

Approximation of offset curves of Bèzier Curves

Patrick Ott

Institute of Computer Science
Anhalt University of Applied Sciences
D-06366 Köthen (Saxony-Anhalt), Germany
Email: Patrick.Ott@inf.hs-anhalt.de

Yu Zhengsheng

Institute of Virtual Reality and Multimedia
Dianzi University
CN-310018 Hangzhou (Zhejiang), China
Email: yuzhengsheng@tom.com

Abstract

Three methods aimed at speed and preciseness for calculating offsets of a Bèzier curve are given. The paper deals with an approximation of offset curves (of a given Bèzier curve of degree n) by a new Bèzier curve (the offset) of degree m . All three methods are based upon the method of least square.

I. INTRODUCING THE PROBLEM

1) *Areas of application:* Calculating offsets, which are also called parallel curves, has a big field of application. Besides computer games and visual design the main reason for using offsets lies within CAD and CAM applications. Another interesting way to use them is to calculate paths of industrial robots. Bèzier curves are usually used in the automobile industry, which is also the area they were first introduced by Pierre Bèzier in the early 1960's. For example varnishing robots may drive around a cars surface (two-dimensional: curve) to paint it from the in- or outside. Parallel curves could also be used to determine tolerance regions or to represent brush strokes. All those examples are covered by the literature survey [2], which works off nearly every method introduced or enhanced since 1992, too. It also gives a good overview over the available techniques to derive (self-) intersections off offset curves. A literature survey on offset curves and surfaces prior to 1992 was conducted by [3]. [4] covers areas of application such as geological modeling, molecular modeling and commercial advertising. Different computer programs, such as Adobe's Photoshop¹ or Corel Draw², use Splines, and especially Bèzier curves to build vector based image data, which can be rescaled as often as needed without losing any details of the picture. Adobe Postscript³ makes a big use of Bèzier curves by using them for the entire drawing model. Offsets become interesting in this way by drawing bold letters.

2) *Basic math:*

Definition 1: A general Bèzier curve is given as a vector $\vec{r}(t)$ by the following statement:

$$\vec{r}(t) = \begin{pmatrix} x_r(t) \\ y_r(t) \\ \vdots \end{pmatrix} = \sum_{i=0}^n \vec{P}_i B_n^i(t)$$

where B indicates the Bernstein Polynomials

$$B_n^i(t) = \binom{n}{i} t^i (1-t)^{n-i} \quad t \in [0, 1]$$

\vec{P}_i are the control points position vector.

The dimensional coherence could be expressed like this:

$$\vec{r}(t) = (\phi_1(t), \phi_2(t), \dots)^T$$

¹ Photoshop is a registered trademark of Adobe Systems, Inc.

² Corel Draw is a registered trademark of the Corel Corporation

³ Postscript is a registered trademark of Adobe Systems, Inc.

where $\phi_k(t)$ is the k 'th component. The mapping of $\vec{r}(t)$ could be described further by $\vec{r}' : \mathbb{R} \ni [0, 1] \rightarrow \mathbb{R}^n$.

Definition 2: An offset is given by the curve along the original curve $\vec{r}(t)$, plus the distance multiplied with the normal vectors $\vec{n}_e(t)$ of the curve.

$$\vec{q}(t) = \vec{r}(t) + d\vec{n}_e(t) = (x_q(t), y_q(t))^T. \quad (1)$$

For plain two-dimensional curves the normal vectors $\vec{n}_e(t)$ have the following representation:

$$\vec{n}_e(t) = \pm \left(\frac{y'_r}{\sqrt{(x'_r)^2 + (y'_r)^2}}, -\frac{x'_r}{\sqrt{(x'_r)^2 + (y'_r)^2}} \right)^T.$$

x'_r and y'_r are the first derivations of the curve in the specified dimensions. The curves first derivation $\frac{d}{dt}\vec{r}(t) = \vec{r}'(t)$ is

$$\vec{r}'(t) = \begin{pmatrix} x'_r(t) \\ y'_r(t) \\ \vdots \end{pmatrix} = \sum_{i=0}^{n-1} B_{n-1}^i(t) n \left(\vec{P}_{i+1} - \vec{P}_i \right)$$

which usually is referred to as the hodograph of the original Bèzier curve.

