International Journal of Computer Graphics & Animation (IJCGA) Vol.3, No.1, January 2013

Iterative Smoothing of Curves and Surfaces

John R Rankin
Computer Science and Computer Engineering, La Trobe University, Australia
j.rankin@latrobe.edu.au

Abstract

Algorithms for drawing smooth curves and surfaces are well-known but may be too expensive in CPU time to use for certain high speed interactive graphics applications. This paper looks at some new geometrically-based smoothing algorithms that arrive at the final smooth curve or surface after the application of an infinite number of iterations of the algorithm using subdivision refinement. When CPU time is scarce low iteration counts can be used to provide an acceptable level of approximation of smoothness and when CPU time is more plentiful higher levels of iteration can be applied for greater visual smoothing accuracy. Three new algorithms are presented and analysed. Convergence is proved geometrically for each and their timings are reported. Mixing iterations provides new opportunities for achieving various different effects in curve and surface smoothing.

Keywords

subdivision curve schemes, subdivision surface computations

1. INTRODUCTION

Smooth curve and surface rendering has a long history in graphics research [FOLEY et al. 1996] with the main emphasis being on spline algorithms [AHLBERG et al. 1967, de BOOR 1978]. Many applications today require animated smooth curves and surfaces in the background with smoothness accuracy not the highest priority in the application and CPU time allocated to several other tasks as well. This leaves little frame time for generating the smooth shapes and traditional high fidelity spline algorithms that produce perfect smoothness could use up too much of the scarce available time. Subdivision schemes were subsequently developed starting from the work of Chaikin [CHAIKIN 1974, JOY 1999] and have been extensively researched in relation to spline algorithms as refinement limit cases or as alternatives [eg JOY 2002, CATMUL et al. 1987, YAP 2006, KARCIAUSKAS 2010, LEVIN 2000, KOBBELT 2000, CASHMAN et al. 2009a & b, PETERS 1997, MULLER et al 1998, HERTZMANN et al. 2000]. In many algorithms the final smooth curve or surface and the starting polyline or polyhedron of control points are far apart: the control points being used only to control the final smooth curve or surface shape. In this paper we look at new algorithms for smoothing a polyline or mesh with the result as close to the polyline or mesh as possible. These are the inner, outer and central algorithms. The central includes the original vertices and is different from the incentre scheme of C Deng and G Wang [DENG et al. 2010], in particular in the definition of polyline vertex tangents. The algorithms are iterative where each iteration produces an improvement in smoothness. By using only a small number of iterations, a satisfactory level of smoothness is possible in less CPU time.

2. INNER SMOOTHING

A polyline is a sequence of line segments (called edges) joined end-to-end where adjoining line segments are not colinear and curve smoothing starts from a given polyline P of vertices P_i for i = 1 to n where n >

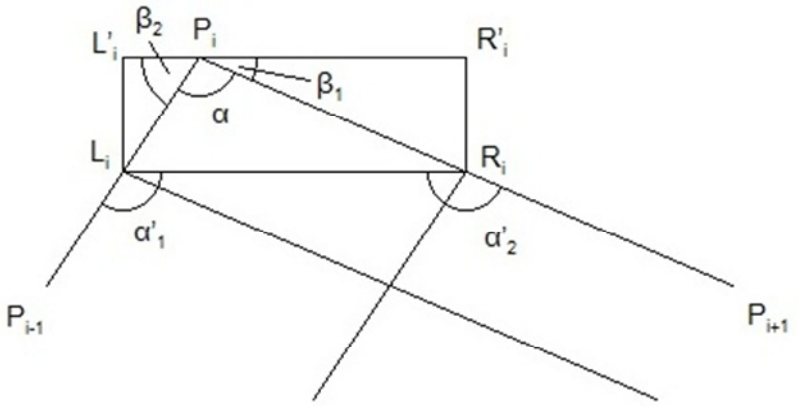
DOI : 10.5121/ijcga.2013.3103

2. (The cases for $n \leq 2$ are trivial and require no smoothing.) The polyline defines a sequence of n-1 edges $E_i = P_iP_{i+1}$ for i = 1 to n-1 and a sequence of n-2 angles α_i where α_i is the angle between edges E_i and E_{i+1} for i = 1 to n-2. From these definitions it follows that all polyline angles are in the range $(0,\pi)$. For the purposes of this paper, the smoothness $\sigma(P)$ of a polyline P is defined to be the minimum angle α_i in the polyline, divided by π (and sometimes expressed as a percentage). For any smoothing algorithm S, the algorithm is convergent if $\sigma(S(P)) > \sigma(P)$ for all polylines P. A fast algorithm that fulfills the requirement of convergence is Inner Smoothing which is now described.

For any polyline P, vertices P_1 and P_n are called the outer vertices and the other n-2 vertices are called the inner vertices. Each angle of P corresponds to an inner vertex. For inner smoothing, each inner vertex P_i is replaced by two vertices by the 2D lerping geometric operator:
$L_i = \text{Lerp}(P_i,P_{i-1},\lambda) \equiv P_i + \lambda(P_{i-1} - P_i)$
$R_i = \text{Lerp}(P_i,P_{i+1},\lambda) \equiv P_i + \lambda(P_{i+1} - P_i)$
so that the new polyline is $P' = \{P_1, L_2, R_2, L_3, R_3, \ldots L_{n-1}, R_{n-1}, P_n\}$.

The proportion factor λ is a constant of the inner smoothing algorithm in the range (0,0.5). The algorithm effectively chops off the corners at each inner vex. Each vertex corner is replaced by two other vertices whose angles we need to compare with the angle at the vertex chopped off in order to determine convergence. Figure 1 shows the new angles α'_1 and α'_2 that are formed. From Figure 1 we see that
$\alpha'_1 = \alpha + \beta_1$
$\alpha'_2 = \alpha + \beta_2$
where
$\alpha + \beta_1 + \beta_2 = \pi$
and $L'_iR'_i$ is the parallel transport of line segment L_iR_i to pass through inner vertex P_i so that $L_iR_iR'_iL'_i$ forms a rectangle. Since α is in $(0,\pi)$ and λ is in (0,0.5), β_1 and β_2 are both strictly positive and therefore both α'_1 and α'_2 are greater than α and yet both less than π. As α tends to its upper limit of π, β_1 and β_2 tend to zero and α'_1 and α'_2 also asymptote to π ahead of α. Therefore all angles of P' are greater than the corresponding angles of P and so the smoothness $\sigma(P')$ of the derived polyline $P' = S(P)$ is greater than the smoothness $\sigma(P)$ of the original polyline P and this means that the inner smoothing algorithm is convergent.

Figure 1. Three vertices in a polyline are shown, P_{i-1}, P_i and P_{i+1}. Points L_i and R_i are lerped from vertex P_i to form an inner smoothing.



.

