Programming Assignment #4

Due Nov 10 by 11:59pm **Points** 4 **Submitting** a text entry box or a file upload
Available until Nov 15 at 11:59pm

This assignment was locked Nov 15 at 11:59pm.

Due: Wednesday, November 10th (at midnight).

Late Policy: Assignment submission will remain open for 5 additional calendar days after the formal deadline. Late submissions will be penalized by a maximum of 1 point on our 4-point scale (i.e. if you submit 5 days late and your assignment was otherwise deserving a "3", you might get a grade as low as a "2" due to this penalty). Some leniency might be exercised on this general rule based on how late the assignment was, and whether turning in an assignment was a repeating occurrence or a one-time event.

Synopsis: You will make a program that uses a parametric curve to define an intricate trajectory, and use this representation both to draw such a curve as well as move/animate an object along it.

Learning Objectives: To exercise your understanding of parametric curves (including cubics) and their representations, their use for motion/animation purposes, and become more familiar with the concept of continuity of piecewise-defined curves.

Evaluation: Based on our 4-point grading scheme, as discussed in our introductory lecture. You get a check (“3”) if you turn in a viable, and complete submission (even if it just draws a rectangle like the example in the tutorial). “Above and beyond” grades (i.e. a "4") will be awarded for people who have crafted something particularly cool. As a general rule, no more than 1/3 of all assignments turned in (the very best ones, that is) will be considered for a "4" grade.

Collaboration policy: This is an assignment to be done individually. Code not written by you needs to include proper attribution (see **this post** **(https://graphics.cs.wisc.edu/WP/cs559-fall2020/2020/09/10/collaboration-policy/)** here). It is always ok to use code provided in our in-class examples as a starting point, but you need to add your own effort to raise those examples (or other sources) to what is asked by the programming assignment (i.e. finding some code on some online forum that does all the job for you that is needed to satisfy the assignment is not the intent, if you haven't added any of your own effort to it). If you use somebody else's code (other than our GitHub examples), make sure to clarify in your submission notes what **you** did, and what you repurposed from the external source.

Hand-in: Electronic turn-in on Canvas. Make sure that you turn in all files needed for your program to run. It is acceptable to turn in a single HTML file with your program, but even preferable to separate your code into an .html file and a separate .js file containing the JavaScript code, similar to the examples in our **GitHub repository** **(https://github.com/sifakis/CS559F21_Demos)** (see, e.g. Demos 0-4 from Week 7). If you submit anything else than a single HTML file, please put everything in a

single ZIP archive. Feel free to use the copy of the glMatrix library included in our examples in the GitHub repository (or use them as a starting point) if it's convenient. ***It is not acceptable to submit a link to JSbin for this assignment!***

Description

In the last few lectures before switching to 3D (especially in lectures 10 through 13) we discussed the representation and implementation of parametric curves, including certain types of cubic curves with convenient properties. We addressed the motivation in using such geometric tools (e.g. creating more intricate shapes than the ones allowed by built-in canvas commands, and having the ability to animate objects along them). We also looked at implementation considerations associated with the use of parametric curves in 2D (most of which are found in our **GitHub repository** (https://github.com/sifakis/CS559F21_Demos) under **Week5/**), such as the drawing of a curve as a finely-refined chain of line segments **[JSBin demo]** (http://jsbin.com/ziletiq), using the parametric representation to control the position and orientation of an object that travels along the curve **[JSBin demo]** (http://jsbin.com/meliyax), and how piecewise-defined curves are implemented and what different degrees of continuity feel like **[JSBin demo]** (http://jsbin.com/vupevab). In Week 6, we also talked about various types of *cubic* polynomial curves (including Bezier, Hermite and B-Splines), and the convenient properties they provide as building blocks of composite (piecewise-cubic) curves (see our **GitHub repository** (https://github.com/sifakis/CS559F21_Demos) under **Week6/**).

