DYNAMIC GEOMETRY AND WEBSITE SETUP BY AUTOMATIC OBJECT RECOGNITION OF FREE-HAND DRAWINGS AND SCANS

Karl-Heinz BRAKHAGE
RWTH Aachen University, Germany

ABSTRACT: Computer software, websites, web applications and the usability on tablets and smartphones have the potential to make significant improvements in how geometry is learnt and taught. As with any approach to teaching, the educational use of information and communication technology as well as computer software needs to be well thought through and carefully planned. Sometimes it is useful to have tools that automatically recognize given curves as lines, circles or conics and splines. In this paper we will concentrate on automatic object recognition from free-hand drawings and / or scans and the conversion into scalable vector graphics. These can be used for animations or exported as vector graphics in various formats. We will give an overview of the analytical background and implementation issues. Furthermore we will demonstrate its use on a couple of selected examples from various areas.

Keywords: Teaching, Descriptive Geometry, Dynamic Geometry, Splines, Approximation, Fairing, Numerical Analysis

1. INTRODUCTION

Nowadays the abilities of computers and software have a significant influence on teaching and learning geometry. Computer software, particular that known as Dynamic Geometry Software, websites, web applications and the usability on tablets and smartphones have the potential to make significant improvements in how geometry is learnt and taught. Our view is that teachers should now have at their disposal an appropriate variety of equipment from which to select. As with any approach to teaching, the educational use of information and communication technology as well as computer software needs to be well thought through and carefully planned. In this paper we will concentrate on some special topics of ***WinCAG*** (Windows version of Computer Aided Geometry) which has been especially designed for teaching and learning geometry (see [2]). With this software package both simple constructions and complicated dependencies can easily be animated. Furthermore the drawings can be exported in well suited formats for further use in publications or on web pages. Here we will particular focus on some special capabilities for automatic object recognition from free-hand drawings and / or scans. Sometimes it is useful to have tools that automatically recognize given curves as lines, circles or conics and splines. We have implemented that in ***WinCAG*** in two ways. For external data we can import pixel oriented bitmaps or polylines in form of coordinates. Furthermore the system has a module that generates traverses by free-hand mouse-drawing.

To clearly demonstrate the goal of our efforts we give an illustrating example. Figure 1 shows a screenshot of such a scenario. We have three different traverses. These were *recognized* by the system as a line, a circle and an ellipse. Figure 2 shows the reconstruction of the corresponding scanned plot of figure 1. A zoom clearly shows a pixelated look in case of figure 1 and the *exact*

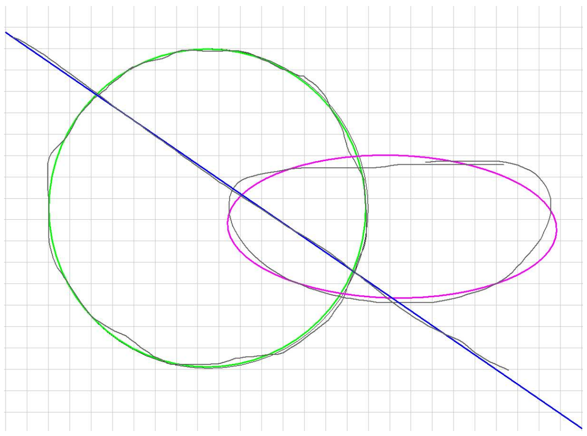

Figure 1: Screenshot of an object recognition: line, circle and ellipse (with raster).

curves in figure 2. Additionally hyperbolas and

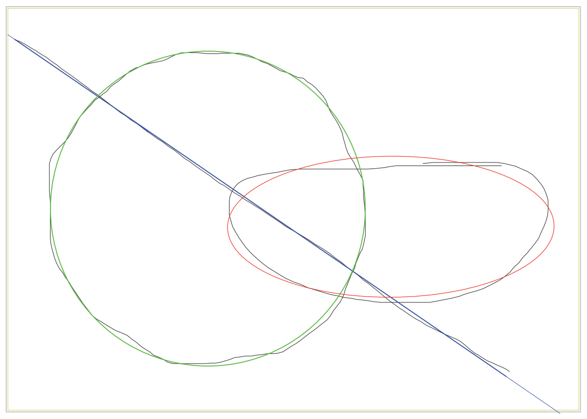

Figure 2: Reconstruction of figure 1 (without raster).

parabolas will be recognized. For this purpose the user has to give tolerances for each object. Since for instance an ellipse normally fits better than a circle, the tolerance for circles must be chosen larger than that for ellipses. The system stores the data traverses and the user can force it to compute a given kind of curve. The fall back option is a smooth B-spline. The curves corresponding to the input in form of polylines or point clouds are computed by least squares solutions. For conics the fit by a linear implicit form does not yield the optimal (non-linear) least squares solution given by the parameterized form. Thus for an initial guess from a linearized scaled problem we determine the parameters for the data points and make some Gauss-Newton steps to improve our result. If we are *close to* a parabola the system computes a least squares solution for this case, too. Then the user can optionally decide to take the parabola. If the distances to all of the above objects are too large the system will compute a B-spline approximation. Figure 3 shows an example for that situation. We have plotted the control polygon for the B-spline (of order 4), too. Additionally we have marked the data points because the data polygon nearly coincides with the spline.

