Chebyshev Polynomials for Boxing and Intersections of Parametric Curves and Surfaces

Alain Fournier and John Buchanan[1]

Department of Computer Science, University of British Columbia, Vancouver, British Columbia V6T 1Z4
{fournier | juancho}@cs.ubc.ca

Abstract

Parametric curves and surfaces are powerful and popular modelling tools in Computer Graphics and Computer Aided Design. Ray-tracing is a versatile and popular rendering technique. There is therefore a strong incentive in developing fast, accurate and reliable algorithms to intersect rays and parametric curves and surfaces.

We propose and demonstrate the use of Chebyshev basis functions *to speed up the computation of the intersections between rays and parametric curves or surfaces. The properties of Chebyshev polynomials result in the computation of better and tighter enclosing boxes. For surfaces they provide a better termination criterion to decide on the limits of subdivision, and allow the use of bilinear surfaces for the computation of the intersection when needed.*

The efficiency of the techniques used depends on the relative magnitude of the coefficients of the Chebyshev basis functions. We show from a statistical analysis of the characteristics of several thousands surfaces of different origin that these techniques will result most of the time in significant improvement in speed and accuracy over other other boxing and subdivision techniques.

Keywords: trimming curves, planarity criteria, ray-tracing, chebyshev polynomials.

1. Introduction

It is hardly necessary to motivate the development of algorithms to intersect parametric curves and surfaces with rays. Even in the context of other rendering methods, such as scanline based ones, boxing such curves and surfaces is a useful technique, and therefore efficient methods to do so is of interest.

Even though ray-tracing *per se* is not normally practiced in two dimensions, we will first develop the concepts and formulae in the plane, and then extend to three dimensions. In fact the 2D case itself has important applications, mainly in the use of *trimming curves* (parametric curves defined in the surface parametric space, where an *inside/outside* test is necessary).

1.1. Geometric vs Parametric Intersection

There can be two goals when computing the intersection with a parametric curve or surface. The first one, which can be called *geometric intersection*, is to obtain the *XYZ* world coordinates of the intersections. The other one, *parametric intersection*, consists in obtaining the *uv* parameters of the intersection. It is clear that the second "contains" the first, since it is easy to go from the parameters to the world coordinates, but the opposite is not true. In this respect the distinction is similar to the difference between geometric and parametric continuity (where the latter implies the former). Most applications require the values of the parameters, to compute the normals, essential to shading and reflection/refraction computations, and to apply texture

1. Current address: Department of Computer Science, University of Alberta.

mapping, where the texture is almost always indexed through the parameters. In the case of trimmed surfaces, the knowledge of the parameters is necessary to determine whether the point belong to the surface or not. It is why in the following we will only consider parametric intersection.

1.2. Direct Intersection

In *direct intersection*, the intersection is computed from some form of the polynomial representation of the surface. Two examples can be found in papers by Kajiya [1] and Sederberg and Anderson [2]. These methods can be very costly. It can be shown that the implicit formulation of the intersection between a ray and a polynomial Cartesian product patch is a polynomial of degree $2 \times M^2$, where M is the degree of the polynomial. This means a polynomial of degree 18 for a bicubic patch. The order of increase as a function of the degree also means that there is a clear advantage in lowering the degree of the polynomial as much as possible. In fact it is only for a bilinear patch, where the intersection equation is quadratic, that it is reasonable to intersect it directly. The intersection algorithms also can fail for high degrees, since they almost always use for efficiency root finding methods, such as Newton-Raphson iteration, which can fail to converge on a root. Another problem is that in their naive implementation they can spend a lot of time determining roots which are not real, or outside of the relevant parameter range. A good solution to the latter problem is *boxing*, where the curve or patch is surrounded by a convex figure easy to intersect, such as a convex polygon, such as the ray cannot intersect the patch if it does not intersect the box.

1.3. Subdivision Methods

In that category of method the curve or surface is subdivided, most often recursively, into simple elements such as polygons, which are then intersected with the ray. The method is most of the time accompanied by boxing. The efficiency of the method is critically dependent on the stopping criterion. In adaptive versions, a test is applied to decide whether to continue subdividing or whether to intersect the simple element (if its box is intersected). Most of the tests used are based of the "flatness" of the surface, that is on how close the patch is to a polygon. One of the problem is how to determine "flatness". This approach has been used in scan-conversion algorithms, by Catmull [3], by Clark [4] and by Lane and Carpenter [5]. The criteria used were straightness of the boundaries, difference between corner normals, etc. In each case it is easy to design examples of surfaces which will pass the test but are obviously not "flat". It should also be pointed out that if curvature is used directly, for many surfaces the curvature does not decrease as the subdivision increases (a cylinder or sphere is a common example), and therefore it is a poor stopping criterion for subdivision.

A different approach is to subdivide until the subpatches are small enough (usually determined with respect to the size of the pixels). An efficient way to subdivide has been introduced by Nishita *et al* [6] who used *Bézier clipping*, which eliminates portions of the surface where there are no intersections, until the remaining sections are small enough.

2. Boxing

As noted above, most implementations, no matter which method they use to compute the intersection, use some form of boxing. A box is an area (in 2D) or a volume (in 3D) which is guaranteed to completely enclose the curve or surface, so that if the ray does not intersect the box, it cannot intersect the curve or surface. If it does intersect the box, then the ray has to be tested further for intersection. This could be against the curve or surface itself, or against other boxes, if a hierarchy of boxes is used. A description and analysis of this approach is to be found in [7] and its application to parametric surfaces in [8] and [9]. A general discussion, with some of the results used here, is to be found in Chapter 6 of [10].

