Assignment 5: Glyph Editor

CS 148 Autumn 2011-2012

Due Date: Monday, October 29th, 2012 at 7pm

Introduction: The goal of this assignment is to design and build a glyph editor that will allow you to create your own piece of digital typography, implementing basic Bézier spline drawing and editing operations along the way.

Background: A scalable digital font is made up of a collection of *glyphs*, one or more tables describing a mapping of characters to these glyphs, and a set of metrics describing various font properties. You can think of a glyph as the shape of the letter you see on the page or on-screen. A glyph consists of a number of *contours* or *outlines*, typically described as closed Bézier paths. Each Bézier path is then defined by a sequence of control points on the Cartesian plane.

A font editor is a software program that allows a font designer to create and edit glyphs for digital typefaces. It provides the ability to design glyphs for each character in the font, usually by allowing the user to define and edit control points that specify the glyph outlines.

Fig. 1 shows two screen shots of the open source font editor FontForge in action, editing the glyph for the letter 'C'. You may wish to play with FontForge to get a feel for how a glyph editor might work in practice.

Your primary task in this assignment is to replicate a basic subset of glyph editing capabilities you might need to implement parts of a font editing program. To be specific, you will be implementing a TrueType glyph editor, which allows editing glyphs one at a time.

TrueType: Fig. 2 are examples of glyphs for the letters 'B' and 'C', showing their outlines and control points (from the Apple TrueType Reference Manual).

A TrueType glyph contour is made up of both straight line segments and quadratic Bézier curves. It can be represented as a list of control points, with each point explicitly marked as either lying on the curve, or off-curve. Any combination of on and off points is acceptable when defining a curve. The path is always closed, meaning that the last point is implicitly connected to the first.

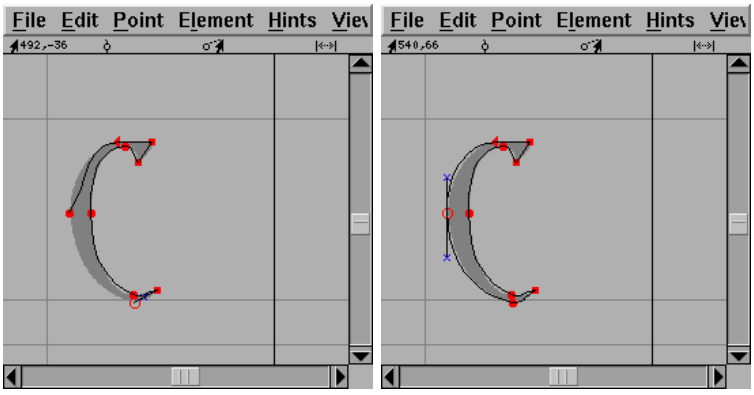

Figure 1: FontForge open source font editor

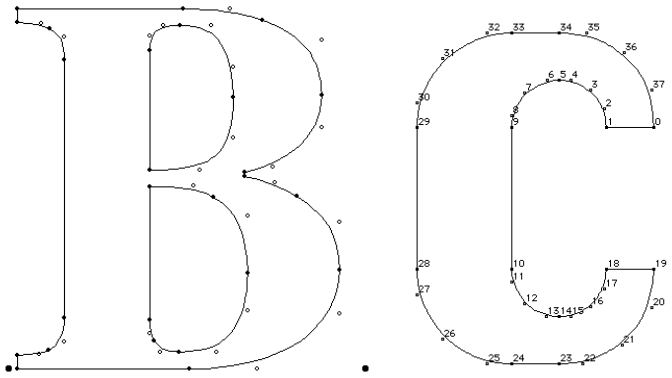

Figure 2: Glyphs for the letters 'B' and 'C' showing their outlines and control points

For example, take a look at the letter 'B' above. The black, filled in dots correspond to control points on the curve, whereas the dots which aren't filled in correspond to control points not on the curve. Note how the letter can be represented by a sequence of these dots. Sometimes you have two on-curve points in a row, and sometimes the points alternate between on-curve and off-curve.

More Information on TrueType: You may also wish to look at some of the resources available to font and font tool developers for more details, or as a reference on how TrueType glyphs are defined. Here are two comprehensive online resources.

1. Apple TrueType Reference Manual
2. Microsoft Typography

Steps: There is a lot to do in this assignment, so we'll do our best to break it down into a nice, simple checklist for you here. Read through and make sure you understand the assignment questions – They will help you plan out your work.