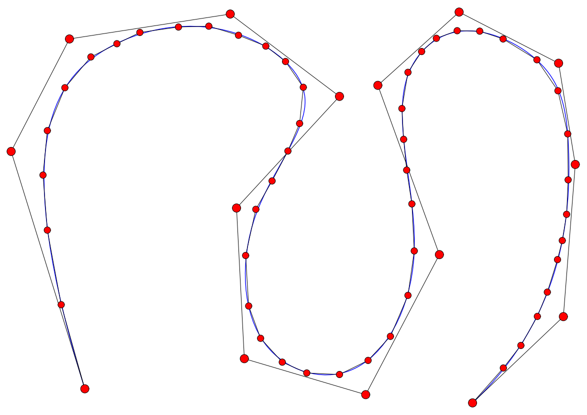

Figure 3: PDF export of an object recognition: B-spline.

The geometric objects of the above processes can either be used within the software package ***WinCAG*** for animations or exported as vector graphics in various formats, for instance Encapsulated Postscript (EPS), PDF and Scalable Vector Graphics (SVG). EPS and PDF files are for use in printed publications. SVGs can be used for websites. The major advantage of all three formats is the very small size of the files and the possibility to zoom without grid pattern effects (see figure igure 1). This is achieved via highly accurate numerical approximation of the given geometrical entities using objects supported in the target format.

We have seen above that raster images like screen shots cannot be scaled up in size without the loose of sharpness – they will look pixelated. Producing files (in the background) with a sufficient resolution for high quality prints results usually in very large files. Thus formats based on raster images are useless for websites – at least if we need higher resolutions. On the other hand vector graphics formats can present lines, curves, fonts etc. at essentially arbitrary precision in different colors and sizes in *small* files. Thus the conversion into these formats yields a significant quality improvement for the representation of curves and other plots.

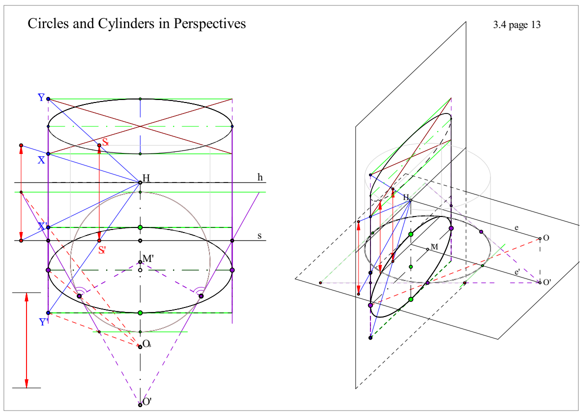

Figure 4: Advanced stage of an example for a perspective view with all layers on.

Figure 4 shows a more complex example. Here we have turned all layers on. Details will become clear by zooming into different areas of the plot.

The rest of this paper is organized as follows. First we give some basic notations and properties of Bézier and B-spline curves. Next we give some details on the used approximation strategies and algorithms for splines. A short summary and an evaluation regarding computation time and accuracy of the algorithms usually applied for these purposes will be given. Then we show how we force the system to favour special entities such as ellipses, hyperbolas or parabolas. Finally a we make some notes on the module that converts bitmaps to vector graphics.

2. SOME BASICS ON BÉZIERS AND B-SPLINES

Bézier curves of order n are given by $n+1$ control or Bézier points $\mathbf{p}_i \in \mathbb{R}^m$, $i = 0 \dots n$ (here $m = 2$), that build the so-called control polygon, are given for $t \in [0,1]$ by

$$\mathbf{x}(t) = \sum_{i=0}^{n} B_i^n(t)\,\mathbf{p}_i \tag{1}$$

with the Bernsteinpolynomials

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}. \tag{2}$$

It is easy to verify

$$\begin{aligned} \mathbf{x}(0) &= \mathbf{p}_0, & \mathbf{x}'(0) &= n\,(\mathbf{p}_1 - \mathbf{p}_0), \\ \mathbf{x}(1) &= \mathbf{p}_n, & \mathbf{x}'(1) &= n\,(\mathbf{p}_n - \mathbf{p}_{n-1}) \end{aligned} \tag{3}$$

