Medieval Playground

Matt Newcomb
Virtual Reality Applications Center
Iowa State University
Ames, IA 50014
talenos@vrac.iastate.edu

Tyler Streeter
Virtual Reality Applications Center
Iowa State University
Ames, IA 50014
streeter@vrac.iastate.edu

Abstract

This paper describes an interactive virtual environment in a medieval setting. The scenery for this virtual environment includes a terrain and castle ruins the user can explore. The user will be able to interact with the environment in exciting ways. Some examples are hang gliding and launching objects with catapults. An extremely interactive world such as this can be a great tool to show people the capabilities of virtual reality and go beyond simple navigation.

Keywords

Medieval playground, virtual reality, virtual environment, interactive, dynamics.

Introduction

The Medieval Playground is a virtual environment with castle ruins (Figure 1) and a surrounding terrain (Figure 2). The application is being developed to be used to demo the interactivity that is possible in virtual environments.

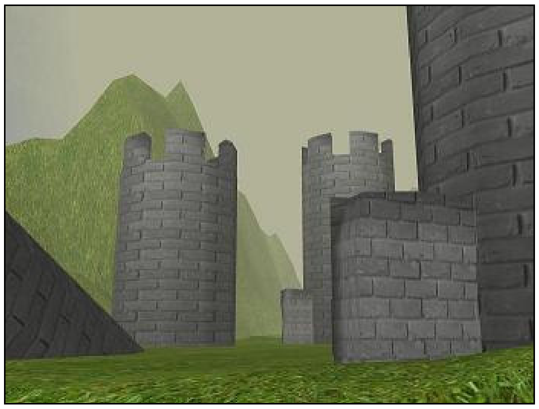
Figure 1: Castle Ruins

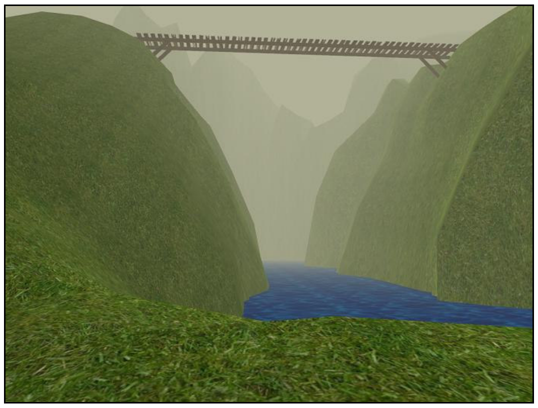
Figure 2: Terrain

Inspiration

The desire to go to interesting places and do dangerous things is the general inspiration behind this project, along with all kinds of medieval stories and films. The world is both realistic but also has a fantasy-like quality to it, which makes it a great setting for a virtual environment where anything can happen.

Some computer games such as Microsoft's Midtown Madness [1] and the popular Sims series by Maxis [2] give players freedom to interact with a world without having any concrete goals. These kinds of games are a taste of the kind of virtual world the user should be able to explore in our application. What the user experiences should be based on what they think up and decide to do, not an experience that is pre-decided by the application programmer.

We are taking this inspiration to create a realistic, virtual world with an action-oriented, adventurous theme that the user can explore freely. The user should be inspired to try new things based on the environment that they are in and the placement of usable objects throughout the world.

Users

The intended users for this project are basically everyone. There are only a few simple gestures that people need to know to move about the world and many people can figure them out just by trying what is natural to them. There will be no menus or any GUI to operate or learn so the complexities of the program will be better hidden from them.

This application is meant to be someone's first experience with virtual reality. They won't need any technical knowledge to interact in the environment so they will be free to take in the entire experience.

Background

Graphics are usually the showcase of high-end virtual reality applications. They can create almost perfect models of real life places or bring to life a world that could never exist. With today's graphics power, realistic images and effects can be rendered in real time making it sometimes difficult to distinguish reality from virtual reality.

The problem that some of these applications fall to is their lack of interactivity. The first time through one of these virtual environments may be very stunning, but the feeling can fade quite quickly and there is very little experience that can be gained from going through the environment a second time.

The Medieval Playground was created to highlight the interaction between the user and the world so they have ways to interact with the world beyond choosing the direction of navigation. This increased interaction gives the user a sense of presence that cannot be achieved by graphics alone.

Concept

We want the user to control where they go, so they can choose to go to places that most people would never think of exploring, like trying to swim through a river (Figure 3). This makes the experience more satisfying to the user because they have ownership of the choices they make; the experience is all theirs, not pre-scripted by a programmer or a virtual tour guide. To achieve this goal, we focused our efforts on a simple, intuitive input system and on a high level of interactivity in the virtual environment.

