Programming Assignment 1: Curves and Surfaces

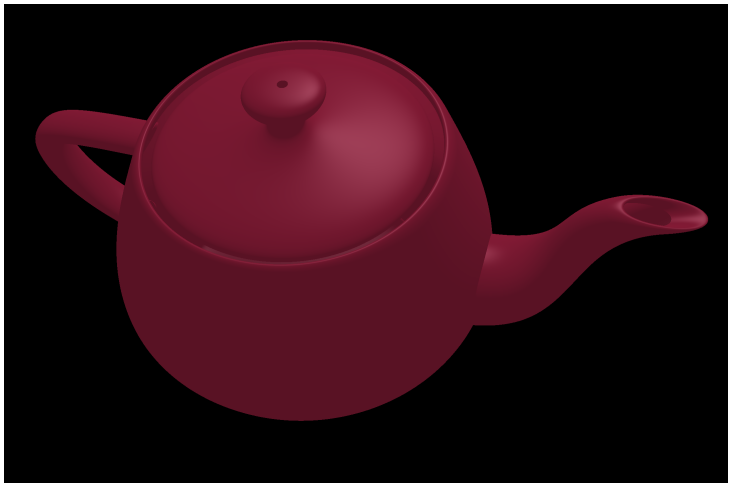
MIT 6.837 Computer Graphics

Fall 2021

Due September 29 at 8pm, Boston Time

In this assignment you will implement spline curves and tensor product surfaces to model interesting shapes. The primary goal is to introduce you to splines and coordinate systems. Upon successful completion, you will be rewarded with a powerful tool for modeling 3D shapes.

To get you motivated, here is an image of the Utah Teapot[1], composed of tensor-product Bézier patches that your program will create:



1 Getting Started

The sample solution is included in the starter code distribution. Look at the `sample_solution` directory for Linux, Mac, and Windows binaries. You may need to change the file mask on the Linux/Mac binaries, e.g., by executing the following command in your terminal:

```
chmod a+x sample_solution/linux/assignment1
```

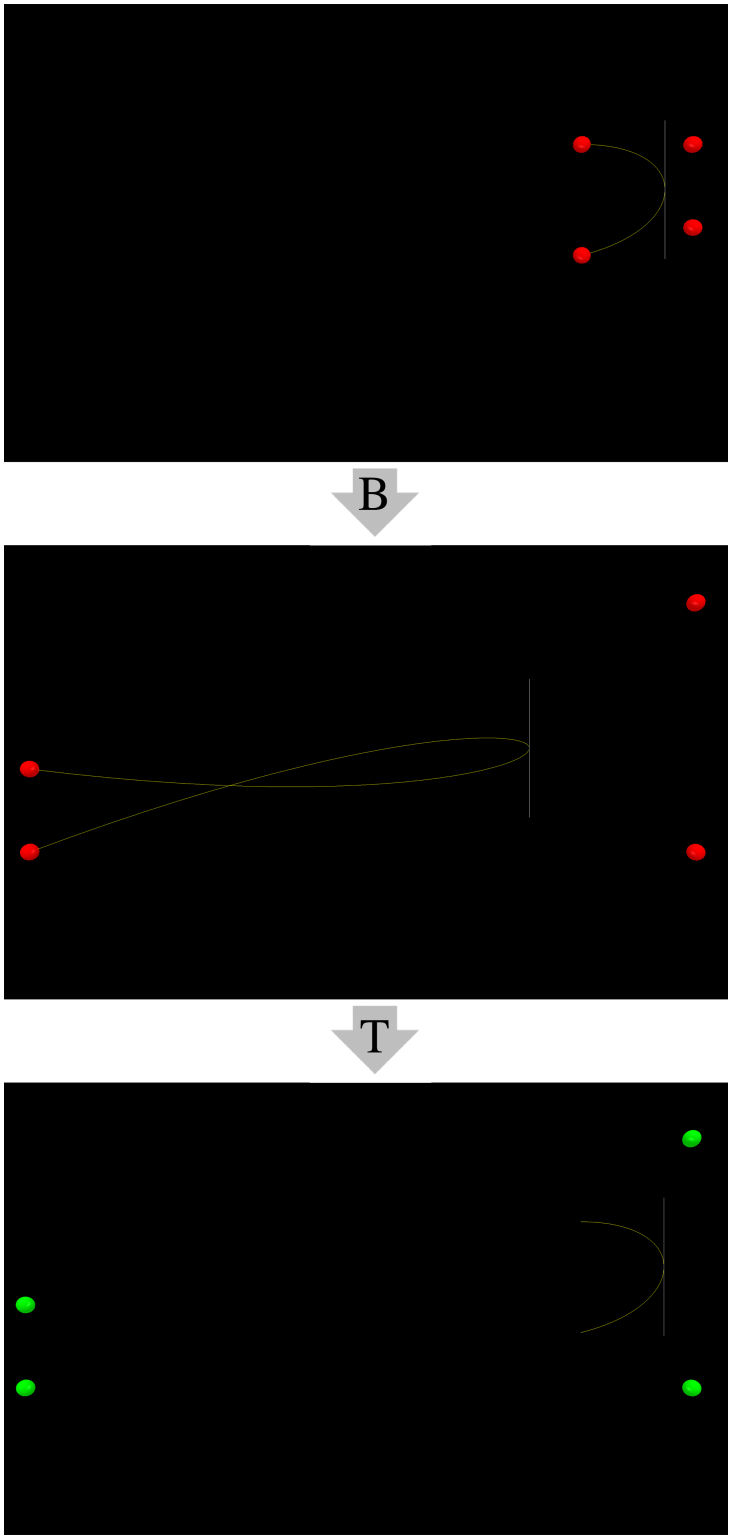
Unlike in Assignment 0, this program comes with a mouse-based camera. Move your mouse while holding down the left button to rotate, the middle button to pan, and the right button to zoom. You can also zoom using the scroll wheel. Additionally, you can press the `A` key to toggle the x-, y-, and z-axes, in red, green, and blue, respectively, on and off. You will not need to implement this camera for the assignment; it is already there for you in the starter code.

