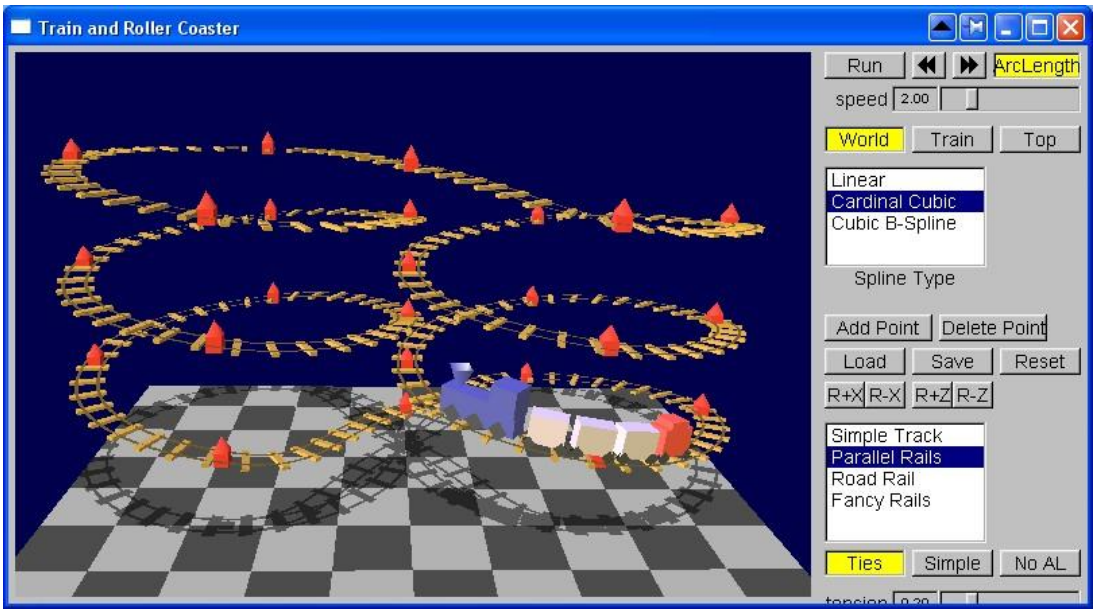
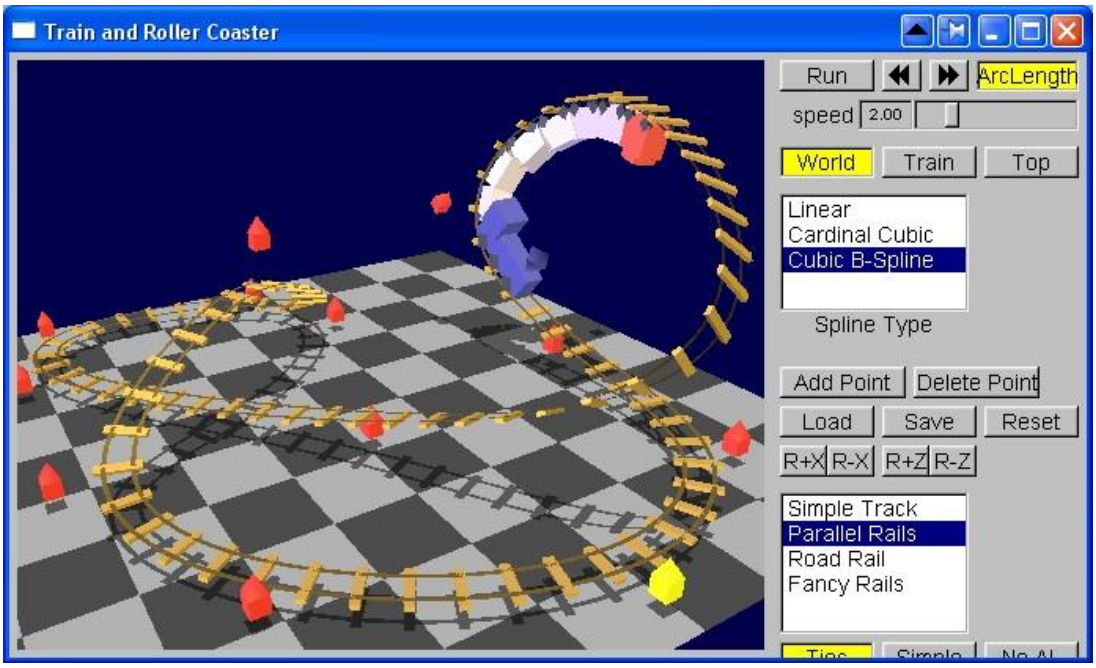
Project 3: Check Point #1

Overview

In this check point, you will create a train that will ride around on a track. When the track leaves the ground(or is very hilly), the train becomes more like a roller coaster.



