Magic Carpet

Chad Austin
2003-12-12
IE 584x

Abstract

A new navigation system using a pressure-sensitive arrow pad for virtual environments has been developed. Technologies such as wireless networks, inexpensive game controllers, and remote device access can come together and make a variety of virtual environment interaction methods more accessible. A smooth navigation method for "magic carpet" pressure-sensitive pads was designed. To give the user something to navigate, a virtual world model has been created. Upon this structure, a game has been built. The components involved in designing and implementing this system, as well as limitations, are discussed.

Introduction

Konami's music-based game Dance Dance Revolution is getting pretty popular, not just in Japan, but all around the world. When playing Dance Dance Revolution (DDR), the player listens to rhythmic music and steps on four pressure-sensitive directional arrows with the beat. Flashing arrows scroll up the screen and let the player know which arrows to step on and when.

As a product gets more popular, prices for the product and related peripherals go down. DDR pads and mats are available on the common market for anywhere between $15 and $200. Since they're designed to be used in PlayStation and PlayStation 2 consoles controller ports, they have limited applicability outside of their intended purpose. However, companies such as LevelSix and Lik-Sang sell adapters that convert PlayStation controllers into USB Human Interface Devices. One such adapter is the EMS USB II; it goes for around $15 [1]. With this adapter, any buttons or axes that the controller provides become accessible from common PCs, or even any device that supports USB.

Now that the device is accessible from a PC, it has uses beyond playing games. It can be used in any application that would benefit from a pressure sensitive pad with many distinct regions. One purpose that jumps to mind right away is navigation in a virtual world by stepping on the arrow, or a combination of arrows, in the intended direction of travel.

In order to use a DDR pad within a virtual environment such as Iowa State University's C6 [2] or C4 [3], it must be accessible by the computer driving the system. Since few CAVEs have USB ports readily available, an external computer must be used to read input from the device, and make that information available across some channel to the virtual environment. The application within the VE can then treat the pad as just another input device, providing a set of analog axes and digital buttons.

Fortunately, the work to connect USB input devices to a virtual environment has already been done by a group of researchers and students at University of North

Carolina. Their work is called the Virtual Reality Peripherals Network (VRPN) [4]. VRPN allows virtual reality devices on several distinct computers to be used within a single virtual reality application. It transfers input data from the device over a network to the application, which can then act upon it.

Finally, after connecting the DDR pressure-sensitive pad to a USB adapter, then to a PC, then to VPRN, then to a virtual environment, the pad can be used within virtual reality.

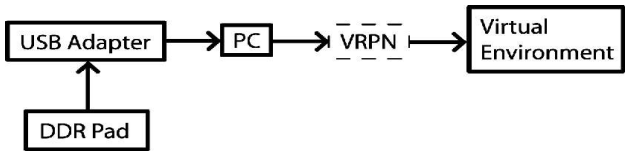
Motivation

Sometimes the need for new applications drives new technology. Other times, new technology inspires the applications for which it's used. The latter is the case here. Let us say that a Dance Dance Revolution pad is in the middle of the floor in a CAVE such as the C6, floating in a virtual world. What does that remind you of? A magic carpet! And thus this project was born.

Two games conceptually similar to this idea are PilotWings by Nintendo and Magic Carpet by Bullfrog Entertainment. They provided the mental imagery to plan out this magic carpet game. This Magic Carpet is inspired by Bullfrog's Magic Carpet, at least when it comes to visuals and navigation. The gameplay aspects are based on PilotWings, and will be explained in further detail later.

Design

Architecture



As shown in the diagram above, the DDR pad is connected to the virtual environment. The application uses inputs from the DDR pad to navigate around within a model. The next two sections discuss navigation and the model in more detail.

Navigation

In Magic Carpet, navigation is performed by stepping on an arrow. This is considered an intuitive mechanism for virtual travel: step in the direction you wish to go. Beyond those four buttons, everything is flexible. Since the DDR pad has buttons in its corners, they can be used for diagonal movement. Also, it has two extra buttons,

"start" and "select", which can be used for additional functionality, such as moving up and down.

There are a variety of ways to implement motion with on-off directional arrows. A naïve way would be to assign a velocity vector to each arrow and, each frame, add the pressed ones to the user's current position. However, this leads to very choppy motion. A better way would account for gradual acceleration and deceleration.

For this project, I derived a velocity system that has acceleration, deceleration, and uses all eight directional pads for more refined motion. The application should store a current position and current velocity, each of which are updated for each frame. Some definitions:

P_t = position vector at current frame
V_t = velocity vector at current frame
dv = normalized vector representing intended movement direction
dt = seconds between frame t and $t+1$
VD = velocity dampening scalar (time for velocity to halve)
VA = velocity affect scalar (how much arrows affect velocity)

And the equations that show how position and velocity are updated:

$$P_{t+1} = P_t + V_t \cdot dt$$
$$V_{t+1} = V_t \cdot VD^{dt} + dv \cdot VA \cdot dt$$

The next frame's position is calculated in the standard Newtonian model – simply add the velocity multiplied by the elapsed time in seconds to the position. Velocity is composed of two factors: the last frame's velocity with an exponential decay plus the intended direction multiplied by a constant factor and the time since the last frame.