First, take a look inside `assets/curve.spline`. This file defines a single spline curve. The first line of the file specifies that it is a (cubic) Bézier curve (as opposed to a B-Spline), and each of the following lines defines each of the four control points.

Now, try loading the file, e.g. by running:

```
sample_solution/macos/assignment1 curve.spline
```

[1] `https://en.wikipedia.org/wiki/Utah_teapot`



You should initially see the first screen above. Each of the four control points is displayed as a red sphere, and the corresponding Bézier curve is displayed in yellow. There is also a white tangent line displayed. Next, press the `B` key. This converts the geometry (i.e., the control points) from the cubic Bézier basis to the cubic B-Spline basis. You should see the middle picture above on your screen. Notice that the curve displayed is **still a Bézier curve**. Although you just converted the control points to B-Spline geometry, the program is still using the Bézier basis to sample the curve. Press `T` to toggle the spline basis. Now, you should see the third screen—the control points from the middle screen are being

interpreted using the B-Spline basis, and, thus, the curve from the first screen is rendered. To indicate that we are now using the B-Spline basis, the control points are rerendered in green. You can press `Z` to convert to Bézier geometry; the curve will be computed using the B-Spline basis until you toggle.

Notice that you can press `B` or `Z` multiple times in a row, and the geometry changes each time. The `B` trigger interprets the current control points as being in the Bézier basis and converts them to the B-Spline basis (and vice versa for `Z`).

Now, take a look at `assets/polybspline.spline`. This file defines a B-Spline curve with six control points. If you render this curve using the program, you will see that all of the control points and the spline are rendered as well as a tangent line for each B-Spline segment. Note that the keyboard triggers are disabled in this view. The file `assets/polybezier.spline` defines a poly-Bézier curve.

Finally, look at `assets/teapot.spline`. This file defines a collection of bicubic Bézier tensor product patches. As before, there is one control point per line. Rendering these patches displays the Utah Teapot.

2 Starter Code

Follow the same steps as in Assignment 0 to build the starter code. For instance, on MacOS or Linux, execute the following:

```
mkdir build
cd build
cmake ..
make
```

The result will compile, but running, e.g., `build/assignment1 curve.spline` will not display much.

The starter code uses the OpenGL Mathematics (GLM) library extensively, especially for linear algebra operations. You may want to familiarize yourself with the API[2]. In particular, classes such as `glm::vec3` or `glm::mat4` and functions such as `glm::cross` and `glm::normalize` will be quite useful.