Definition 3: A Bèzier curves k 'th derivative is

$$\vec{r}^{(k)}(t) = \left(\prod_{j=1}^{k+1} (n-j) \right) \sum_{i=0}^{n-k} B_{n-k}^i(t) \psi_i^k$$

where ψ represents finite differences. ψ_i^0 is the control point \vec{P}_i for $0 \leq i \leq n$. The first level finite difference is $\psi_i^1 = \psi_{i+1}^0 - \psi_i^0 = \vec{P}_{i+1} - \vec{P}_i$ for $0 \leq i \leq n-1$. The second level difference will have $n-1$ points, so repeating the procedure will lead to the k 'th level difference

$$\psi_i^k = \psi_{i+1}^{k-1} - \psi_i^{k-1} \text{ for } 0 \leq i \leq n-k.$$

Offsets of space curves $n \geq 3$ are not possible since there is an infinite number of normal vectors bothering the curve. Fixing one or more dimensions of the normal vectors or extending the curve to a surface (and higher-dimensional mathematical constructs) is another possibility. [5] covers those topics and gives a deep insight into directional offsets of three-dimensional curves. However, the offset-calculation in this paper will focus on two-dimensional Bèzier curves.

$$\vec{r}(t) = (\phi_1(t), \phi_2(t))^T = (x_r(t), y_r(t))^T$$

II. BERNSTEIN POLYNOMIAL TRANSFORMATION

Since integrating and repeatedly differentiating Bernstein polynomials can be a tough job compared to general polynomials (monom basis), it would be useful to transform the Bernstein polynomials to the vector space of monom based polynomials $\{a_i t^i \mid i \in \mathbb{N}\}$. A Bernstein Polynomial could be expressed to monom basis in the following way.

$$B_n^i(t) = \left(\binom{n}{i} \sum_{k=0}^{n-i} (-1)^k \binom{n-i}{k} t^{i+k} \right) \quad (2)$$

A whole Bèzier curve can also be transformed in this way.

$$\sum_{i=0}^n \left((-1)^i \binom{n}{i} \sum_{k=0}^i (-1)^k \vec{P}_k \binom{i}{k} \right) t^i = \sum_{i=0}^n \vec{a}_i t^i \quad (3)$$

We call this transformation the Bernstein Polynomial Transformation or simply the Bernstein transformation. Besides simpler ways to differentiate and integrate the Bernstein transformation offers some other interesting advances. In addition to calculating the points of a Bzier curve with the Bernstein polynomials or using DeCasteljau's algorithm we now can use a third method: Horner's scheme (formally known as Horner's method). On the other side calculating the roots, (dimensional) extreme- and interception points got much easier. Actually it is now possible to apply polynomial division on the curve, so it is possible to split roots from it, too.

III. OFFSET CALCULATION

A. Error consideration

The approximated total error E as well as the average error $AVGE$ between two curves $\vec{p}(t)$ and $\vec{q}(t)$ could be expressed in the following way

$$E = \sum_{u=0}^{\lambda} |\vec{p}(u/\lambda) - \vec{q}(u/\lambda)|^2 \quad (4)$$

$$AVGE = \varepsilon = \frac{\sum_{u=0}^{\lambda} |\vec{p}(u/\lambda) - \vec{q}(u/\lambda)|^2}{\lambda} \quad (5)$$

Another possibility is to solve or numerical approximate the integral

$$\varphi = \int_0^1 |\vec{p}(t) - \vec{q}(t)|^2 dt \quad (6)$$

B. Least square method

We know a set of $\chi \in \mathbb{N}^+$ equidistant points on the original Bèzier curve. The multiple distances between the approximated offset $\vec{q}(t)$ and the perfect offset $\vec{p}(t)$ could be expressed commensurate to Formula 4 by the following formula.