Once it becomes a roller coaster, loops, corkscrews, and other things become possible



The main purpose of this project is to give you experience in working with curves (e.g., the train and roller coaster tracks). It will also force you to think about coordinate systems (to make sure that things move around the track correctly). Thus, we will provide you framework code so that you don't need to worry about that so much.

The core of the project is a program that creates a 3D world, and to allow the user to place a train (or roller coaster) track in the world. This means that the user needs to be able to see and manipulate a set of control points that define the curve that is the track, and that you can draw the track and animate the train moving along the track. We'll provide the framework code that has a world and manages a set of control points. You need to draw a track through those points, put a train on that track, and have the train move along the track.

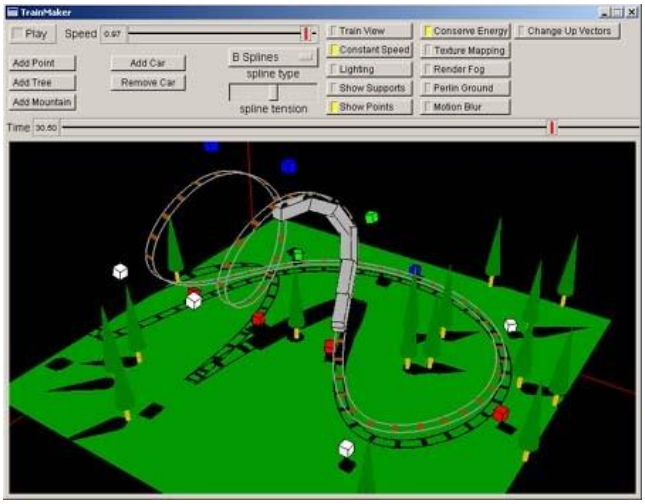
Basically, in this project you will need to:

1. Find your way around the framework code.
2. Add the basic functionality: draw a track (curve) based on the control points and draw a train on that track. If you do the latter part correctly, the framework will make it easy to animate the train going around the track. You will also need to implement a "train view" (so the user can "ride" your train).
3. Add more advanced features: nicer drawing of the track, arc-length parameterization, more kinds of splines, physics, ...
4. Add special effects and extra features to make it really fun. Really nice-looking train cars, scenery, better interfaces for creating complex tracks, ...

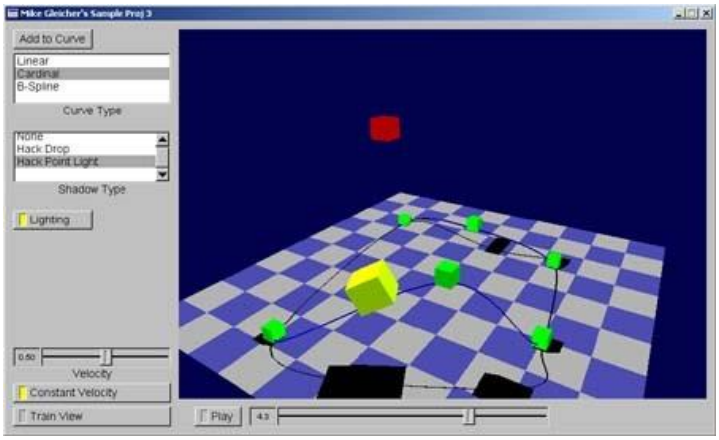
The basic functionality is most important, and the core advanced features (arc-length parameterization) are the next most important things. Fancy appearance (like using textures and pretty lighting) isn't the focus here - add them only if you have time after doing the more important things.

We have provided a sample solution of the possible features (at least the most common ones). We recommend that you play with it a bit to understand how it works. The example also has options that lets you see some of the most common mistakes and simplifications that students make.