Figure 2b shows the third iteration of inner smoothing for $\lambda = 0.3$. Every iteration of the inner smoothing algorithm reduces the length of every edge of the polyline. If in the initial polyline an edge between inner vertices is of length L_0, in the first derived polyline the corresponding edge will be reduced to length $L_1 = (1-2\lambda)L_0$. After i iterations the corresponding edge will be of length $L_i = (1-2\lambda)^i L_0$ which tends to zero as i tends to infinity because λ is in the range (0,0.5). The starting point for the edge of length L_1 is at distance λL_0 from P_k along inner edge E_k of the original polyiine (for the edge number k under consideration). After the second iteration, the starting point has moved further to distance $\lambda L_0 + \lambda L_1$ from P_k along the original edge E_k of the initial polyline P. After i iterations the starting point is at $\lambda L_0 + \lambda L_1 + \ldots + \lambda L_i$ from P_k and as i tends to infinity it is easy to demonstrate that this distance asymptotes to $L_0/2$. So therefore the limit curve of inner smoothing passes through the midpoints of all the edges as well as the outer vertices of the initial polyline P. These points are termed here "sticking points" where the derived polylines remain attached to the initial polyline. It was observed that after N = 3 iterations the changes in the derived curves are virtually too small to see. This leaves apparent discontinuities in slope for λ above 0.3 at the sticking points which are indeed smoothed however but at sub-pixel scales. For low values of λ less than 0.2 the curves appear (at larger than pixel scales) to smoothly join to a section of the original polyline's edges surrounding the sticking points on those edges as seen in Figure 2a. If $\lambda < 0$ then it is out of the valid range for λ and it generates "knots" in the curve. The knots can be avoided by reversing the order of L_i and R_i in the derived polyline P' for i = 2 to n-1. If $\lambda > 0.5$ then it is out of the valid range for λ and it generates "snitches" in the curve which are small introduced z-turns or cross-overs. The snitches can be removed by interchanging points R_i and L_{i+1} for i = 2 to n-2. Even so, there is no guarantee of convergence for these values of λ. The inner smoothing algorithm is more general than polygon or polyline rectification and the Chaikin curves. Polyline rectification corresponds to $\lambda = 0.5$ and points R_i are not used so that $P' = \{P_1, L_2, L_3, \ldots L_{n-1}, P_n\}$. Repeated rectification results in the Bezier curve whose sticking points are P_1 and P_n only. The Chaikin curves use $\lambda = 0.25$ and replace each edge with the derived pairs of points rather than replacing each inner vertex with the derived pairs of points as in the inner smoothing algorithm. The Chaikin curves do not therefore retain the P_1 and P_n as sticking points but do retain the midpoints of the original polyline edges as sticking points. Repeated application results in a quadratic uniform B-spline curve [RIESENFELD 1975].

Applying the inner smoothing algorithm to an initial polyline $P^{(0)}$ of n_0 vertices produces polyline $P^{(1)}$ of n_1 vertices and after N iterations produces polyline $P^{(N)}$ with n_N vertices where:

$$n_{i+1} = 2n_i - 2$$

or in general after N iterations

$$n_N = 2^N n_0 - 2^{N+1} + 2$$

The time for iteration i is approximated by

$$log_{10}(T_i) = a + bn_i$$

where the constants a = 0.79 and b = 0.42 have been experimentally determined from the data shown in Table 1. Based on the amount of frame time Δt available and the initial polyline size n_0 one can therefore compute the maximum number of iterations N possible by solving:

$$\Sigma_{i=1}^{N} T_i < \Delta t$$

For large N or n_0 this is approximately

$$N \approx log_2\left(\frac{(log_{10}\Delta t - a)/b - 2}{(n_0 - 1)}\right)$$

for smoothness with upper limit

$$\sigma_N \simeq 1 - 2^{-N}(1 - \sigma_0)$$

where σ_0 is the smoothness of the initial polyine. For example if $\Delta t = 0.5s$ and $n_0 = 5$, $\sigma_0 = 0.25$ then $N < 5$ iterations are possible and the final smoothness factor could be expected to be $\sigma_4 = 0.953$ which is visually acceptable as in Figure 2a. We should note that inner smoothing has the property that all derived polylines $P^{(j)}$ for j = 1 to N are inside the convex hull of the starting polyline $P^{(0)}$.

1	17
2	42
3	80.6
4	191
5	458
6	986
7	2958
8	13356
9	40010
10	160899

Table1. Iteration count and execution ticks (milliseconds) for a polyline of $n_0 = 5$ vertices.