$$\varrho(\chi) = \sum_{u=0}^{\chi} |\vec{p}(\omega) - \vec{q}(\omega)|^2 \quad \omega = u/\chi$$

with $\vec{q}(t) = \sum_{i=0}^n \vec{Q}_i B_n^i(t)$. Our goal will be to minimize the squares of the distances with respect to the offsets control points $\varrho(\chi) \rightarrow \min$. Minimizing the expression will result in partial differentiating of the whole term. We don't need to solve the equations for the first and last control point because it is possible to calculate them exactly. The tangent vector of the first point is the straight line between the first and the second point; the tangent vector of the last point is the straight line between the last and the point before the last point. Figure 1 illustrates this relationship. Solving the equation will result in two systems of linear equations.

$$K\vec{Q}_x = \vec{b}_x \quad K\vec{Q}_y = \vec{b}_y$$

The matrix K consists of

$$K \ni k_{i,j} = -2 \sum_{t=0}^{\chi} B_m^i(\omega) B_m^j(\omega)$$

for $i = j = 1, 2, \dots, m-1$. $\vec{Q}_x = (q_1^x, q_2^x, \dots, q_{m-1}^x)^T$ and $\vec{Q}_y = (q_1^y, q_2^y, \dots, q_{m-1}^y)^T$ hold the control points of the offset curve in the different dimensions. The vectors \vec{b}_x and \vec{b}_y are defined as

$$\vec{b}_x = (\xi_1, \xi_2, \dots, \xi_{m-1})^T \quad \vec{b}_y = (\varsigma_1, \varsigma_2, \dots, \varsigma_{m-1})^T$$

with

$$\xi_i = \xi_i(\omega) = 2 \sum_{t=0}^{\chi} (B_m^i(\omega) (-x_r(\omega) - dx_{n_e}(\omega) + q_0^x B_m^0(\omega) + q_m^x B_m^m(\omega)))$$

and

$$\varsigma_i = \varsigma_i(\omega) = 2 \sum_{t=0}^{\chi} (B_m^i(\omega) (-y_r(\omega) - dy_{n_e}(\omega) + q_0^y B_m^0(\omega) + q_m^y B_m^m(\omega)))$$

q_0^x, q_0^y are the first control point's components of the original curve, q_m^x, q_m^y are the last point's components. Hoschek [6] introduced a comparable method with some further consideration of geometric continuity and some closer looks at error-handling.

Because differentiating the curve and calculating the points along the curve is slow the usage of Bernstein polynomial transformation could be wise.

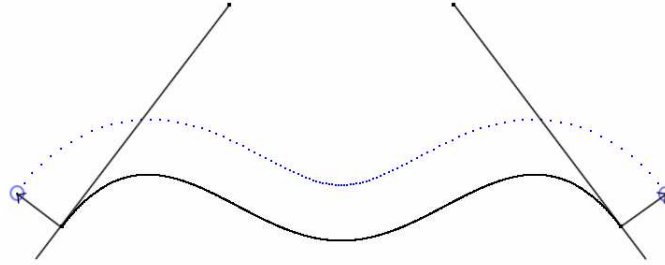


Fig. 1. The connection between the control points of the original and the offset curve

Note 1: $\vec{q}(t)$ is, in our case, a Bèzier curve, too. However, since the calculation is independent from the curve model it is possible to calculate other curve models than just Bèzier curves.

Finding the right number of new control points m of the offset curve can be a tough job. Usually a trial and error process could solve this problem. For some cases the number of control points of the original curve (+1,+2) is a good choice.

C. Polynomial least square fitting

We want to abandon from the classical Bèzier curve model. When allowing the least square fitting to result in general monom-based polynomials, two systems of linear equations will arise. Using the Vandermonde Matrix to drop some of the summations leads to two new matrix equations (one for every dimension). One of these equations is

$$\vec{f} = V \vec{a}$$

with $\vec{a} = (a_0^x, a_1^x, \dots, a_m^x)^T$ and

$$V \ni v_{i,j} = \left(\frac{i}{\chi} \right)^j \quad i = 0, 1, \dots, \chi \quad j = 0, 1, \dots, m$$

as well as $\vec{f} = (\nu_0, \nu_1, \dots, \nu_\chi)^T$ and

$$\nu_i = x_r(i/\chi) + dx_{n_e}(i/\chi)$$

The second equation is build in an equal way. The systems could be solved using any numerical method, using Gauss-Jordan ($m = \chi$) or, if the matrix equation $V^T \vec{f} = V^T V \vec{a}$ is well formed, by solving the direct formula

$$\vec{a} = (V^T V)^{-1} V^T \vec{f} \quad (7)$$

Note 2: In contrast to III-B polynomial least square fitting doesn't offer a fixture of the first and last control point but in comparison it offers a better numerical stability for a big χ because a lot of summations are not done. As a drawback the solutions loose on stability for high m (normally $m > 12$), since the matrix operations used in Formula 7 are numerical unstable for high m .

