simulation software and data desperately needs a common and accurate mathematical base. Computers are now powerful enough to provide an answer using functional representation, 3D point sets, and voxels. Fortunately, there is already extensive work being done in these areas, with the HyperFun[4] language being the best all-encompassing solution. The language is simple, open-ended, and heterogeneous, includes data sets like voxels, and it is multi-dimensional. Using functional representation and continuous functions means that a model can now be as accurate as the computer or modeler can make it. It is no longer limited to arbitrary polygons along a surface that define its volume, but instead is defined by a real function that mathematically represents its internal structure. Materials and properties can be applied, allowing for very accurate simulation and analysis of models. We are currently working to incorporate the HyperFfun language and specification into the Quake game engine and web technologies.

In functional representation any object in three-dimensional space is defined by a function of point coordinates F(x,y,z). This continuous real-valued function is positive inside the object, negative outside, and takes zero value on its surface. Similarly, a multidimensional object is defined by a function of several variables F(x1, x2, x3, ..., xn). For example, an object changing in time can be defined by F(x,y,z,t) with t representing time. A great variety of shapes can be modeled using this approach. You can find many examples at HyperFun.org

HyperFun provides us with an open framework for accurate transferable models and a great foundation for simulation, but does not provide us with an open framework for the simulation logic itself. 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 [2] 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.

SEDRIS is a collaborative project between the US government, industry, and academia, to provide a common standard for representation of environmental data, and loss-less, non-proprietary interchange of such data.

2 AI "Bots" in Quake - Using procedurally generated movements as well as input from motion capture, these entities can be programmed to interact intelligently with each other and with human players.

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.

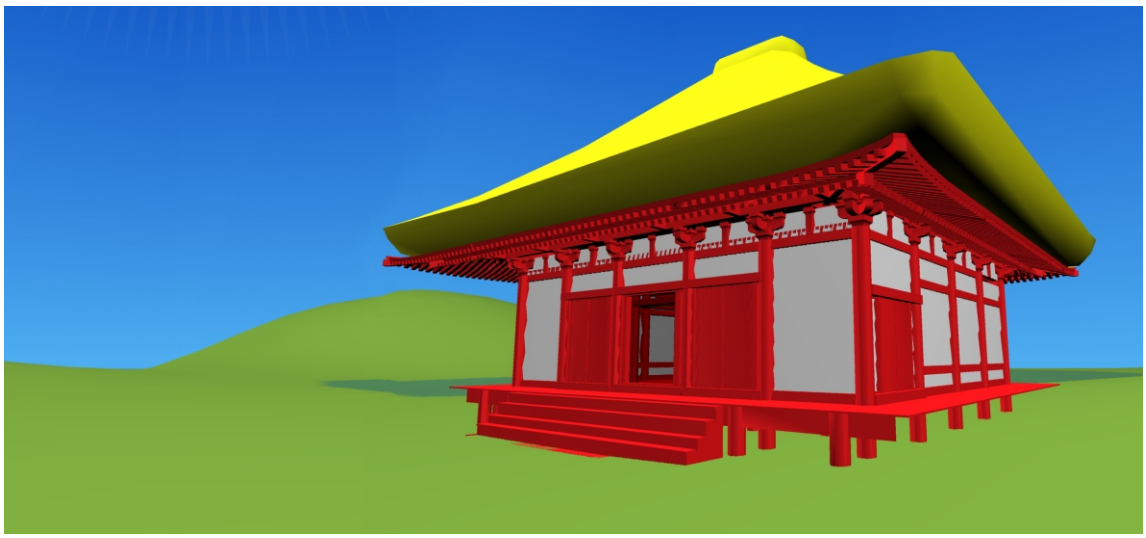
For this kind of simulation to be really effective, a cross-disciplinary approach is necessary. For example, with a historical simulation, only part of the "rule set" will fall within the bounds of what we would ordinarily call "history", and the rest would fall into other academic fields, such as physics, architecture, geography, and botany.