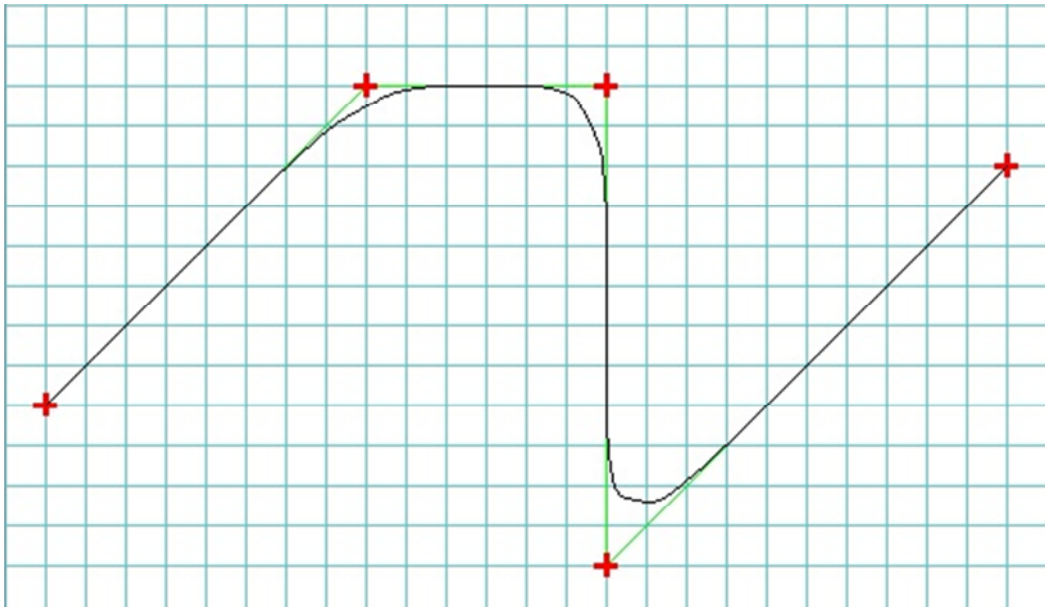


Figure 2a. Inner smoothing (green) for $\lambda = 0.15$ after i = 4 iterations

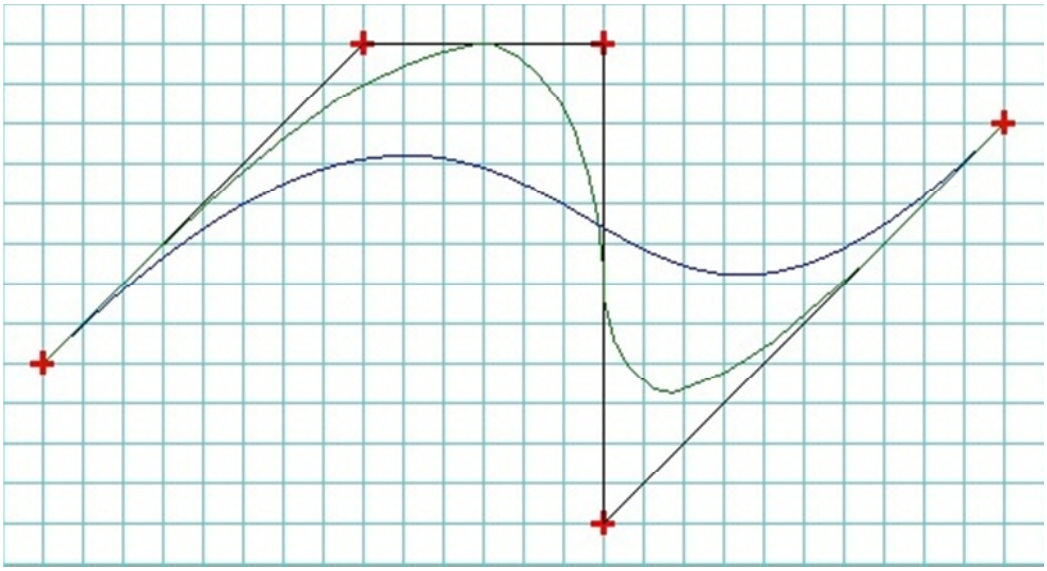
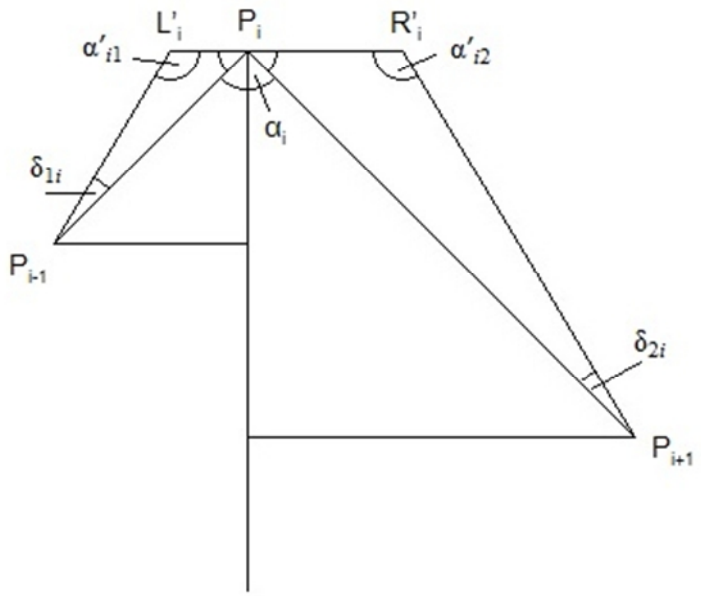


Figure 2b. Inner smoothing (green) for λ = 0.3 after i = 3 iterations vs Bezier (blue)

3. OUTER SMOOTHING

The name of the inner smoothing algorithm derives not from the fact that it uses the inner vertices of a polyline, but that if the polyline happens to be closed and forms a convex polygon, then the curves generated by the inner smoothing algorithm are all inside the polygon so formed. Also in general each polyline in the sequence of polylines generated by the inner iterations is inside the convex hull of the initial polyline. Sometimes however we need to generate a smooth curve which for a closed polyline forming a convex polygon, lies outside of this polygon. An algorithm to do this is here called an outer smoothing algorithm. A way of achieving outer smoothing is to construct a "tangent" at each inner vertex P_i. The tangent at an inner polyline vertex is here defined as a line $L'_iR'_i$ passing through inner vertex P_i such that the perpendicular to $L'_iR'_i$ through P_i bisects the polyline angle α_i at P_i. This is depicted in Figure 3. The vertices L'_i and R'_i now replace vertex P_i in the polyline. The outer smoothing algorithm, like the inner smoothing algorithm, introduces two new angles for the new polyline in replacement of the one angle at P_i. For convergence of the algorithm we need to see that these two angles are greater than the angle α_i at P_i and yet still less than π. For L'_i and R'_i as given in the inner smoothing algorithm and constant λ this condition is not true and therefore we introduce a different smoothing parameter called ν below.

Figure 3 The geometric construction of points L'_i and R'_i which replace vertex P_i in the outer smoothing algorithm.



Denoting angles $P_{i-1}L'_iP_i$ as α'_{1i}, angle $P_iR'_iP_{i+1}$ as α'_{2i}, $P_{i-1}P_iP_{i+1}$ as α_i and $P_iP_{i-1}L'_i$ as δ_{1i} and $R'_iP_{i+1}P_i$ as δ_{2i} we have (see Figure 3):

$$\alpha'_{i1}=\frac{\pi}{2}+\frac{\alpha_i}{2}-\delta_{1i}$$

$$\alpha'_{i2}=\frac{\pi}{2}+\frac{\alpha_i}{2}-\delta_{2i}$$

We will choose

$$\delta_{1i}=\delta_{2i}=\delta_i=\nu(\frac{\pi}{2}-\frac{\alpha_i}{2})$$

where ν is a small but positive constant in the range (0,1) so that

$$\alpha'_{1i}-\alpha_i=(1-\nu)(\frac{\pi}{2}-\frac{\alpha_i}{2})>0$$

Since ν is in (0,1), the factor (1-ν) is positive. Since $\alpha < \pi$, the second factor above is also positive and hence α'_{i1} is greater than α_i and similarly for α'_{i2}. In this way, the introduced angles α'_{1i} and α'_{2i} which replace α_i are each guaranteed to be greater than α_i and therefore the smoothness of the polyline σ(P) must increase with each iteration of the outer smoothing algorithm asymptoting to a smoothness of unity. Therefore the outer smoothing algorithm is convergent.

As with inner smoothing, the application of the outer smoothing algorithm once to polyline $P^{(0)}$ of n_0 vertices produces a polyline $P^{(1)}$ of n_1 vertices where $n_1 = 2n_0 - 2$ so that the formula for n_N remains the same as for inner smoothing. The ith iteration of the algorithm takes time T_i where $\log_{10}(T_i) = a + bn_i$ where the constants were experimentally found to be a = 1.16 and b = 0.398 for ν = 0.1 and $n_0 = 5$. So outer smoothing is computationally faster than inner smoothing for higher iteration values.

Figure 4 shows outer smoothing after 6 iterations for ν = 0.1. Clearly outer polylines are not contained within the convex hull of the starting polyline.