Note 3: Because of the abandonment of the classical form of a Bèzier curve this method offers a much higher speed if implemented in a computer program.

D. Integral error approximation

Formula 6 states the error φ between two curves. The motivation for solving this integral is beyond the scope of this paper. However, the approach might be used for offset calculation. Since a integral is nothing more than a infinite sum, $\varphi \rightarrow \min$ will result in two systems of equations as well as III-B did.

$$L \vec{R}_x = \vec{c}_x \quad L \vec{R}_y = \vec{c}_y$$

L consists of the following integral

$$L \ni l_{i,j} = -2 \int_0^1 B_m^i(t) B_m^j(t) dt \quad (8)$$

whichs solution is

$$l_{i,j} = \frac{-2}{1+i+j} \left(t^{1+i+j} \binom{m}{i} \binom{m}{j} {}_2F_1(1+i+j, i+j-2m, 2+i+j, t) \right) \Big|_0^1$$

where ${}_2F_1$ is a hypergeometric function, also known as Gauss's hypergeometric function. Another, and obviously better way of integration is to use the Bernstein Polynomial Transformation to transform the Bernstein Polynomials and then integrate them. This gives

$$l_{i,j} = 2 \binom{m}{i} \binom{m}{j} \sum_{g=0}^{m-i} \sum_{k=0}^{m-j} \frac{\binom{m-j}{k} \binom{m-i}{g} (-1)^{k+g}}{j+k+i+g+1}$$

$\vec{R}_x = (r_1^x, r_2^x, \dots, r_{m-1}^x)^T$ and $\vec{R}_y = (r_1^y, r_2^y, \dots, r_{m-1}^y)^T$ hold the control points of the offset curve. The vectors \vec{c}_x and \vec{c}_y are defined as

$$\vec{c}_x = (\alpha_1, \alpha_2, \dots, \alpha_{m-1})^T \quad \vec{c}_y = (\beta_1, \beta_2, \dots, \beta_{m-1})^T$$

with

$$\alpha_i = \alpha_i(t) = 2 \int_0^1 (B_m^i(t) (-x_r(t) - dx_{n_e}(t) + r_0^x B_m^0(t) + r_m^x B_m^m(t))) dt \quad (9)$$

as well as

$$\beta_i = \beta_i(t) = 2 \int_0^1 (B_m^i(t) (-y_r(t) - dy_{n_e}(t) + r_0^y B_m^0(t) + r_m^y B_m^m(t))) dt \quad (10)$$

Expanding the term α_i and applying the sum-rule changes it, too:

$$\alpha_i(t) = 2 \left(- \int_0^1 B_m^i(t) x_r(t) dt - d \int_0^1 B_m^i(t) x_{n_e}(t) dt + r_0^x \int_0^1 B_m^i(t) B_m^0(t) dt + r_m^x \int_0^1 B_m^i(t) B_m^m(t) dt \right)$$

with

$$\int_0^1 B_m^i(t) x_r(t) dt = \binom{m}{i} \sum_{k=0}^{m-i} \binom{m-i}{k} \sum_{j=0}^n \binom{n}{j} \sum_{h=0}^j \frac{(-1)^{h+j+k} \vec{P}_h(j)}{j+i+k+1}$$

and

$$\int_0^1 B_m^i(t) B_m^0(t) dt = \binom{m}{i} \sum_{j=0}^m (-1)^j \binom{m}{j} \sum_{k=0}^{m-i} \frac{(-1)^k \binom{m-i}{k}}{i+k+j+1}$$

and

$$\int_0^1 B_m^i(t) B_m^m(t) dt = \binom{m}{i} \sum_{k=0}^{m-i} \frac{(-1)^k \binom{m-i}{k}}{i+k+m+1}$$