The correctness of the method depends on the fact that the contained object cannot be intersected if the box is not intersected. The efficiency of the method is dependent on two properties: testing intersection with the box is easier than testing intersection with the surface, and there are relatively few cases where the box is intersected, in other words the box is small. If at a given stage of the intersection process a ray/box intersection is performed at a cost C_B, the probability of intersecting the box is P_B, and the cost of the next action to be taken is C_N, the cost of this stage of the intersection is:

$$C = C_B + P_B \times C_N$$

The two box-dependent quantities are obviously C_B and P_B. The probability of intersecting the box is directly related to its size (we will make this statement more precise in the next sections), and it is clear that the size of the box cannot be less than the size of the enclosed object. In many circumstances there is a trade-off between the cost of intersection and the probability of intersection, as smaller boxes usually mean less simple boxes, which are harder to intersect.

If the next step in the case of a ray/box intersection is to intersect the surface itself, C_N can be many times the cost C_B. In this case even a small decrease in the probability of intersecting the box is worthwhile, even if it increases C_B. It is important to note, however, that in this case even a large box decreases the total cost. For example, assuming that $C_N = 10\ C_B$ (more realistic ratios are of the order of 500 to 1, see[11] for example) the cost of intersection with the box is less than C_N, the cost of intersection without the box, if $P_B < 0.9$. In other word even if almost 90% of the rays intersect the box, it is still useful, and this regardless of how loosely the box encloses the object inside.

If the next step in the intersection process is to test against another box (the box does not have to be nested or hierarchical), then the cost can be written:

$$C = C_{B1} + P_{B1} \times (C_{B2} + P_{B2|1} \times C_{N2})$$

In this expression, the subscript $B1$ refers to the first box, $B2$ to the second box, and $P_{B2|1}$ is the probability of intersecting the second box given that the first box has been intersected. C_{N1} is the cost of the next step using the first box, and C_{N2} the cost of the next step using the second box. It is easy to see that the second box makes the intersection testing more efficient if:

$$\frac{C_{B2}}{C_{N1}} < 1 - P_{B2|1}\,\frac{C_{N2}}{C_{N1}} \qquad \text{or in terms of the probability:} \qquad P_{B2|1} < \frac{C_{N1} - C_{B2}}{C_{N2}}$$

This is not a very stringent condition. Assuming $C_{N1} = C_{N2}$, which is the case if the next step in each case is to intersect the enclosed object directly, and as above $C_{N1} = C_{N2} = 10C_{B2}$, then, for the second box to be useful one has to have $P_{B2|1} < 0.9$. We will see later that in the case where $B1$ encloses $B2$, $B2$ has only to be less than 90% the size of $B1$ to be effective. Another question is whether it would be more efficient to use the second box directly. In this case one has to compare $C_{B1} + P_{B1} \times C_{N1}$ and $C_{B2} + P_{B2} \times C_{N2}$. The second box alone is better if:

$$\frac{C_{B2} - C_{B1}}{C_{N1}} < P_{B1} - P_{B2}\,\frac{C_{N2}}{C_{N1}}$$

If $C_{N1} = C_{N2} = C_N$, then it simply means that the relative increase in cost has to be less than the decrease in probability of intersection.

2.1. Probability of Intersection

To determine the probability of a ray intersecting an object[2], one has to make some assumptions about the distribution of rays. In the absence of additional information, we will assume that the unit direction vectors of the rays are equally distributed over the unit circle (sphere in 3D), and that the rays with the same direction vector are uniformly distributed over any finite length line segment (area in 3D) intersecting them. Note that we are actually concerned about the relative ray/object orientation. The above assumptions do not require that the rays are uniformly distributed in this fashion in object space (the primary rays, for instance, are far from equally distributed in all directions), but that they are if the rays are mapped to the objects own coordinate system (or if the objects are mapped into the rays coordinate system).

2. Since this applies to enclosed objects as well as to boxes, in the following discussion we will use the term "object" to designate either a box or its enclosed curve or surface.

Using the assumption about the distribution of rays with the same direction, the probability of rays with a direction (θ, ϕ) intersecting an object is proportional to the *projected area* of the object in the direction (θ, ϕ), and given by:

$$A_P(\theta, \phi) = \int_u \int_v G(u, v) \ [\ \mathbf{N}(u, v) . \mathbf{R}(\theta, \phi)\ ] \ dA$$

in which $G(u, v)$ is the visibility characteristic function, which is 1 if the point (u, v) is "visible" in the direction of the projection, and 0 otherwise, $\mathbf{N}(u, v)$ is the unit vector normal to the surface at (u, v), $\mathbf{R}(\theta, \phi)$ is the unit vector in the direction (θ, ϕ), given by $(\sin\theta \cos\phi, \sin\theta \sin\phi, \cos\theta)$ and dA is the differential of area, given by:

$$dA = \|\mathbf{N}(u, v)\| \ du \ dv$$

In the case of an unblocked polygon, the projected area simplifies to:

$$A_P(\theta, \phi) = A \cos\psi$$

where A is the area of the polygon, and ψ is the angle between the direction (θ, ϕ) and the normal to the polygon. Using the assumption about the distribution of directions, the *average projected area* $\bar{A}_P$ is:

$$\bar{A}_P = \frac{1}{4\pi} \int_{\theta=0}^{\pi} \int_{\phi=0}^{2\pi} A_P(\theta, \phi) \sin\theta \ d\theta \ d\phi$$