This model should provide smooth acceleration, deceleration, and graceful direction changes, at least if dt is reasonably close to zero. But it still lacks something... When a user's weight is put on one half of the pad, it should lean in that direction. So virtual lean must be accounted for in the navigation model. Lean is calculated as an axis-angle rotation of the user (or, inversely, the scene). Lean should be more pronounced the faster the user travels. Also, the carpet should tilt in the direction it is traveling. The following equations calculate the lean for the current frame:

U = up vector
$H_t = V_t$ projected onto horizontal plane
LF = lean factor in $m/(rad \cdot s)$
A = lean axis = $H_t \times U$
C = lean amount = $|H_t| \cdot LF$

The lean angle is directly proportional to velocity, and the lean axis is the cross product of the current horizontal velocity and the up vector. With lean, acceleration, and deceleration taken into account, the movement should feel much smoother and more interesting than the naïve approach.

Visuals

For this project, the visuals consist of a static model of the world and a model of the magic carpet itself.

To fit the theme of a magic carpet ride, the static model is of an area of desert. The desert has dunes, raised hills, and small trenches. The entirety of the model isn't intended to be totally visible at all times. In fact, some regions should be hidden as much as possible. This makes the scene more interesting and encourages exploration.

The carpet is drawn in the center of the floor, exactly where the DDR pad lies. The rationale here is that it will inform the user where the real pad should lie, in case it gets slightly off. The pad also emits particles, which trail behind it and give the user another type of motion cue.

Game Play

A simple game has been designed around this structure. Based on PilotWings, the game involves navigating through different targets in the world in the smallest amount of time. Each world is also assigned a starting location. The player begins the game at the start location and pressing the start button on the pad. The timer starts, and the user must navigate through all of the targets. Targets are represented as circles that change their orientation to always face the player. When the user passes through a target, it changes from green to red. When the last target is reached, the timer stops. Pressing select at any time restarts the game from the beginning.

Implementation

Engine

The magic carpet application itself is built upon the VRJuggler framework for virtual reality applications [5]. VRJuggler provides stereo, multi-pipe rendering on VR systems that support it, as well as access to a myriad of input devices. It also allows applications to be developed on common workstations. At least 90% of the development occurred on a standard Windows workstation, so this was a necessity.

All graphics capabilities are managed with the OpenSG scene graph [6]. OpenSG provides model loading, efficient graphics rendering and culling, and saves a lot of time for graphics programmers. In fact, very little code within Magic Carpet involves graphics. A simple scene graph is constructed from files loaded from disk and some game setup logic. OpenSG handles the rest.

Not only does OpenSG provide graphics capabilities, it does intersection testing for collision detection as well. Magic Carpet can take advantage of this to implement a hover mode where the carpet tries to stay at a constant distance above the ground. Regrettably, this functionality is disabled due to performance issues.

In order to make the DDR pad accessible to the Magic Carpet application, it is plugged in, via an EMS USB II adapter, to the Intermec portable computer, which is running Windows 98. Windows 98 supports reading USB Human Interface Devices

as joysticks. The VRPN software comes with a server that reads data from the joystick. However, it does not send the data anywhere unless specifically asked for over the network by a client. Fortunately, VRJuggler knows how to communicate with VRPN right out of the box. VRJuggler then acts as a client, connecting to the server which is specified in a configuration file and reading input data from the device. The Intermec has wireless networking capabilities, so nearly any networked VR system can access it without needing a wired connection. This also means it can be used from within the C6, even when the C6 is totally closed in.

Unfortunately, VRPN has some performance issues when used in conjunction with with VRJuggler. Every 500 milliseconds or so, one frame would take extra long, giving the illusion of the application running very choppily. I tried several different techniques to overcome this problem, or even see what was causing it, but could not find a solution. Eventually, I ended up not using VRPN at all and writing a simplified version of it called joyserver. joyserver consists of a simple Windows program to read joystick data and periodically transmit it to connected clients, as well as a small amount of code that lies within the VRJuggler application to receive this data. Ideally, the VRPN problem should be addressed and overcome, but that was not feasible in the interest of time.

Modeling

This is the first real project that has required me to do any sort of significant modeling. All modeling was done with discreet's 3ds max 5.1.

For this project, there were two main objects that needed to be modeled: a magic carpet and the virtual environment. The magic carpet is trivial. It involved applying a texture to a single small plane object.