The solution of $\int_0^1 B_m^i(t) x_{n_e}(t) dt$ is too complex to be obtained in this work, that's why a numerical integration of $x_{n_e}(t)$ could be useful. This can only be done by approximating the function. A closer look to the graphs of some example-curves reveals that the development of a Taylor-Series (even a very high-ordered) is useless. That's why only a (transformed) Lagrange interpolating polynomial $\sum_{j=0}^p b_j t^j \approx x_{n_e}(t)$ with the order n is capable of giving a useful approximation of the curve.

$$\int_0^1 B_m^i(t) x_{n_e}(t) dt \approx \sum_{j=0}^p b_j \binom{m}{i} \sum_{k=0}^{m-i} \frac{(-1)^k \binom{m-i}{k}}{i+k+j+1}$$

E. Examples

1) *Example 1:* A simple two-dimensional Bèzier curve defined by the following control points is given:

$$(-4, 2), (-2, -2), (0, 2), (2, 4), (4, -1), (2, -4), (0, -2), (-2, 2), (-4, -2)$$

Also the offset of degree m is given at a distance of $d = 1$; it was computed using least square method (III-B) with $\chi = 100$. In comparison another offset was derived using the integral approximation method described in III-D with $\chi = 100$ and $p = 8$. In addition to III-B we use polynomial least square fitting (III-C) to determine when and how numerical integration becomes a problem. The error between the perfect and approximated offset was derived by using the average error (formula 5) ε with $\lambda = 100000$. Table I confronts the three methods for different m 's. Both, the integral approximation as well as the method of polynomial fitting suffer from problems in numerical stability. Integral approximation already loses stability at $m = 12$, while polynomial fitting becomes unstable at $m = 14$. As a result of this only the classical least square method described in III-B is able to give a very small error. So if the application dictates a very small tolerance of error the least square method is the best solution. However, for lower m 's than 10 all methods offer a good approximation of the offset curve, while the polynomial fitting is always a step ahead of the others. This could be explained by the fact that it doesn't fix the first and the last control point and so is more free to align. Integral approximation offers the same error as the least square method but is much faster to compute. Figure 2 shows the offset curves for $m = 6$. Figure 3 explains the problem of numerical stability.

TABLE I
COMPARISON OF ERRORS

m	Least Square III-B	Integral aprox. III-D	Poly. fitting III-C
5	0.639567	0.639568	0.475383
6	0.25721	0.257197	0.201414
7	0.104892	0.104895	0.0980275
8	0.04991527	0.0499465	0.0434329
9	0.0491972	0.0492346	0.042636
10	0.042636	0.0412266	0.0303407
12	0.0218384	0.0400858	0.0248584
14	0.0171043	0.040525	0.409886
16	0.0106697	0.0677359	0.467254
18	0.00469752	1.4803	9.28474
20	0.00269357	8.30264	0.463058

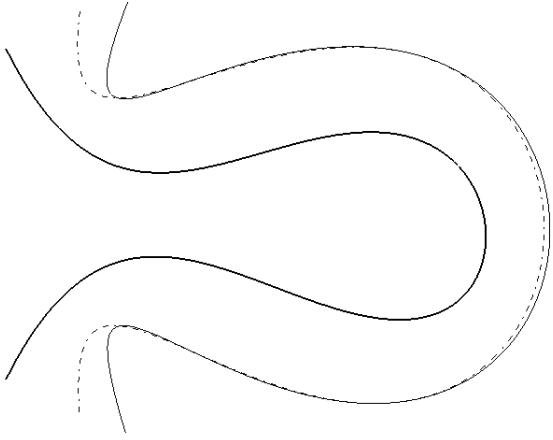


Fig. 2. Original curve and offset curves at $m = 6$; dotted line: Integral approx. and Least Square, continuous line: Poly. fitting

