Computer Graphics
Exercise 5 - Splines

Handout date: 16.12.2011

Submission deadline: 31.12.2011, 23:59h

You must work on this exercise **individually**. You may submit your solution by:

- Emailing a .zip file of your solution to `introcg@inf.ethz.ch` with the subject and the filename as `cg-ex4-firstname_familyname`

The .zip file should contain:

- A folder named `source` containing your source code.
- A `README` file inside the `source` clearly describing what you have implemented for each module and compilation instructions for your code.

Your submission must be self-contained: include all additional data and the code required to load them.

No points will be awarded unless, your source code can:

- Compile (without any external dependencies besides GLUT)
- Run

Your submission will be graded according to the quality of the images produced by your renderer, and the conformance of your renderer to the expected behavior described below for each module.

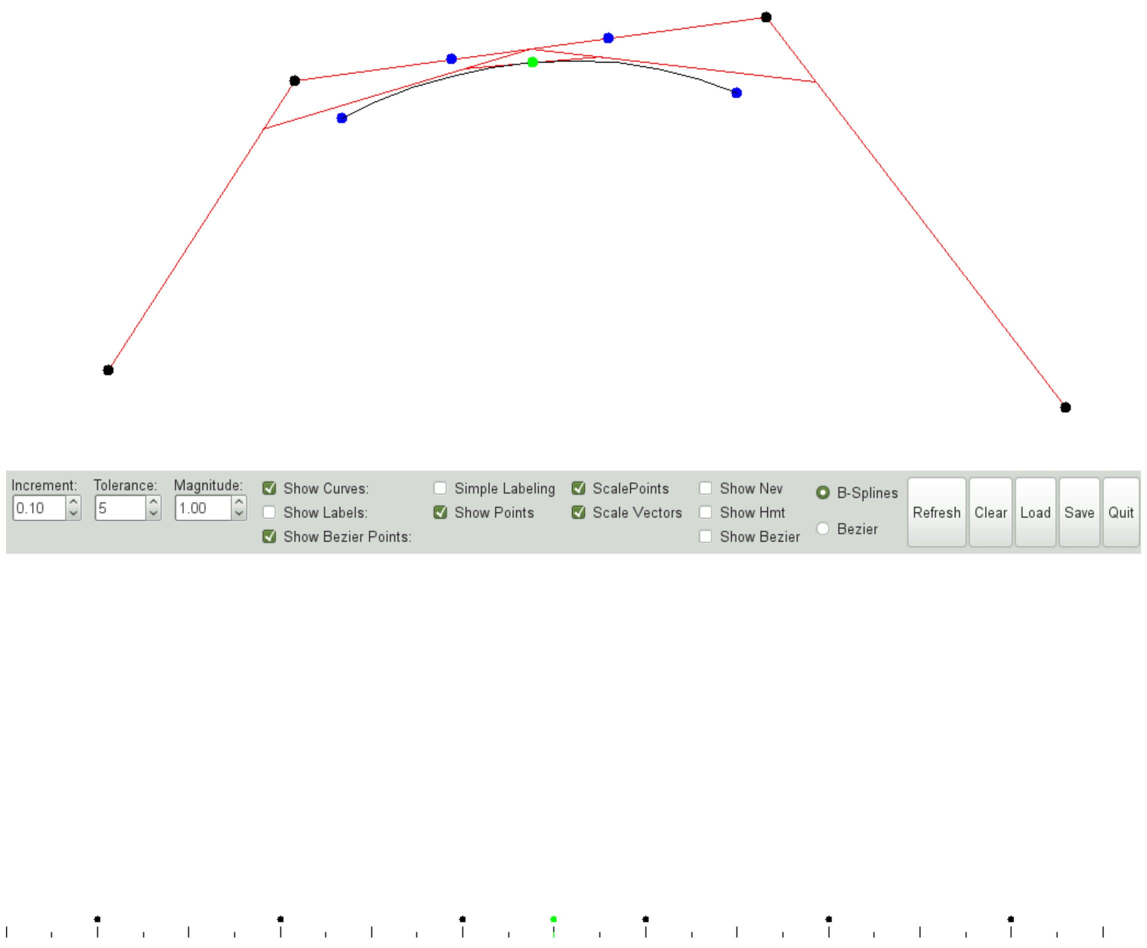
Goal

The goal of this homework is to familiarize yourself with the B-Spline and Bezier curves. The homework contains a programming exercise that requires you to implement a cubic B-Spline curve editor accompanied by some theoretical questions.

The programming exercise you can submit as before, the theoretical part you can either submit it as a PDF with your programming part or submit it in class. Please specify your choice in your README.