The term $1/4\pi$ is the normalization factor. In the case of a polygon, we can take, without loss of generality, a polygon whose normal is in the Z-axis direction, and consequently where $\psi = \theta$. If we count only the intersections on the "front" side of the polygon, we integrate only over the hemisphere and its average projected area is :

$$\bar{A}_P = \frac{1}{4\pi} \int_{\theta=0}^{\frac{\pi}{2}} \int_{\phi=0}^{2\pi} A \cos\theta \sin\theta \ d\theta \ d\phi = \frac{1}{4} A$$

A direct consequence of the above formula is that for a convex polyhedron its average projected area is one-fourth of its total area. In the case of a sphere, the average projected area is by symmetry the area of its great circle, πr^2, and its area is $4\pi r^2$, which gives the same ratio 4 of total area to average projected area than for a polygon. To convert the areas into probabilities, one has to divide the projected area by the total area covered by the rays in the (θ, ϕ) direction, and then include this divisor in the expression for the average projected area. In the simplest case, if the area covered by the rays is the equal in all directions (that is the rays are equally distributed within a sphere), then all the areas are divided by a constant (the area of the great circle of the sphere) to be converted into probabilities.

In the case of nested boxes, one has to determine the probability of intersecting a box given that the ray has already intersected another one. It is easy to see that for a direction (θ, ϕ) it is the ratio of the projected area of the intersection of the two projections over the projected area of the first one. In the case where the second box is totally enclosed within the first, the probability is simply the ratio of the two projected areas.

2.2. Cost of Intersections

It is interesting to compare the cost of intersecting various types of boxes, and the cost of intersecting a bilinear patch. While there is no claim that the number of operations we obtained are minimal, they have been carefully coded, and they represent the actual cost in our implementation. The relative cost of the floating point operations For a SGI Crimson with a MIPS R4000 processor and a MIPS R4010 floating point co-processor, the values are as in the following table. The square root is a function call, and this reflects in a rather high cost. The actual cost depends on the value, and we have taken an average for the range of values found in this application. "Non Aligned Rect." refers to an orthogonal parallelepiped whose edges are not parallel to the coordinate axes. "Aligned Rect." refers to the same object with edges parallel to the coordinate axes.

Operation	Plus/Minus	Multiply	Divide	Square Root	Total
Relative Cost	1	1	4	50	
Sphere	10	9	0	0	19
Aligned Rect.	9	6	3	0	27
Non Aligned Rect.	24	24	3	0	60
Bilinear Patch (one root)	37	77	1	1	168

There are many interesting conclusions stemming from these numbers. One is that boxing a bilinear patch with a sphere pays off if less than (168-19)/168 = 89% of the rays intersect the sphere. Another is that for a non-aligned rectangular box to be better than a sphere, the probability of intersecting the sphere has to be (60-19)/168 = 24% more than the probability of intersecting the box.

3. Chebyshev Polynomials

The basis polynomials used with parametric curves and surfaces are chosen primarily because they facilitate the design process, and additionally because they are easy to evaluate. There is no reason to believe that the same formulation will facilitate the computation of the intersection with a ray (or any intersection, for that matter). Since the design and the rendering are normally two separate phases, and the rendering is done many times for every design change, there is little or no penalty incurred in a system if the internal representation uses a polynomial basis different from the one used for design. In fact this is widespread practice: for example systems using β-splines often use Bézier-Bernstein polynomials for rendering, since these give simpler subdivision formulae. It is therefore worth exploring if there is some polynomial basis (we want a basis so that the same space of curves or surfaces is represented) which would speed up intersection calculations.

3.1. Definition

Chebyshev polynomials are orthogonal polynomials usually denoted $T_n(x)$ such that:

$$T_0(x) = 1 \qquad\qquad T_1(x) = x$$

and with the recurrence relation:

$$T_n(x) = 2\ x\ T_{n-1}(x) - T_{n-2}(x)$$

It is immediate that each polynomial $T_n(x)$ is of degree n. A remarkable relation makes apparent many of their interesting properties:

$$T_n(\cos\theta) = \cos(n\ \theta)$$

The polynomials are best used when the parameter varies in the closed interval $[-1, 1\ ]$.

Since Chebyshev polynomials are orthogonal, any polynomial of degree $\le n$ can be written as a linear combination:

$$P_n(x) = \sum_{k=0}^{n} a_k\ T_k(x) \tag{1}$$

In the case of a polynomial of degree 3, using standard notation:

$$P_3(t) = [t^3\ t^2\ t\ 1\ ][T\ ]\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

The 4x4 matrix $[T\ ]$ is:

$$T = \begin{bmatrix} 0 & 0 & 0 & 4 \\ 0 & 0 & 2 & 0 \\ 0 & 1 & 0 & -3 \\ 1 & 0 & -1 & 0 \end{bmatrix}$$

In geometric modelling with parametric curves and surfaces, the parameter(s) range is usually [0, 1], so to convert a standard parametric representation from their original basis to the Chebyshev basis, one has to use the following matrix to convert from the [0, 1] range to the [–1, 1] range:

$$R = \begin{bmatrix} 8 & 0 & 0 & 0 \\ -12 & 4 & 0 & 0 \\ 6 & -4 & 2 & 0 \\ -1 & 1 & -1 & 1 \end{bmatrix}$$

This matrix premultiplies $[T\ ]$.