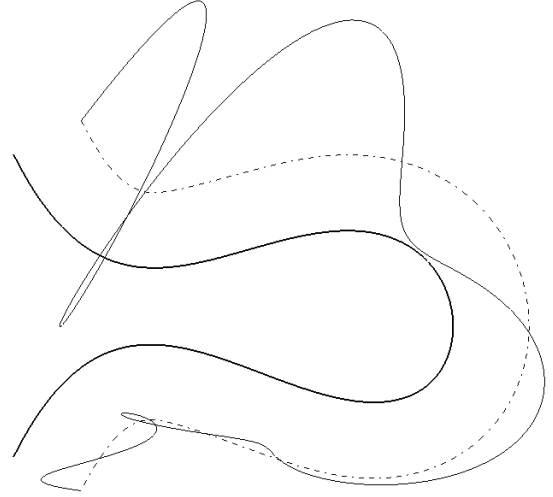


Fig. 3. Original curve and offset curves at $m = 18$; dotted line: Least Square, continuous line: Integral approx. The continuous line is numerical unstable

2) *Example 2:* A Bèzier curve $\vec{r}(t)$ with the following control points is given

$$(-4, -4), (0, -2), (4, -4), (4, 4), (0, 2), (-4, 4), (-4, -4)$$

Integral approximation (III-D) is used to approximate the offset. Table II shows how the number of points χ picked from the curve $\int_0^1 B_m^i(t)x_{n_e}(t)dt$ and the degree p of the interpolating polynomial affect the average error ε . The offset makes use of 10 control points. The spreadsheet shows that a high χ is not able to make a significant change to the error. It also shows us that it is important to select the highest possible p to get the best result. If the chosen p is too high, numerical stability will become a problem, since the matrix operations used to determine the fitting polynomial of $\int_0^1 B_m^i(t)x_{n_e}(t)dt$ are numerical unstable for a high number of coefficients. Figure 4 shows the original curve and its normal vectors as well as two offset curves.

TABLE II
THE COMPARISON OF ERRORS II

p	$\chi = 20$	$\chi = 100$	$\chi = 1000$
4	0.082235	0.0754217	0.0744377
6	0.0275552	0.0255903	0.0249442
8	0.0155298	0.0143054	0.0137077
10	0.00957423	0.00960248	0.0096366
12	0.0455372	0.0739152	0.0411033

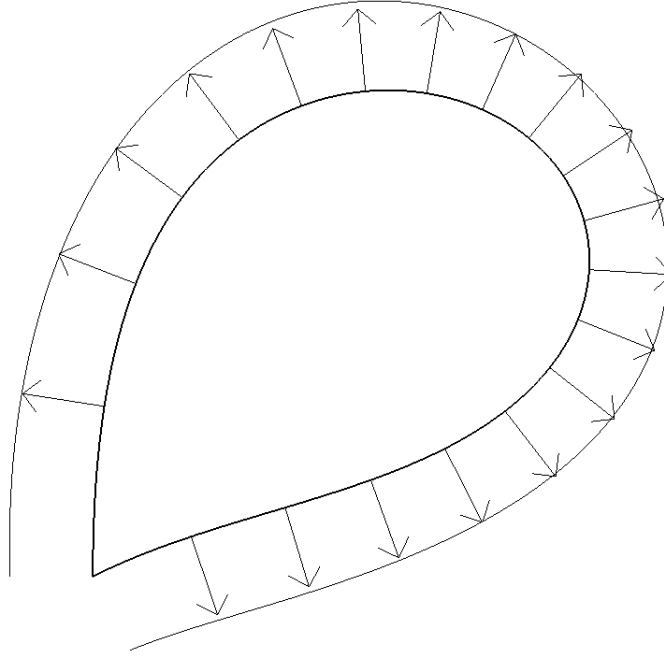


Fig. 4. Original curve (and normal vectors), Offset at $p = 8, \chi = 100$

IV. A BRIEF SURVEY TO THE BÈZIER SURFACE

Definition 4: A Bèzier Surface $\vec{s}(u, v)$ with the degrees m and n is given by

$$\vec{s}(u, v) = \begin{pmatrix} x_s(u, v) \\ y_s(u, v) \\ \vdots \end{pmatrix} = \sum_{i=0}^m \sum_{j=0}^n \vec{P}_{i,j} B_m^i(u) B_n^j(v)$$

$\vec{P}_{i,j}$ are the control points of the surface.