In this assignment you will create your own program, in the spirit of the demos we saw in class, extending them of course into something more interesting and hopefully exciting! You are welcome to take inspiration or even adopt some of the implementation practices from the in-class demos; your program, however **must** fulfill the following requirements in order to get a *satisfactory* "3" grade:

- **Requirement #1**. Your drawing has to include, at minimum, one of the following:
  (a) A piecewise-defined curve, that shifts from one formula to another (see the example in this **[JSBin demo]** (http://jsbin.com/vupevab), that is also closed in the sense that the curve returns to the place where it started, forming a closed "loop"; note that this is different from the example we saw in class, where all curves, including those that had piecewise-definitions, were "open".
  or (you can do both if you wish!)
  (b) Multiple separate curves, out of which one has to have a piecewise-defined formula (i.e. at least two components with different formulas, joined together), but in this case the curves don't have to be closed (unless you want them to be!).
- **Requirement #2.** Your drawing needs to have at least one object that is animated using one of the curves defined. It is sufficient to have the parametric curve control only the *position* of the moving object (as in this **[JSBin demo]** (http://jsbin.com/ziletiq), but if you want to be extra fancy you can also control the *orientation* of the moving object (see this **[JSBin demo]** (http://jsbin.com/meliyax).
- **Requirement #3.** You must "draw" the path of at least one of the curves involved in your scene (consider the way we discussed in our demos, by splitting the curve up into small line segments). It's ok to have a "switch" of sorts (e.g. the value of a slider) that "turns off" drawing the curve line, if

you feel your scene looks better without it! As long as there is the option to display that path, you are good!

- **Requirement #4.** At the point where the different formulas of a piecewise-defined curve in your scene, you must enforce at least C0-continuity (G1-continuity or higher would be nice; C2 might be overkill, but you are welcome to do it). This is just for *one of the junctions* in one of your piecewise-defined curves; you are free to use any degree of continuity for all other cases.
- **Requirement #5.** At least one of the curves you use in your implementation has to be a parametric cubic (Hermite, Bezier, B-spline, etc). It doesn't have to be the case that all components of a piecewise curve will have to be a cubic ... just that *somewhere* in your scene there must be a curve that has *at least one component* (if it's a piecewise/component curve) that's a cubic. Frankly, you might find that one of the most natural ways to enforce C1 continuity (if you choose to have at least that degree of smoothness) is to resort to cubics anyhow ...

You are welcome to use either sliders, or automatic animation to “move” your objects along any curve trajectories; use whatever feels best for your vision of the scene you want to create!

As always, you are encouraged to try and exceed these requirements, and if you do well, you can compete for a “4” *above-and-beyond* grade. Here are some recommendations for that:

- Controlling position *and* orientation of objects in your scene.
- Having controls/sliders that alter the shape of your curved trajectories.
- Create intriguing motions and shapes.
- Enhance the appearance of curved profiles by giving them “thickness” or drawing multiple parallel lines that go alongside your curve; think of a train track with 2 parallel rails, that swerve together!
- Maybe ... if you want to be ambitious you might contemplate having your curves live in 3D? We saw some examples of that in Week 7, and on Lecture 16 we will wrap up most of the tools you are likely to need. Note however that most of the examples from Week 7 had curves that still were restricted onto a plane (even if that plane was placed somewhere in 3D). You certainly don't have to be restricted to that.

Readings

The following will be useful references for the topics discussed in weeks 4 through 7:

- Foundations of Computer Graphics (Chapter 6, section 1 **Link**) discusses transformations and representations in 2D. Later sections in this chapter venture into 3D, which will be useful very soon!
- Chapter 5 of Hart’s Big Fun Computer Graphics Book (**Link**) is also very useful for 2D geometry and transforms.
- Foundations of Computer Graphics Chapter 15 (**Link**) covers the representations of parametric curves, and many other concepts as we discuss in class! Our lectures were very closely aligned to the notation used in this chapter.