The virtual environment was much more difficult. Since a desert is mostly sand, modeling a terrain was the important part. There are several ways to model a terrain in 3ds max, and I tried three of them. One way is to build the terrain with NURBS surfaces and manipulate them into the desired shape. The advantage to this approach is that it is easy to make an initially smooth terrain. Unfortunately, it is hard to add detail: the more control points a NURBS surface has, the more computational power it takes to render and manipulate it. It doesn't take long for it to become infeasible to work on.

Another technique to model terrains is conceptually simpler than NURBS, but doesn't look quite as good. Instead of using a NURBS surface, a highly tessellated plane object can be used as a base. Then, individual vertices can be shifted up and down to form the terrain. So that vertices don't have to be edited individually, 3ds max provides an option called soft selection where groups of vertices can be edited all at one time. The word soft in soft selection refers to the fact that any modification to vertices outside of your selected set is only partially modified. So if a single vertex is selected and raised above the base ground level, the vertices around it are raised. Soft selection allows for bigger, gentler changes to the terrain.

Finally, one of the most useful techniques for modeling terrains in 3ds max is displacement mapping. Displacement mapping lets you open a bitmap file and assign

it as a height map for a specific piece of geometry. The total luminance value of each pixel within the image represents the offset of the geometry at that point. As above, a plane is created for the base terrain structure. When the displacement modifier is applied with a suitable height map, the plane shifts to represent a terrain. Bright areas in the image become raised hills and plateaus, and dark areas become trenches and sunken holes.

For magic carpet, I experimented with all three of these techniques, and used each in the final result.

The last component of each model is a set of objects representing locations in the game. Custom 3ds max objects called "dummies" must be inserted into the model and given specific names. The object called "start" represents the starting location for that model. Objects whose names are "target" are targets for the player to reach.

Limitations

As mentioned above, the use of VRPN created strange performance characteristics within the application. This was unacceptable for the illusion of real-time performance in a virtual environment, so a limited temporary replacement for VRPN was written. Eventually, the VRPN performance issue should be addressed. At that time, the custom replacement will no longer be necessary.

Also, constant-height hovering over the terrain and collision detection are both disabled in this implementation due to performance issues in OpenSG. OpenSG provides a way to do ray intersection within the scene. To do constant-height hovering, this intersection test must be done once per frame. However, it's documented (and holds true in practice) that intersection tests cannot currently be done in real time and maintain performance.

The way navigation is currently implemented, it requires a fully immersive virtual environment, such as the C6. When using a facility such as a PowerWall [7], the C4, or a desktop system, the user can only look in one direction, north. Magic Carpet does not yet provide a navigation system for systems such as these.

Conclusion

When enabling technologies such as wireless networks, portable computers, inexpensive input devices, VRJuggler, and VRPN are combined, the number of possible interaction methods goes way up. Using a Dance Dance Revolution pressure-sensitive pad for navigation seems to be an intuitive way to navigate within a virtual space, at least from initial tests.

I have learned a great deal about OpenSG and VRML, especially when it comes to getting them to work together. My experience in modeling terrain, and using 3ds max in general, has improved greatly. I have overcome many problems I had with applying texture mapping within max and having it show up in OpenSG, and thus, my application. My competency with NURBS surfaces has also improved.

Future Work

Immediate future work lies within improving the performance of OpenSG and VRPN so that collision detection, hover mode, and VRPN support can be enabled. Replacing my custom joyserver solution with VRPN would enable navigation with any button-axis device understood by VRPN.

The magic carpet game would be enhanced by the addition of spatialized audio, improved graphics, and new levels. Audio could provide a greater sense of immersion, as well as improving the player's sense of depth. The closer the carpet gets to a target, the louder the target's noise would be. A dinging sound could be played when a target is reached, so that the user would not necessarily have to see it change colors to know that they successfully got to it.

Finally, the concept of "magic carpet navigation" using a pressure-sensitive pad can be applied to other applications. In the Visualization of the Joint Battlespace [8] application, traveling around the battlefield using a "magic carpet" might be more intuitive, especially if a traditional navigation device such as a wand or a controller is used at the same time.

References

[1] EMS USB II at LevelSix: http://www.levelsix.com/detail.aspx?ID=2

[2] Iowa State University's C6: http://www.vrac.iastate.edu/about/labs/c6/index.php

[3] Iowa State University's C4: http://www.vrac.iastate.edu/about/labs/c4/c4page.php

[4] University of North Carolina's VRPN: http://www.cs.unc.edu/Research/vrpn/

[5] VRJuggler: http://www.vrjuggler.org/

[6] OpenSG: http://www.opensg.org/

[7] PowerWall: http://www.lcse.umn.edu/research/powerwall/powerwall.html

[8] Joint Battlespace Visualization http://www.vrac.iastate.edu/~sannier/battlespace.htm