Note that GLM has a **column-major API** for matrices. What this means is that, for example, the matrix $B_{\text{Bézier}}$ from above translates to:

```
const glm::mat4 BBezier(
       1, 0, 0, 0, -3, 3, 0, 0, 3, -6, 3, 0, -1, 3, -3, 1
);
```

This also means that the matrix will expect the column index first when accessing values:

```
const glm::mat4 BBezier(
       1, 0, 0, 0, -3, 3, 0, 0, 3, -6, 3, 0, -1, 3, -3, 1
);

// This gets the value in the first column (0 index) and fourth row (3 index)
float value = BBezier[0][3];
```

The `main.cpp` file contains the code for launching your program; you do not need to modify it. You will need to make changes to the `SplineViewerApp`, `CurveNode`, and `PatchNode` implementations.

2.1 `SplineViewerApp`

This class is responsible for reading in and parsing the user-specified `.spline` file. These `.spline` files contain the type curve they represent in the first line (i.e. Bezier curve, B-spline curve), and then the control points of the curve in the subsequent lines. The class is also responsible for creating the corresponding curve or patch nodes so as to render the curve or patch onto the canvas. The parsing part is taken care of for you. The first line of the input file is stored in `spline_type`, and `control_points` is a `vector` of `glm::vec3` objects corresponding to the specified control points.

When parsing a spline curve, it is recommended that you create as many curve nodes as there individual curve segments rather than creating a single curve node for the entire spline.

[2] `http://glm.g-truc.net/0.9.9/api/index.html`

2.2 `CurveNode`

This node should plot a single curve as well as its control points and a tangent line at at least one point on the curve. We provide a starter implementation for a `PlotTangentLine` function. Currently, this function just plots a single fixed line on the canvas. You will want to change it to plot a tangent line for the particular curve instance. The code shows how you might go about rendering a line on the screen. In particular, note that `positions` is a vector of points, and `indices` indexes into to `positions`—if `indices` is `[0, 1]`, this corresponds to a line segment between the first two points.

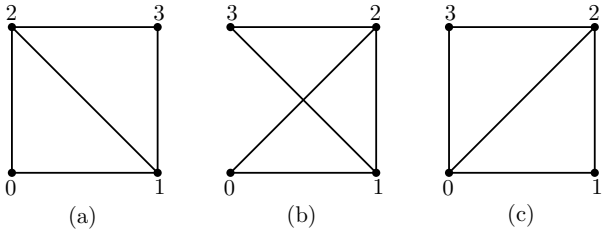
In order to display the curve, you will need to approximate it by subdividing it into line segments. The constant `N_SUBDIV_` corresponds to the number of line segments that you should use to render a single curve. Similar to how you display the tangent line, your `indices` can correspond to a polyline, e.g., `[0, 1, 1, 2, 2, 3, 3, 4]`.

You also need to implement toggling of spline bases and conversion of control point geometry. The keyboard triggers are implemented in the `Update` function.

The starter code declares some variables and functions that you may find useful to fill in. Note that you might want to update some of the function signatures to better suit the desired functionality.

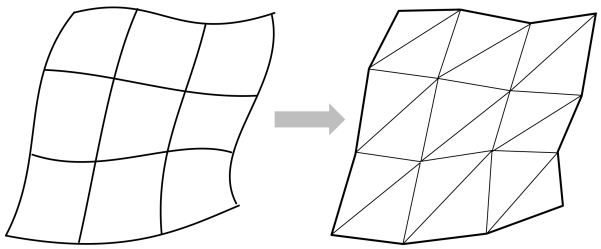
2.3 `PatchNode`

This node should plot a single tensor product patch. Similar to how you subdivided each curve into line segments, you'll want to tessellate your patch into a triangle mesh. The `indices` for a triangle mesh work similarly to those for a polyline, e.g., `[0, 1, 2, 2, 1, 3]` corresponds to two triangles, one with vertices at indices 0, 1, 2 of `positions`, and the other at indices 2, 1, 3.



It is important to get the order of the indices correct. `[0, 1, 2, 2, 1, 3]` corresponds to (a) above. If your vertices were ordered as in (b) or (c), this set of indices would yield overlapping triangles (b). Instead, you could do `[0, 1, 2, 2, 0, 3]`, which corresponds to (c).

Think carefully about how to create the tessellation for each patch. You will likely want to construct a triangle mesh like the one shown below. The patch mesh should be subdivided into `N_SUBDIV_` triangles along each parametric direction (in the example below, `N_SUBDIV_` $= 3$).