Geometric continuity is an important property of the shape of curves and surfaces. For dynamic applications, e.g. path of a cutter tool for NC milling, we need curvature continuity. This is due to the physical effect of mass inertia. For our purposes here it its enough to claim tangent continuity. Thus we only define GC^1-continuity in terms of the parametrization. Two curves $\mathbf{x}(t)$, $t \in [t_0, t_1]$ and $\mathbf{y}(s)$, $s \in [s_0, s_1]$ are said to be GC^1-continous at the junction $\mathbf{x}(t_1) = \mathbf{y}(s_0)$ (that is called GC^0-continuity) if there exists a γ with

$$\mathbf{x}'(t_1) = \gamma\,\mathbf{y}'(s_0) \tag{4}$$

From (3) it is easy to see that the composition of two Bézier curves $\mathbf{x}^1(t)$ and $\mathbf{x}^2(t)$ is GC^1-continous, iff $\mathbf{p}^1_{n-1}$, $\mathbf{p}^1_n = \mathbf{p}^2_0$ and $\mathbf{p}^2_1$ lie in a straight line.

One main advantage for interpolation and approximation with curves like (1) is the linearity in the control points $\mathbf{p}_i$. B-spline curves have these property, too. They are given by

$$\mathbf{x}(t) = \sum_{i=0}^{N} N_{i,p,T}(t)\,\mathbf{p}_i \tag{5}$$

where $N_{i,p,T}(t)$ is the i-th normalized B-spline function of order p (degree $p-1$) corresponding to the generally non-uniform knot vector $T = (t_0, t_1, \ldots, t_{N+p})$. We usually assume that T is clamped, i.e., $t_0 = \ldots = t_{p-1}$ and $t_{N+1} = \ldots = t_{N+p}$. For the sake of simplicity we write $N_{i,p}$ instead of $N_{i,p,T}$ since it becomes clear from the name of the function argument what the knot vector is. Surfaces are represented by B-spline tensor products of the form

$$\mathbf{x}(u,v) = \sum_{i=0}^{N} \sum_{j=0}^{M} N_{i,p}(u)\, N_{j,q}(v)\, \mathbf{p}_{ij} \tag{6}$$

and the Bézier representation is

$$\mathbf{x}(u,v) = \sum_{i=0}^{N} \sum_{j=0}^{M} B_i^n(u)\, B_j^m(v)\, \mathbf{p}_{ij}. \tag{7}$$

These representation are linear in the control points, too. The same applies for stationary subdivision curves and surfaces. The algorithms presented below can be used for all these curve and surface classes.

Further information on Bézier- and B-splines can be found in standard literature on CAGD (Computer Aided Geometric Design), e.g. [5]. Furthermore several web page with applets regarding splines can be found. Here are some examples

- Wikipedia-Bezier: `http://de.wikipedia.org/wiki/B%C3%A9zierkurve`
- Prautsch-Applets: `http://i33www.ibds.uni-karlsruhe.de/applets/mocca/html/noplugin/inhalt.html`
- Farin-Applets: `http://www.vis.uni-stuttgart.de/~kraus/LiveGraphics3D/cagd/index.html`

3. APPROXIMATION OF CURVES AND SURFACES

Postscript, PDF and SVG have the entities quadratic an cubic Bézier curves but no higher degrees. Therefore we will restrict ourselves to polynomial degree two and three in the following if it reduces technical investments for the description.

B-splines can be split into Béziers by successive knot insertion. If we use cubic B-splines we get GC^2-continuity and in general approximations with smaller errors than those from GC^1-continuous Bézier approximations. But the linear systems of equations we have to solve are larger.

To be more precise we have to introduce some technical notations. For an optimal approximation of a given curve $\mathbf{y}(v)$, $v \in [v_{min}, v_{max}] \in \mathbb{R}$ by a Bézier-spline we have to determine the control points $\mathbf{p}_i$ (see (1)) in such a way that

$$\max_{v \in [v_{min}, v_{max}]} \min_{t \in [0,1]} \|\mathbf{y}(v) - \mathbf{x}(t)\|_2 \tag{8}$$

is minimized. For B-splines the only difference is that in (8) now $t \in [t_{p-1}, t_{N+1}]$. Since these problems are to hard to solve we switch to discrete interpolation or approximation problems. For Béziers the whole interval $[v_{min}, v_{max}]$ is normally a-priori splitted into a couple of subintervals. For simplicity we use the interval$[v_l, v_r]$ for these subintervals and define

$$v(s) := v_l + s\,(v_r - v_l). \tag{9}$$

Since we want to construct over all GC^1-continuous curves we further assume that for all parts $\mathbf{y}(v_l)$, $\mathbf{y}'(v_l)$, $\mathbf{y}(v_r)$ and $\mathbf{y}'(v_r)$ are given or computable. Then for quadratic Béziers from (3) we get

$$\begin{array}{lll} \mathbf{p}_0 = \mathbf{y}(v_l) & , \quad \mathbf{p}_1 = \mathbf{p}_0 + \alpha\, \mathbf{y}'(v_l) & , \\ \mathbf{p}_2 = \mathbf{y}(v_r) & , \quad \mathbf{p}_1 = \mathbf{p}_2 + \beta\, \mathbf{y}'(v_r). & \end{array} \tag{10}$$

