Hypershot: Fun with Hyperbolic Geometry

Praneet Sahgal

December 19, 2012

Introduction

Hypershot is a first person shooter in hyperbolic space. The program is designed to run in the Cube at the Illinois Simulator lab through the Syzygy application framework. The application allows a user to fire projectiles out of the Cube's wand and watch them move according to the rules of hyperbolic space.

Using the application

When the application starts, a test shot of three projectiles is fired from right to left. This shot can be reset by pressing the 0 button on the wand. Additional shots can be fired from the wand by aiming and pressing the 1 button. Navigation using the wand and joystick work as per Syzygy default behavior. There is no software-implemented limit on the number of shots that can be fired; the only limitation is the amount of memory on the computer. Pressing 0 will remove any fired projectiles and reset to the test shot.

1 The Source Code

The source code for the application follows object-oriented design principles; it is made up of several classes and a main function. The classes/files are as follows (class names are in bold).

The **Framework** class defines the main Syzygy framework within the application. This handles the Syzygy event loop; pre-exchange, post-exchange,

drawing, button handling, etc. This class is what sets up projectiles, updates them, and fires them on command.

The **Projectile** class describes a projectile object that can be fired out of the wand. In the current build of the application, this is in the form of green cubes. This class also handles updating the velocity and position of a given projectile. Most importantly, this class also applies hyperbolic trajectories.

The **HMath** files define a simple vector and point structure that is used by the other classes of the application to store velocity and position. These files also define a function to rotate a given vector by a rotation matrix, a useful function for firing projectiles.

The **Environment** and **Target** classes describe the surrounding environment and a red target on the map, respectively. Currently this consists of a floor and a red wall in the middle of the 3D space. This is intended to be later expanded upon.

The **RodEffector** class defines a rod that follows the position of the wand. This is intended to be later replaced with a weapon. This object is used to determine where newly fired projectiles spawn from. New projectiles spawn from the tip of the wand.

Lastly, **Hypershot** contains the main function loop for the application. This is nearly identical to the main function found in the oopskel.cpp template program.

Hyperbolic Trajectories

The most important function in the program is the application of hyperbolic trajectories to objects. This is done in the applyTrajectory function in the Projectile class. This function takes the projectile and uses matrix multiplication to generate a new velocity that is based on the characteristics of hyperbolic space. This results in a model of hyperbolic translation.

There are three matrices used by the application. For x translation, the following matrix is used:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cosh(x) & 0 & sinh(x) \\ 0 & 0 & 1 & 0 \\ 0 & sinh(x) & 0 & cosh(x) \end{bmatrix}$$

For y translation, this matrix is used:

$$\begin{bmatrix} cosh(y) & 0 & 0 & sinh(y) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ cosh(y) & 0 & 0 & sinh(y) \end{bmatrix}$$

And for z translation:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & cosh(z) & sinh(z) \\ 0 & 0 & sinh(z) & cosh(z) \end{bmatrix}$$

Each of these matrices, when applied to a matrix describing the velocity of an object, result in a new matrix containing the new velocity. For example, translating an object a velocity v in the z direction would be done like so:

$$\begin{bmatrix} 1 & 0 & 0 & z \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & cosh(v) & sinh(v) \\ 0 & 0 & sinh(v) & cosh(v) \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 0 & xsinh(v) & xcosh(v) \\ 0 & 1 & ysinh(v) & ycosh(v) \\ 0 & 0 & e^{-v}(\frac{1}{2} - \frac{z}{2}) + e^{v}(\frac{1}{2} + \frac{z}{2}) & e^{-v}(\frac{z}{2} - \frac{1}{2}) + e^{v}(\frac{z}{2} + \frac{1}{2}) \\ 0 & 0 & sinh(v) & cosh(v) \end{bmatrix}$$

For the purposes of the program, we are only interested in the new velocity that results from this operation. We extract the velocity from the 4×4 matrix to get a 1×3 velocity column vector:

$$\begin{bmatrix} xcosh(v) \\ ycosh(v) \\ e^{-v}(\frac{z}{2} - \frac{1}{2}) + e^{v}(\frac{z}{2} + \frac{1}{2}) \end{bmatrix}$$

This vector is then used as the new velocity of the object.

