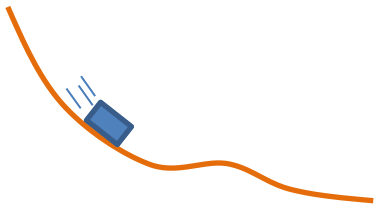


24-780 B—ENGINEERING COMPUTATION

Assigned: Tues. Nov. 15, 2022
Due: Tues. Nov. 22, 2022, 11:59pm

Problem Set 9: Track and Slide

In PS03 and PS04 (with a bit in PS06), we build a somewhat sophisticated 2D shape viewer and editor. Now, we want to build on that to create a movement simulator that can be used to visualize a box sliding down a track (spline curve).

I expect this assignment to take about 3-5 hours only, so that you can keep pushing hard on the development of your team project.

Task 1 (Study Track2D and TrackManager classes in lecture)

We already have the classes to manage most of the data and user interface for a Shape2D which can do the following:

- load from file and save to file,
- moving, deleting, and adding points (Although we never did the latter two graphically, I included them in my start-up code. Delete can be applied to any highlighted point. Adding to highlighted point creates a new point on edge, 20% of the way back to previous point).
- zooming and panning
- some color control

We are going to *inherit* from *Shape2D* to create a *Track2D* class that can manage the additional requirements;

- whereas a shape is closed and non-intersecting, a track is open and is allowed to intersect (making loops)
- a spline has a beginning and an end, so things like perimeter no longer make sense. However, it does make sense to think of distance *along* the spline
- calculating the spline shape, with the points we maintain in *Shape2D* (vector *thePoints*) used as "control points" for the spline. Note that we will only be able to extend the spline from the second control point to the second-to-last control point, with the "outer" control points used only to set the slope of the end. (The coding for cubic splines is mathematically rigorous, but it is a "solved" problem we can essentially download.)

Let's do everything possible to minimize changes to Shape2D. We will, however have to **over-ride** some functions in Track2D:

- paint() will now paint a line along *splinePoints* instead of along *thePoints*
- isIncluded() will now always return false
- selfIntersects() will now always return false
- recalcShape() will now also include recalculation of spline points

We also need some additional functions such as:

- Point2D getCoords (float givenLength) // measured along spline
- float getAngle(float givenLength) // measured from back point to forward point

Task 2 (Add/Remove Sliding Boxes To/From Model)

I have started you up by creating a "draft" class definition for *SlideBox* class. Your task in this assignment, therefore, is to add to this code and you should NOT duplicate any of the coding that is provided for you (although you can edit as needed).

As you can see from the attached code, the *TrackManager* class has the ability to store several objects of the *SlideBox* class. You need to provide a means to have the user create slideboxes that can be added to the model (see SlideBox::loadFromConsole). There is no limit to how many boxes can be added. The parameters listed in the draft version of the *SlideBox* class are a good guide to what you should ask from the user, but try to keep it as simple as possible at first. You can always come back later and add more features (like label, material, squeak sound 😊) none of which are required or expected (see time budget above).

We also need the ability for the user to remove a slidebox.

There is no requirement for the slideboxes to be readable/writeable from/to permanent storage (see time budget above).

Take a few minutes to think about what functions are in the *Track2D* class that can be used to accomplish the necessary calculations to paint the box (global coords, slope, etc.).

There is no prescribed look that the boxes should have, but their scale should match the scale of the track. Obviously, you need to code the SlideBox::paint() function to complete this task since the call to the function is already part of the TrackManager::manage() function.

Task 3 (Simulation)

We don't want to just look at nice, colorful, editable spline curves and have some boxes show up on the screen. For this task, you are asked to simulate how boxes will move along the track. For simplicity, assume that the boxes are "stuck" to the track and cannot detach from the track (e.g., the way a roller-coaster might be). You do not have to worry about what happens once the boxes move past the end of the track **except to reset the boxes at their initial spot**. Notice that the draft class *SlideBox* contains parameters for the "current" distance, velocity, and acceleration which you should take as a hint that you need only focus on movement along the track, updating these parameters for each time increment, not worrying about global position of the box. Yes, it's a dynamics problem, but we'll keep it simple by assuming the box is a "particle" (i.e., no rotation) with only gravity and friction working on it.

Obviously, you need to code the SlideBox::move() function to complete this task since the call to the function is already part of the TrackManager::manage() function.

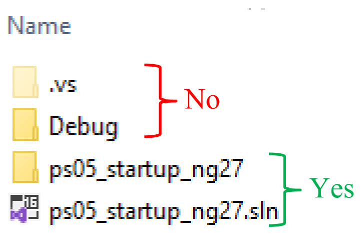
Deliverables

7 files (zipped together):

TrackManager.h, TrackManager.cpp
Track2D.h, Track2D.cpp
SlideBox.h, SlideBox.cpp
ps09_track_andrewID.cpp

The 7 files above are the only ones that will require changes from you, so don't need to include all the others unless you made some changes. Upload the zip file to the class Canvas page before the deadline (Tuesday, Nov. 22, 11:59pm).

Alternatively, if you are using Visual Studio, it may be easier to submit your entire solution rather than a collection of files. To do this, create a *zip file* of the whole project (the .sln file and the associated folder), being careful NOT to include the hidden folder called ".vs". This folder is used only to manage the IDE and is typically huge (100MB). Erasing or omitting it will just force Visual Studio to rebuild it when needed. The Debug folder should be kept out of the zip file too to avoid including executable files that some firewalls may disallow. *The name of the project should include your AndrewID*



Learning Objectives

Use of classes and objects in C++.

Modeling animation of real-world objects.

Increasing understanding of graphical data representation.

Implementing algorithms developed by others.

Changes to Shape2D class

Here are the changes I made to Shape2D class in order to have them work appropriately

- Changed anything private to protected
- Changed isContained(), selfIntersects(), selfIntersects(), and recalcShape() to virtual, so that when I over-ride these functions in Track2D, they are called appropriately
- Instead of directly reading a file using a constructor, I copied the code to a separate function called readFile(). I needed to clear any old points before reading the new points. The old constructor simply calls this new function. This change allows for a shape to be reloaded from a file without having to delete it and create a new one.
- Found and corrected error in my implementation of addPoint(int index, float ratio). The first if statement was incorrect.