While the assignment was a little bit different in 1999, the basic idea was the same. For a totally crazy solution to this assignment, check out RocketCoaster. It was what happened when I let two students work as a team. There are two more "normal" example solutions to this project which are by and Rob, both from 1999. One is a version that I wrote (called mikes-Train) and another was written by a really good student (robs-train).



Mike's sample train, circa 1999



Rob Iverson's A+ assignment from 1999

Ground Rules

You can complete the assignment individually or in pair. We allow (encourage) you to discuss the project with your classmates, but the things you turn in must be substantively your own.

Your program must use OpenGL and run on the classroom computers (like everything else in this class). While we strongly recommend you use the framework code, this is not a requirement. The Framework/Example solution uses FlTk, but you can use any UI toolkit that you wish, providing its available in the Lab (talk to us) as we need to be able to build your program.

If you want to use other external libraries/code, please ask us. Something like a math library or data structures library is probably OK, but please check.

Please print Project3-Grading.doc, let TA to check your score.

Submission

Upload it to the FTP site.

The Basic Functionality and Framework

The most basic part of this assignment is to provide a "track" for the train track. Your program must do the following things.

You can use other framework, for example: glut, glfw + nanogui, Qt...

1. Provide a user interface to look around the world, as well as providing a "top down" view.
2. You must have a "ground" (so the track isn't just in space).
3. Provide a user interface that allows control points to be added, removed, or repositioned. Note: even if you do a very advanced interface, you should display the control points and allow for them to be edited manually.
4. Allow for the control points to be saved and loaded from text files in the format used by the example solution.
5. Provide lighting.
6. Allow things to be animated (have a switch that allows the train to start/stop), as well as allowing for manually moving the train forward and backwards.

If you make your own framework, please make sure you can do all of these things. You don't get any extra points for writing it yourself, but you can't get any points if you don't have the basic features.

The basic/essential features you must add:

1. Have a Cubic B-Spline track. Your program should draw the track. The track should be a loop, always, and should be either interpolate or approximate the control points.
2. Have a train that goes around the track (with a play button to start/stop it). The train should always be on the track. Your train need not be fancy, but it should be obvious which end is the front. And your train should not distort in weird ways as it moves (if it is not rigid, it should be for a good reason).
3. Have the train oriented correctly on the track. The train should always face forward if the track is flat, and mostly face forward on a 3D track. Getting 3D orientation correct in the hard cases (like loops) is a more advanced feature (see below).
4. Allow the user to "ride" the train (look out from the front of the train). There should be a button or keystroke to switch to this view.
5. Have some scenery in the world besides the ground plane.
6. Your program is properly documented, is turned in correctly, and has sufficient instructions on how to use it in the readme file.
7. You should have a slider (or some control) that allows for the speed of the train to be adjusted (how far the train goes on each step, not the number of steps per second).

The framework code is designed to make it easy to add all of those things. In fact, there are "TODO:" comments explaining where to plug them in. See the discussion of it here.

The framework code was used to make the sample solution. We didn't give you all of the files, but you can see the "hooks" to the parts we didn't give you (they are turned off with a macro). In some places, we intentionally left extra code for you to look at as a hint. The framework has some spiffier features (like drop shadows), and some features you may not need (the control points have "orientation").

Basic requirements

Technique	Details	Points
Basic system	• User Interface with top-down view • User Interface with train view • A track with C1/G1 • Have a train • Lighting • Animated the things • Program is proper documented • Train is proper oriented • Some scenery • Slider to control the speed of the train	10 **(If you did not finish this, you cannot get any score.)**
Arc-Length Parameterization	Having your train move at a constant velocity (rather than moving at a constant change of parameter value) makes things better. Implementing this is an important step towards many other advanced features. You should allow arc-length parameterization to be switched on and off to emphasize the difference, you should also provide a speed control.	
Draw nicer looking tracks	The most basic track is just a line. To make something nicer (to make atube or a ribbon), you need to consider the geometry of the curve. 1. Parallel rails (2): for parallel rails, simply offsetting the control points (in world space) doesn't work. You need to know the local coordinates as you go around the track. 2. Rail ties (2): ties are the cross pieces on railroad tracks. Getting them right(uniformly spaced) requires good arc-length parameterization. In the example code, you can turn the arc-length parameterization on and off to see the difference.	

Advanced Features (Train-related techniques)

Meeting the basic requirements will get you a basic grade of 10. But to get a better grade, and toreally make the assignment fun, you should add some advanced features to your train.

