Strider: A Simple and Effective Terrain Navigation Controller

Jarno van der Linden, Jing Li, Richard Lobb, Kevin Novins, and Burkhard Wünsche

Graphics Group
Department of Computer Science, University of Auckland.
jvan006@cs.auckland.ac.nz

Abstract

We present a simple controller for terrain navigation called Strider. The controller is based on the concept of walking and is implemented using a standard mouse device as used with virtually any desktop computer. Terrain can be explored at any detail by changing the height of the Strider. By applying a scale factor to the rate of change in position and Strider height based on the current height above the terrain, large terrains can be navigated quickly while still allowing fine control when examining small details. In contrast to other user interfaces for terrain navigation, such as flight simulators, no keyboard input is required.

Keywords: terrain navigation, controllers, walking

1 Introduction

Terrain visualisation is one of the mainstays of computer graphics. The challenge of showing large scale detailed terrain in real-time has given rise to numerous algorithms and rendering systems.

While investigating various terrain rendering systems, it became clear to us that while some produce impressive visuals, there is often little thought put into how the user is to navigate effectively over the terrain. Common methods range from the use of a keyboard with each basic movement mapped to a key, to the simultaneous use of both keyboard and mouse to control a realistic simulation of a complex fighter aircraft.

Frustrated with mechanisms that provided insufficient control, too much control, were too complex, required specialised hardware, or had no relation with the task of terrain exploration, we developed the Strider terrain navigation controller. Unlike other controllers, Strider can be used to explore a terrain at any scale ranging from local to global. Strider uses only a standard mouse and is operated single-handedly.

2 Related work

The controller is often an afterthought in terrain visualisation systems when the emphasis is on the rendering. The topic of navigating in a virtual environment is being studied extensively in the computer-human interaction field. See for example [1].

Ware and Fleet [2] have investigated the use of distance to objects to scale the sensitivity of the controller. This is similar in concept to our work where we apply a scaling to changes in position and height depending on distance to the terrain. However, we wish to avoid the use of the flying control metaphor. Our application is also targeted at terrain exploration, rather than more general environments.

Walking is a common way of navigating through human-scale environments. Studies such as [3] show that walking is a better method of exploring human-scale virtual environments than flying.

To achieve a more immersive and intuitive experience, custom hardware has been built to support the use of a walking interface, for example [4]. Such equipment is expensive and not commonly available.

A mouse device is specialised for 2D navigation, and as such does not have the degree of freedom required for general 3D navigation. Devices such as the SpaceBall [6] provide the necessary degrees of freedom, but take more time to master than a standard mouse. Furthermore, we wanted to use the devices present on a typical computer system rather than depend on specialised devices.

3 The Strider Controller

As part of a terrain visualisation project, we required a means to easily navigate the virtual terrain. After investigating the control methods used

in some other terrain rendering demonstrations, we found them to be insufficient for one or more of the following reasons:

- Requiring the use of a keyboard, sometimes in combination with a mouse.
- The need to memorise numerous non-intuitive key assignments and key combinations.
- Non- and counter-intuitive rotation controls, for example with gimbal locking problems, or controls which invert when going backwards.
- Overly simple controls, for example allowing only one or two speeds.
- Overly complex controls, for example an accurate simulation of an advanced fighter aircraft or helicopter.
- Inability to use the controls for both small scale and large scale examination, for example requiring minutes to cross the entire terrain, or always moving too fast to concentrate on small details.
- Requiring the use of non-standard hardware, for example a 6 degree of freedom mouse.

Having come to the conclusion that existing methods were unsuitable for our terrain navigation applications, we developed the Strider controller.

3.1 The model

In order for the controller to be intuitive it should be based on a model that the user is naturally familiar with. A common choice is to model the operation of a vehicle. In these models the user controls the movement of some airborne or ground-based vehicle, such as a helicopter or car. Keyboard and mouse controls are mapped to simulations of real-world controls, such as a steering wheel, gas and break pedals, or rudder.