The application should be simple enough so that a child could operate it safely and without any problems [3], but still have enough interactivity in it that even an experienced VR veteran will be able to have fun while performing some of the interactions inside the environment. The interactions and dynamics in the world should all be realistic so that they are more believable to the user.

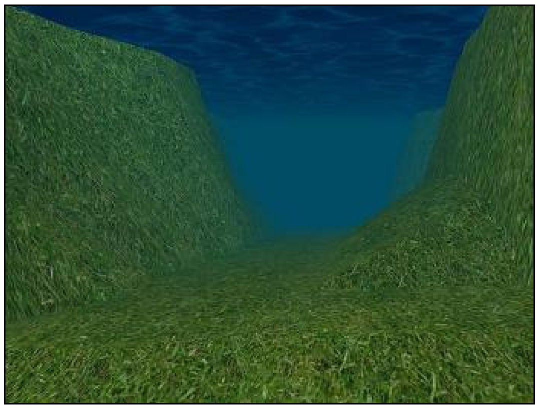
Figure 3: Underwater

Most people play a passive role during their first virtual reality experience because often the controls are difficult and hard to master, or only in the hands of an experienced VR user, or there are many people in a group. We would like to make this role more active with our project. To do this, we created a virtual reality application with a simple user interface that still presents users with a wide array of interactive components.

Previous Work

People, especially children, are flocking to see virtual reality applications due to the hype that has been generated by the media lately and their desire to see new and cutting edge technology [3]. This means that applications meant for a wide public audience need to be very simple to use so that even a child could operate the system.

Implementation

At first, the most daunting portion of the project was the interactive components. We asked ourselves how we could maximize the level of user interactivity in the virtual environment and stay within the time constraints of a semester-long project. We decided to focus on simulated physics in our virtual world to give users this high level of interactivity. Therefore, the three major components of our project are: the graphics, the dynamics, and the input method. In addition, we decided to make both a VR version and a desktop version of the application.

Graphics

We chose to use OpenGL for our project and decided not to use a scene graph. Because our virtual environment does not have a complex hierarchy of objects, a scene graph would not have been beneficial, although we did initially begin our project using one. Initially the project started in an OpenSG scene graph, but it wasn't worth the extra complexity.

The 3D textured models in our project were created using Discreet's 3D Studio Max. We chose to export our models into the WaveFront OBJ format for loading into our project because it is a simple format that is widely supported. We tried to optimize the size of our texture files by using JPG compressed pictures.

Other graphical effects in the project were done using OpenGL. An example of one of these effects is the fog. Our application normally uses gray fog, but the fog color changes to blue when the user goes underwater.

Dynamics

To create realistic interactions we needed a way to simulate physics. We decided to use the Open Dynamics Engine [4] to help with these realistic simulations. This open source software handles collision detections between primitive shapes (e.g. spheres and cubes) and arbitrary triangular meshes (e.g. rolling terrains). It also simulates realistic physical interactions among 3D objects, allowing programmers to deal only with forces while the dynamics software handles positions, orientations, velocities, etc. The dynamics engine then returns all objects' positions and orientations to be used by the graphics API. Though this system can be used to handle the trivial task of keeping the user from walking through walls, it can also empower the user to interact in new ways with objects in the environment.

Objects in our application have two parts: a graphical part and a dynamics part. The dynamics part is the part that deals with all the forces and collision, while the graphics part controls how the objects are displayed on the screen. This means that a complex shape like an acorn can be simply represented by a sphere in the dynamics world. This is very useful for shapes that can be approximated by one of the dynamics primitives. The towers in our application have intricate tops, but their representation in the dynamics world is simply a capped cylinder.

Open Dynamics Engine includes a powerful array of functions. Some of the most common functions we used were:

<u>dWorldCreate</u> – creates a simulated physical world in which to insert 3D objects.

