In this assignment you are to implement 2D Bézier curves in an OpenGL program. You are provided with C++ source code for starting a simple OpenGL program and moving some points around on the screen. You are also provided with a basic vector class to get you started faster. Finally, a reference implementation is available. You should refer to that reference implementation for clarification of any ambiguities in this assignment specification.

You will implement Bézier curve evaluation, the subdivision algorithm, and piecewise Bézier curves.

We have provided a simple user interface for manipulating curves and changing modes. You should be able to create and move control points by clicking and dragging with the mouse. This functionality is provided by the PointCanvas class.

Right-clicking brings up a menu to control the modes of the program. There are three options under the “Mode” submenu: “Basic”, “Subdivision”, and “Piecewise”. These correspond to the three components of the assignment you will need to implement.

- In Basic mode, a Bézier curve is drawn using all the control points on the canvas.
- In Subdivision mode, the Bézier curve is approximated using the subdivision algorithm. The < and > keys can be used to increase and decrease the subdivision level.
- In Piecewise mode, the control points are interpolated/approximated using piecewise Bézier curves of a chosen degree. The number keys 1-9 can be used to choose the degree of the piecewise curves.

In Piecewise mode, the continuity of the curves can be set in the “Continuity” menu. We have provided menu options for C0 and C1 continuity, although only C0 continuity is mandatory for the assignment.

Finally, there are menu buttons to clear all points from the canvas, and to exit the application.

Assignment 3.1: Bézier Curve Evaluation (5 Points)

Implement simple evaluation and rendering of Bézier curves. You are provided with an incomplete BezierCurve class. Implement BezierCurve::evaluate() to return the point along the Bézier curve at the given t-value. We have provided a function BezierCurve::binomialCoefficient() to compute \( \binom{m}{i} \). You may also find it useful to use std::pow() to compute exponential expressions. Note that these computations are prone to overflow in the presence of large numbers. As such, we do not expect that your code be robust for Bézier curves of arbitrarily high degrees.
Next, implement `BezierCurve::draw()`. You should draw the curve by evaluating it at uniformly-spaced t-values, and then drawing lines between those points. We have provided a function `BezierCurve::drawLine()` to do the line-drawing for you. It is up to you to choose how many different points to evaluate in order to produce a reasonable-looking curve.

**Assignment 3.2: Subdivision (5 Points)**

Implement `BezierCurve::subdivide()`. Follow the subdivision algorithm described in class to produce two `BezierCurve` objects that have control polygons representing the two halves of the initial `BezierCurve`. For the purpose of this assignment, subdivision is always at the center of the parameter interval.

Next, implement the section of `CurveManager::drawCurves()` labelled “Subdivision Mode”. You should start with a `BezierCurve` composed of the full set of points. Then subdivide the curve repeatedly until the number of subdivisions is equal to the `subdivisionLevel` variable. Once all subdivisions have been performed, draw the control polygons of the resulting curves using `BezierCurve::drawControlPolygon()`. The control polygons should converge to the true Bézier curve.

You should be able to subdivide the control polygon any number of times, like the sample implementation. **There should be no hard limit on the number of subdivision steps that can be performed.**

**Assignment 3.3: Piecewise Curves (5 Points)**

Implement the section of `CurveManager::drawCurves()` labelled “Piecewise Mode”. Create $\lceil (N - 1)/d \rceil$ `BezierCurves` where $N$ is the total number of control points and $d$ is the degree of each piecewise curve. Use the `piecewiseDegree` variable to determine what the degree of the Bézier curves will be (the final curve may need to have lower degree than the others). The curves must have at least C0 continuity. You may earn bonus points by implementing C1 continuity (see below). Note that this will likely require adding additional control points in the code to ensure that the C1 condition is enforced.

**Assignment 3.4: Extensions (2 Points)**

You can get up to two bonus points for this assignment (i.e. the maximum grade is 17 out of 15!). Extra points will carry over so you will be able to use these points for compensating for points you loose on midterm, final, or other homework. Examples for what you can do are:

- Smooth transitions between multiple Bézier curves, i.e., C1 continuity.
- Animation/Implementation of the De Casteljau evaluation algorithm (this will require tweaking the UI accordingly).
- Whatever else you can think of.

Bonus points are given at the discretion of the TAs.
Getting Started

You should first download the provided template and unpack it (use `tar -xzf` or `unzip` depending on whether the extension is `.tar.gz` or `.zip`, respectively). The sample implementation is available under the names `a3_example` and `a3_example.exe` for Windows and Linux, respectively. Run it to get a feeling for what you are about to be doing.

After you have run the sample implementation, compile the provided source code. For Windows we have provided Visual Studio 2017 files and the required glut library files. Simply open `a3.sln` using Visual Studio and build the code. For Linux use the Makefile by running `make`, which will compile your code. Note that if you are not using the CS Linux machines you need to install the GL, GLU, and freeglut libraries using your package manager.

The initial version of the code contains all user interface components and point manipulation mechanics. It does not draw any curves until you begin implementing the assignment.

You should not need to modify any files except for `bezier.cpp`. You are welcome to modify anything you like while implementing your extensions (though you must preserve all functionality from the mandatory components of the assignment if you want to receive full marks).

If you create new files, make sure you update the Makefile (or the Visual Studio files if you are using Windows) appropriately. Points will be taken off for solutions that require tinkering to get them compiled.

Submission

In addition to your source code, you should also have a README file that details how far you have gotten in your implementation. You should also mention any extensions you may have implemented.

Once you have all your files together, submit them with the `handin` command that you should know from other courses:

```
handin cs424 A3
```

Note that the `handin` command will ignore object files and the executable.