There are several problems with vehicular models that makes them unsuitable for our task. In the real world, driving a car or flying an aircraft typically requires the use of both hands and feet. In contrast, interacting with a computer is commonly done through keyboard and mouse, both hands-only devices. Familiar vehicles also have a range of motion that is either too limited or too broad for easy use. For example, a car can not turn around its own axis, and is restricted to movement on the surface of the terrain. A helicopter on the other has has a large number of degrees of freedom, but requires the simultaneous use of a number of controls for stable flight, and many hours of practice to master.

For Strider we chose one of the most basic and familiar forms of locomotion: walking. Not only is walking a model that is instantly familiar to everyone, it is also one of the most common ways of real-world exploration of terrain.

There are two disadvantages with a simple walking model however. First, walking is slow compared with many other modes of transport, especially when considering that the typical size of the terrain of interest is tens to hundreds of kilometres square. The second disadvantage of walking is that the height above the terrain is fixed, or limited to a few metres variation at best by crouching and jumping. Overviews of large areas of the terrain can only be obtained by walking to the top of a high mountain.

We alleviate both these disadvantages with a single elegant solution: the Strider can grow and shrink in size at will. A larger Strider sees a larger area of the terrain by being taller. Furthermore, a taller Strider also takes larger strides thereby traversing the terrain faster (figure 1).

3.2 Basic movements

The Strider controller has the following movements:

- Move forward and backward
- Turn left and right
- Grow and shrink
- Look up and down

The movements are designed to minimise any possible confusion about what direction the Strider is moving in. For this reason there is no look left and look right movement. Looking left, for example, would be visually indistinguishable from turning left. The difference only shows up when the user attempts to move forward. The apparent direction of motion would differ depending on what direction the user is looking at. Unless there is a visual or other cue to indicate whether or not the user is facing in the direction of forward movement, it is difficult to anticipate where the camera is going to move to. We therefore defined Strider to always move forward relative to the camera viewing direction. Note that there is no such ambiguity in relation with looking up and down. Walking always occurs parallel to the terrain, no matter how far up or down the user is looking.

The four basic movements are mapped to a commonly available mouse pointer device (figure 2). Moving forward and backward is linked with moving the mouse forward and backward. Turning is

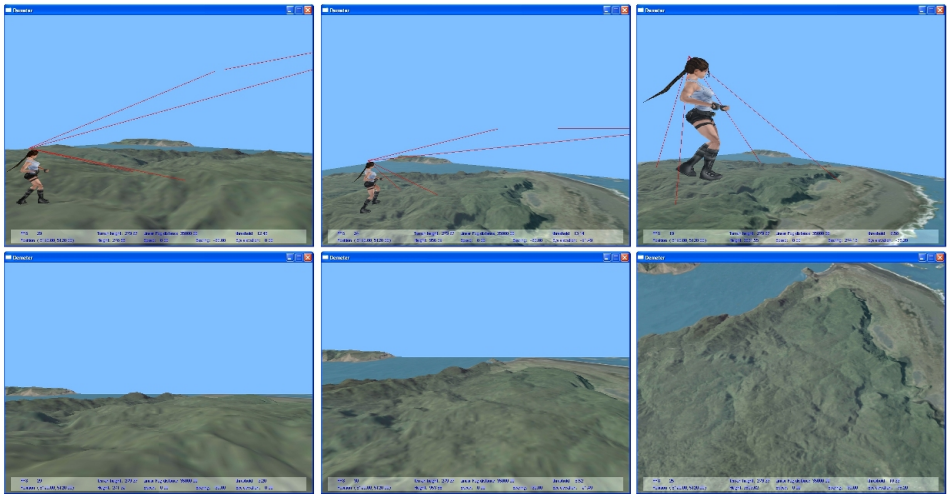

Figure 1: The Strider concept: as you grow larger, you see the terrain at a larger scale and move faster.

Figure 2: Strider movements mapped to a mouse.

done by moving the mouse left and right. Growing and shrinking is achieved by pressing the left and right mouse buttons. Looking up and down is controlled through the use of the scroll wheel.

Moving the mouse forward and backward acts like an accelerator. The more forward the mouse is moved, the faster the Strider goes. As it is common for the user to want to stand still and look around, there is a "dead-zone" for movement when the Strider is standing still. The mouse needs to be moved forwards or backwards by some minimum amount in order for the Strider to once again start moving. Whenever the Strider movement changes from forward to backward or vice versa the speed is set to zero, thereby activating the dead-zone. This size of the dead-zone is one of a number of parameters that can be set which are explained later.