α and β are already determined by the two equations for $\mathbf{p}_1$. For cubic Béziers (3) implies the conditions

$$\begin{array}{lll} \mathbf{p}_0 = \mathbf{y}(v_l) & , \quad \mathbf{p}_1 = \mathbf{p}_0 + \alpha\, \mathbf{y}'(v_l) & , \\ \mathbf{p}_3 = \mathbf{y}(v_r) & , \quad \mathbf{p}_2 = \mathbf{p}_3 + \beta\, \mathbf{y}'(v_r). & \end{array} \tag{11}$$

Thus we only have the two free (scalar) parameters α and β. With this notations we can formulate two strategies for curve approximation with cubic Béziers.

1) Determine α and β by the condition $\mathbf{y}(v(0.5)) = \mathbf{x}(0.5)$ – interpolation in between.

2) Determine α and β by the condition $\sum_{i=1}^{n}(\mathbf{y}(v(s_i)) - \mathbf{x}(s_i))^2 \to \min$ with $0 < s_1 < s_2 < \ldots < s_n < 1$ – (discrete) mean square approximation. Here we only use $n = 3$ with $s_1 = 1/4$, $s_2 = 1/2$ and $s_3 = 3/4$.

We further use the error estimators

$$\delta = \max_{i \in \{1,3\}} \left\| \mathbf{y}\left(v\left(\frac{i}{4}\right)\right) - \mathbf{x}\left(\frac{i}{4}\right) \right\|_2 \tag{12}$$

in 1) and

$$\delta = \max_{i \in \{1,3,5,7\}} \left\| \mathbf{y}\left(v\left(\frac{i}{8}\right)\right) - \mathbf{x}\left(\frac{i}{8}\right) \right\|_2 \tag{13}$$

in 2). If the error is two large, we (recursively) subdivide $[v_l, v_r]$ into two equal parts.

We use $\mathbf{y}_h := \mathbf{y}(v(1/2))$, $\mathbf{t}_0 := \mathbf{y}'(v_l)$ and $\mathbf{t}_1 := \mathbf{y}'(v_r)$. Strategy 1) results in an equation

$$Ax = b \tag{14}$$

with

$$A = \left(\tfrac{3}{8}\mathbf{t}_0 \quad \tfrac{3}{8}\mathbf{t}_1 \right), \tag{15}$$

$$b = \left(\mathbf{y}_h - \tfrac{1}{2}\mathbf{p}_0 - \tfrac{1}{2}\mathbf{p}_3 \right) \tag{16}$$

and

$$x = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} . \tag{17}$$

With $\mathbf{z}_i := \mathbf{y}(v(s_i))$ strategy 2) leads to an overdetermined linear system

$$\|Ax - b\|_2 \to \min \tag{18}$$

with

$$A = \begin{pmatrix} \frac{27}{64}\mathbf{t}_0 & \frac{9}{64}\mathbf{t}_1 \\ \frac{3}{8}\mathbf{t}_0 & \frac{3}{8}\mathbf{t}_1 \\ \frac{9}{64}\mathbf{t}_0 & \frac{27}{64}\mathbf{t}_1 \end{pmatrix}, \tag{19}$$

$$b = \begin{pmatrix} \mathbf{z}_1 - \frac{27}{32}\mathbf{p}_0 - \frac{5}{32}\mathbf{p}_3 \\ \mathbf{z}_2 - \frac{1}{2}\mathbf{p}_0 - \frac{1}{2}\mathbf{p}_3 \\ \mathbf{z}_3 - \frac{5}{32}\mathbf{p}_0 - \frac{27}{32}\mathbf{p}_3 \end{pmatrix} \tag{20}$$

and

$$x = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} . \tag{21}$$

Figure 5 shows an example for the approxima-

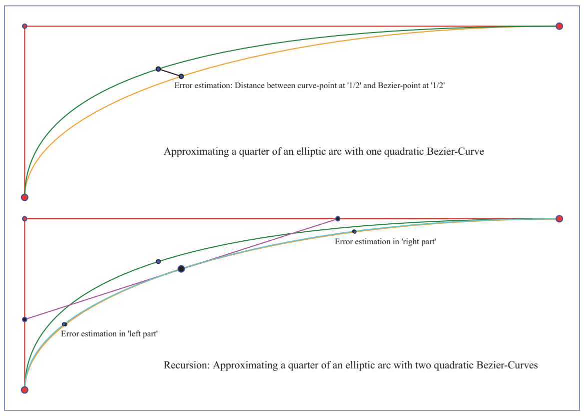
Error estimation: Distance between curve-point at '1/2' and Bezier-point at '1/2'
Approximating a quarter of an elliptic arc with one quadratic Bezier-Curve
Error estimation in 'right part'
Error estimation in 'left part'
Recursion: Approximating a quarter of an elliptic arc with two quadratic Bezier-Curves