Grading

If you have partial solutions in your submission, describe them clearly in your `README`, otherwise there will be no partial credit. Describe what you have implemented or tried to implement for each module.


Increment: 0.10 Tolerance: 5 Magnitude: 1.00

[✓] Show Curves: [ ] Show Labels: [✓] Show Bezier Points: [ ] Simple Labeling [✓] Show Points [✓] ScalePoints [✓] Scale Vectors [ ] Show Nev [ ] Show Hmt [ ] Show Bezier [✓ B-Splines] Bezier

Figure 1: Screenshot of a sample solution.

Programming

For this part you need 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. Also from the course web-site you can download a Linux executable of th sample solution.

0.1 Codebase

Note that for this homework the code base compiles only in Linux and we only support this platform. We tested it on the Linux machines in your labs. If you wish to work from home and you do not have a Linux machine you can log in remotely. On a Mac this is trivial, just type ssh -X favourite machine in the lab and it wil work. On Windows you need to install an X-server.

Download the file H5_Codebase.tar.gz from the web page and untar it (tar xzvf H5_Codebase.tar.gz). This will create an empy directory and copy all the files in it. Type `make` to compile the program. This will create an executable called `bse`. The code given already have some basic functionality implemented.

0.2 Functionality

Your editor should allow the user to create a set of control points of a B-Spline curve and its corresponding knot vector. The user should be allowed to interactively edit both the control points and knot vector. The editor should render the B-Spline curve corresponding to the control points.

As learned in class, any polynomial curve has different control points depending on the set of basis function used to represent it. In this case you will be asked to display in a distinctive color the Bezier control points corresponding to your curve.

In addition to this, the editor should have an **active point** along the curve. This point can move along the curve by modifying its **t** value. This point and its coresponding **t** value should be rendered in a distinctive color. The **t** value should be displayed as a cirlce in the knot window and the user should be able to change it using mouse dragging. Your editor should illustrate the pyramidal algorithm showed in class by rendering the affine lines in the de Casteljau/de Boor algorithm as illustrated in the figure above.

If you are not able to implement all the features, please document any limitations in the `README` file.

0.3 User Interface

The basic UI coinsists of 2 windows. The top window will show your B-SPline curve editor where you will render your B-Spline and other information required by the specificatione above. The bottom window will be the knot editor where you will show and edit the knot vector. Between these windows there is a set of widgets that will togle on and off different features.

In the curve window (top) you need to render the B-Spine control points, control polygon, B-Spline curve and the active point. Toggled by the check box controls provided in the GUI, you should show the B-Spline control points, Bezier control points and the affine transformation between the control points as illustrated in the screenshot.

In the bottom window you need to render the current knot vector and the knot of the **active point**. The user should be allowed to change the knot points by dragging them similar to the behaviour of the B-SPlines control point functionality already provided for the B-Spline window.

The UI contains 3 heck box controls. All these togle 3 rendering features. **Show Construction** togle the rendering of the construction lines of the **active point** as shown in the screenshot. **Show Bezier Points** togles ythe rendering of the Bezier pointes corresponding to the curves. **Show Points** togles rendering of the B-Spline control points. All these check boxes are already oined for your convenience to 3 global variables as explained in the next section.

The UI also has 4 buttons. The **Refresh** button triggers a re-rendering of all windows. The **Clear** button resets the B-Splines (i.e. removes all the control ponts and knot vector). The **Load** and **Save** button provide a simple load/save functionality. This is an optional functionaliy that can be useful for debugging.

To differentiate between altering the control points and knot vector of the B-Spline curve from the **active point**, the manipulation (i.e. dragging) of the **active point** should be done using a differet mouse button (i.e. right mouse button).

0.4 Under the Hood

The code provided is structured over several files. A simple Makefile is provided. You are allowed to add additional files as needed.

The two most important files are **BSplines.h.cpp**, **UI.cpp** and **KnotVector.h.cpp**. They encapsulate the functionality of the two widget windows where you have to render the B-SPline curves and knot vectors respectively.

The **UI.cpp** file contains the mouse and keyboard hooks as well as the rendering callbacks for the BSpline window. The **BSplines.h.cpp** file encapsulate the abstract functionality of a BSplines curve. The **KnotVector.h.cpp** contains the ui hooks and the rendering callback for the knot vector sequence.

In order to facilitate communication between **BSplines** and the **KnotVector** classes, the **KnotVector** class contains a pointer to the **BSpline** object associated to the window.

For the rendering you are given a few functions: DrawLine, FillEllipse and SetColor. They are implemented in main.c. Also you can call **RepaintAll** function in order to trigger an update for all the windows.

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?