Looking up and down is limited to a range of $\pm 90°$. This avoids any confusion that may be caused by walking while looking upside down and backwards, and approximates the natural range of head movement.

4 Height and speed

Two of the issues that have occupied us the most in the development of Strider are the rate of change in Strider height as a function of height, and the relation between height and Strider speed.

First we have to define what height means. There are two possible interpretations of height when used in terrain visualisation:

1. The vertical distance between the camera and the terrain, with positive height meaning above the terrain.

2. The vertical distance of the camera above some fixed datum, such as sea level.

We refer to the height above terrain as the Strider's eye height, while the second defines the datum height or altitude.

As the Strider moves, do we keep the eye height fixed or the datum height? There are advantages and disadvantages to both methods. Fixing eye height matches the intuitive walking model, but subjects the viewer to up and down movement for every bump in the terrain. This becomes particularly acute when moving fast where small scale variations in terrain height cause high frequency bouncing of the Strider. The alternative of fixing datum height avoids this bouncing, but does not correspond with the natural way of walking.

Various intermediate representations and fixes can be used to lessen the disadvantages of either height method. For example, some speed- and height-sensitive smoothing could be used with a fixed eye height to reduce the high frequency bouncing. Or a fixed eye height could be used when the camera is near the ground, and a fixed datum height when the camera is far above the terrain, with an interpolation used in between the two regimes. As yet we have not explored all these possibilities. In our currently implemented Strider controller, we fix the datum height.

4.1 Changing height

The height of the Strider is changed by using the left and right mouse buttons. Holding down one button causes the Strider to grow, while the other makes the Strider shrink. Having a constant rate of change is simple to implement, but proved frustrating to use in practice while exploring terrain. If the rate of change is set to be small, fine corrections while low to the ground are easy, but growing to a great height takes too long. Similarly, a large rate of change means that the Strider can go through the entire range of heights at which the user wants to explore quickly, but small changes in height are virtually impossible. Even the briefest of button clicks can cause the height of the Strider to jump by an overly large amount.

When the user wishes to explore small scale details of the terrain the height of the Strider is typically set to be close to the terrain. It is here that the rate of change in height should be small. In contrast, when high up looking at an overview of the terrain, changes in height should be large. The higher one is above the terrain, the greater the height of the Strider has to change in order for the change to become noticeable.

It is therefore clear that the rate of change in the height of the Strider should be a function of the current height, with greater height leading to greater change. A simple way to achieve this is by setting the change in height Δh per time step to be linear with respect to the eye height h_{eye} above the terrain:

$$\Delta h = a \; h_{\text{eye}} + b$$

Rather than a linear relationship, after some experimentation we chose a polynomial:

$$\Delta h = a \;\; (h_{\text{eye}} + 1)^{\frac{1}{2}}$$

This results in a natural feel, with a perceptually fairly constant rate of change. This phenomenon seems to be related to other visual perception tasks, such as area and volume perception, which have been shown to be non-linear (see for example Steven's Law [5], which states that the perceived scale of many attributes is a power of their actual scale).

4.2 Changing position

As with the change in height, the amount of change in position per time step should be dependent on the height. With "position" we mean the location of the Strider projected onto the horizontal plane. It is independent of height. The speed of the Strider is the change in this position. Speed does not include any changes in height, only the change in projected position.

The perception of speed is related to the size of detail seen, and hence the eye height of the Strider. A user exploring a terrain from a great height would want to move faster (in absolute terms) than looking at the terrain from near ground level. The rate of change in horizontal position is the Strider's speed s. Reasoning that it should be similar in scale to the rate of change in height, we use the same relation as with height:

$$s = s' \;\; (h_{\text{eye}} + 1)^{\frac{1}{2}}$$

where s' is the perceptual speed set by the user. The perceptual speed is controlled by moving the mouse forwards and backwards. It remains constant with eye height, unlike the actual speed s.

