CIS 462/CIS562 Computer Animation (Fall 2015)
Homework Assignment 2

(Curve Editor)

Due: Tuesday, Sept. 29, 2015 (must be submitted to Canvas before midnight)

Please note the following:

- No late submissions or extensions for this homework.
- This is a programming assignment. You will need to download the *AnimationToolkit* code framework from the CIS462/562 Canvas site (*https://canvas.upenn.edu/courses*).
- Use MS Visual Studio 2013 (available on CETS computers; or download the software from Dreamspark):
  - o http://e5.onthehub.com/WebStore/ProductsByMajorVersionList.aspx?ws=24207847-db9b-e011-969d-0030487d8897&vsro=8).
- Unzip the AnimationToolkit.rar file and open the solution *AnimationToolkit.sln* in Visual Studio 2013 to see the associated projects and files.
- Double click on the file "A1-CurveViewer-Demo.exe" in the AnimationToolkit\bin directory to see a demo of the Curve Editor functionality
- If it is not already there, upload the *AnimationToolkit* folder to the GitHub CIS462-562 repository that has been set up for you.
- When you are finished with this assignment, update your GitHub project files. Also, zip your code under the *AnimationToolkit* directory along with a *Readme.txt* file and name it *hw02_name.zip*, then upload your solution to Canvas before the deadline. *Do not* include the directories *Debug* or *Release* in your zip file - we will recompile your project from the source files.
- Work within the *AnimationToolkit* code framework provided. Feel free to enhance the GUI interface if you desire.
- You only need to implement the functions in the aSplineVec3.cpp file marked with "TODO" to complete this assignment.
- **NOTE: THIS IS AN INDIVIDUAL, NOT A GROUP ASSIGNMENT. That means all code written by you for this assignment should be original! Although you are permitted to consult with each other while working on this assignment, code that is substantially the same as that submitted by another student will be considered cheating.**

ANIMATION TOOLKIT CURVE EDITOR

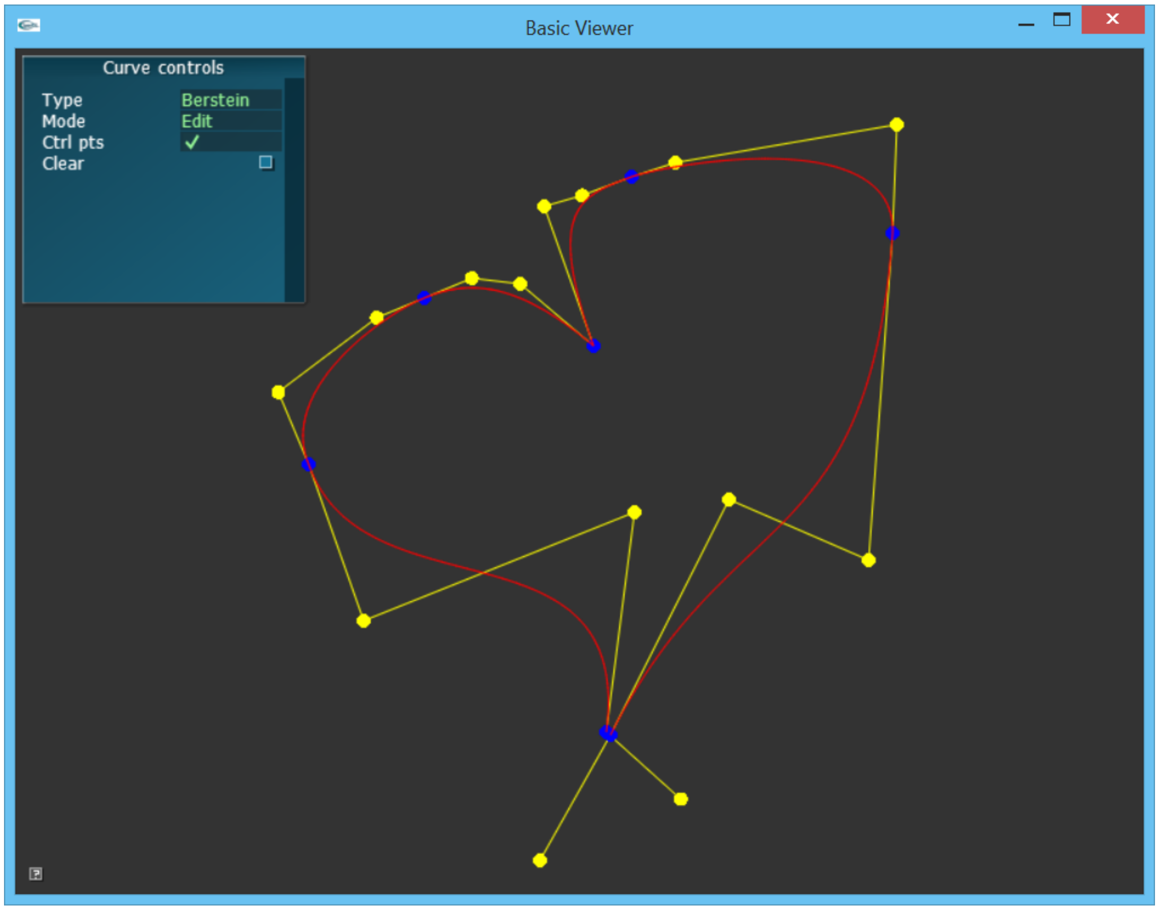
In this assignment you will develop an interactive 2D curve editor by implementing various functions in the Animation Toolkit. The base code you receive for the Animation Toolkit will be used throughout the semester, with each subsequent programming assignment adding new features and functionality to the toolkit libraries. The Animation Toolkit base code includes the following directories:

