Modeling Curves in Vector Fonts

Chen Yihang
Peking University
1700010780

***Abstract*—In this report, we investigate the method to model curves in computational graphics, especially in vector fonts. We mainly consider the Bezier curve and its generalized version, NURBS.**

I. Introduction

A computer font (or font) is implemented as a digital data file containing a set of graphically related glyphs, characters, or symbols such as dingbats. There are three basic kinds of computer fonts

- **Bitmap fonts** consist of a matrix of dots or pixels representing the image of each glyph in each face and size, which is faster and easier to use, yet is not scalable.
- **Vector fonts** use Bézier curves, drawing instructions and mathematical formulae to describe each glyph, which make the character outlines scalable to any size. Vector fonts are the most popular choice in computers nowadays.
- **Stroke fonts** use a series of specified lines and additional information to define the profile, or size and shape of the line in a specific face, which together describe the appearance of the glyph.

The contribution of Wang Xuan mainly lies in reducing the storage and facilitate the output of a character, which is rather a technical than a mathematical work. Besides, due to the significant improvement in the hardware, despite its prominent historical meaning, such improvement is not vital in the present. In this report, we will focus on the developments and underlying mathematical insights in the vector fonts.

II. Developments of vector fonts

Vector fonts are collections of vector images, consisting of lines and curves defining the boundary of glyphs. Early vector fonts were used by vector monitors and vector plotters using their own internal fonts, usually with thin single strokes instead of thick outlined glyphs. The advent of desktop publishing brought the need for a universal standard to integrate the graphical user interface of the first Macintosh and laser printers. The term to describe the integration technology was WYSIWYG (What You See Is What You Get). The universal standard was (and still is) Adobe PostScript. Examples are PostScript Type 1 and Type 3 fonts, TrueType and OpenType.

The primary advantage of vector fonts is that, unlike bitmap fonts, they are a set of lines and curves instead of pixels; they can be scaled without causing pixellation. Therefore, vector font characters can be scaled to any size and otherwise transformed with more attractive results than bitmap fonts, but require considerably more processing and may yield undesirable rendering, depending on the font, rendering software, and output size. Even so, vector fonts can be transformed into bitmap fonts beforehand if necessary. The converse transformation is considerably harder, since bitmap fonts requires heuristic algorithm to guess and approximate the corresponding curves if the pixels do not make a straight line.

Vector fonts have a major problem, in that the Bézier curves used by them cannot be rendered accurately onto a raster display (such as most computer monitors and printers), and their rendering can change shape depending on the desired size and position. Measures such as font hinting have to be used to reduce the visual impact of this problem, which require sophisticated software that is difficult to implement correctly. Many modern desktop computer systems include software to do this, but they use considerably more processing power than bitmap fonts, and there can be minor rendering defects, particularly at small font sizes. Despite this, they are frequently used because people often consider the processing time and defects to be acceptable when compared to the ability to scale fonts freely.

We give a brief summary of the three popular commercial type fonts.[1]

a) Type 1 and Type 3 fonts: Type 1 and Type 3 fonts were developed by Adobe for professional digital typesetting. Using PostScript, the glyphs are outline fonts described with cubic Bezier curves. Type 1 fonts were restricted to a subset of the PostScript language, and used Adobe's hinting system, which used to be very expensive. Type 3 allowed unrestricted use of the PostScript language, but didn't include any hint information, which could lead to visible rendering artifacts on low-resolution devices (such as computer screens and dot-matrix printers).

b) Truetype fonts: TrueType is a font system originally developed by Apple Inc. It was intended to replace Type 1 fonts, which many felt were too expensive. Un-

[1]https://en.wikipedia.org/wiki/Computer_font

like Type 1 fonts, TrueType glyphs are described with quadratic Bezier curves. It is currently very popular and implementations exist for all major operating systems.

c) Opentype fonts: OpenType is a smartfont system designed by Adobe and Microsoft. OpenType fonts contain outlines in either the TrueType or Type 1 (actually CFF) format together with a wide range of metadata.

III. Bezier curve

In the vector fonts, Bezier curve is adopted to modeling the curve of the characters. TrueType fonts use composite Bézier curves composed of quadratic Bézier curves. Other languages and imaging tools (such as PostScript, Asymptote, Metafont, and SVG) use composite Béziers composed of cubic Bézier curves for drawing curved shapes. OpenType fonts can use either kind, depending on the flavor of the font

A. General formulation

A Bézier curve is a parametric curve used in computer graphics and related fields. The curve, which is related to the Bernstein polynomial, is named after Pierre Bézier, who used it in the 1960s for designing curves for the bodywork of Renault cars. Other uses include the design of computer fonts and animation. Bézier curves can be combined to form a Bézier spline, or generalized to higher dimensions to form Bézier surfaces.

A recursive definition for the Bézier curve of degree n expresses it as a point-to-point linear combination of a pair of corresponding points in two Bézier curves of degree $n-1$.