In analogy to Bèzier curves, the least square method could be used to derive a Bèzier surface's offset by minimizing the error between both, the perfect offset and the surface which we want to calculate.

$$\sum_{q=0}^{\chi} \sum_{r=0}^{\delta} \left| \vec{s} \left(\frac{q}{\chi}, \frac{r}{\delta} \right) - \vec{t} \left(\frac{q}{\chi}, \frac{r}{\delta} \right) \right|^2$$

with $\vec{t}(u, v) = \sum_{i=0}^m \sum_{j=0}^n \vec{T}_{i,j} B_m^i(u) B_n^j(v)$. Therefor three systems of linear equations need to be solved.

Each system holds the unknown control point's in the single dimensions.

$$M\vec{T}_x = \vec{d}_x \quad M\vec{T}_y = \vec{d}_y \quad M\vec{T}_z = \vec{d}_z$$

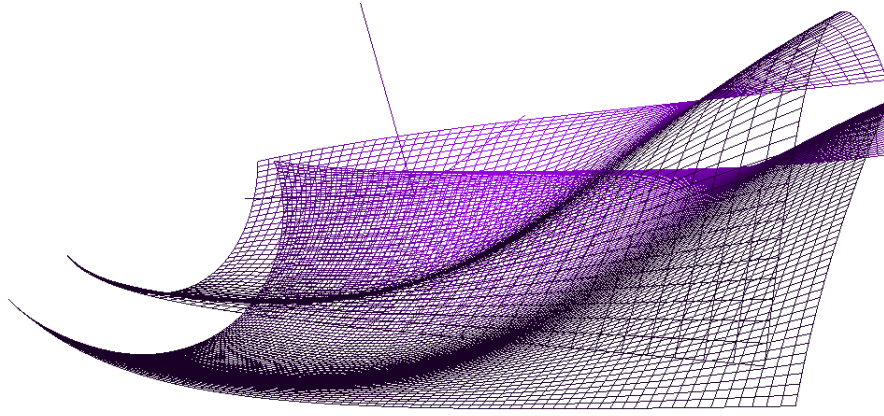


Fig. 5. Original surface (on the top) and the offset at a distance $d = 1.5$

with $m_{i,j} \in M$ as

$$m_{i,j} = -2 \sum_{u=0}^{\chi} \sum_{v=0}^{\delta} B_m^i(\gamma) B_n^j(\gamma) B_m^i(\kappa) B_n^j(\kappa) \quad \gamma = \frac{u}{\chi}, \kappa = \frac{v}{\delta} \quad (11)$$

for $i = j = 0, 1, \dots, mn$. The vectors \vec{d}_x , \vec{d}_y and \vec{d}_z are defined as

$$\vec{d}_x = (\psi_0, \psi_1, \dots, \psi_{mn}) \quad \vec{d}_y = (\varpi_0, \varpi_1, \dots, \varpi_{mn}) \quad \vec{d}_z = (\tau_0, \tau_1, \dots, \tau_{mn})$$

with $\psi_k =$

$$2 \sum_{u=0}^{\chi} \sum_{v=0}^{\delta} (B_m^i(\gamma) B_n^j(\kappa) (x_s(\gamma, \kappa) + dx_{ne}(\gamma, \kappa))) \quad (12)$$

and $\varpi_k =$

$$2 \sum_{u=0}^{\chi} \sum_{v=0}^{\delta} (B_m^i(\gamma) B_n^j(\kappa) (y_s(\gamma, \kappa) + dy_{ne}(\gamma, \kappa))) \quad (13)$$

and $\tau_k =$

$$2 \sum_{u=0}^{\chi} \sum_{v=0}^{\delta} (B_m^i(\gamma) B_n^j(\kappa) (z_s(\gamma, \kappa) + dz_{ne}(\gamma, \kappa))) \quad (14)$$

with $k = i + j$ but always $i \geq j$.

V. CONCLUSION

Three different methods based (and further developed) on the method of least squares, to derive offset of two-dimensional Bèzier curves have been introduced. All of them offer advantages, like calculation speed and disadvantages, like numerical instability in different variations. The integral error approximation (III-D) offers the best combination of speed and numerical stability. Furthermore it is highly variable and could give very precise offsets, as well as a very fast calculation processes. The method also could be optimized by finding a solution for the approximated sub-integral $\int_0^1 B_m^i(t) x_{ne}(t) dt$ if procurable at all or at least give a better numerical approximation. The Gaussian Quadrature [10] could help in this case.