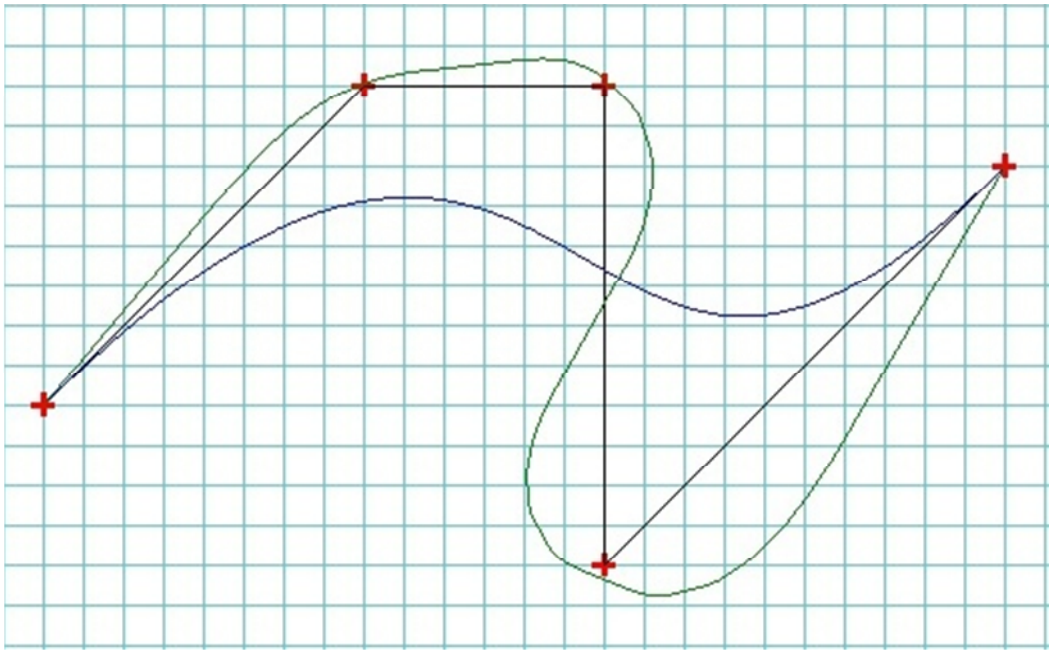


Figure 4. Outer smoothing (green) for v = 0.1 after N = 3 iterations vs Bezier (blue) for $n_0 = 5$.

4. CENTRAL SMOOTHING

The more iterations of inner smoothing that are done, the shorter the resultant smoothed curve total length is and the greater λ is the more it is pulled away from the original polyline except at the sticking points as seen in Figures 2a and b. After N = 4 iterations no further changes to the curve are visually apparent. Likewise the more iterations of outer smoothing that are done, the longer the resultant curve is and the larger v is the more it is pulled away from the original polyline in the other direction as seen in Figure 4. After N = 4 iterations no further changes to the curve are visually apparent. In both cases the original vertices are no longer part of the smoothed curve except for the two outer vertices P_1 and P_n which remain fixed. However in many applications we want the smoothed curve to continue to use the original vertices and not be pulled away from the original polyline. A central smoothing algorithm which achieves this is now described.

At each inner vertex P_i of the polyline P, construct a tangent which is a line $L'_iR'_i$ passing through P_i such that the perpendicular to $L'_iR'_i$ through P_i bisects the vertex angle α_i at P_i. The angle between the tangent $L'_iR'_i$ at P_i and the edge $P_{i-1}P_i$ is called here the tangent angle ϕ_i at P_i:

$$\phi_i = \frac{\pi}{2} - \frac{\alpha_i}{2}$$

From each inner vertex, a pair of new vertices A_i and B_i is generated where angles $A_iP_iP_{i-1}$ and $P_{i+1}P_iB_i$ equal $\theta_i = \mu\phi_i$ where μ is a real number in the range (0,1), and dropping A_i vertically onto P_iP_{i-1} gives point A'_i at proportion λ along P_iP_{i-1} and dropping B_i vertically down to P_iP_{i+1} gives point B'_i at proportion λ from P_i along edge P_iP_{i+1} as shown in Figure 5. The new polyline consists of the vertices P_1 A_2 P_2 B_2 A_3 P_3 B_3 … B_{n-1} P_n. The Central Smoothing iteration therefore converts a polyline of n vertices into a new polyline of n' = 3n - 4 vertices which include the n vertices of the previous polyline. Therefore after the ith iteration, the derived polyline has $n_i = 3^i n_0 - 2(3^i) + 2$ vertices.

The Central Smoothing algorithm generates 3 new angles for each vertex angle of the original polyline and in order to prove smoothing convergence, each angle must be compared with the original vertex

angle. We will set restrictions on λ in order to prove convergence. Figure 6 shows the new angle at A_2 which is called α'_2 and is given by

$$\alpha'_2 = \pi - \theta_2 - \theta_1$$

By requiring $\lambda < 0.5$ we ensure that $\theta_1 < \theta_2$ so that

$$\alpha'_2 - \alpha_2 > (1-\mu)(\pi - \alpha_2) > 0$$

Therefore the new vertex angle at A_2 exceeds the vertex angle α_2 at P_2. Considering the angle α_2 at P_2 and the new angle α'_3 at $P'_3 = P_2$ in the derived polyline, it is clear that since $\mu > 0$ it follows that $\theta_2 > 0$ and $\alpha'_3 = \alpha_2 + 2\theta_2 > \alpha_2$ and so the new vertex angle at P_2 (i.e. the angle at vertex P'_3 in the derived polyline P') exceeds the previous vertex angle at P_2.

Next we look at the possibilities for the angle at B_2. There are four cases to consider and Figure 7a shows the related triangles for the first of these. In this case $\theta_3 < \theta_2$ so that $h_2 > h_3$ and $\beta_3 > 0$. From $\lambda < 0.5$ it is easy to prove that $\theta_2 > \beta_3$. B_2 is vertex 4 of the next polyline P', and the angle at this vertex is therefore

$$\alpha'_4 = \pi - \theta_2 - \beta_3 > \pi - 2\theta_2 = \alpha_2 + (1-\mu)(\pi - \alpha_2) > \alpha_2$$

Hence the angle at B_2 in P' exceeds the angle at P_2 in P. The similar case shown in Figure 7b where $\theta_3 \geq \theta_2$ has $\beta_2 \geq 0$ and

$$\alpha'_4 = \pi - \theta_2 + \beta_2 \geq \pi - \theta_2 = \alpha_2 + (1-\mu/2)(\pi - \alpha_2) > \alpha_2$$