<u>dBodyCreate</u> – creates a physical object (just a point mass).

dCreateSphere/dCreateBox – creates the physical representation of a physical object.

dBodySetPosition – sets the position of a physical object in world coordinates.

dBodySetMass – adjusts the mass of a particular physical object.

dJointCreateHinge – creates a hinge joint between two physical objects; this joint constrains the motion of the two objects to keep them from moving far apart.

dBodyAddForce – add a force to a physical object in a particular direction; this type of action is very common and replaces the typical 3D graphics paradigm of setting objects' positions explicitly.

dSpaceCollide – checks for collisions among all objects in the world and adjust the forces on each object appropriately.

dWorldStep/dWorldStepFast1 – moves the physical world ahead by a given amount of time (in our case, we adjusted this time delta based on the application's frame rate to make the physics work in real time); the dWorldStep function is more accurate, but the dWorldStepFast1 is more optimized for real time applications and adjusts the amount of time spent calculating physics based on a value set by the programmer.

Using a robust dynamics engine creates endless ways for users to interact with the virtual environment without being forced to see pre-scripted animations. For example, users in the Medieval Playground can pick up bricks or pieces of wood, carry them to new locations, and throw them (Figure 4). There is also the possibility of using objects as tools to perform some task (e.g. stacking blocks to climb onto a high ledge or using a hang glider to float across a chasm).

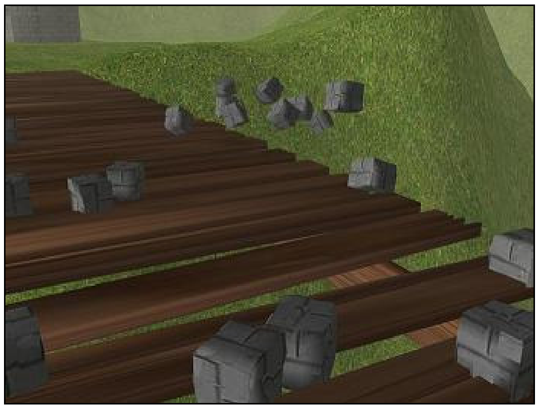
Figure 4: Throwing Bricks

The dynamics engine allowed us to create a realistic trebuchet (Figure 5) that works based on physical principles rather than pre-scripted animation. This trebuchet lets users launch various objects found in the world. It even lets users launch themselves through the air if they decide to. The dynamics engine lets a lot of interesting interactions happen and it allows realistic interactions that the users comes up with on their own, without the programmer having to worry about every action that the user might do.

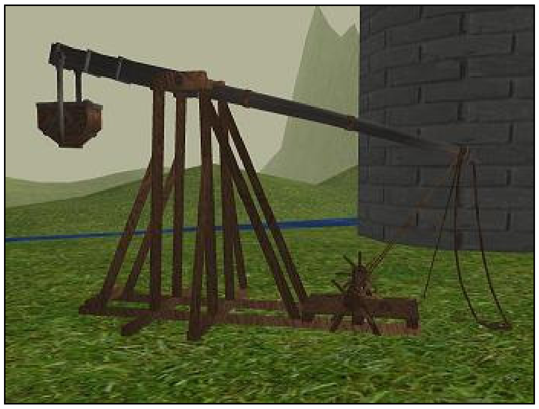
Figure 5: Simulated Trebuchet

Input Method

We chose to use a pair of gloves as the sole interface because we thought it would be more natural to use gestures than a wand or other device. The user will have no other visual interface to interact with, so other input devices don't make sense in our situation.

We wanted to keep the number of gestures as low as possible and make them very simple and intuitive. This would give the user more freedom to explore and interact with things since they don't have to worry about learning all the gestures and don't need to waste as much time training to use the equipment.

For navigation we used the pointing gesture, which would move the user in the direction they were pointing and allow them to rotate while they were pointing as well. Rotating the user while moving them forward makes it difficult to orient the world specifically, but it is smoother than having a separate rotate and move command. Ideally the user should be able to just walk where they would like to go.

To implement the throwing action, we came up with a simple method to approximate the throwing direction and strength. In our code we maintain a list of the past five positions of the user's hand. Then, we the user opens his or her hand (when throwing or dropping an object), we calculate a throwing vector, using the past five hand positions to determine the strength and magnitude of this vector. This effectively gives us how much force to apply to the object and which direction it should be forced.

The most natural gesture for interacting with objects in the world is a closed fist. All of the objects in the environment are used by grabbing or grasping the object. Even our idea for using a hang glider would just require the user to grip on to the glider by making two closed fists.

These two gestures, grabbing and pointing, are the only ones that are needed for moving and interacting in our world, so there are a small number of simple gestures for people to remember. Most people probably won't even need to be told what they are to figure it out.

List of Classes

The following list of classes presents the general structure of our project implementation:

Base3DObject – base object from which all other objects in the world are derived; has a position in world coordinates.