The possibilities for simulation are as infinite as the complexity of the natural world. The choice 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 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.

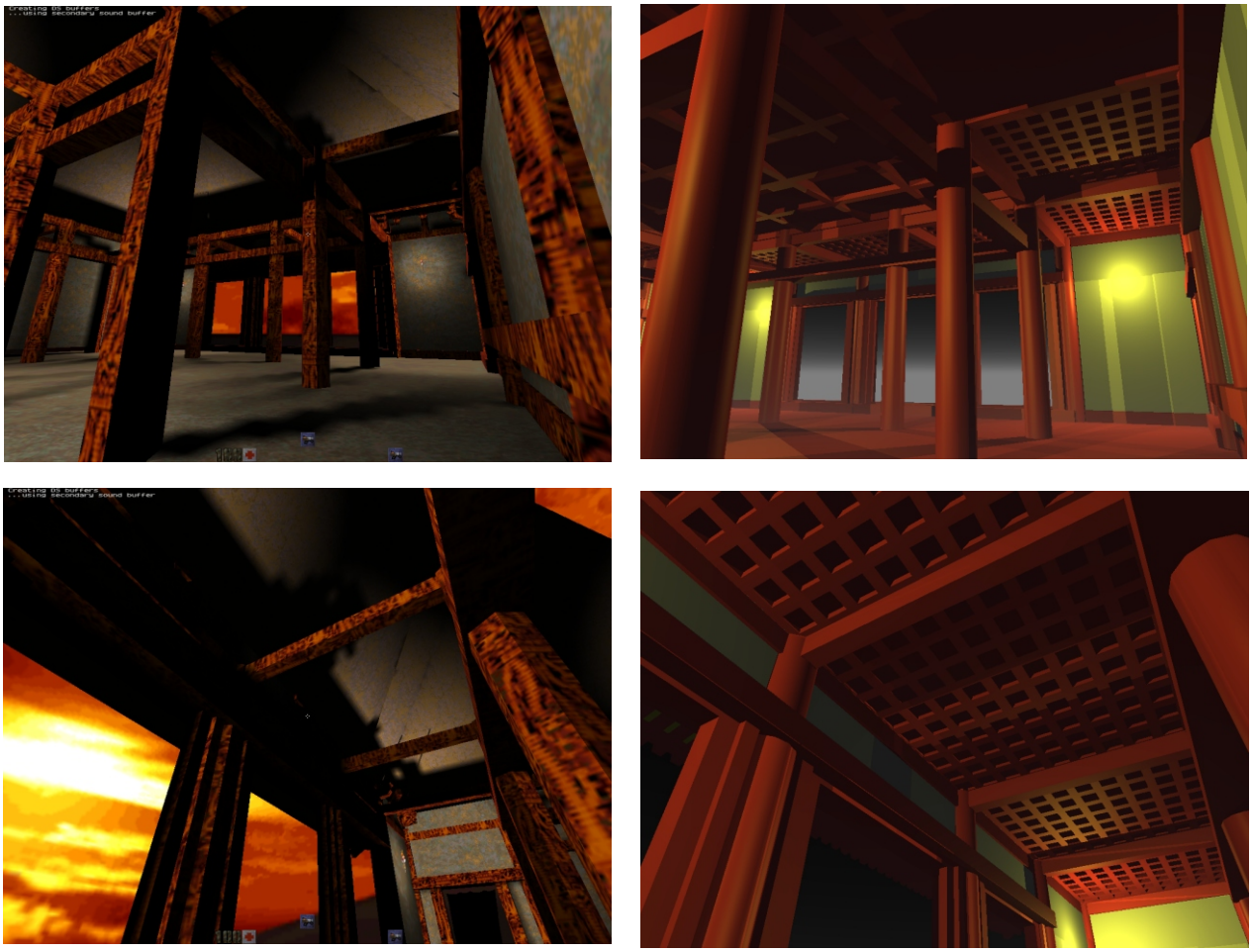
Currently, at the University of Aizu, we have modeled such a system using Quake as the front end client and Povray as the backend server. Our test case is a model of Enichiji temple, from the Aizu region of northern Japan. (For an early version of the Enichiji model, see [3].) In order to provide an immersive environment, we have created a model of this temple which runs in Quake, and allows the user to climb the stairs, inspect the internal architecture, and move under, over, through, or around the temple in full realtime.