Let $\mathbf{B}_{\mathbf{P}_0,\mathbf{P}_1,\cdots,\mathbf{P}_n}$ denote the Bezier curve determined by the points $\mathbf{P}_0,\mathbf{P}_1,\cdots,\mathbf{P}_n$, then to start

$$\begin{aligned}&\mathbf{B}_{\mathbf{P}_0}(t)=\mathbf{P}_0,\quad \text{and}\\ &\mathbf{B}_{\mathbf{P}_0,\mathbf{P}_1,\cdots,\mathbf{P}_n}(t)=(1-t)\mathbf{B}_{\mathbf{P}_0,\mathbf{P}_1,\cdots,\mathbf{P}_{n-1}}(t)+t\mathbf{B}_{\mathbf{P}_1,\cdots,\mathbf{P}_n}(t)\end{aligned} \tag{1}$$

From the definition, the formulae can be explicitly expressed as

$$\mathbf{B}(t)=\sum_{i=0}^{n}\binom{n}{i}(1-t)^{n-i}t^i\mathbf{P}_i:=\sum_{i=0}^{n}b_{i,n}(t)\mathbf{P}_i \tag{2}$$

where $b_{i,n}(t)$ is called Bernstein basis polynomials of degree n.

We can also write the curve into polynomial form

$$\begin{aligned}&\mathbf{B}(t)=\sum_{j=0}^{n}t^jC_j,\quad \text{where}\\ &C_j=\frac{n!}{(n-j)!}\sum_{i=0}^{j}\frac{(-1)^{i+j}\mathbf{P}_i}{i!(j-i)!}\end{aligned} \tag{3}$$

If we assign weight w_i onto point $\mathbf{P}_i$, we get the rational Bezier curve

$$\mathbf{B}(t)=\frac{\sum_{i=0}^{n}b_{i,n}(t)\mathbf{P}_iw_i}{\sum_{i=0}^{n}b_{i,n}(t)w_i} \tag{4}$$

By adjusting weight, (4) can provide closer approximation.

B. De Casteljau's algorithm and splitting

The algorithm utilize recursive relation (1) to calculate Bernstein basis polynomial (2), which is slower than the direct polynomial computation (3), yet more stable.

Define

$$\begin{aligned}\beta_i^{(0)}&:=\mathbf{P}_i\\ \beta_i^{(j)}&:=\beta_i^{(j-1)}(1-t_0)+\beta_{i+1}^{(j-1)}\end{aligned}$$

we can have

$$\mathbf{B}(t_0)=\beta_0^{(n)}$$

Besides, the curve $\mathbf{B}(t)$ can be split into two curves at t_0

$$\mathbf{B}_1(t)=\sum_{i=0}^{n}\beta_0^{(i)}b_{i,n}\left(\frac{t}{t_0}\right),\quad t\in[0,t_0]$$

and

$$\mathbf{B}_2(t)=\sum_{i=0}^{n}\beta_i^{(n-i)}b_{i,n}\left(\frac{t-t_0}{1-t_0}\right),\quad t\in[t_0,1]$$

which amounts to two Bezier curve with respect to two group of control points.

C. Degree elevation

A Bézier curve of degree n can be converted into a Bézier curve of degree $n+1$ with the same shape. This is useful if software supports Bézier curves only of specific degree. For example, systems that can only work with cubic Bézier curves (such as PostScript) can implicitly work with quadratic curves (such as TrueType) by using their equivalent cubic representation.

We use the relation

$$(1-t)b_{i,n}=\frac{n+1-i}{n+1}b_{i,n+1},\ tb_{i,n}=\frac{i+1}{n+1}b_{i+1,n+1}$$

and introducing $\mathbf{P}_{-1}=\mathbf{P}_0,\mathbf{P}_{n+1}=\mathbf{P}_n$

$$\begin{aligned}\mathbf{B}(t)&=\sum_{i=0}^{n}b_{i,n}(t)[(1-t)\mathbf{P}_i+t\mathbf{P}_i]\\ &=\sum_{i=0}^{n+1}\left(\frac{i}{n+1}\mathbf{P}_{i-1}+\frac{n+1-i}{n+1}\mathbf{P}_i\right)b_{i,n+1}(t)\end{aligned}$$

Hence, the new control points are

$$P_i'=\frac{i}{n+1}\mathbf{P}_{i-1}+\frac{n+1-i}{n+1}\mathbf{P}_i,\quad ,i=0,\cdots,n+1$$

Repeated elevation on a control polygon $\mathbf{R}$ to get $\mathbf{R}_r$ have the relation to the underlying Bezier curve: $\lim_{r\to\infty}\mathbf{R}_r=\mathbf{B}$