- /3rdparty - this large directory contains additional toolkits to support UI, images, FBX, and more
- /bin - executables, both solutions and the applications you create will be here
- /include - header files, these are copied from the libsrc directory. Applications look here for headers
- /lib - library files, both solution libraries and the libraries you implement are here, Applications look here for libraries
- /libsrc - library source, the basecode you will extend is here
- /models - geometry assets for each assignment
- /motions - additional motion files in BVH format

The top level AnimationToolkit.sln file should contain the relevant project files. Note: because the assignments are new for this semester, the Animation Toolkit base code may need to be updated occasionally during the semester.

User Interface Overview

The Animation Toolkit includes a simple orthographic view which will allow you to test the Curve Editor code being developed in this assignment. A sample screenshot is shown below.



Type Berstein
Mode Edit
Ctrl pts ✓
Clear □

Using the "Curve Controls" panel on the top left, you can create and edit different types of spline curves. The blue circles represent the input data points (also called "keys") which are to be interpolated using various curve types. The yellow circles represent the spline control points which are computed based on the curve type.

- The **Type** drop-down menu specifies how the input data points are to be interpolated. The choices which you will implement include: Linear, Bernstein, Castlejau, Matrix, Hermite, and B-Splines.
- The **Mode** drop-down menu specifies how left mouse clicks are interpreted:
    - **Add** will append points to the spline curve when and where you click the left mouse button;
    - **Edit** will allow you to drag existing key points (blue) and control points (yellow) using the left mouse button;
    - **Delete** will remove key curve points (blue) when you select one with the left mouse button.
- The **Ctrl pts** button toggles the display of control points (yellow) on/off.
- The **Clear** button deletes and removes the entire spline curve.

Spline Curve Overview

The splines you implement in this assignment will be the foundation for animating objects, characters, and crowds in the assignments that follow in the course. In this assignment, keys are the locations in space that determine the shape of the spline curve. However, in future assignments, the spline curves will be used to specify changes in the state of an object over time. For example, a key could be the position of a character or the color of a particle at a particular point in time. This assignment asks you to implement spline curves that can interpolate 3D vector (i.e. vec3) key points. In the next assignment, you will implement similar splines for interpolating rotations using quaternions.

In the implementation that follows, spline curves (also referred to as "splines" or "animation curves") are constructed using "keyframes", where each keyframe is a time/value pair. In this assignment, we will assume that the keyframe data points are uniformly spaced in time (e.g. the first key is at time 0.0, the second key is at time 1.0, etc.). The "framerate" of an animation spline curve determines the number of samples between each key. For example, if the framerate is 10 frames per sec (fps), the corresponding timestep is be 1.0/10 = 0.1, which will result in 10 curve samples being computed between each key.