Therefore in this case also the angle at B_2 in P' exceeds the angle at P_2 in P. In cases three and four shown in Figures 7c and 7d, the new polyline crosses the old polyline and the angle at B_2 is $\alpha'_4 = \pi - \theta_2 - \beta_3$. In Figure 7c, $\theta_3 < \theta_2$ which results in $\beta_3 < \theta_2$ provided $\lambda \leq 0.25$ which leads to $\alpha'_4 - \alpha_2 > (1 - \mu)(\pi - \alpha_2) > 0$ and so $\alpha'_4 > \alpha_2$. In case 7d it can be proven from $\lambda < 0.25$ that $\beta_3 < \theta_3$. From this the following formula can be derived

$$\alpha'_4 - \frac{\alpha_2 + \alpha_3}{2} > (1-\mu)\left\{\pi - \frac{\alpha_2 + \alpha_3}{2}\right\} > 0$$

Since in this case $\theta_3 > \theta_2$ it follows that $\alpha_2 < \alpha_3$ so that

$$\alpha'_4 > \frac{\alpha_2 + \alpha_3}{2} > \alpha_2$$

In summary we have found that the new angle at B_2 exceeds the old angle at P_2 for all cases provided $\lambda \leq 0.25$. The treatment for the new angles at A_3, P_3 etc follow similarly and we conclude that the application of the Central Smoothing algorithm produces a smoother polyline that the previous polyline.

Figure 5 The construction of new edges A_iP_i and P_iB_i for generating the central smoothing algorithm.

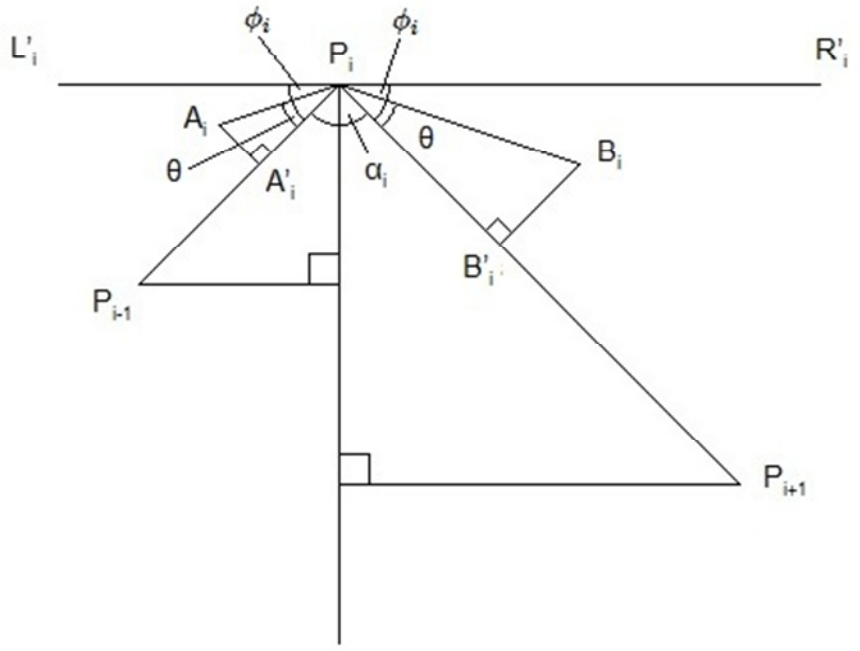


Figure 6 The angles relating to the new vertex A_2.

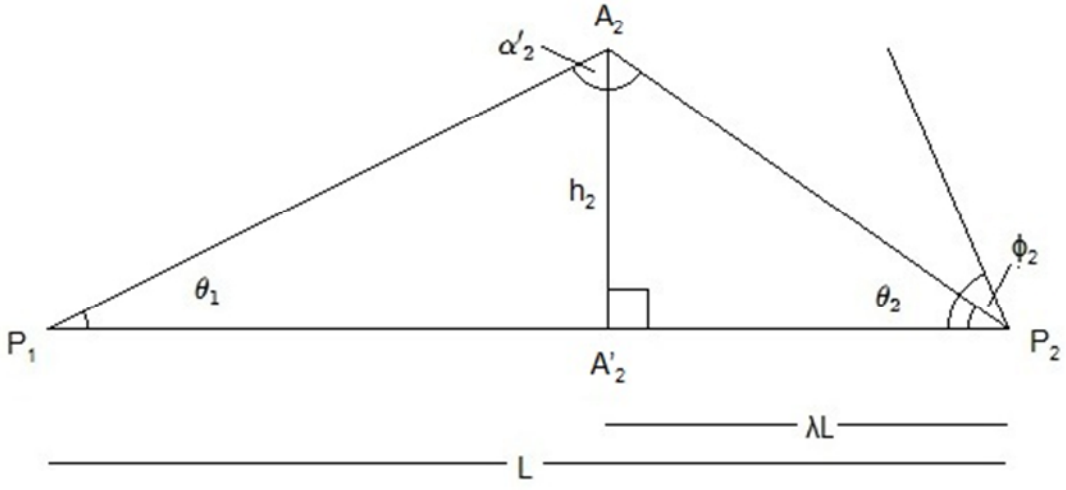
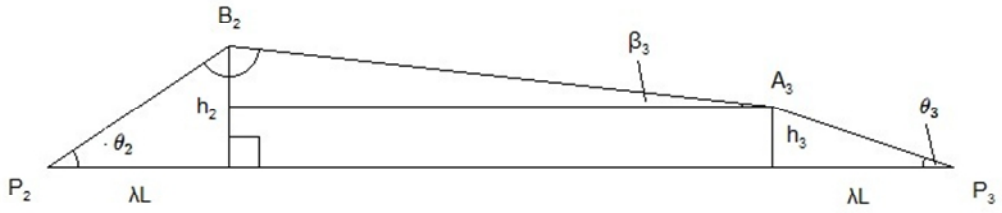
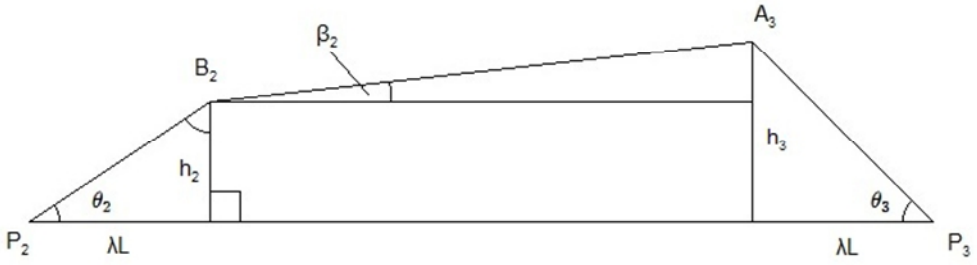
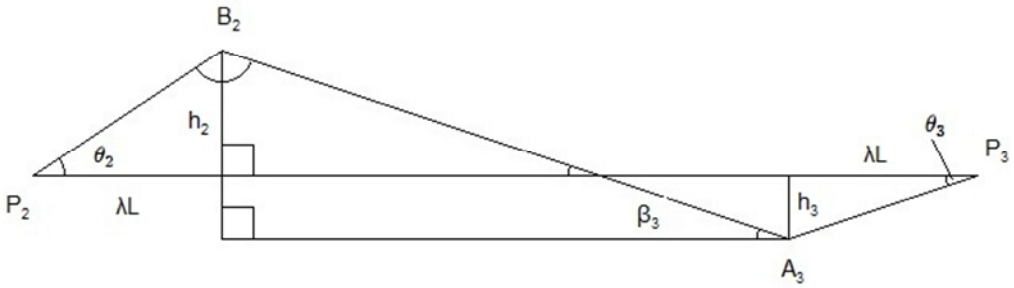


Figure 7a, b, c and d show the 4 cases to consider in determining the relative size of the polyline angle at B_2.