IV. NURBS: Beyond the Bezier

A. Formulation

A NURBS (Non-uniform rational basis spline) curve is defined by its order $n+1$ (degree n), number of weighted control points k, and a knot vector. NURBS curves and surfaces are generalizations of both B-splines and Bézier curves and surfaces, the primary difference being the weighting of the control points, which makes NURBS curves rational. By using a two-dimensional grid of control points, NURBS surfaces including planar patches and sections of spheres can be created. These are parametrized with two variables. This can be extended to arbitrary dimensions to create NURBS mapping $\mathbb{R}^n \to \mathbb{R}^n$. We will discuss NURBS curve in the following, and a NURBS surface is obtained as the tensor product of two NURBS curves.

a) Order: The order of a NURBS curve defines the number of nearby control points that influence any given point on the curve. The curve is represented mathematically by a polynomial of degree one less than the order of the curve. The number of control points k must be greater than or equal to the order of the curve $n+1$. In practice, cubic curves are the ones most commonly used.

b) Control point: Each point of the curve is computed by taking a weighted sum of k control points. The weight of each point varies according to the governing parameter. For a curve of degree n, the weight of any control point is only nonzero in $n+1$ intervals of the parameter space. Within those intervals, the weight changes according to a polynomial function (basis functions) of degree n. At the boundaries of the intervals, the basis functions go smoothly to zero, the smoothness being determined by the degree of the polynomial.

c) Knot vector: The knot vector is a sequence of parameter values that determines where and how the control points affect the NURBS curve. The number of knots is always equal to the number of control points plus curve degree plus one. The knot vector divides the parametric space in the intervals mentioned before, usually referred to as knot spans. Each time the parameter value enters a new knot span, a new control point becomes active, while an old control point is discarded. It follows that the values in the knot vector should be in nondecreasing order. We assume the ith knot is $a_i, 1 \le i \le n+k+1$.

d) Basis function: The B-spline basis functions used in the construction of NURBS curves are usually denoted as $N_{i,n}(u)$, in which i corresponds to the ith control point. The definition of these basis functions is recursive in n. The degree-0 functions $N_{i,0}$ are piecewise constant functions. They are one on the corresponding knot span and zero everywhere else. Effectively,$N_{i,n}$ is a linear interpolation of $N_{i,n-1}$ and $N_{i+1,n-1}$. The latter two functions are non-zero for n knot spans, overlapping for $n-1$ knot spans. For $a_i \le u \le a_{i+n+1}$, the function $N_{i,n}$ is computed as

$$N_{i,n}(u) = \frac{u-a_i}{a_{i+n}-a_i} N_{i,n-1}(u) + \frac{a_{i+n+1}-u}{a_{i+n+1}-a_i} N_{i+1,n-1}(u)$$

e) General form: In the one dimensional case, we denote the control point $\mathbf{P}_i,\ 1 \le i \le k$, and corresponding weight w_i. The basis function is denoted as $N_{i,n}$. Then, a NURBS curve takes the following form

$$C(u) = \sum_{i=1}^{k} \frac{N_{i,n} w_i}{\sum_{j=1}^{k} N_{j,n} w_j} \mathbf{P}_i := \sum_{i=1}^{k} R_{i,n}(u) \mathbf{P}_i$$

where $R_{i,n}$ are known as rational basis function.

B. Continuity

Surface continuity is a measure of how smoothly two surfaces flow into each other. A car hood, for example, can be composed of multiple small surfaces that appear to be one because of the smoothness of the surface continuity. In addition, different positions of a character may require different continuity.

a) Positional continuity (G^0): holds whenever the end positions of two curves or surfaces coincide. The curves or surfaces may still meet at an angle, giving rise to a sharp corner or edge and causing broken highlights.

b) Tangential continuity (G^1): requires the end vectors of the curves or surfaces to be parallel and pointing the same way, ruling out sharp edges. Because highlights falling on a tangentially continuous edge are always continuous and thus look natural, this level of continuity can often be sufficient.

c) Curvature continuity (G^2): further requires the end vectors to be of the same length and rate of length change. Highlights falling on a curvature-continuous edge do not display any change, causing the two surfaces to appear as one. This level of continuity is very useful in the creation of models that require many bicubic patches composing one continuous surface.

C. Comparison

While non-rational splines or Bezier curves can only approximate a circle, NURBS can represent a circle exactly. A circle can be represented as a NURBS curve of degree 2. The general parameterization is

$$\begin{aligned}&(\hat{x}(u), \hat{y}(u), w(u))\\ =&w_0(1-u)^2(1,0,1) + w_1 2u(1-u)(1,1,1) + w_2^2 u^2 (0,1,1)\end{aligned}$$

then we project it onto the 2-D plane $(x(u), y(u)) = (\hat{x}(u)/w(u), \hat{y}(u)/w(u))$. For symmetric, we set $w_0 = w_2 = \sqrt{2} w_1$.

A Python Package "NURBS-Python"[2] provides convenient data structures and highly customizable API for rational and non-rational splines along with the efficient and extensible implementations of the multiple algorithms.

[2]https://github.com/orbingol/NURBS-Python