4.3 Terrain avoidance

The Strider should not go below the terrain. For rendering purposes it is desirable that the camera remains at least some small distance above the terrain. This distance is the minimum eye height of the Strider. When the Strider hits the side of a mountain for example, it needs to be moved up by changing the datum height.

We distinguish between a requested datum height and the actual datum height. The requested datum height is set by the user using the Strider controls. The actual datum height is set based on the requested height and the terrain height.

When the user changes the requested datum height, the new requested height is set to be at least the height of the terrain at the current Strider position plus the minimum eye height. If this were not done, there would be a confusing delay when increasing the requested datum height until it catches up with the actual Strider height.

5 Implementation

The implementation of the Strider controller consists of four routines:

- `MouseMove`: process a change in mouse position into a new perceptual speed and heading.
- `MouseButton`: process a change in mouse button state into a change in requested height.
- `MouseWheel`: process mouse wheel scrolling into a new look up/down direction.
- `MoveStrider`: to move the Strider with a given timestep using the current speed, heading, height, and look direction.

Pseudo-code for the Strider controller is given in the appendix. In total the controller requires about 30 to 40 lines of code to implement. We have successfully implemented Strider in C++ and Java. The important state variables for the controller are given in table 1. There are a number of parameters that control the sensitivity of the controller and ranges of the state variables. These may be tuned to the application and user.

6 Results

The Strider controller has been implemented in several terrain exploration applications, both existing and new. We used Strider to replace the F16 fighter airplane controller in the JCanyon terrain visualisation demonstration Java program [7], and implemented Strider in a custom terrain renderer used to investigate aspects of terrain rendering.

Although entertaining, using an F16 to explore a terrain proved limiting. The user was required to spend a significant amount of time concentrating on flying the aircraft rather than effectively exploring the terrain. Empirical observation showed that using the Strider controller allowed to user to easily navigate to any point on the terrain and explore at all scales from small detail to large overviews.

Strider also proved more friendly for onlookers, who weren't subjected to frequent loss of view of the terrain as the user tries to loop an aircraft back towards an area of interest, or had to be bored passengers on a long road trip.

Visitors to the research group who were unfamiliar with the Strider controller were asked to try it out with little or no explanation of its operation. In general the Strider controller was mastered within a period of seconds to a minute.

One feature that some users asked for is the ability to move in the viewing direction while looking up or down. This is more in line with an aircraft or spacecraft model. At the same time, the users liked the ability to move over the terrain with the movement direction uncoupled from the look up/down direction. We haven't yet resolved these conflicting requirements while maintaining simplicity.

The parameters of the Strider controller can be tuned to suit the user, but in practice we have found it unnecessary to change them from their initial settings.

7 Conclusion

Strider has proven for us to be an easy and effective controller for exploring virtual terrain. Informal experiments have shown that people unfamiliar with the controller can use it effectively within a very short time. The use of an intuitive walking metaphor means that the user can pay more attention to exploring the terrain rather than concentrating on using the controller. Fast terrain navigation at any scale is achieved by scaling the rate of change in position and Strider height as a function of the eye height above the terrain. The Strider controller has no special hardware or software requirements, being based on a standard mouse device, which has allowed us to implement it in several existing and new applications.

There are still many areas that remain to be investigated. A formal useability study is yet to be done. The perceptual effect of various speed scaling functions needs to be looked at more closely. There is a significant body of research on the effects of perspective and scale on perceived motion. It is our hope to be able to derive a scaling function based on these models of perception.

References

[1] D. S. Tan, G. G. Robertson, and M. Czerwinski: Exploring 3D navigation: combining speed-coupled flying with orbiting. *Proceedings of the SIGCHI conference on Human factors in computing systems*. (2001) 418–425. ACM SIGCHI.

[2] C. Ware, and D. Fleet: Context sensitive flying interface. *Proceedings of the 1997 symposium on Interactive 3D graphics*. (1997) 127–ff. ACM SIGGRAPH.

[3] M. Usoh et al.: Walking > walking-in-place > flying, in virtual environments. *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. (1999) 359–364. ACM SIGGRAPH.