Step 0: Answer Assignment Questions

1. What is the difference between a quadratic Bézier curve and a cubic Bézier curve? Which curves do TrueType fonts use? Which curves do Adobe Type 1 fonts use? What about OpenType fonts?
2. Glyphs for certain roman characters are special in that they are hollow inside. How does one use Bézier paths to describe such glyphs while ensuring that the inside remains unfilled, as must be done for the letter 'o' for example?
3. When we use several Bézier curve segments to describe a curved part of a glyph outline, we usually want to ensure a certain degree of smoothness so that no "kinks" show up on the curve. For example, two adjoining quadratic Bézier curve segments can be specified using 5 control points (P0 ... P4) with alternating on/off curve flags (ON-OFF-ON-OFF-ON). In this example,
   - What condition must hold for the full curve to have C1 (first derivative) continuity?
   - Suppose we move one of the off-curve control points. What must happen to the other points to preserve this condition?
   - Suppose we move the middle on-curve control point (P2). What (if anything) should happen here to preserve continuity?
4. Would the use of cubic Béziers provide advantages over the quadratic Béziers we're using for the purposes of editing outlines? If so, what are these advantages?

Step 1: Download and Build the Starter Code

1. Download the starter project for your platform/environment (below), and ensure that it compiles.

Step 2: Support Glyph Creation In the previous assignment, you have already implemented a polygon-editing UI. In this assignment, you will additionally implement the functionality required to create a TrueType glyph from scratch.

1. Design a way for a user of your glyph editor to interactively specify the control points that make up a TrueType glyph, whether they are "on-curve" or "off-curve". On-curve control points should look different than off-curve points when displayed to the user.

Step 3: Loading and Saving a Glyph

1. Write code that allows a user to load a single glyph from a file on disk, create a new glyph, or save a single TrueType glyph to disk. You can make up your own file format for this step, but you should support loading and saving of a glyph that is made up of more than one contour/outline.

Step 4: Support Glyph Rendering

1. If you recall, a single TrueType glyph outline is made up of a sequence of on-curve and off-curve control points. In order to draw an outline, it can be decomposed into a series of straight line segments and quadratic Bézier curves that can then be rendered on-screen. Make sure you understand how this works and how Bézier curves work.
2. Write the code to draw TrueType glyph outlines from the on-curve and off-curve control points the user has already specified.

Step 5: Allow Bézier Curve Editing

1. Given an existing TrueType contour/outline, allow the user to interactively add, move, or delete control points with the mouse.
2. Users often want to control tangency at an on-curve control point, as this determines the shape of the curve. Implement a method to edit the "tangent curve" at these on-curve control points, making sure that the larger TrueType contour/outline remains valid. (This means that the change should be local. It is permissible to move nearby points to maintain tangent continuity, but it should not alter tangents at other on-curve points.)

Starter Code: When you get the starter project running, you should see a window that looks like Fig. 3.

Commands A number of guides (as labeled in Fig 3) will be displayed on top of the background "template" image with writing on it. Some basic navigation controls have already been implemented to help you work with the template.

- Right-click and drag to pan the image
- Hold down the middle mouse button and move up/down to zoom in/out
- '+' and '-' keys for coarse zoom in/out
- '[' and ']' keys for finer zoom in/out

The idea is to position each letter in turn to fit within the guides, so that you can create an outline for a character over the template. Play around a bit to get a feel for the controls.

You may notice the starter project will already have lots and lots of code, but do not let that worry you! However, there are a few classes and data structures that could be useful.

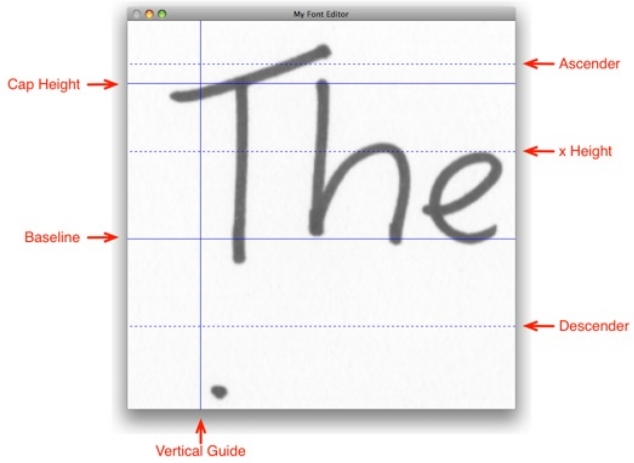

Figure 3: Initial window for starter project

Source Files

- `main.cpp`: You will need to edit this file to complete your assignment.
- `TemplateLayer.h/TemplateLayer.cpp`: Holds and draws the background image and guide lines. You probably don't have to touch this class.