In this assignment you will need to implement two core functions for each curve type. First, you will need to compute the control points based on the spline curve's keys. Second, you will need to implement the associated curve interpolation function to compute the intermediate curve values between each key. In the curve editor, whenever the user adds, deletes or edits a key, the control points and interpolating curve values are recalculated using calls to the following functions: *`ASplineVec3::computeControlPoints()`* and *`ASplineVec3::cacheCurve()`*. These functions then call the current spline algorithm based on the curve type selected, which are implemented as subclasses of the class `AInterpolatorVec3`. Below is the class hierarchy of the interpolators you will implement:

```
AInterpolatorVec3
    ALinearInterpolatorVec3
    ACubicInterpolatorVec3
       ABernsteinInterpolatorVec3
       ACasteljauInterpolatorVec3
       AMatrixInterpolatorVec3
       AHermiteInterpolatorVec3
       ABSplineInterpolatorVec3
```

Given values for the control points and keys of the spline, values of the spline curve at each point in time are computed by going through curve segment (defined by a pair of keys) then calculating intermediate values by interpolating between the keys (which is implemented in the `AInterpolatorVec3::interpolate()` function).

The pseudocode for this looks like:

```
clearSplineCurve();
for each curve segment
    key1 = segment.start
    key2 = segment.end
    time = key1.time
    while time < key2.time
        u = fraction of time between key1.time and key2.time
        time += timeStep
        value = interpolate(data, controlPoints, segment, u)
        splinecurve.append(value);
splinecurve.append(lastKey);
```

Assignment Details

1. *(5 points)* **Linear Splines.** To get warmed up, you will implement a piece-wise linear spline. Given the keyframe data points $\mathbf{p}_i = [x_i, y_i, z_i]^T$ (i = 0 to m) input by the user, complete the implementations for:

   1. The general base class interpolation function in: `AInterpolatorVec3::interpolate()`
   2. The specifics for the linear interpolation function in: `ALinearInterpolatorVec3::interpolateSegment()`

2. *(50 points)* **Cubic Catmul-Rom Splines.** Given keyframe data points of the form $\mathbf{p}_i = [x_i, y_i, z_i]^T$ (i = 0 to m), construct a cubic Catmul-Rom spline comprised of Bezier curve segments that interpolates the $\mathbf{p}_i$ data points.

   1. (20 points) **Compute the Catmul-Rom spline control points.** This is accomplished by implementing the function: `ACubicInterpolatorVec3::computeControlPoints()`

You also will need to implement the three equivalent Bezier curve computation methods.

   2. (10 points) **Bernstein.**
      Implement `ABernsteinInterpolatorVec3::interpolateSegment()`
   3. (10 points) **De Casteljau.**
      Implement `ACasteljauInterpolatorVec3::interpolateSegment()`
   4. (10 points) **Matrix**.
      Implement `AMatrixInterpolatorVec3::interpolateSegment()`

3. *(20 points)* **Hermite Spline**. Implement a cubic Hermite spline having C^2 continuity which supports either clamped or natural end point conditions. This can be accomplished by implementing the following functions:

   `AHermiteInterpolatorVec3::computeControlPoints()`

   `AHermiteInterpolatorVec3::interpolateSegment()`

4. *(25 points)* **Cubic B-Spline**. Implement a cubic spline curve with natural end point conditions using a B-Spline formulation. This can be accomplished by implementing the following functions:

   `ABSplineInterpolatorVec3::computeControlPoints()`

   `ABSplineInterpolatorVec3::interpolateSegment()`

Feel free to add helper functions to the `ABSplineInterpolatorVec3` class to make the implementation easier.

5. *(10 points)* **Extra credit**. Implement an additional interpolation type as a subclasses of AInterpolatorVec3. For example a staircase interpolator (i.e. degree = 0), a Catmul-Rom spline comprised of quantic (i.e. degree = 5) Bezier curves or something of your own choosing.