Figure 5: Recursive approximation with quadratic Bézier curves

tion of one quarter of an ellipse with quadratic Béziers. Only one level of recursion is shown. The error estimator – at least for higher recursion level – is rather pessimistic. This is due to the fact that the error vectors become more and more tangential to the curves with higher recursion level. For a measurement of the real curve to curve distance the error vectors should be orthogonal to them. We will fix this lack later on with parameter corrections. This simple example already shows the power of spline approximation. For quadratic splines we expect the convergence rate $\mathcal{O}(h^3)$. This can only be achieved if the underlying curve is in $C^3[v_{min}, v_{max}]$. Thus for every recursion the error will be (asymptotically) reduced by a factor of 8. In figure 6 we have used a cubic Bézier with strategy 2) for a Lissajous figure with dimension 20×14. We made two approximation. One with a desired absolute error of 10^{-1} and as reference one with an absolute error of 10^{-6}. Additionally the control polygon is shown. Very few control points lead to a good approximation of the Lissajous curve. For cubic splines we expect the convergence rate $\mathcal{O}(h^4)$. Here the curve for approximation has to be in

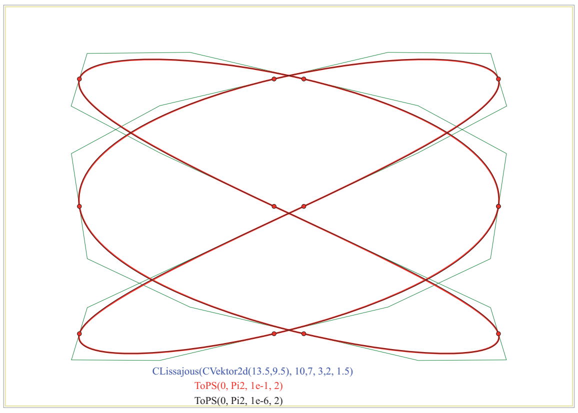

Figure 6: Approximation with cubic Bézier curves

$C^4[v_{min}, v_{max}]$.

Figure 7 shows the approximation and estimation strategy. Again we have use a quarter of an ellipse. Only the entrance level is shown. If we

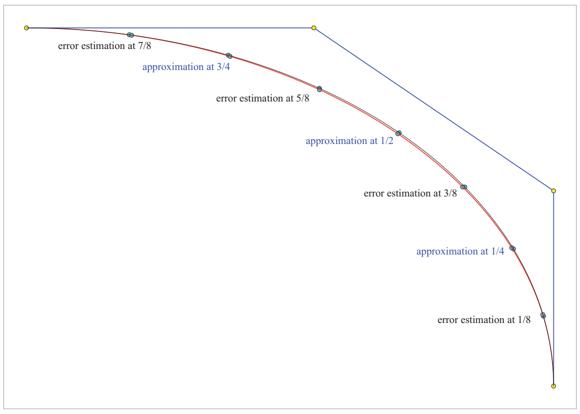

Figure 7: Approximation with cubic Bézier curves – strategy and error estimation

use the error estimation as shown in figure 5 and figure 7 with several recursion levels we will not recognize the expected convergence rates. This is due to the fact that the used error estimators do not measure the distance orthogonal the curves. To overcome this lack we have to correct the parameter values for v originally given as $v(s_i)$ according to (9) in such a way that we measure orthogonal. Using $\mathbf{p} = \mathbf{x}(s_i)$ this can be formulated as

$$f(v) = (\dot{\mathbf{y}}(v))^T \, (\mathbf{y}(v) - \mathbf{p}) \overset{!}{=} 0. \qquad (22)$$

The derivative of this function is

$$f'(v) = (\ddot{\mathbf{y}}(v))^T \, (\mathbf{y}(v) - \mathbf{p}) + (\dot{\mathbf{y}}(v))^T \, \dot{\mathbf{y}}(v) \quad (23)$$

and it is easy to implement Newton's method for (22). For surfaces this works similarly. (22) is replaced by the system

$$f(u,v) = \begin{pmatrix} \mathbf{x}_u^T(u,v) \, (\mathbf{x}(u,v) - \mathbf{p}) \\ \mathbf{x}_v^T(u,v) \, (\mathbf{x}(u,v) - \mathbf{p}) \end{pmatrix} \qquad (24)$$

and the derivative is given by (we omit the parameters and use $\mathbf{d} = \mathbf{x}(u,v) - \mathbf{p}$)

$$f' = \begin{pmatrix} \mathbf{x}_{uu}^T \mathbf{d} + \mathbf{x}_u^T \mathbf{x}_u & \mathbf{x}_{uv}^T \mathbf{d} + \mathbf{x}_u^T \mathbf{x}_v \\ \mathbf{x}_{uv}^T \mathbf{d} + \mathbf{x}_u^T \mathbf{x}_v & \mathbf{x}_{vv}^T \mathbf{d} + \mathbf{x}_v^T \mathbf{x}_v \end{pmatrix}. \qquad (25)$$

This parameter correction already leads to an error estimator that reflects the expected convergence rates.

