INTERACTIVE GEOMETRY LAB

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Graphics Laboratory ETH Zurich

Computer Graphics
Exercise 5 - Splines

Handout date: 22.11.2013

Submission deadline: 13.12.2013, 14:00h

The goal of this homework is to familiarize yourself with the Bezier and B-Spline curves. This homework consists of two parts: first, a set of theoretical exercises, and, second, a programming exercise that requires you to implement a simple cubic B-Spline curve editor.

You must work on this exercise **individually** as usual. Your solution of the theoretical part can be handed in either in class or together with the programming part electronically, which you do by:

- Emailing a .zip file containing your solution to `introcg@inf.ethz.ch` with the subject and the filename as `cg-ex5-firstname_familyname`

The .zip file should contain:

- A folder named `source` containing your source code.
- A `README` file clearly describing what you have implemented for the programming part.
- A typeset or scanned version of your solution of the theoretical part unless you submit it in class.

No points will be awarded unless your source code can:

- Run on the Firefox web browser on student lab PCs at CAB H 56 or 57.

Grading of the programming part

Your submission will be graded according to the conformance of your editor to the expected behavior described below. If you have partial solutions in your submission, describe them clearly in your `README`, otherwise there will be no partial credit.

You are required to attend the **exercise session on 13.12.2013 at CAB H 56** to get your solution graded. Those absent will receive no point for the programming part. You are required to email your solution before the session as well.

Theoretical questions

1. Prove that Bezier curves are affine-invariant.

2. Compute the minima and maxima of a Bernstein polynomial $B_{\mathtt{in}}(t)$ (parameter values and function values).

3. One wishes to change the tangent direction of one of the endpoints of a Bezier curve. What modifications need to be made to the control polygon so that the prescribed tangent is attained? How about changing the tangent of an arbitrary point on the curve?

4. One wishes to move a point on the Bezier curve (say, with parameter value $t_{\mathtt{move}}$) to a prescribed location while keeping the endpoints of the curve fixed. Is it possible to find a corresponding control polygon that will achieve this? If yes, how?