B2
β3
A3
h2
θ3
θ2
h3
P2
λL
λL
P3


A3
β2
B2
h3
h2
θ2
θ3
P2
λL
λL
P3


B2
h2
θ3
θ2
λL
P3
P2
λL
h3
β3
A3

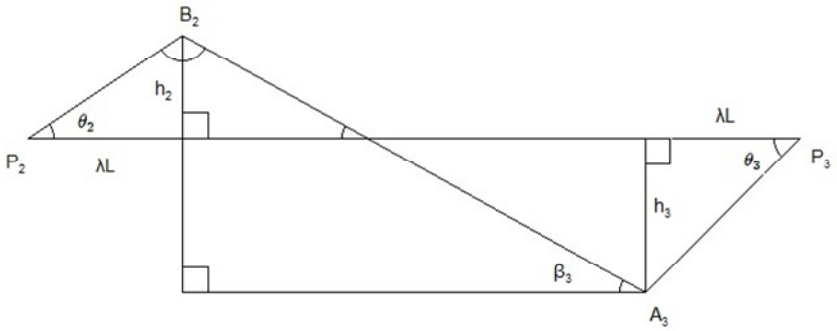
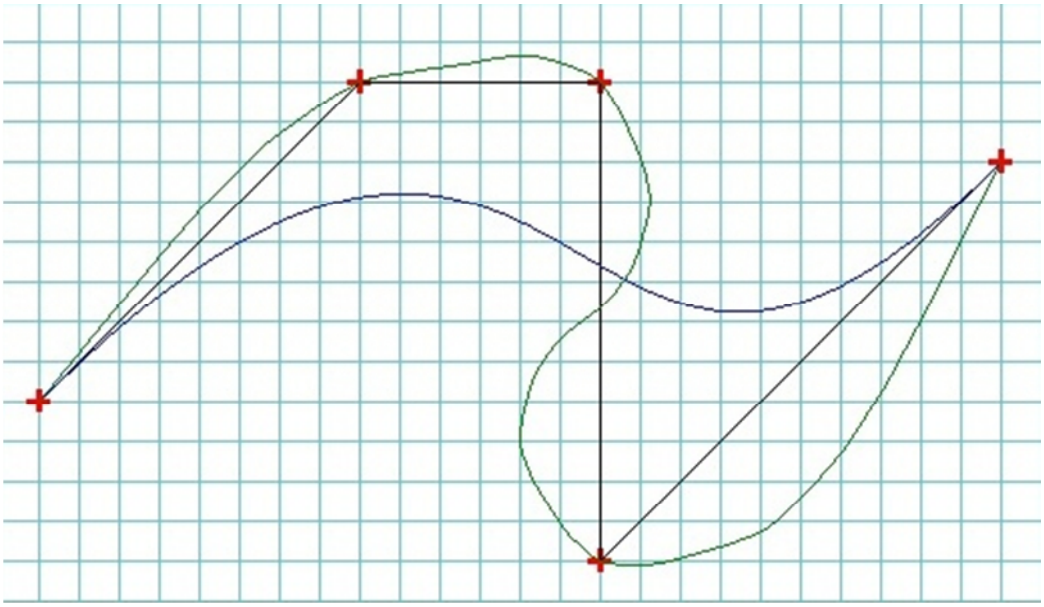

B2
h2
θ2
λL
P2
λL
θ3
P3
h3
β3
A3

Figure 8 below shows central smoothing for 4 iterations for $\mu = 0.5$ using the same initial polyline with $n_0 = 5$. Results showed the same formula for T_i with a = 0.60 and b = 0.75 and 4 iterations are sufficient for visual smoothness with no visible changes for i > 4..



5. MIXING THE ALGORITHMS AND 3D

The iterative approach to smoothing curves provides increased flexibility over how the final smoothed curve is developed: the curve parameters may be changed between iterations and we have the opportunity to mix and combine the algorithms at each iteration. For example inner smoothing after a single iteration of outer smoothing produces a smoothed curve which is closer to the original polyline than repeated outer smoothing produces. For each of these algorithms, inner, outer and central, there is also an inverse algorithm. The algorithm to reverse inner smoothing uses pairs of vertices forming alternate edges of $P^{(i)}$. These edges are extended and intersected pairwise and the intersection points replace the adjacent before and after vertices of $P^{(i)}$. The new vertices (including the two outer vertices) form the new polyline $P^{(i-1)}$. The central algorithm, like the in-centre algorithm [18], retains the vertices of the previous polyline yet differs markedly from that algorithm firstly in the definition of what is a tangent at a vertex, secondly in the number of vertices added per iteration and thirdly in how those additional vertices are formed. The tangent to a circular arc through P_{i-1}, P_i and P_{i+1} at P_i can differ from the perpendicular to the angle bisector at P_i by up to almost ninety degrees.

If a three-dimensional poyline (a polyline whose vertices are in three dimensions) is coplanar (i.e. all its vertices lie in the same plane) then application of any of the described iterative smoothing algorithms within its plane maintains this coplanarity property in the resultant polyline. To develop the 3D versions of the three smoothing algorithms for 3D polylines, the 3D polyline is broken into a sequence triplets of vertices $\{P_1,P_2,P_3\}$, $\{P_2,P_3,P_4\}$,... $\{P_{n-2},P_{n-1},P_n\}$. Each triplet defines a plane in 3D so a sequence of n-2 3D planes is thereby generated. The 2D planar algorithm of each of the three smoothing algorithms of 2D geometric constructions is applied in the sequence of planes to generate the 3D vertices for the derived 3D polyline. When the generated 3D points after N iterations are drawn, no 3D discontinuities appear. For the inner smoothing algorithm this only means generalizing the 2D point lerping formula in a straight-

forward manner to 3D point lerping. Outer smoothing additionally requires 3D lerping by distance rather than by proportion together with 3D parallel transport of the base of the isosceles triangle to the apex and this is easily implemented using the 3D point displacement geometric operator. The central smoothing algorithm also needs the construction of perpendiculars in 3D. The geometric operator for this takes two points A and B, the distance d and unit normal **n** to the plane containing A and B and returns point C in the plane which is at distance d from point B and such that BA and BC are perpendicular vectors. This is a straight-forward generalization of the 2D perpendicular geometric operator which does not have the parameter **n**. Generally 3D polylines are not coplanar and neither will the derived polylines be coplanar.