Data Structures We provide you sample contour data structures, but you do not need to use them. Here is some example code that will create and add a simple glyph to a TrueType font object:

```
TTContour *contour = new TTContour();
contour->AddPoint(new TTPoint(10, 0, true));
contour->AddPoint(new TTPoint(10, 250, true));
contour->AddPoint(new TTPoint(20, 250, true));
contour->AddPoint(new TTPoint(20, 0, true));
TTGlyph *glyph = new TTGlyph();
glyph->AddContour(contour);
gMyFont.SetGlyphForCharacter('|', glyph);
```

The last question that remains is where you might put all the code you're going to write for this assignment. That is an architectural decision that we'll leave up to you. Writing a new class for the editing functionality may prove to be the cleanest solution, but putting the bulk of your code in `main.cpp` is also viable.

Requirements: Your glyph editor should be able to

1. Define both straight line segments and Bézier curve segments on a TrueType glyph outline.
2. Correctly draw Bézier paths as outlines for a TrueType font glyph.
3. Allow the user to add, remove, and change the position of control points using the mouse.
4. Allow the user to edit tangents at on-curve points, while preserving continuity (and smoothness) of the curve.
5. Load and save a single TrueType glyph.

You must use your editor to draw the glyph outlines for a single "interesting" character, such as the letters A, B, Q, &, or perhaps something more complex such as 天 or 水. It should have multiple contours and a mix of on- and off-curve control points.

Your glyph editor should preserve C1 (first derivative or tangential) continuity of the curve through the point while editing. We realize that with certain manipulations of the control points it may not be possible

to preserve continuity through all parts of the entire contour. Any reasonable effort to preserve continuity through at least the adjacent segments will earn you full points.

Grading:

- 1/4 – Correct answers to assignment questions
- 2/4 – Correct answers to assignment questions and the program has ability to edit control points and draw curves
- 3/4 – Correct answers to assignment questions, editor correctly represents all curve types (lines, Béziers), and provides means of adding/moving/deleting them to specify font glyphs
- 4/4 – Correct answers to assignment questions, editor has all capabilities above, and allows editing curve tangents while preserving (tangential) continuity

In the world of fonts and typography, we've only just hit the tip of the iceberg. There are a number of things you can do now to really make your simple glyphs into a first class font. If you'd like, feel free to allow for editing multiple glyphs and exporting them into a TrueType "TTF" font file.

Some examples are adding kerning, ligatures, hinting, and bold or italic variants. Those of you who write other languages may wish to explore compound glyphs for adding characters with accents, or even attempt a non-Roman font. Be aware that most of these extra tasks can be extremely challenging, as you'll need to augment (or write your own) support code to export these features in the font file.

There are many things you can do to streamline the writing digitization process. An automatic method of snapping the contours to the image, or a method of correctly filling the glyphs so that you see more than just outlines would be neat.

Showcase any of these or other extras you pursue to your grader during the grading session. We may showcase anything that, to our best discretion, is especially awesome.

Tips, Tricks, & FAQs

- What is all this font code? `SetGlyphForCharacter`? In this assignment, you do not need to use all the extra code provided. However, if you would like to go beyond the call of duty there is enough code given to support assigning glyphs to alphanumeric characters and exporting to TrueType fonts. You could, for example, use this to digitize your own handwriting into a personal TrueType font!
- How should I render the quadratic Bézier curve? We are not mandating any particular sampling or rasterization scheme, but encourage you to choose an efficient yet accurate method. See the lecture notes for more information.
- How do I convert a sequence of on- and off- control points into a list of straight line segments and quadratic Bézier curves, for rendering? Please read through the TrueType documentation links we have given you. The basic idea is that you may need to have "implicit" control points for a subsequence of off- control points. More specifically, if consecutive off-curve points occur in the path definition, the TrueType engine implicitly adds an on-curve point at the midpoint between any two consecutive off-curve points, so that the entire path can be drawn using straight line and quadratic Bézier segments. It may be good to note this little representation detail when creating your editor. More details can be found on Apple's excellent article on digitizing letterforms. Note that the editor should still support a subsequence of off- control points–the implicit on- control point is just for the purposes of rendering.
- What does editing the tangent at a point mean? A tangent line is defined at any points where the underlying function is differentiable, or sufficiently smooth. The common case that we'd like you to support is to editing of this tangent line when you know that the curve is smooth at that point. In other words, if your "tangent editing" mode works by allowing users to move off-curve control points, then you should preserve tangency at any adjacent on-curve control points. That is, you should not allow users to introduce kinks where they do not exist already in this mode. For this assignment, you are not strictly required to add off-curve control points to maintain tangency, for example in the case of a straight line segment followed smoothly by a curve.