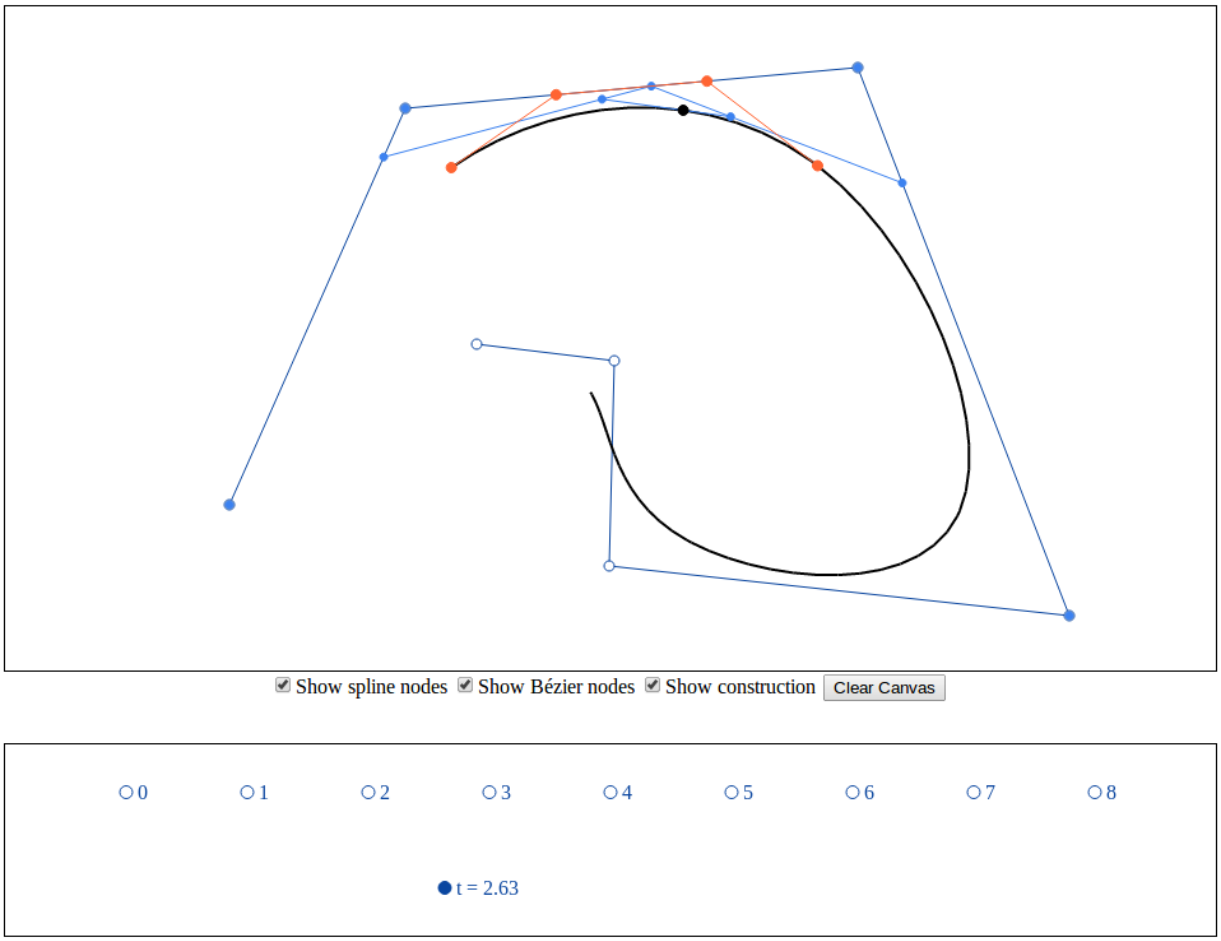

Figure 1: Screenshot of a sample solution.

Programming exercise

You are required to implement a cubic B-spline editor using the code base provided. Your editor should support the functionality described in the next sections. A screenshot of an example solution is shown above.

The codebase is written in JavaScript. The code should run on Firefox or Chrome web browser on all major platforms (Windows/Mac/Linux), but we only support and have tested the code with Firefox on Windows and Linux. Note that your solution should run out of the box on one of the two environments. You are encouraged to use the lab machines at CAB H 56 and H 57 to void any platform-specific complications.

The code template can be downloaded from the course web site. It implements the GUI and already gives you some basic functionality.

User interface

The GUI consists of two windows. The top window shows your B-Spline curve editor. The basic implementation that you have to extend allows the user to add nodes or control points by clicking in the window. This will automatically also add knots in the knot vector that are displayed in the window below. The default knot sequence consists of integers starting at 0. The number of knots is always the number

of control points plus two. The number displayed near the knots is their index and *not* their value. The values of both the knots and control points can be modified using a mouse select/drag interface. Control points and knots cannot be deleted. Additionally, at the bottom of the knot window there is an additional active knot. The value of this knot will be used to illustrate the de Boor algorithm as illustrated in the image.

Functionality

Given the UI you have to enhance the functionality of this codebase. In the editor window (top) you need to render the B-Spline curve corresponding to the control points and knot sequence specified by the user. The application has to be interactive, meaning that changes in the control points, knot vector or active knot should be reflected visually automatically.

In addition to the B-Spline, you have to render in this window, toggled by check boxes between the two windows, two additional features:

1. Construction lines of the de Boor algorithm corresponding to the active knot value as shown in the screenshot.
2. Bezier control points of the relevant interval that includes the value of the active knot.

Under the Hood

The code provided is structured over several files. The relevant files for this homework are the following:

1. `node.js`: Encapsulates the functionality of a control point.
2. `knot.js`: Encapsulates the functionality of a knot. Note the boolean `_isTimeKnot` marks the special active knot.
3. `curve.js`: Perhaps the most important of all, it encapsulates the B-Spline curve and its visualization. All rendering has to be done inside its draw member function. The class has references to all the necessary information: control points, knot vector, active knot, etc. The toggle variables for the various drawing modes are also available here. Inside the draw member function you have hints how to render points and lines and how to change the color using helper functions.
4. `bSplineTools.js`: An empty file that you can use to separate your B-Spline specific implementation from the drawing routine.