For surfaces, the cartesian product of the polyline smoothing algorithms is used. The initial surface is a height map above an mxn grid in the x-y plane. The grid has equally spaced lines in the x and y directions with a spacing of Δx in the x direction and a spacing of Δy in the y direction. Thus the 3D position of the (i,j) grid point is $(x_0+i\Delta x, y_0 + j\Delta y, 0)$ for i = 0 to m and j = 0 to n. The height of the surface above grid point (i,j) is z_{ij} giving the surface vertex $P_{ij} = (x_0 + i\Delta x, y_0 + j\Delta y, z_{ij})$. The surface can be drawn as m+n+2 polylines where m+1 polylines are parallel to the x-axis and n+1 polylines are parallel to the y axis. The polylines are 3D polylines and as the smoothing algorithm progresses these polylines change and may well become non-planar. The inner and outer smoothing algorithms have disadvantages for surface smoothing in that vertices of the polylines and therefore polylines of the surface are removed in each iteration and new ones added. However the central smoothing algorithm doesn't have this disadvantage: the existing polyline vertices and therefore their corresponding polylines of the surface remain and new ones are added. This means that central smoothing of surfaces is a process of adding more and more polylines across the surface producing a patchwork of quadrilaterals with increasing numbers and smaller sizes of quadrilaterals in each iteration. Since each quadrilateral might not be planar, it is rendered as two triangles by the diagonal (roughly) parallel to the plane y = x. The central smoothing algorithm produces more quadrilaterals per iteration than either the inner or the outer smoothing algorithms.

Figure 9(a) shows a random 10x10 height map illuminated by a mainly diffuse light source in OpenGL. In Figure 9(b) the surface of Figure 9(a) has had one application of the Inner Smoothing algorithm for λ = 0.3. Figures 9(c) and (d) show the results of a second and third application of the algorithm. In Figure 9(e) the refinements in smoothing are barely noticable so that 3 iterations are sufficient in this case.

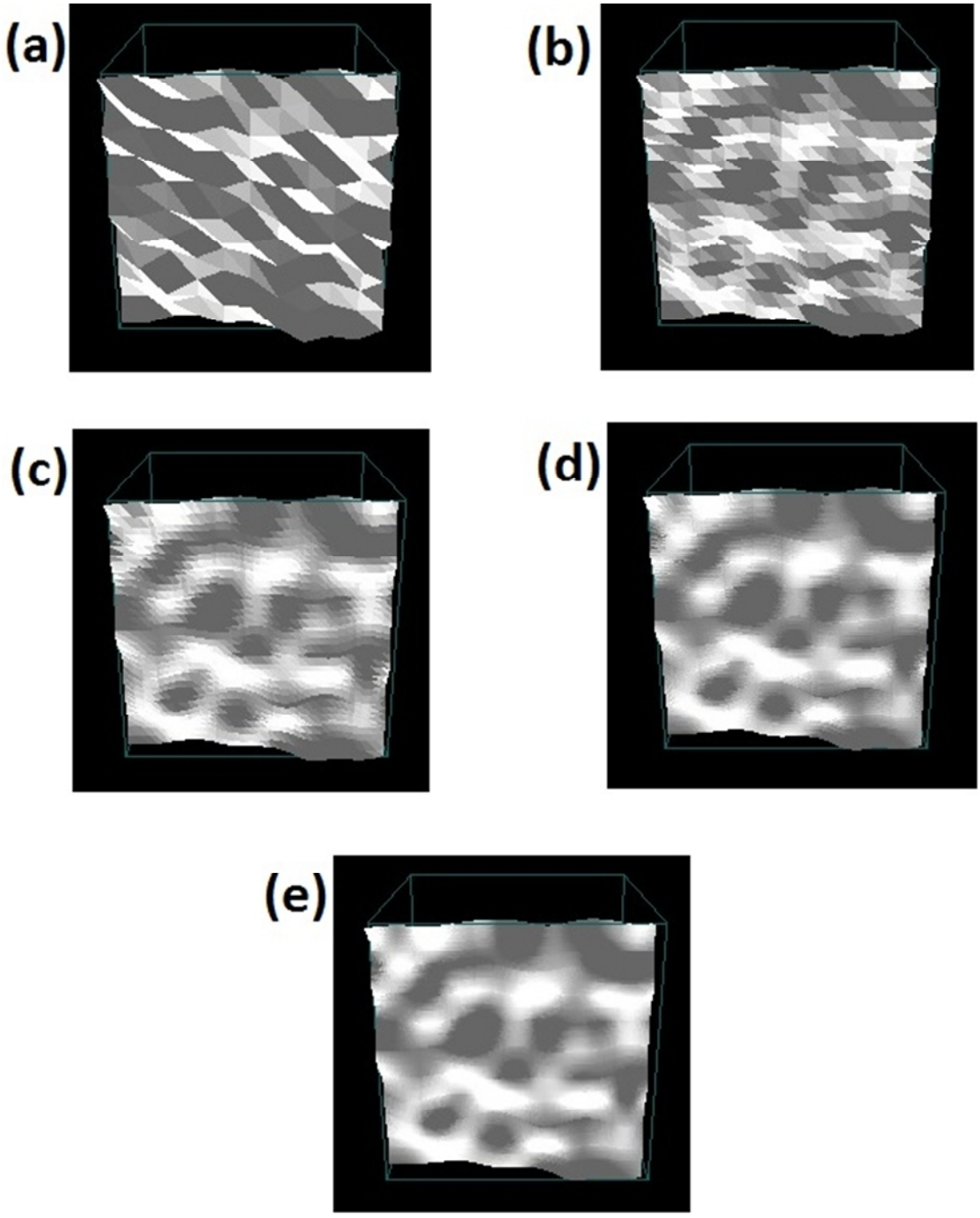


Figure 9

6. DISCUSSION

Inner smoothing produces a smoothed curve tightly fitting to the given set of initial points. Outer smoothing produces a smoothed curve which pulls away from the original polyline in the opposite side from the inner smoothed curve. Central smoothing on the other hand maintains the original points in the smoothed curve neither pulling away from the original polyline on one side or the other. The timing measurements done although precise to the tick have two sources or error. Firstly the measurements are made in an unavoidably multiprocess environment where other higher priority system processes can steal CPU cycles and at random moments of time so that repeating the tests can give different measurements and therefore average values over many tests were used. Secondly the timings included the times for construction, copying and destruction of dynamic arrays to hold the polyline data which changes at every iteration. By using the formulas for n_N and the timing results, we can decide on the maximum number of iterations N in advance and allocate sufficient memory in static arrays for the polyline data before any iterations are done. This would be more efficient than using dynamic arrays and save on the housekeeping chores involved with dynamic arrays.