ODEObject – derived from Base3DObject; adds Open Dynamics Engine functionality to the object; each ODEObject can optionally be represented visually by an OBJ model.

OBJModel – encapsulates an OBJ model.

LightObject – an OpenGL light object derived from Base3DObject; because its parent class has a position, it can be moved easily by the programmer.

ODEWorld – encapsulates the ODE world functions; includes such universal factors as gravity, friction, and overall physics accuracy.

ODECamera – represents the position of the user; uses an invisible sphere to keep the user from walking through objects – the user moves around by applying forces to this object.

ODEBox – simple class to represent ODE's primitive box shape.

ODESphere – simple class to represent ODE's primitive sphere shape.

ODECappedCylinder – simple class to represent ODE's primitive capped cylinder shape.

ODETrimesh – uses geometry loaded from an OBJ file to create an arbitrary triangular mesh; useful for creating terrains.

SkyBox –simplifies the creation of a skybox.

Timer – used to get the amount of time elapsed since the previous frame.

ODEHuman – composite object made from primitive ODE shapes joined together to represent a human body.

Neuron – represents an artificial neuron; used solely by the NeuralNet class.

NeuralNet – a collection of simulated neurons that can be used to control an ODEHuman.

Desktop Version

The desktop version of the application will allow people to interact with the environment in a 2D window on a PC. Navigation and interaction must be mapped to the keyboard and mouse, so it is less intuitive but allows someone to get a feel for the environment without having to enter a CAVE environment. We designed the code for the desktop version similar to an ordinary VRJuggler application so that we can port code between the desktop and VR versions easily.

The desktop version will be the testing bed for the improvements that we plan to implement in our project because it is slightly more stable and our dynamics engine works better on Windows and Linux than it does with Irix.

VR Version

The VR version uses most of the same code as the desktop version but runs in VRJuggler so that it can be configured to run in most VR facilities. We originally designed the application to be used with a HMD, but we later decided that it would be better to use a CAVE environment because it gives the user more space to move around freely.

We recommend using gloves for interaction because it is more natural than a wand device for our type of application. The two main interactions the users will perform are navigating (by pointing) and picking up and releasing objects (by grabbing). Other devices, like a wand, can still be implemented, but would not be as natural to use. We hope that we can continue working on this project in the future so that it can be used in tours and demos to show the more interactive side of what virtual reality can accomplish.

Difficulties

The most notable difficulty we had was porting our software to the Irix operating system. Open Dynamics Engine would not compile with the Irix MipsPro compiler, and we could only compile VRJuggler with MipsPro on Irix. Thus, our two main pieces of software would not work together. Because of this problem, we eventually decided to work with Linux instead.

There was initially some trouble with the 5DT wireless gloves at first because the transmitters for them were switched around, so when we tested the gloves they gave us slightly odd results at first. Since we decided to switch to Linux and run the application on the baby cave, we decided wired gloves would work because the user won't be turning around so much. We still haven't been able to get the gloves working in Linux since there isn't much support for this at the VRAC. Presently, we are developing our project with wand-based interaction. When we have time, we will implement our glove-based interaction ideas.

A big concern from the beginning was if the dynamics engine's collision detection would perform well if our terrain was a triangular mesh. After the terrain was implemented we determined that there was not a major performance loss. Collisions between the user and the terrain worked well, but some of the primitive dynamics objects did not collide as well, specifically the cubes and capped cylinders. Spheres seemed to work the best, so a possible change is to try to represent most objects as spheres. A lot of the accuracy in the collision is given up for speed, but we can get collisions that are realistic enough that most people will not notice.

Conclusion

We started this project with very ambitious goals, but because it was the first time that either of us has developed a virtual reality application, we didn't know exactly how much we could accomplish. We are happy with the results that we achieved this semester and that we got most of the main aspects of the application to work like we wanted them to.

Once we got over the initial hurdle of learning the insides of VRJuggler and dealing with some hardware issues, we were able to get a lot of work done. Future VR projects that we work on should go a lot smoother after going through this learning process.

Future Work

We plan to continue work on this project by adding more kinds of interaction and enhancing the visual effects. Specifically, we would like to focus on interactions, modeling and textures, sound effects, dynamic environmental effects, and artificial intelligence/artificial life. These are incremental changes that can be added to the project without having to change the overall structure. The medieval playground could eventually be modified to have certain goals that the user needs to accomplish or perhaps even tell a story.

As far as the software goes, we would eventually like to get a version that works on Irix so that we can utilize the C6 like we had originally intended. This will still allow use to use the same source code because of the portability of VRJuggler.

