William Baker
Spring 2005
Math 198

Splinester: A real-time interactive application for building and riding CR splines

Operating Instructions:

There are three stages of the splinester program, and each has its own controls an operating instructions. I have made sure to use a three button configuration, so that splinester runs correctly with a variety of wand devices.

To start splinester:

In the cave type: "dex cave splinester.py"

This starts the program, and the user will begin in what I call the "Creation Stage."

Creation stage instructions:

The first stage (the creation stage) is used to create the spline that you will ride as a roller coaster. You create the spline by placing control points, and you need to have at least 3 control points drawn to make a spline you can ride. Splinester automatically closes your spline into a loop when you choose to enter preview mode.

Controls:

Wand:	Move the wand to move your cursor. Control points are placed at the location of the cursor.
Left Button:	Place a control point.
Middle Button:	Adjust the tension parameter of the spline (cycles in 0.1 increments, with a minimum of 0.3)
Right Button:	Enter preview mode and close the spline to form a loop.

Preview stage instructions:

The preview stage give a very rough preview of how your ride will be. Once you have entered the preview mode it is impossible to add or remove control points. The only modification that can be made is changing the tension parameter.

Controls:

Left Button:	Ride the coaster (first person view).
Middle Button:	Adjust the tension parameter.
Right Button:	Ride the coaster (same as pressing left button).

Riding the coaster instructions:

Now you get to enjoy all your hard work in designing a coaster. This mode keeps you riding on your track until you choose to exit.

Controls:

Left Button:	Reset your ride to the beginning of the track, with your original orientation.
Middle Button:	Not used.
Right Button:	End your ride, delete the track, and return to the creation stage to make a new track.

Narrative and Further Details:

Roller coasters for the CAVE and CUBE are not a new idea, and several different types of roller coaster programs are already running in the CAVE and CUBE today. However, my program is the first coaster program for the CAVE/CUBE that I have seen that lets users interactively design their own roller coaster, and then have the opportunity

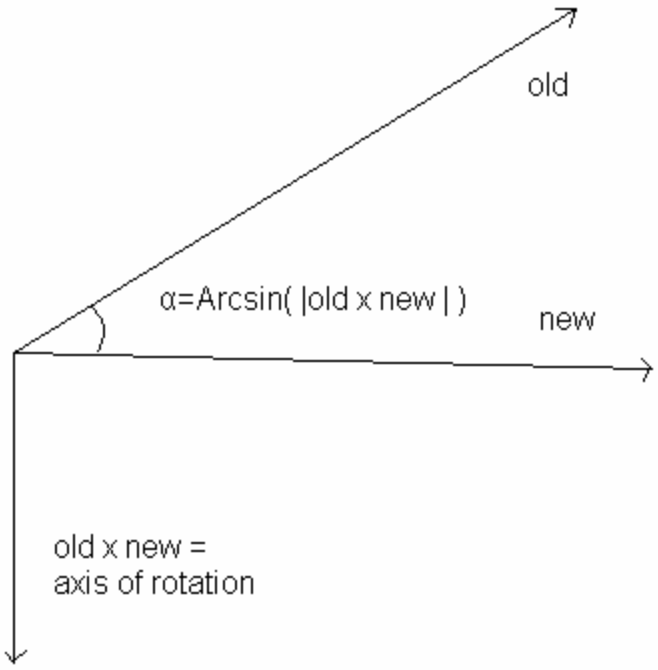
to ride the coaster that they have created. I have written the splinester program using Python and Syzygy, and its built inside of Syzygy's master/slave framework. I started with PyBolt code I had written last summer as a base, and then added and removed code to create the splinester program. Two particular areas of the program warrant further discussion and those are how the splinester program creates the track, and how the program computes how to ride on the track.

Users have quite an easy time designing and creating roller coaster tracks in the splinester program, and the use of Catmull-Rom splines is one of the primary reasons that this is possible. I decided to use CR splines because of the guarantee that the track a user draws will travel through all of their control points, and because CR splines guarantee C^1 continuity, so we have a guarantee that transitions between control points will be smooth. After the user draws their control points, the splinester program creates a CR spline that interpolates through all of them (including the starting and ending points, because the program creates a loop). Once the track is complete, and users are ready to ride their coaster, splinester applies a scaling factor to all of the control points, and then computes the spline again for the scaled control points. Splinester must do this scaling up so that the track will be large enough to ride.

Splinester decides how users should ride on the track by framing the spline with the Darboux-Bishop frame. Framing is the process of deciding what direction should be up, right, and forward, at every step of riding on the spline. The forward vector is simply our direction, and the Bishop frame lets us determine what directions should be up and right. Up and right are initially set in a somewhat arbitrary manner when we first start to ride the coaster (a preference is given toward having the riders of the coaster sitting

upright at the start). Then, after the coaster starts to move, the Bishop frame lets us decide how to rotate our initial up and right vectors to point in new directions. The Bishop frame itself is best described as a set of rules about how we can move through space. The Bishop frames lets us pitch and yaw, but we cannot roll along our axis (unlike the Frenet frame, which lets us pitch and roll, but not yaw). Prof. McCreary explains, "One way to envision the flight of the Bishop frame is to think of it as a bullet shaped vessel flying forward along the direction of its main axis. It can only apply propulsion directly *outward* from its nose. Thus, its propulsion cannot increase nor can it slow down its speed. It can only change direction" (emphasis in original). We should also note that although the Bishop frame does not allow us to roll directly about our direction axis, a slow roll can be induced if we travel in a helical path (the spin will be opposite the twisting of the helical coil).

Applying the Bishop frame to our spline is rather straightforward. We keep track of our current frame in an OpenGL style 4x4 matrix, and then apply and accumulate rotations to the matrix each frame. To apply our rotations, we need to know both the axis of our rotation, and the angle that we need to rotate about that axis. To calculate the axis of rotation, we simply need to take the cross product of our old direction vector, and our new (direction vector). Then, to calculate the angle of the rotation, we just need to take the inverse sine (arcsin) of the magnitude of the cross product of the old direction vector and the new direction vector (I make sure they are unit vectors). See the figure on the next page for more details.



After obtaining the rotation axis and angle, I let OpenGL apply the rotation to my matrix, and that completes the work that needs to be done for framing the curve.

References:

Angel, Edward. Interactive Computer Graphics: A Top-Down Approach Using OpenGL. Third Edition. Addison Wesley. 2003.

Francis, George K. Math198: Hypergraphics Lab and Class Notes. 2003.

Garland, Michael. "CS418: Computer Graphics" course notes. http://www-courses.cs.uiuc.edu/~cs418/. University of Illinois Urbana-Champaign, Dept. of Computer Science.

Hart, John. "CS419: Advanced Topics in Computer Graphics" course notes. http://www-courses.cs.uiuc.edu/~cs419/. University of Illinois Urbana-Champaign, Dept. of Computer Science.

Kaufman, Jake "virt", "X-BALLS" music for the splinester. http://virt.vgmix.com/

McCreary, Paul. "Towards an Understanding of the Frenet and Bishop Framings of Space Curves; An Application of a Real-Time Interactive Computer Animation (RTICA) To Model Airplane Flight and DNA Conformations."