It is noted that all three algorithms presented have local shape control and this can be used to construct the curves piecewise thereby using less memory allocations. Consider for instance the inner smoothing algorithm. Denoting the sticking points as $Q_i = \text{Lerp}(P_i,P_{i+1},0.5)$ for i = 1 to n-1 we note the following pieces of the limit curve. From P_1 to Q_1 the limit curve is the straight line segment P_1Q_1. The limit curve piece from Q_1 to Q_2 is uniquely determined by the initial polyline vertices P_1, P_2 and P_3 alone. The limit curve piece from Q_2 to Q_3 is uniquely determined by the initial polyline vertices P_2, P_3 and P_4 alone and so forth. From Q_{n-1} to P_n the limit curve is the straight line segment $Q_{n-1}P_n$. So the total curve can be rendered by computing and drawing these n+1 pieces in turn and this approach requires much less array space allocation. Likewise if a vertex of the original polyline is moved then only a part of the curve needs to be recomputed and redrawn. Moving P_1 changes Q_1 and requires recomputing and redrawing the curve from P_1 to Q_2 only i.e. the first two pieces. Moving P_2 changes Q_1 and Q_2 and therefore requires recomputing and redrawing only the first three pieces from P_1 to Q_3. In general moving inner vertex P_k means that three curve pieces must be recomputed and redrawn namely the pieces from Q_{k-2} to Q_{k-1}, from Q_{k-1} to Q_k and from Q_k to Q_{k+1}. Moving P_n requires recomputing and redrawing only the last two pieces from Q_{n-2} to Q_{n-1} and from Q_{n-1} to P_n.

The vertical projection of the initial height map to the plane z = 0 is a grid of regular spacing in the x and y directions. However after applying any of the three smoothing algorithms to the cross pattern of polylines on the surface, the vertical projection to z = 0 of the surface mesh will no longer be a regular equally-spaced grid of parallel and perpendicular lines. The polylines used to draw the surface have become general 3D polylines rather than planar polylines so their 2D projection is an irregular grid (although for the central smoothing, while most of the polylines will be non-planar and their projections irregular, the original polylines will remain and they will remain planar with rectilinear projections). Another observation to make about the smoothed surfaces developed by crossed sets of 3D polylines, is that the final mesh is not order dependent. The set of 3D mesh vertices produced by subdividing the x line polylines and then the y line polylines will be the same as the set of vertices produced by first subdividing the y polylines followed by subdivision of the x polylines. This is due to the symmetric nature of the formulas for multi-dimensional lerping.

7. CONCLUSIONS

Three algorithms for iterative smoothing have been presented: Inner Smoothing, Outer Smoothing and Central Smoothing. Each algorithm converges geometrically and generates after sufficient iterations different smoothed approximations of the original starting polyline or polyhedron. The curve smoothness

measure σ(P) defined in this paper, for visually acceptable smoothness has a value of about 0.95. The number of iterations for visually acceptable smoothness has been found to be very small such as N = 3 or 4. Iterative smoothing can consume less CPU time that traditional spline curve and surface methods. Since only a few applications of the algorithm are needed for visually acceptable smoothness, these algorithms can take less computation time than traditional methods such as Bezier curves and NURB surfaces. Additionally it is possible to predict in advance how many iterations N can fit into a given time interval so that smoothness computations can be tuned to the available time in an animation and sufficient memory space allocated in advance. Using iterative smoothing algorithms also allows for alternating between the algorithms with varying parameter values to gain different smoothed curve or surface effects.

REFERENCES

[1] FOLEY J D, VAN DAM A, FEINER S K, HUGHES J F "Computer Graphics Principles and Practice (Second Edition in C)",Addison-Wesley, 1996, pp 478-529.

[2] AHLBERG J H & NILSIN E N, "The Theory of Splines and Their Applicqations", Academic Press Inc, 1967

[3] de BOOR C "A Practical Guide To Splines", Springer-Verlag, 1978, pp113-114.

[4] CHAIKIN G, "An Algorithm For High Speed Curve Generation", Computer Graphics and Image Processing 3 (1974), 346-349.

[5] JOY K I, "Chaikin's Algorithms For Curves", On-line Geometric Modelling Notes, Visualization and Graphics Research Group, Department of Computer Science, University of California, Davis, 7pp, 1999.

[6] JOY K I, "Quadratic Uniform B-Spline Curve Refinement", University of California, Davis, Computer Science Department web pubication, 2002, 5pp.

[7] CATMUL E & CLARK J, "Recursively Generated B-Spline Surfaces On Arbitrary Topological Meshes", Computer-Aided Design, Vol 10, 6th of November 1987, pp 350-355.

[8] YAP C K "Complete Subdivision Algorithms, I: Intersection of Bezier Curves", SCG '06 Proceedings of the 22nd Annual Symposium on Computational Geometry, 2006 pp 217-226.

[9] KARCIAUSKAS K & PETERS J "Curvature of Approximating Subdivision Schemes", University of Florida web publication, pp 1-13 and Proceedings of Curves and surfaces'2010. pp.369~381.

[10] LEVIN D, WARTENBERG I, "Convexity-Preserving Interpolation by Dual Subdivisions Schemes", Saint-Malo Proceedings XXX, 2000, pp 1-10

[11] KOBBELT L, "Root 3 Subdivision", SIGGRAPH '00, Proceedings of the 27th annual conference on Computer graphics and interactive techniques, pp 103-112

[12] CASHMAN TJ, DODGSON NA, Sabin MA, “A symmetric non-uniform refine and smooth subdivision algorithm for general degree B-splines”, Computer Aided Geometric Design, 26 (2009) pp94-104.

[13] CASHMAN TJ, DODGSON NA, Sabin MA, “Selective knot insertion for symmetric non-uniform refine and smooth B-spline subdivision”, Computer Aided Geometric Design, 26 (2009) pp472-479.

[14] PETERS J, REIF U “The simplest subdivision scheme for smoothing polyhedra”, ACM Transactions on Graphics, 16, 4, Oct 1997, pp 420-431.

[15] MULLER H, JAESCHKE R “Adaptive Subdivision Curves and Surfaces”, Computer Graphics International 1998 Proceedings, 1998, pp 48-58.

[16] HERTZMANN A & ZORIN D, “Illustrating Smooth Surfaces”, SIGGRAPH ‘00 Proceedings of the 27th annual conference on Computer Graphics and Interactive techniques,10pp.

[17] DENG C, WANG G “Incentre Subdivision Scheme For Curve Interpolation”, Computer Aided Geometric Design, 27 (2010), pp 48-59.

[18] RIESENFELD R, “On Chaikin’s Algorithm”, IEEE Computer Graphics and Applications 4, 3 (1975), pp 304-310.