To obtain the Chebyshev coefficients as a column vector $[A\ ]$ from any basis representation whose coefficient column vector is $[P\ ]$ (often called the control points), whose 4x4 matrix is $[M\ ]$ (the unit matrix in the case of the power basis), and whose parameter is in the range [0, 1], one has to compute:

$$[A\ ] = [T\ ]^{-1}\ [R\ ]^{-1}\ [M\ ]\ [P\ ]$$

where

$$R^{-1} = 1/8 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 3 & 2 & 0 & 0 \\ 3 & 4 & 4 & 0 \\ 1 & 2 & 4 & 8 \end{bmatrix} \qquad T^{-1} = 1/4 \begin{bmatrix} 0 & 2 & 0 & 4 \\ 3 & 0 & 4 & 0 \\ 0 & 2 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

The product of the 3 matrices has only to be computed once for a given basis. For a surface, $[A\ ]$ and $[P\ ]$ are matrices, and the computation is:

$$[A\ ] = [T\ ]^{-1}\ [R\ ]^{-1}\ [M\ ]\ [P\ ]\ [M\ ]^{T}\ [R\ ]^{-T}\ [T\ ]^{-T}$$

If we call $[C\ ]$ the basis conversion matrix:

$$[C\ ] = [T\ ]^{-1}\ [R\ ]^{-1}\ [M\ ]$$

then the transformation is:

$$[A\ ] = [C\ ]\ [P\ ]\ [C\ ]^{T}$$

The basis conversion matrices for the power basis $[C_p\ ]$ and the Bézier basis $[C_b\ ]$ are

$$C_p = 1/32 \begin{bmatrix} 10 & 6 & 6 & 10 \\ -15 & -6 & 3 & 15 \\ 6 & -6 & -6 & 6 \\ -1 & 3 & -3 & 1 \end{bmatrix} \qquad C_b = 1/32 \begin{bmatrix} 10 & 12 & 16 & 32 \\ 15 & 16 & 16 & 0 \\ 6 & 4 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

3.2. Properties

The basic properties of Chebyshev polynomials are to be found in [12]. The most important in our context is the *minimax property*. From Equation (1) any polynomial $P_n(x)$ of degree $\le n$ can be written as a linear combination of $T_i(x)$. If we drop the $T_n(x)$ term from that sum, we obtain a polynomial of degree $\le n-1$:

$$P^{*}_{n-1}(x) = \sum_{k=0}^{n-1} a_k\ T_k(x)$$

$P^*_{n-1}(x)$ has the property that of all the polynomials of degree $\leq n-1$ the maximum of the absolute difference:

$$E_n = MAX(|P_n(x) - P^*_{n-1}(x)|)$$

is minimum over the interval $[-1, 1]$. Moreover, $E_n = |a_n|$, that is the maximum difference is the absolute value of the coefficient of the Chebyshev polynomial dropped. This property is central to the use of these polynomials in boxing and intersection. It means that if we want to replace a polynomial with another of smaller degree, then the way to minimize the maximum error is to convert the polynomial into its Chebyshev representation and drop the Chebyshev polynomial of highest degree.

A problem occurs because in practice we often want to reduce the degree by 2, for example from a cubic to a straight line segment. In general Chebyshev polynomials do not give the best approximation. In the case of a reduction by 2 degrees, the answer are polynomials whose value depends on the ratio of the coefficients of the two highest degree monomials, and are called *Zolotarev* polynomials of order n (if the original polynomial is of degree $n+1$) [12]. Fortunately Chebyshev polynomials are close approximations of the best, and, for instance, in the case of going from degree 3 down to degree 1, the Chebyshev approximation is $a_0 + a_1 x$, and we have: $|P_{best}(x) - (a_0 + a_1 x)| \leq |a_2| + |a_3|$. Since it is so easy to compute the reduced polynomials, and as we will see the cost of not having the best possible polynomial is minor, Chebyshev polynomials still should be used where they are not theoretically optimal.

4. Subdivision

In most algorithms it will be necessary to subdivide curves and surfaces. One could of course subdivide in whatever basis the curves are initially represented, or other basis efficient for subdivision, but the conversion to and from the Chebyshev basis would more than offset the savings. Moreover subdividing using the Chebyshev basis directly is relatively easy.