Note that the shader used in rendering the patch mesh requires a normal for each vertex in your mesh—you should compute this. You must compute the normals to the actual surface of the tensor product patch analytically; do not compute the normals to the flat triangles in your tessellation.

The mesh should be assembled in the `PlotPatch` function.

3 Summary of Requirements

This section summarizes the core requirements of this assignment. You will find more details regarding the implementation of these requirements later in this document.

For this and future assignments, you are free to start your implementation from scratch in C++. The starter code, however, implements a number of these requirements already, which **must** be present in your submission; missing features can cause your submission to receive no credit.

3.1 File Input

This is fully implemented in the starter code. Your program must read in a specific file format that allows users to specify curves ((poly)-Bézier and (poly)-B-Spline) and tensor product patch (Bézier and B-Spline) surfaces. Several such `.spline` files are provided in the `assets` directory. The specification of the file format is as follows.

The first line of the `.spline` file contains one of `Bezier curve`, `Bezier patch`, `B-Spline curve`, and `B-Spline patch`. Each subsequent line controls the coordinates for a single control point, i.e., `x y z`. The program interprets the control points depending on the type specified on the first line. Note that the "curves" can either be single curve segments or splines (piecewise Bézier or piecewise B-Spline). You can assume that the splines are cubic and that the patches are bicubic. In particular:

- A `Bezier curve` file with n curve segments has $3n + 1$ control points.
- A `B-Spline curve` file with n curve segments has $n + 3$ control points.
- A `Bezier patch` file with n patches has $16n$ control points.
- A `B-Spline patch` file with n patches has $16n$ control points.

For grading, we may use `.spline` files that are *not* included with the starter code.

3.2 User Interface

This is fully implemented in the starter code. Your program must provide functionality to rotate the model using mouse input, convert control points between Bézier and B-Spline geometry, and toggle betwen bases. If you choose to implement the assignment from scratch, play with the sample solution and support equivalent functionality.

3.3 Curves (55% of grade)

Your program must be able to generate and display piecewise cubic Bézier and B-Spline curves. In addition, for single curves you must be able to convert control points between Bézier and B-Spline bases and toggle which basis is used to compute the curve. Your program should indicate which basis is being used to display the curve at any given time, e.g., via the color of the control points. The bases that you should implement are precisely the ones covered in class lecture:

$$B_{\text{Bézier}} = \begin{pmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad B_{\text{B-Spline}} = \frac{1}{6} \begin{pmatrix} 1 & -3 & 3 & -1 \\ 4 & 0 & -6 & 3 \\ 1 & 3 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Recall that each segment of a piecewise B-Spline curve shares three control points with the previous segment, unlike piecewise Béziers, in which each segment shares only one control point with the previous. Thus, for the same number of control points the piecewise B-Spline will have a different number segments than the piecewise Bézier.

You should render each of the control points, and at least one tangent line for **each** cubic segment in the spline.

You will receive 25% if you implement one type of curve correctly, 15% for the other, and 7.5% each for converting the geometry and toggling the spline basis.

3.4 Surfaces (45% of grade)

Your program must be able to display bicubic Bézier and B-Spline tensor product patches. It should also compute the correct surface normals, which are required to render the patches. You do not any need any of the basis conversion functionality implemented for surfaces. Note that you only need to support Bézier × Bézier and B-Spline × B-Spline tensor products, similar to what was shown in class.

You will receive 20% for one type of surface, 15% for the other, and 10% for proper computation of normals.

3.5 Artifact (5% of grade)

Finally, you are to use your program to create at least two complex geometric models, which are substantially different from those distributed with the assignment, and submit them in `.spline` format. They can model real-life objects or they can be abstract. It is also fine if your models are based on a design made by someone else that you find online. Just make sure that the you submit them in a format compatible with your program, and mention where you found the original model in your `README.txt`. Note that at most one of the two models can be a spline curve—at least one must comprised of tensor product patches.

In addition to the two or more `.spline` files, you should submit screenshots of these shapes in either PNG or JPEG format.