The results can be improved if we iterate according to (18) – (21) with the corrected v's and thus modified $\mathbf{z}_1$, $\mathbf{z}_2$ and $\mathbf{z}_3$ in (20). This decoupled iteration only slowly converges towards the optimal solution. This is due to the fact that approximation (18) minimizes a point distance and not the curve distance.

In [8] and [9] an approach based on the minimization of a quadratic approximant of the squared distance function was introduced. Unfortunately in general the second order Taylor approximant does not lead to symmetric positive definite system matrices and for this reason we can not guarantee the existence and uniqueness of the minimum. For this reason the second order Taylor approximation was modified to guarantee positive definiteness. But this modification destroys the second order and thus the claimed quadratic convergence of that method. Nevertheless such methods need less iterations. But the drawback is a computational overhang. For instance the curvature – the principal curvature and directions for surfaces – has / have to be computed. Furthermore the parametrization is not really totally avoided. In [4] the behaviour of the point distance minimization (PDM) and the squared distance function minimization (SDM) was studied. Regarding run time the authors conclude: *Our experiments show that the time used by SDM on iterative optimization is about 30% to 50% more than PDM because SDM needs extra time to set up the more complex SDM error function.*

We have used a different approach. This avoids the curvature computation. Furthermore our method can still be written in the form of (18) which is not the case for SDM. We will demonstrate the idea of our method for the approximation with B-splines. Due to (5) and (6) B-spline curve or surface points are linear combinations of the control points.

$$\mathbf{x}_i = \mathbf{x}(t_i) = \sum_j a_{ij}\mathbf{p}_j. \tag{26}$$

The $\mathbf{x}_i$ correspond to curve points $\mathbf{y}_i = \mathbf{y}(v_i)$ and we have to minimize the distances $\|\mathbf{x}_i - \mathbf{y}_i\|_2$. We collect the control points $\mathbf{p}_ij$ in a vector $\mathbf{p}$, the $\mathbf{y}_i$ in a vector $\mathbf{y}$ and coefficients a_{ij} in a sparse matrix A. Now PDM is simply

$$\|A\,\mathbf{p} - \mathbf{y}\|_2 \to \min. \tag{27}$$

For the solution of (27) we can use orthogonal transformations or the normal equations. For more details especially on surfaces an volumes see ([1]). To reduce the number of parameter corrections (the *outer loop*) we minimize the distances of the $\mathbf{y}_i$ to the linear approximation of $\mathbf{y}(v)$ ad the point $\mathbf{y}_i$. Note that this coincides with the squared distance function for points on the curve. Thus we have to minimize

$$\|\mathbf{n}_i^T\,(\mathbf{x}_i - \mathbf{y}_i)\|_2 \tag{28}$$

where $\mathbf{n}_i = \mathbf{n}(v_i)$ is orthogonal to $\dot{\mathbf{y}}(v_i)$ and normalized. Notice that (28) is a minimization over the control points $\mathbf{p}_j$. To use matrix notations we have to separate the x and y components. We define $N_x = \operatorname{diag}\{\mathbf{n}_{i,x}\}$, $N_y = \operatorname{diag}\{\mathbf{n}_{i,y}\}$ and collect the terms $\mathbf{n}_i^T\,\mathbf{y}_i$ in a vector $\mathbf{d}$. Now we can write this part as

$$\|N_x A\,\mathbf{p}_x + N_y A\,\mathbf{p}_y - \mathbf{d}\|_2 \to \min. \tag{29}$$

For our final minimization problem we use (29) and a scaled portion of (27). Here we have to split $\mathbf{p}$ into its x-components $\mathbf{p}_x$ and y-components $\mathbf{p}_y$. The same way we split $\mathbf{y}$ into $\mathbf{y}_x$ and $\mathbf{y}_y$. With

$$\hat{A} = \begin{pmatrix} \lambda A & 0 \\ 0 & \lambda A \\ N_x A & N_y A \end{pmatrix} \tag{30}$$

$$\hat{\mathbf{p}} = \begin{pmatrix} \mathbf{p}_x \\ \mathbf{p}_y \end{pmatrix}, \tag{31}$$

and

$$\hat{\mathbf{y}} = \begin{pmatrix} \lambda\,\mathbf{y}_x \\ \lambda\,\mathbf{y}_y \\ \mathbf{d} \end{pmatrix} \tag{32}$$

our minimization problem now reads.

$$\|\hat{A}\,\hat{\mathbf{p}} - \hat{\mathbf{y}}\|_2 \to \min. \tag{33}$$

Thus again we can use orthogonal transformations, the normal equations or iterative methods. Especially for recursive approximation of surfaces and volumes the iterative methods are much faster. If only equidistant knot vectors are used a matrix free implementation is possible. The parameter $\lambda \geq 0$ is chosen depending on the error estimator. For small errors we use smaller λs. If the oversampling is high enough we even choose $\lambda = 0$ for high accurate approximations.