4.1. Subdivision of Curves.

Given a curve C defined by the Chebyshev coefficients (a_0, a_1, a_2, a_3), the following matrices, when applied to the coefficient vector will produce the coefficients of the two half curves:

$$M_{(-1,0)\to(-1,1)} = \begin{bmatrix} 1 & 1/2 & -1/4 & -1/4 \\ 0 & 1/2 & 1 & 3/8 \\ 0 & 0 & 1/4 & 3/4 \\ 0 & 0 & 0 & 1/8 \end{bmatrix} \qquad M_{(0,1)\to(-1,1)} = \begin{bmatrix} 1 & -1/2 & -1/4 & 1/4 \\ 0 & 1/2 & -1 & 3/8 \\ 0 & 0 & 1/4 & -3/4 \\ 0 & 0 & 0 & 1/8 \end{bmatrix}$$

If the coefficients of the two sub-curves are called A' and A'', where A' corresponds to the $[-1, 0]$ interval in parameter space and A'' corresponds to $[0, 1]$, then the computation of the new coefficients is made more efficient by the use of temporary variables:

$$t_0 = a_0 / 8\ ,\ t_1 = a_3 / 4\ ,\ t_2 = t_1 * 4\ ,\ t_3 = t_1 + t_0\ ,$$

$$t_4 = a_2 / 4\ ,\ t_5 = a_0 - t_4\ , \qquad t_6 = a_1 / 2,\ \ t_7 = t_6 - t_1\ , \qquad t_8 = t_6 + t_3\ ,$$

$$a'_0 = t_0\ , \quad a'_1 = t_4 - t_1\ , \quad a'_2 = t_8 - a_2\ , \quad a'_3 = t_5 - t_7\ ,$$

$$a''_0 = t_0\ , \quad a''_1 = t_4 + t_1\ , \quad a''_2 = t_8 + a_2\ , \quad a''_3 = t_5 + t_7\ ,$$

5. Boxes from Chebyshev Polynomials

In what follows, we assume that we use some cubic parametric representation for curves (bicubic for surfaces), which has been converted to Chebyshev polynomials using equation (1) for curves or (2) for surfaces. This conversion of course is done only once. Figure 1 shows an example of a Bézier curve with its various boxes.

If we apply the Chebyshev degree reduction to each of the coordinates separately, we do not necessarily obtain a minimum maximum Euclidean distance. However we still can exploit the basic property when we use a rectangular box. It is better shown in 2 dimensions. Assume we reduce the parametric polynomials to degree 1. The curve is then reduced to a line segment. Rotating the curve until this line segment is parallel to the X or Y axis, one obtains from the magnitude of the terms dropped a box whose dimensions is such that the curve is guaranteed to be inside, and is the smallest possible if each coordinate polynomials is the best approximation.

5.1. Boxes in 2D

Given a curve defined by the Chebyshev coefficient vectors $(\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3)$, three bounding boxes are defined. A circle, an aligned rectangle (whose sides are parallel to the axes) and a non-aligned rectangle (in a direction defined by the Chebyshev line segment).

Circle

The centre of the circle is defined by the point $\mathbf{P}_{centre} = \mathbf{a}_0$ and the radius is[3] $radius = \| \sum_{i=1}^{3} |\mathbf{a}_i| \|$

Rectangle aligned with axes

From the curve definition $(\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3)$ the Chebyshev line segment is $\mathbf{l} = (\mathbf{a}_0 - \mathbf{a}_1, \mathbf{a}_0 + \mathbf{a}_1)$. The aligned rectangle is constructed by extending the min-max bounding box for the Chebyshev line segment by α_x and α_y, where

$$\alpha_x = |a_{2_x}| + |a_{3_x}| \quad \text{and} \quad \alpha_y = |a_{2_y}| + |a_{3_y}|$$

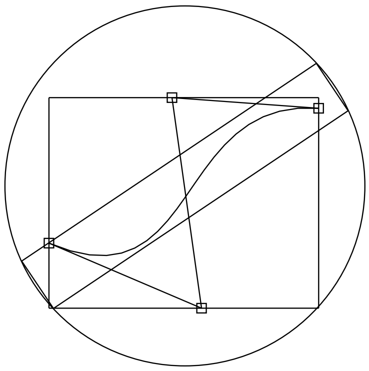

Figure 1. Cubic curve with associated bounding boxes.

Non-aligned rectangle

This rectangle is aligned with the Chebyshex line segment. We rotate the control points $\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$ so that the resulting Chebyshev line segment $\mathbf{l}'$ lies along the X axis. Call the angle of rotation Θ. If $\mathbf{l}' = ((X_{\min}, 0) , (X_{\max}, 0))$, $\delta_x = |a'_{2_x}| + |a'_{3_x}|$ and $\delta_y = (|a'_{2_y}| + |a'_{3_y}|$ the bounding box B' is defined by the bottom left and upper right corner.

3. Here the absolute value $|\mathbf{a}_i|$ means the absolute value of each of the coordinates of $\mathbf{a}_i$.

$$B' = [(X_{\min} - \delta_x\ ,\ - \delta_y), (X_{\max} + \delta_x\ ,\ \delta_y)]$$

The bounding box B for the curve in its original position is then obtained by rotating B' by $-\Theta$.

5.2. Termination Criteria

The Chebyshev line segment is an acceptable approximation of the cubic curve with error $< \varepsilon$ when:

- $\| (\delta_x\ ,\ \delta_y\ )\| < \varepsilon$ in the case of the non-aligned rectangle
- or $\|(\alpha_x\ ,\ \alpha_y\ )\| < \varepsilon$ in the case of the aligned rectangle.

Note that it does not necessarily mean that the curve is "almost straight". It only means that it is never far from the Chebyshev line segment.

5.3. Boxes in 3D

Given a patch defined by the Chebyshev vectors $(p_{00},\ p_{01},\ \cdots,\ p_{33})$, three bounding boxes are easy to define: a sphere, a rectangular parallelepiped aligned with the axes, and a rectangular parallelepiped aligned with the Chebyshev bilinear approximation of the patch (see Figure 2 for illustration).