To translate in x, y and z directions, we must apply each individual translation matrix. We first travel a small distance $dx(dt)$ in the x direction, then a distance $dy(dt)$ in the y direction proportional to the x direction, and finally $dz(dt)$ in the z direction proportional to the x direction, where dt is a small time step. In the program, dt is arbitrarily chosen to be 0.01. So, assuming that dx, dy, and dz are nonzero, to get from a matrix with Euclidean velocity A to a matrix with hyperbolic velocity B, we do the following sequence of operations:

$$[A]\,[X]\,(dx)(dt) = [A_1]$$

$$[A_1]\,[Y]\,(dy)(dt) = [A_2]$$

$$[A_2]\,[Z]\,(dz)(dt) = [B]$$

where X, Y, and Z are the translation matrices described above. While these operations may not be completely accurate, this method generates a good model of hyperbolic trajectories.

Firing out of the Wand

Another operation of importance is described by the functions used to spawn the projectiles from the wand. A projectile object in Hypershot is primarily defined by a center point and a velocity vector. Getting the starting velocity vector from the wand also requires some matrix multiplication. There is a function within Syzygy that returns a matrix representation of the tip of the rod effector attached to the wand. This matrix is of the form:

$$\begin{bmatrix} & & & x \\ & R & & y \\ & & & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where R is a rotation matrix and x, y, and z represent the position of the object. The starting position of a fired projectile can be easily defined as (x, y, z). However, the starting velocity vector is harder to define. To find the starting velocity vector, we first extract the rotation matrix R and apply a rotation transformation to a given vector (i, j, k):

$$\begin{bmatrix} i \\ j \\ k \end{bmatrix} \begin{bmatrix} & & \\ & R & \\ & & \end{bmatrix} = \begin{bmatrix} i' \\ j' \\ k' \end{bmatrix}$$

where (i', j', k') is the rotated vector. This vector is then used as the starting velocity of the object, and the rotation matrix is stored within the Projectile object as the starting orientation of the projectile. The operation to rotate the vector is spread between the Projectile class and the Vector3 structure

defined in the HMath files. The input vector does not need to be a unit vector; in fact, a non-unit vector will result in a higher initial speed, which is desirable for the project. The arbitrary input vector used for Hypershot is $(0, 1, 5)$.

Projectile Behavior

The behavior of the projectiles in Hypershot illustrates an interesting characteristic of hyperbolic space: space in the middle of the Poincaré sphere is larger than space away from the middle. This is seen by the appearance of acceleration of the objects as they leave the center. In hyperbolic space, the velocity of the projectiles are constant; the space itself is shrinking. The paths of the objects appear to be the same as they would in Euclidean space. However, an observer outside of hyperbolic space would see some very strange paths.

Expanding the Application

There is a lengthy list of features that were initially proposed for this project but, due to time constraints, were never implemented. Here is a list of proposed extra features, in order of priority:

1. Path tracing for projectiles as they move.
2. The ability to switch to an observer in Euclidean space outside of the Poincaré sphere. This would allow a person to view hyperbolic trajectories as seen in Euclidean space.
3. Multiple weapons with different firing patterns. Specifically, a double-barreled shotgun in hyperbolic space would result in an interesting trajectory in Euclidean space.
4. Proper collision detection. As of this writing, collision detection in Hypershot is not working properly, and detracts from the application.
5. Better 3D models for the weapon, target, projectile, and environment. While not mathematically significant, it makes for a better demonstration.

Conclusion

For me, this was an interesting introduction to the world of non-Euclidean geometry. There was a lot of abstract theory involved, and the result is some very strange behavior. Hypershot is a start of what could be a very interesting game, and a great demonstration of the oddities of hyperbolic space. It is a way to convey what hyperbolic space would feel like, which is something hard to convey to people used to living in a Euclidean world. In this way the project succeeded in its goal: provide a sandbox for players in hyperbolic space.