As iterative solver we normally use CGLS, a Conjugate Gradient method for linear Least Squares (also called CGNR in [10]). To apply CGLS we only need an effective implementation to multiply the system matrix A and its transpose A^T with vectors. It should be noted that in our applications A is very sparse and for recursive strategies we have very good starting vectors. For further informations and details on the use of CGLS see [10] or [3].

4. OBJECT RECOGNITION

Sometimes it is useful to have tools that recognize given curves as lines, circles or conics. We have implemented that in ***WinCAG*** in several ways. In this section we will report on the module that generates data polygons by free-hand mouse-drawing and then determines the best fit from a given list of objects. An example is shown in figure 1. The three recognized objects are a line, a circle and an ellipse. Hyperbolas and parabolas will be recognized too. For this purpose the user has to give tolerances for each object. Since for instance an ellipse normally fits better than a circle, the tolerance for circles must be chosen larger than that for ellipses. The system stores the data polygons and the user can force it to compute a given kind of curve. The curves corresponding to the data polygons (x_i, y_i) $i = 0, 1, \ldots, N$ are computed by least squares solutions. For this reason we start with the implicit form of lines, circles and conics:

$$\begin{aligned} ax + by + c &= 0 \\ (x - x_m)^2 + (y - y_m)^2 - r^2 &= 0 \\ \sum_{i,j=0}^{i+j\leq 2} a_{ij} x^i y^j &= 0 \end{aligned} \tag{34}$$

For circles the equations seem to be non-linear. But

$$\begin{gathered} (x - x_m)^2 + (y - y_m)^2 - r^2 = \\ -2xx_m - 2yy_m + x_m^2 + y_m^2 - r^2 + x^2 + y^2 = \\ -2xx_m - 2yy_m + \alpha + x^2 + y^2 \end{gathered} \tag{35}$$

with the substitution $\alpha = x_m^2 + y_m^2 - r^2$ it is linear in x_m, y_m and α and has the same solution as the original system. If we plug in the (x_i, y_i) we get a linear least squares problem for the coefficients in all cases. For lines and conics the coefficients have an arbitrary non zero scaling factor. We can choose one of them to be one. In very few cases the chosen coefficient must be zero and we have to revise our guess and have to solve the system once more. The principal axis transformation is used to get the kind of the conic. The conic fit by the implicit form does not yield the optimal (non-linear) least squares solution given by the parameterized form. Thus we determine the parameters for the data points and make some Gauss-Newton steps to improve our result. If we are *close to* a parabola the system computes a least squares solution for this case, too. Then the user can optionally decide to take the parabola.

If the distances to all of the above objects is to large the system will compute a B-spline approximation. Figure 3 shows an example for that situation. Details on spline approximation have been given in the previous section. The only difference here the determination of the parametrization of the knot vector. We use a chord length parametrization since it is known that in general this leads to small approximation errors. We refer to [6]

If we want to force the system to compute a special conic section we have have to make some extra efforts for parabolas and in the case the principal axis transformation does not lead to the right kind of the conic. Parabolas do not cause a real problem. in general the implicit solution leads to feasible initial values for the non linear iteration.

Next we describe the approach for forcing an elliptic fit. With the *normal* form

$$\mathbf{x}_n(t) = \begin{pmatrix} a\cos(t) \\ b\sin(t) \end{pmatrix} \tag{36}$$

and the rotation matrix

$$D(\psi) = \begin{pmatrix} \cos(\psi) & \sin(\psi) \\ -\sin(\psi) & \cos(\psi \end{pmatrix} \tag{37}$$

the general form of an ellipse is given by

$$\mathbf{x}(t) = \begin{pmatrix} x_m \\ y_m \end{pmatrix} + D(\psi)\mathbf{x}_n(t). \tag{38}$$

We have five parameters a, b, x_m, y_m and ψ and denote the corresponding Ellipse by $E(a, b, x_m, y_m, \psi)$. Given a point $\mathbf{x}_i$ and the parameters a, b, x_m, y_m and ψ we can compute the distance $d_i(\mathbf{x}_i;\, a, b, x_m, y_m,\, \psi)$ from the point $\mathbf{x}_i$ to the ellipse $E(a, b, x_m, y_m,\, \psi)$. For a point cloud $\mathbf{x}_i,\ i = 1, \ldots, N$ we can define the distance function as (in d_i we have omitted the arguments)

$$dist(a, b, x_m, y_m,\, \psi) = \sum_{i=1}^{N} d_i. \qquad (39)$$

This function is highly non linear. For the minimization of $dist(a, b, x_m, y_m,\, \psi)$ we use the Nelder-Mead-method published in [7]. This method has only linear convergence but is very robust and does not need any derivatives. As starting values we use $\psi = 0$, $a = b = r$ with r and x_m and y_m from a circle approximations. In our examples this method never fails.