Figure 2. Patch with enclosing sphere and parallelepiped.

Sphere

The sphere is defined by a centre point $\mathbf{P}_{centre} = \mathbf{P}_{00}$, and

$$radius = \|(\sum_{i=0}^{3} \sum_{j=0}^{3} |\mathbf{P}_{ij}|) - |\mathbf{P}_{00}|\|$$

Rectangular parallelepiped aligned with axes

A *Chebyshev bilinear approximation* of the patch is defined as the bilinear patch obtained by dropping all the $\mathbf{P}_{ij}$ terms where either $i > 1$ or $j > 1$. Its corners are called: $(\mathbf{b}_{00}, \mathbf{b}_{01}, \mathbf{b}_{10}, \mathbf{b}_{11})$. It can be shown easily that the distance between points on this patch of parameter (u, v) to the point on the original bicubic with the same parametric values is less than or equal to:

$$\alpha = (\alpha_x, \alpha_y, \alpha_z) = \sum_{i=0}^{3} \sum_{j=0}^{3} \left[\begin{array}{c} i > 1\ or\ j > 1\ :\ |\mathbf{P}_{ij}| \\ else\ :0 \end{array} \right]$$

The bounding box is calculated by extending the min-max bounding box for the Chebyshev bilinear patch (which is given by the bounding box of its corners) by $\alpha_{x,}\ \alpha_y,\ \alpha_z$.

Rectangular parallelepiped aligned with Chebyshev bilinear patch

An *average* plane for the Chebyshev bilinear patch is calculated using the algorithm found in [13]. The control points are then rotated so that the average plane of the transformed patch lies parallel to the $X - Y$ plane. The control points are again rotated (around the Z axis) so that the line defined by $(\mathbf{b}_{00} + \mathbf{b}_{01})/2$ and $(\mathbf{b}_{10} + \mathbf{b}_{11})/2$ lies parallel to the X axis. Using an orthogonal projection of the bilinear patch into the $X - Y$ plane a min-max bounding rectangle R for the patch is defined in the $X - Y$ plane. A bounding parallelepiped is then constructed for the transformed patch by extending R by $\delta = (\delta_x, \delta_y, \delta_z)$, where

$$\delta = \sum_{i=0}^{3} \sum_{j=0}^{3} \left[\begin{array}{c} i > 1 \ or \ j > 1 \ : |\mathbf{P}'_{ij})| \\ else \ : 0 \end{array} \right]$$

The inverse transformation can be applied to the bounding box to return to the original position of the patch.

5.4. Termination Criteria

A bicubic patch can be approximated by its Chebyshev bilinear patch within a tolerance ε if $||\delta|| < \varepsilon$ or $||\alpha|| < \varepsilon$ for boxes aligned with the axes.

6. Applications

6.1. Adaptive subdivision of curves.

A simple yet effective use of the Chebyshev basis is to subdivide adaptively the curve until the curve sections can be approximated by their linear representation with less than a given tolerance ε (typically half a pixel). The algorithm for rendering a curve is then as follows.

```
DrawCurve(C,ε)
      δ = Δ(C)
      if δ <= ε
            DrawLinearRepresentation(C)
      else
            left = LeftSplit(C)
            right = RightSplit(C)
            DrawCurve (left,ε)
            DrawCurve (right,ε)
      endif
end DrawCurve
```

In the preceding, drawing the linear representation means drawing the line segment between the real end points of the sub-curve. For more details and results on this approach, see [14].

6.2. Ray-Curve Intersection

Given a tolerance ε the curve is subdivided until each curve can be approximated by its Chebyshev line segment within an error of ε. The intermediate subdivisions are stored in a tree. The Chebyshev line segments of the leaf nodes are replaced by an interpolating line segment. The algorithm for intersecting a curve with a ray (note that this works as well to intersect a scan line with the curve) then proceeds as follows.

```
Curve = record
      left  ^Curve
      right ^Curve
      Circle
      Box
      Line
```

```
end Curve
RayCurve(Ray,C)
      if not Hit(Ray, Circle) then
            return
      if not Hit(Ray, Box, Intersection1, Intersection2) then
            return
      if C.left != NIL
            RayCurve(Ray, C.left)
            RayCurve(Ray, C.right)
      end if
      if Hit(Ray, Line, Intersection) then
            SetPixel(Intersection)
            SetIntensity(Neighbours)
      else
            for Pixels in (Intersection1, Intersection2)
                  calculate distance from line
                  SetIntensity of pixel according to the distance
                  SetPixel(CurrentPixel)
            end for
      end if
end RayCurve
```

The purpose of the code in italic is to provide some measure of filtering. If when building the boxes the minimum size is never set below half a pixel, one can guarantee that the curve segment will not be missed, and one can compute an intensity for the pixels in the neighbourood of the curve.

6.3. Ray-Surface Intersection

Once we have computed the Chebyshev bilinear patch which approximates a bicubic patch to a predetermined tolerance we can use the intersection of the ray with the bilinear patch to approximate the intersection of the patch with the ray.

The intersection between a bilinear patch and a ray is given by the roots of a quadratic equation. In our computations we solve in u. There can be 0, 1 or 2 real intersections within the patch domain.