3 Enichiji Temple Model

However, to overcome the unavoidable limitations on complexity of a realtime application, the user may at any point choose a view or a path through a scene to be rendered in greater detail. Using Povray on a server, a much more complex model of the chosen scene is rendered, creating still images or animation to be sent back to the client and viewed in a separate web browser 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 offloading complex simulation logic to an external server. 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. However, imagine 1000 of these cups in the environment, each calculating its “swirl”. This is almost certainly too much computation to run in a realtime gaming environment. Instead, it could be updated not in realtime, but in keyframes. A server would do the more processor intensive simulation, allowing the client to do the visualization and morph between the keyframes sent from the server. The client would update the server as to what was happening in the environment, “Did someone just stir their coffee?” and the server would modify the simulation. Using this method a simulation would only be limited by how often the simulation needs updating, for realism, and how much computing power is available on the server.



4 Inside the Enichiji Temple Model: on the left are views of the realtime model in low detail able to run on even a Pentium class processor, on the right are the high detail raytracings delivered to the client in seconds after the views were selected

Tools and Web Interface

For an offline rendering application, we chose to use Povray [7], 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 Povray 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 extendability 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, 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.

We are also using a backend database server for storing and sharing simulation data. We have developed an advanced HTML interface to manipulate and manage this database over the web. This allows anyone in the world to make complex changes in the environment by simply editing the database through a browser.

Current Work

In addition to adding links to SEDRIS, we are incorporating a much improved skeletal animation system utilizing a genetic algorithm to minimize energy expenditure (See the work of Schmitt and Kondoh, [5]) and have just finished adding multi-weighted “bones” to Quake. Our system now allows each vertex to be influenced by many different bones, making smooth "blended" mesh deformation possible in realtime. We also working to increase our use of particle and voxel systems for solids representation, and to add links to the HyperFun library for function representation of solids (F-Rep).

Our main focus at the moment is to extend our current network rendering framework using Quake and Povray. We are working on adding distributed computing capabilities to server side simulations. This will allow us to do things like large scale weather simulations with billions of particles and then update the Quake client with relatively little data. We are also working on making a web browser plugin version of the Quake client which will allow people to view these simulations right alongside HTML and not just in separate window.

Conclusion

We see the combination of simulation tools with realtime gaming to provide any number of new ways to involve people 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 field, exhibition or focus. Some applications might include:

- Users participate in an industrial process, run an airplane or automobile factory, or take part in the operation of an early coal-fired electric plant.

- A school class becomes the population of a farming village, and spends 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 guarantee that the players must build a kiln hot enough to fire the clay that the villagers dug 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 simulation, users could view the orbits of the planets around the sun, or stars around the galactic core. They could navigate a virtual spaceship or modify the masses of the stars and planets and observe the changing gravitational forces.

The content in these facilities could be updated regularly by non-programmers, using an HTML interface to the simulation database. Constantly changing content could keep people interested and returning to the site to see what is currently "going on" in the simulation.

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 scientists, educators, and institutions to have access to better tools with which to reach out to the general public. Using realtime game engines can create immersive and entertaining environments with which to inform 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

[1] 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

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

[3] 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

[4] V. Adzhiev, R. Cartwright, E. Fausett, A. Ossipov, A. Pasko, V. Savchenko, (1999). HyperFun project: a framework for collaborative multidimensional F-rep. *Implicit Surfaces '99*, Eurographics/ACM SIGGRAPH Workshop, J. Hughes and C. Schlick (Eds.), pp. 59-69, http://www.hyperfun.org

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