5. RECONSTRUCTION FROM IMAGES

For the reconstruction of curves from images we only assume that the input format can be transferred into RGB-format. In a first step we cluster the contained colors in the 3D RGB color space and compute local color centroids. The pixels of each centroid are handled separately looping over all centroids. Starting with a low number of neighbored points we use the methods from the previous section to compute initial objects of different kinds. In a next step we insert further points that have a short distance to the previously computed objects and recompute the minimization problems with more points. If no further points close the actual objects are found and the extension is not negligible we store the best fit and drop the used points from our list. If at the end of this process a significant number of points is remaining we try to use B-spline fits with standard methods for point clouds to get connect curve parts. This may fail for some points and we have to leave this points untreated. In our examples this normally only happens for noisy pixels that do not belong do any curve part. The methods only fail for scenarios with a very large number of intersections or very close curves. This is due to the fact that we remove the used points and for the last curves in this cases we do not have enough points remaining for the curve reconstruction.

6. CONCLUSIONS

We have demonstrated the potential of effective approximations with Bézier or B-spline curves applied on scattered data or the conversion of curves given in other formats. One of the main aspects is the preparation of websites and / or further use in a CAGD system. The major advantage of the used vector graphics formats is the very small size of the files and the possibility to zoom without grid pattern effects. This leads to the usability on tablets and smartphones and the potential to make significant improvements in how geometry is learnt and taught.

INTERNET SOURCES ON VECTOR GRAPHICS

- PostScript Adobe: `http://partners.adobe.com/public/developer/ps/index_specs.html`
- Wikipedia-PostScript: `http://de.wikipedia.org/wiki/PostScript`
- Scalable Vector Graphics: `http://www.w3.org/TR/SVG11/`
- The authors website on this topic: `http://www.igpm.rwth-aachen.de/brakhage/ICGG2014`

REFERENCES

[1] K.-H. Brakhage. High quality mesh generation and sparse representation using b-splines. In B.K. Soni. et al, editor, *7th International Conference on Numerical Grid Generation in Computational Field Simulations*. Resort Whistler, British Columbia, Canada, September 25-28 2000.

[2] K.-H. Brakhage. WinCAG - dynamical geometry for teaching and learning. In *13th International Conference on Engineering Computer Graphics and Descriptive Geometry*. Dresden, Germany, August 3-8 2008.

[3] K.-H. Brakhage. Grid generation and grid conversion by subdivision schemes. In B.K. Soni et all, editor, *11^{th} International Conference on Numerical Grid Generation in Computational Field Simulations*. Montral, Canada, May 24-28 2009.

[4] K. Cheng, W. Wang, H. Qin, K.-Y. Wong, H. Yang, and Y. Liu. Fitting subdivision surfaces to unorganized point data using sdm. In *In Pacific Conference on Computer Graphics and Applications 2004*, pages 16–24. 2004.

[5] G. Farin. *Curves and Surfaces for CAGD. A Practical Guide*. The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling, fifth edition, 2002.

[6] M. Floater. Arc length estimation and the convergence of polynomial curve interpolation. *BIT Numer. Math.*, 45: 679–694, 2005.

[7] J. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4): 308–313, 1965.

[8] H. Pottmann and M. Hofer. Geometry of the squared distance function to curves and surfaces. In *VISUALIZATION AND MATHEMATICS III*, pages 223–244. 2003.

[9] H. Pottmann and S. Leopoldseder. A concept for parametric surface fitting which avoids the parametrization problem. *Computer Aided Geometric Design*, 20: 343362, 2003.

[10] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, second edition, 2003.

ABOUT THE AUTHOR

Karl-Heinz Brakhage is member of the Institute of Geometry and Numerical Mathematics at the Aachen University of Technology. His research interests are Computer Aided Geometric Design, CAx Technologies, Grid Generation, Scientific Computing, Computer Graphics, and Development of Education Software. He can be reached by e-mail: brakhage@igpm.rwth-aachen.de, by Fax: +49(241)8092317, by phone: +49(241)8096591, the postal address: Inst. of Geometry and Numerical Mathematics / RWTH Aachen / Templergraben 55 / D-52056 Aachen, Germany, or through the website: `http://www.igpm.rwth-aachen.de/brakhage`.