In the current implementation we subdivide the patch storing all the intermediate patches in a tree. Each node corresponds to either a split in the u or the v parameter. After the construction of the tree each leaf node contains a cubic patch which can be approximated by a bilinear patch within an ε tolerance. Since the Chebyshev bilinear patches do not interpolate the four corners of their bicubic patches we then replace these patches with an interpolating bilinear patch, thus ensuring $C^{[0]}$ continuity.

The following algorithm can then be used to calculate the intersection of a ray with a patch.

```
Patch = record
    left  ^Patch
    right ^Patch
    Sphere
    Box
    Bilinear
end Patch
RayPatch(Ray,P)
    if not Hit( Ray, P.Sphere)
       return NO_HIT
    if not Hit( Ray, P.Box)
       return NO_HIT
    if P.left != NIL
```

```
        return ClosestHit(RayPatch(Ray, P.left), RayPatch(Ray, P.right))
    else
        return Hit(Ray, P.Bilinear)
end RayPatch
```

This approach therefore allows us to cull intersections by enclosing the patch in a hierarchical bounding volume.

6.4. Crack Prevention

As with all adaptive subdivision algorithms, *cracking* is a potential problem (see [15] for a review). In our current implementation, intra-patch cracking is easy to avoid since we have the subdivision tree, and a search for neighbours tells us whether the neighbouring subpatch has been subdivided or not, and therefore if a cubic interpolation or a linear interpolation is appropriate. The prevention of cracking at patch boundaries is more difficult, and for the time being we rely on the Chebyshev tolerance (this is not foolproof, since in the absence of filtering any difference between boundary points might cause a crack).

7. Statistics

It is clear from the previous discussion of boxing and Chebyshev polynomials that the usefulness and efficiency of a method depends greatly on the characteristics of the patches used. This is why we present here statistics derived from applying our methods to three different patch databases, a familiar small one and two large ones.

The statistics were gathered by shooting random rays uniformly distributed in direction (by a uniform distribution over any dA on the unit sphere) and by position (a uniform distribution of the intersection with the great circle perpendicular to the ray direction). For each patch 5000 rays were shot, uniformly distributed over its Chebyshev sphere. This means that every ray intersects the first enclosing Chebyshev sphere. If necessary these statistics can be weighted as a function of the projected area of the patch, to obtain the effect of globally uniformly distributed rays. The tolerance used is a global tolerance (that is the same for all the patches of the same set). It is chosen so that it corresponds to a tolerance of 1/4 pixel if the total object occupies the whole screen at a 1Kx1K resolution.

The data collected for each patch include the areas of the min/max box (aligned with the axis (because it is a commonly used box), the area of the Chebyshev sphere (as computed in Section 5.3), the area of the Chebyshev box aligned with the bilinear patch (as computed in Section 5.3), the number of subdivision levels in u and v, the number of subpatches obtained, the total number of intersections with Chebyshev spheres and boxes, the number of intersection with the last box, which corresponds to the number of times the intersection with a bilinear patch has been attempted, and the number of actual intersections. This last number is proportional to the average projected area of the patch (assuming a correct uniform distribution of our rays), and together with the area of the initial enclosing sphere can be used to compute the projected area of the patch.

In the following tables we gave a summary of these statistics, with the minimum, arithmetic mean, maximum and standard deviations of some quantities. The first three quantities are ratios of areas, to compare efficiencies of various boxes. The ratio Hits/Attempts is useful to show how effectively boxing avoids direct intersection. Note that at this point one could, if so inclined, intersect with the original patch to get "exact" intersection.

7.1. Teapot Patches

The first database is the (in)famous teapot, used here because most readers will be familiar with its shape and characteristics, and can compare our results to any other method. Of course the teapot has only 28 patches, and the statistics obtained from such a small and biased database are indicative but not conclusive.

Number of inital patches	28
Number of final patches	4428
Tolerance	0.0016
Chebyshev box < MinMax Box	18

Quantity	Minimum	Mean	Maximum	Standard Deviation
Chebyshev Box/Sphere	0.193236	0.364025	0.624821	0.134691
Chebyshev Box/MinMax Box	0.611157	1.03148	1.70142	0.368823
Chebyshev Box/Patch	2.15685	3.06972	4.07315	0.637729
Number of Divisions in U	3	3.92857	5	0.716399
Number of Divisions in V	3	3.85714	4	0.356348
Total Subpatches	96	158.143	271	56.9247
Hits/Attempts	0.264167	0.641641	0.87524	0.180599
Sphere Intersections	11086	19460.4	30513	5222.38
Box Intersections	4724	10685.6	18579	3707.9
Patch Intersections	246	615.393	990	230.72

7.2. Baby Patches

The second set consists of 2484 Bézier patches from the *Tin Toy* baby (these are half of the patches; the other half was obtained by symmetry, and would give the same result). These patches were designed "by hand", using a modelling system part of Pixar's *menv*.

Number of inital patches	2484
Number of final patches	9202
Tolerance	0.0027
Chebyshev box < MinMax Box	2440

Quantity	Minimum	Mean	Maximum	Standard Deviation
Chebyshev Box/Sphere	0.0411916	.299494	.578314	0.0736104
Chebyshev Box/MinMax Box	0.0914471	.562541	1.68573	.180807
Chebyshev Box/Patch	.990175	1.92094	20.7687	.983098
Number of Divisions in U	0	.932367	5	.925897
Number of Divisions in V	0	.79066	3	.846005
Total Subpatches	1	3.76047	93	4.68425
Hits/Attempts	.123438	.698736	.922049	.128061
Sphere Intersections	5000	7823.68	20526	2346.9
Box Intersections	346	3216.55	12161	1651.69
Patch Intersections	25	880.85	1713	310.225

7.3. Dragon Patches

The second large set consists of 3082 B-splines patches defining a dragon's head obtained by a hierarchical refinement method [16]. The main difference from the previous set is that they are of widely different sizes.