These are really advanced features. It is much better to have the advanced features (that are really central to the assignment) than these frills, but frills can be fun. And we will give you points for them (**but remember, you can't get points for frills unless the more basic stuff works**)

Note: the exact point values for each of these is not given. The rough guide here will give you some relative importance (big features are worth more than small ones).

We will only check the features that you say that you have implemented correctly. Partial credit will begiven for advanced features, but negative credit may be given for really incorrect features. (So, it's better to not say you implemented a feature than to show us something that is totally wrong).

Also, remember that in your demo, you will have to show off the feature, so think about what demonstration will convince us that it works. For example, with arc-length parameterization, you're best off being able to switch it on and off (so we can compare with the normal parameterization), and think about a track that really shows off the differences. You should probably turn in example tracks that show off the features.

Technique	Details	Points
Have Real Train Wheels	Real trains have wheels at the front and back that are both on the track and that swivel relative to the train itself. If you make real train wheels, you'll need arc-length parameterization to keep the front and rear wheels the right distances apart (make sure to draw them so we can see them swiveling when the train goes around a tight turn). In the sample solution,the wheels are trucked (they turn independently), but each car still rotates around its center (so its as if they are floating above the wheels). You can do better than that.	1
Implement simple physics	Roller coasters do not go at constant velocities - they speed up and slow down. Simulating this (in a simple way) is really easy once you have arc-length parameterization. Remember that Kinetic Energy – Potential Energy should remain constant (or decrease based on friction). This lets you compute what the velocity should be based on how high the roller coaster is. Even Better is to have "Roller Coaster Physics" - the roller coaster is pulled up the first hill at a constant velocity, and "dropped " Where it goes around the track in "free fall." You could even have it stop at the platform to pick up people. Note: you should implement arc-length first. Once you get it right it is much easier.	2
Adaptive	To draw the curves (or to compute arc-length parameterizations), you need to sample along the curve	4

subdivision (or sampling)	(for example, to draw lines connecting the points). The curves are simple enough that simply sampling them uniformly and densely is practical. 1. Adaptive sampling is a more advantageous approach when the curve is straight, fewer line segments are needed. (2) 2. Additionally, the level of detail in the subdivision can be determined by the viewing distance; more details are shown when the distance is closer, and conversely, it is simpler when further away. (2) However, the benefits of adaptive sampling may be hard to see. If you implement adaptive sampling, be sure to have some way to demonstrate that it really works."	
Improve the User Interface of the framework.	Below are some suggestions for improving the UI framework: 1. Allowing users to load brushes of different styles (used for altering terrain) and displaying each type with an image. 2. Offering a simple and intuitive editing method to manage multiple tracks or trains, such as switching the view between different trains or altering the train's chosen path at a junction.	1
Multiple tracks and trains.	The track could have branches, you'd need to have some way to tell the trains which way to go, and some way to deal with branching curves. The framework is pretty much set up to have one track and train, but you could change this without too much hassle. You would need to make multiple World objects, but the hard part would be adapting the UI. 1. The railroad tracks have junctions and switches, allowing you to choose the direction. (1) 2. There will be multiple trains operating simultaneously, and measures are in place to prevent any collisions between them. (1)	2
Direct Manipulation and Interpolating C2 Curves	The easier form of C2 curves is approximating. Making it so that the user has direct control of the C2 curves (either having it interpolate the control points, or having the user adjust the track and having the system adjust the control points to make the track fit. Note: to get full credit for this feature, you need to provide for total control over the curve - just translating a control point when the user tries to move the curve doesn't apply.	2
Sketch-based interface	Allow the user to sketch a rough shape, and then create a smooth curve from that. This is difficult to do well, but if you're interested, we can suggest some interesting things to try.	1~5
Have People on Your Roller Coaster	Little people who put their hands up as they accelerate down the hill are a cool addition. (I don't know why putting your hands up makes roller coasters more fun, but it does). The hands going up when the train goes downhill is a requirement.	1
Headlight for the Train	Have the train have a headlight that actually lights up the objects in front of it. This is actually very tricky since it requires local lighting, which isn't directly supported.	1
Have the train make smoke	Steam trains are the coolest trains, even if they are being a roller coaster. Having some kind of smoke coming from the train's smoke stack would be really neat. Animate the smoke (for example, have "balls of smoke" that move upward and dissipate).	1
Non-Flat Terrain for the ground	This is mainly interesting if you have the train track follow the ground (maybe with trestles or bridges if the ground is too bumpy). 1. The scene features uneven terrain. 2. Multiple brushes of different shapes and sizes are provided to modify the terrain's elevations in the scene. (1) 3. After adjusting the terrain, other objects like trains or tracks won't penetrate the ground. (1) 4. When the terrain is too bumpy, trestles or bridges will be generated automatically to connect the landscape, allowing trains to pass. (1)	3
Support Structures	When the track is in the air, you could create trestles or supports to hold it up (like a real roller coaster). Of course, you'd want to handle the case where the track crosses.	1
Tunnels	Make hills with tunnels through them for your train to go through. The tunnel should adapt its shape to the track (so it should curve like the track curves).	1
Model loading	Loading an OBJ model is a complex task. You need to handle faces and material information, and due to the high number of faces, it is usually drawn via GPU. The project requires loading at least three different models, such as trains, trees, or characters.	3
Make totally over-the-top tracks	1. If you're really into trains, you could have different kinds of cars. In particular, you could have an engine and a caboose. (1) 2. Have Multiple Carriages on Your Train (1): Having multiple carriages on your train (that stay connected) is tricky because you need to keep them the correct distance apart. You also need to make sure that the ends of the carriages are on the tracks (even if the middles aren't) so the cars connect. 3. 3D rails (1): You need to ensure that the system will not experience lag issues when the track length is longer. (Hint: **Instancing** is a good way to repeatedly draw the same object.)	3