[4] L. Bouguila, M. Ishii, and M. Sato: Realizing a new step-in-place locomotion interface for virtual environment with large display system. *Proceedings of the workshop on*

State	Meaning and range	Controlled by
position	Horizontal position on terrain $[(-\infty, -\infty)$ .. $(\infty, \infty)]$	
reqspeed	Perceptual rate of change in position [−maxbackwardreqspeed .. maxforwardreqspeed]	Mouse forward/backward
speed	Actual rate of change in position [−maxbackwardspeed .. maxforwardspeed]	
changeheight	Requested perceptual change in height {−heightsensitivity, heightsensitivity}	Mouse buttons
reqheight	Requested height of Strider from datum [terrain height at position + eye height .. maxheight]	
height	Actual height of Strider from datum [reqheight .. maxheight]	
heading	Heading direction of travel $[0..\ 2\pi)$	Mouse left/right
pitch	Look up/down direction $[-\pi/2\ ..\ \pi/2]$	Mouse wheel

Table 1: Important state variables for the Strider controller.

Virtual environments 2002. (2002) 197–207. Eurographics.

[5] W. S. Cleveland: *The elements of graphing data*. Murray Hill, N.J. : AT&T Bell Laboratories, 1985.

[6] The SpaceBall motion controller: `http://www.3dconnexion.com/-spaceball5000.htm`.

[7] JCanyon: Grand Canyon for Java: `http://java.sun.com/j2se/1.4/-demos/jcanyon/`.

Appendix: Strider pseudo-code

```
MouseMove(delta.{x,y})
{
  // Update absolute pointer position
  mouse.{x,y} := mouse.{x,y} + delta.{x,y}

  // Change speed if current speed is not zero, or if
  // zero, when the mouse has moved out of the deadzone
  if((reqspeed != 0) || (|mouse.y - stop.y| >= deadzone))
  {
    // Compute new speed
    newreqspeed := reqspeed - delta.y * speedsensitivity
    // If sign has changed, force Strider to stop
    if(((reqspeed > 0) && (newreqspeed <= 0)) ||
       ((reqspeed < 0) && (newreqspeed >= 0)))
    {
      newreqspeed := 0
      stop := mouse
    }

    // Limit speed
    reqspeed := Limit(newreqspeed,
                      -maxbackwardreqspeed,
                      maxforwardreqspeed)
  }

  // Change heading, modulo 2 PI
  heading := header + delta.x * headingsensitivity
  heading := Wrap(heading, 0, 2*PI)
}


MouseWheel(delta_wheel)
{
  // Change pitch with wheel movement
  pitch := pitch + delta_wheel * pitchsensitivity
  // Limit pitch between looking straight up and down
  pitch := Limit(pitch, -PI/2, PI/2)
}
```

```
MouseButton(button, state)
{
  // Left button adds to height
  if((button == LEFT) && (state == DOWN))
    changeheight := heightsensitivity

  // Right button subtracts from height
  else if((button == RIGHT) && (state == DOWN))
    changeheight := -heightsensitivity

  // Stop changing height when button released
  else
    changeheight := 0
}


MoveStrider(delta_time)
{
  // Compute desired movement direction
  aim.x := sin(heading)
  aim.z := cos(heading)

  // Compute movement scale based on current height
  // above the terrain
  terrain_height := GetTerrainHeight(position.{x,z})
  scale := sqrt(max((height - terrain_height + 1) / 2, 1))

  // Scale the movement
  speed := Limit(reqspeed * scale,
                 -maxbackwardspeed, maxforwardspeed)
  scaled_changeheight := changeheight * scale

  // Change strider position by adding scaled movement
  position.{x,z} := position.{x,z} +
                    aim.{x,z} * speed * delta_time
  reqheight := reqheight + scaled_changeheight * delta_time
  reqheight := min(reqheight, maxheight)

  // Terrain avoidance
  terrain_height := GetTerrainHeight(position.{x,z})
  min_height := terrain_height + eye_height
  if(reqheight < min_height)
  {
    // Always stay above the terrain
    height := min_height

    // When changing height, make sure requested
    // height is at least the actual height
    if(changeheight != 0)
      reqheight := height
  }
  else
  {
    height := reqheight
  }
}
```