4 Submission Instructions

You are to include a `README.txt` file or PDF report that answers the following questions:

- How do you compile and run your code? Specify which OS you have tested on.
- Did you collaborate with anyone in the class? If so, let us know who you talked to and what sort of help you gave or received.
- Were there any references (books, papers, websites, etc.) that you found particularly helpful for completing your assignment? Please provide a list. In particular, mention if you borrowed the model(s) used as your artifact from somewhere.
- Are there any known problems with your code? If so, please provide a list and, if possible, describe what you think the cause is and how you might fix them if you had more time or motivation. This is very important, as we're much more likely to assign partial credit if you help us understand what's going on.
- Did you do any of the extra credit? If so, let us know how to use the additional features. If there was a substantial amount of work involved, describe how you did it.
- Do you have any comments about this assignment that you'd like to share? We know this was a tough one, but did you learn a lot from it? Or was it overwhelming?

You should submit your entire project folder including your source code and the executable, but **excluding** `external/` since it takes too much space. Make sure your code is compilable so that we will be able to compile it from scratch. You should also exclude auxiliary build files that are too big (e.g. the `ipch` folder if you are using Visual Studio).

To sum up, your submission should be your entire project folder containing:

- The `README.txt` file or PDF report answering the questions above. Leave this file at the root directory of the project folder.
- At least two `.spline` files that define your new artifacts. Please name these `artifact1.spline`, `artifact2.spline`, and so on, and place them in the `assets/assignment1` subdirectory.
- Images of your artifacts in PNG or JPEG format. You can either embed them in your PDF report or leave them at the root directory of the project folder.

Please compress your project folder into a `.zip` file and submit using Canvas.
We will follow the late day policy explained in Lecture 1.

5 Extra Credit

As with the previous assignment, the extra credits for this assignment will also be ranked *easy*, *medium*, and hard. However, these are all relative to the individual assignment; an easy extra credit for this assignment will probably be harder than an easy one from the last assignment (and we will assign credit accordingly). Furthermore, these categorizations are only meant as a rough guideline for how much they'll be worth. The actual value will depend on the quality of your implementation—a poorly-implemented medium may be worth less than a well-implemented easy. We will make sure that you get the credit that you deserve.

If you do any of these extra credits, please make sure that your program does not lose any of the required functionality described above.

5.1 Easy

- (3%) Render your model more interestingly. Change the materials, or even better, change the material as some function of the surface. You might, for instance, have the color depend on the sharpness of the curvature. Or jump ahead of what we are covering in class and read about texture mapping, bump mapping, and displacement mapping.
- (3%) Implement another type of spline curve, and extend the `.spline` format and parser to accommodate it. We recommend Catmull-Rom splines, as they are C^1-continuous and interpolate all the control points, but feel free to try others (Bessel-Overhauser, Tension-Continuity-Bias). If you implement this extension, please make sure that your code can still read standard `.spline` files, since that's how we're going to grade it. Additionally, provide some `.spline` files that demonstrate that they work as they're supposed to.
- (3%) Implement another kind of primitive curve that is not a spline. Preferably, pick something that'll result in interesting surfaces. For instance, you might try a corkscrew shape or a Trefoil Knot[3]. Again, you'll probably want to extend the `.spline` format to support these sorts of primitives.
- (3%) Implement the ability to export your model in OBJ format[4]. Files in this format can be imported into a variety of 3D modeling and rendering packages, such as MeshLab[5] or Maya[6]. This software has been used for a number of video games, feature films, and television shows/

5.2 Medium

- (6%) Implement a user interface so that it is easy for users to specify curves using mouse control. A user should be able to interactively add, remove, and move control points to curves, and those curves should be displayed interactively.
- (6%) In this assignment, the suggested strategy for discretizing splines involves making uniform steps along the parametric curve. However, it is difficult to choose the appropriate number of steps, since the curvature of splines can vary quite a bit. In fact, any choice of a uniform step size will result in either too few steps at the sharpest turns, or too many in the straight regions. To remedy this situation, implement a recursive subdivision technique to adaptively control the discretization so that more samples are taken when needed.

5.3 Hard

- (9%) The techniques covered in this assignment are just one of many ways that modelers create shapes. Another popular technique is that of subdivision surfaces. At a high level, you can think of this technique as starting with a polygon mesh and defining the desired surface as the limit of a set of subdivision rules that are applied to the mesh. Implement this technique.

[3] `https://mathworld.wolfram.com/TrefoilKnot.html`
[4] `https://en.wikipedia.org/wiki/Wavefront_.obj_file`
[5] `http://meshlab.net`
[6] `https://www.autodesk.com/products/maya/overview`