Documentation

Your README.txt should explain the following:

1. A abstract of your work
2. A list of all the features that you have added, including a description, and an explanation of how you know that it works correctly.
3. An explanation of the types of curves you have created
4. A discussion of any important, technical details (like how you compute the coordinate system forthe train, or what method you use to compute the arc length)
5. Any non-standard changes that you make to the code
6. Anything else we should know to compile and use your program

What to turn in

By the deadline you must turn in:

1. Everything needed to compile your program (.cpp files, .H files, .vcproj files, .sln files, and UI filesor other things your program needs). Be sure to test that your program can be copied out of this directory and compiled on a Storm computer.
2. Executable file.
3. Your README file.
4. Tech document.
5. A video (1 ~ 2 min) capturing from your program to show your work.
6. Some examples track files. You should not turn in the track files that we distribute (we give you abunch) - only turn in ones that you made.

Some Hints

Use the framework. It will save you lots of time.

In case it isn't obvious, you will probably use Cardinal Cubic splines (like Catmull-Rom splines). Cubic Bezier's are an option (just be sure to give an interface that keeps things C1. For the C2 curves, CubicB-Splines are probably your best bet.

You should make a train that can move along the track. The train needs to point in the correct direction. It is acceptable if the center of the train is on the track and pointing in the direction of the tangent to the track. Technically, the front and back wheels of the train should be on the track (and they swivel with respect to the train). If you implement this level of detail, please say so in your documentation. It will look cool.

In order to correctly orient the train, you must define a coordinate system whose orientation moves along with the curve. The tangent to the curve only provides one direction. You must somehow come up with the other two directions to provide an entire coordinate frame. For a flat track, this isn't too difficult. (You know which way is up). However, when you have a roller coaster, things become more complicated. In fact, the sample solution is wrong in that it will break if the train does a loop.

The sample solution defines the coordinate frame as follows: (note: you might want to play with itunderstand the effects)

1. The tangent vector is used to define the forward (positive Z) direction.
2. The "right" axis (positive X) is defined by the cross product of the world up vector (Y axis) and the forward vector.
3. The local "up" axis is defined by the cross product of the first two.

Doing arc-length parameterizations analytically is difficult for cubics. A better approach is to do them numerically. You approximate the curve as a series of line segments (that we know how to compute the length of). A simple way to do it: create a table that maps parameter values to arc-lengths. Then,to compute a parameter value given an arc length, you can look up in the table and interpolate.

Alternatively, you can do a little search to compute the arc length parameterization. If you have a starting point (u) in parameter space, and a distance you want to move forward in arc length space,you can move along in parameter space, compute the next point, find the distance to it, and accumulate.