Interactions

The purpose behind this application is to provide lots of different ways that the user can interact with the world. Users should be able to have a huge arsenal of different objects throughout the world that they can interact with.

Some other interactions we might try to implement are ones that also could be figured out by using gestures that make sense. A person could fire a bow and arrow at a target by holding the bow in one hand (a closed fist) and the arrow in the other (grasping with their pointer and middle finger). The user might be able to find a magic wand and cast spells by performing a gesture over time, like a figure eight.

Modeling and Textures

To create a more believable environment, we will spend more time creating 3D models with realistic textures. Making our existing models more detailed and creating new scenery objects will surely add to the realism which will make the user feel more absorbed in their environment.

Some notable things to fix are: the texture of the ground underwater, more foliage, a path up to the castle from the starting point, as well as models for the other forms of life that we hope to include like fish, birds, and other animals.

Sound Effects

Sound effects and music are a very important part of any virtual environment. We would like to make use of high quality music from medieval times to play in the background. Additionally, sound effects occurring when objects collide would naturally increase the level of realism in our virtual world such as splashes, footsteps, and other object-to-object impacts.

Eventually, human models may be added to the environment, so speech capabilities are also a possible upgrade to the application. The user could have conversations with them or participate in a live action story by listening to what they have to say.

Dynamic Environmental Effects

This category could include effects such as rippling water or shifting sun rays that change with the time of day (along with a fog effect that clears when "morning" ends). A particle engine would also be useful to create rain, snow, leaves that can be blown by the wind, perhaps even fall off if the wind has a great enough force, and of course fire. This will make each experience in the environment a little different.

Artificial Intelligence/Artificial Life

Artificial Intelligence and Artificial Life are things we would like to include in our project. This would allow us to put realistic looking behavior into our environment without using too much processing power. This will add a life-like quality to the world and will hopefully make for a richer experience.

One of these additions involves virtual people that can walk around the world without using pre-scripted animations [5, 6]. This idea already been tested (Figure 6) with virtual people that are controlled by neural networks.

Other ideas include a path finding squirrel that will run to an acorn (using the A* path-finding algorithm) if you throw one on the ground, realistic flocks of birds flying overhead, fish swimming through the water in schools, and bees swarming [6] the user if he or she were to upset them by maybe disturbing their virtual bee hive with a rock or stick. Perhaps users will even be able to fight with a virtual knight in shining armor with a sword that uses the dynamics engine to calculate the force of each blow. This would be along the lines of similar applications [7] that focus on virtual environments with artificially intelligent agents.

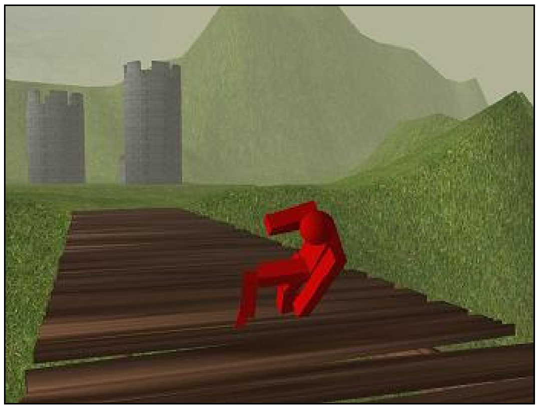

Figure 6: Artificially Intelligent Human

References

1. *Midtown Madness.*Microsoft Corporation. http://www.microsoft.com/games/default.aspx

2. T*he Sims*. Maxis. http://thesims.ea.com

3. M. Roussou, (2000). Immersive Interactive Virtual Reality and Informal Education. In Proc. of i3 spring days workshop on User Interfaces for All: *Interactive Learning Environments for Children*, Athens, Greece.

4. R. Smith. Open Dynamics Engine. opende.sourceforge.net

5. T. Reil, Husbands. Evolution of Central Pattern Generators for Bipedal Walking in a Real-Time Physics Environment. *IEEE Transactions in Evolutionary Computation.* pp. 159-168, April 2002.

6. K. Sims. Evolving Virtual Creatures. ACM Computer Graphics (SIGGRAPH '94). pp. 15-22, 1994.

7. D. Thalmann, C. Babski, T. Capin, N. Magnenat Thalmann, I. S. Pandzic. Sharing VLNET worlds on the WEB. *Computer Networks & ISDN Systems.* Vol. 29, No. 14, pp. 1601-1610, 1997.