The topic of Bèzier surfaces and their offsets was alluded. It was shown that it is also possible to derive a offset of a Bèzier surface, using the same methods used with the curves. Those methods could

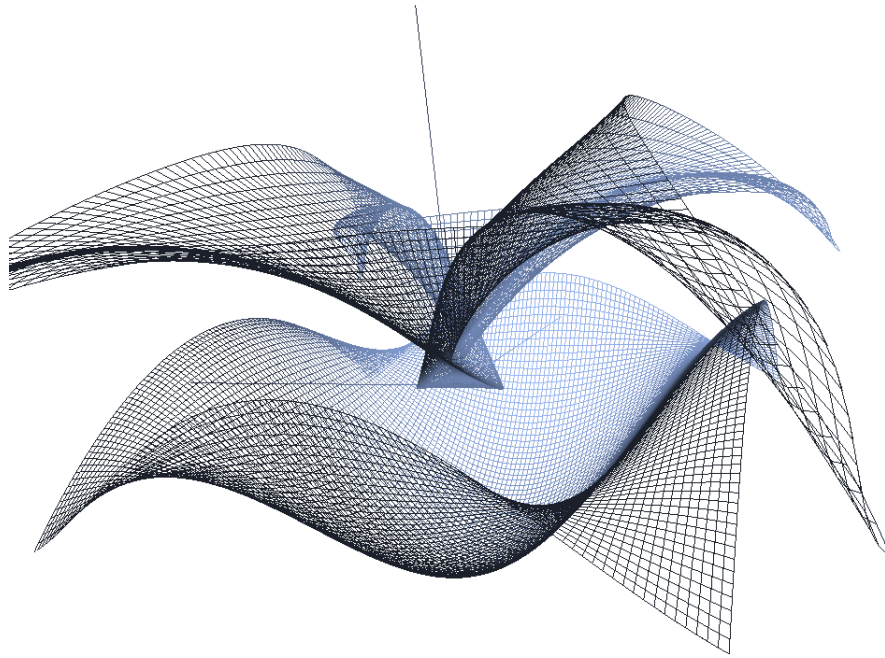


Fig. 6. Original surface and the (self intersecting) offset at a distance $d = 2.75$

be enhanced in the same way we enhanced the methods for the curve. By extending the formulas 11, 12, 13 and 14 to integral expressions much better offsets could be derived. Those integrals could be solved by applying Bernstein Polynomial Transformation to the surface. The transformation would be very equal to the one we used with the curves.

ACKNOWLEDGMENT

The authors would like to thank the Hangzhou Dianzi University for the good time while doing this work. We also acknowledge the Institute of Graphics and Image of Hangzhou Dianzi University for providing support and motivation for some of this work.

REFERENCES

- [1] Adrien-Marie Legendre. *Nouvelles méthodes pour la détermination des orbites des comètes*. 1805.
- [2] Takashi Maekawa. *An overview of offset curves and surfaces*. 1998.
- [3] J. Pham. *Computer Aided Design* 1992; 24(4):223-9. 1992.
- [4] Peter Chambers Alyn Rockwood. *Interactive Curves and Surfaces*. 1996.
- [5] Su K. Cho Hayong Shin. *Directional Offset of a 3D Curve*. 2002.
- [6] J. Hoschek. *Spline approximation of offset curves*. *Computer-Aided Geometric Design* 5. 1988.
- [7] G. Yu J. Wallner T. Sakkalis T. Maekawa H. Pottmann. *Self-Intersections of Offset Curves and Surfaces*. 2001.
- [8] T. Sederberg T. Nishita. *Curve Intersection using Bézier clipping*. *CAD*, Vol. 22 No. 9. 1998.
- [9] T. Nishita. *Applications of Bézier clipping method and their Java applets*, *Proceeding of Spring conference on computer graphics*. 1998.
- [10] Eric Weisstein. *Gaussian Quadrature*, *From MathWorld—A Wolfram Web Resource*. <http://mathworld.wolfram.com/GaussianQuadrature.html>. 2006.