Number of inital patches	3082
Number of final patches	35290
Tolerance	0.15
Chebyshev box < MinMax Box	2255

Quantity	Minimum	Mean	Maximum	Standard Deviation
Chebyshev Box/Sphere	0.00565475	.24598	.6339	.115023
Chebyshev Box/MinMax Box	0.0315184	.797853	3.11497	.400146
Chebyshev Box/Patch	1.10319	6.52643	55.5284	4.32297
Number of Divisions in U	0	1.58112	4	1.06647
Number of Divisions in V	0	1.65217	5	1.12178
Total Subpatches	1	11.4504	182	16.81
Hits/Attempts	0.0201265	.311294	.898305	.12614
Sphere Intersections	5000	8785.55	20769	2772.47
Box Intersections	104	3214.79	10864	1915.32
Patch Intersections	5	251.956	1553	193.505

7.4. Discussion

The general conclusion from these statistics is that our method works well. The number of patches generated is moderate, and each patch covers between 200 (for the teapot) and 50 (for the dragon) pixels, which means they are sizeable. A subdivision into pixel-size polygons would generate two orders of magnitude more polygons. Also from the maximum level of subdivision one can see that a non-adaptive method would have to generate 20 to 30 times more patches. The initial Chebyshev boxes are quite efficient. Their average area is about 60% of the area of the minmax box (and is smaller in more than 80% of the cases), and it is quite close to optimal, since the ratio of its area to the patch area is about 2 for the baby, and 6.5 for the dragon (this reflects the higher average curvature of the dragon patches). The Chebyshev spheres do what is expected of them. They have about three to five times the area of the Chebyshev boxes, which means that if the rays cover an area more than 20 times the area of the sphere (which is very likely for the initial patches), the sphere is better than the orthogonal box. In the subdivision step the situation is different since the rays tested against the boxes are the rays which intersect the ancestors, and are therefore more "concentrated". The ratio of area of the boxes to the area of its children would help decide here, but we have not computed these. It is clear than the method is very efficient in avoiding intersecting the patch itself (or its bilinear representative). On the average only 4 to 16% of the rays are tried on the patch, and it should be remembered that all the initial rays intersect the top Chebyshev sphere. At the assumed size of the object on screen, only one in a few hundreds ray would intersect that sphere. One last interesting point: the ratio hits/attempts is also the reciprocal of the ratio of the area of the enclosing box to the subpatch at the last level. One can then compare it to the similar ratio at the top of the tree. One finds that it is not very different. It is 1.56 vs 3.0 for the teapot, 1.43 vs 1.92 for the baby, and 3.21 vs 6.5 for the dragon. That means that the patches do not really "go flat" as the subdivision increases, but keep a fairly constant curvature.

7.5. Performance

The intersection technique *per se* is efficient, independently of boxing. To obtain a limited comparison, we use the "standard" patch, from Toth [11] also used by Nishita *et al* for comparison [6]. The number of subdivisions is for a 120x120 resolution, and the timings are for a SUN Sparc 1 workstation (non-optimized code).

Algorithm	$\varepsilon = 2^{-6}$	$\varepsilon = 2^{-10}$
Ours (subdivisions)	4.8	8.5
Time (seconds)	14.1	18.7
Nishita (subdivisions)	8.6	11.4
Time (seconds)	24.0	26.0
Toth (subdivisions)	na	19.6

Timings for the teapot, without shadows (T1), with shadows (T2), with about 3000 trimming cubic curves (T3), and with the dragon head and trimming curves (T4) are given in the following table. Figure 3 shows the image corresponding to T4. The figures in Nishita *et al* are given for their Figure 17, which corresponds to

Figure 3. Model with 3,000 trimming curves and 3,000 patches.

T1 wit the primary rays and T2 with shadows. All times are in seconds.

Machine	T1	Nishita	T2	Nishita	T3	T4
IBM RS/6000 560	47	na	78	na	518	10,440
SUN Sparc 1	248	na	467	na	1,656	na
SGI 4D/70	286	189	513	367	2,396	na

The SGI 4D/70 was chosen in spite of its advanced age to be able to compare with similar images in Nishita *et al*. One can see that the timings are very similar, even though without detailed knowledge of each machine configuration these are only semi-quantitative. Figures 4 and 5 (colour section) show two more examples of images produced using our algorithms.

8. Conclusions

The main points of this work are that the Chebyshev basis polynomials are useful for boxing and intersection, that in general it pays to use the proper basis for the proper task (most practitioners already know that, but that is another example), that a combination of boxing and approximation with bilinear patches can lead to an efficient integration of parametric surfaces within a ray-tracer, and that one can usefully quantify the trade-offs involved in boxing. Our basic ray-tracer *optik* [17] has been used in this work, and is in fact the source of the illustration in this paper. It should be stressed that the techniques presented here can be (should be) used in conjunction with other techniques such as space subdivision to speed up ray-tracing, or operations on parametric surfaces in general. Further work is necessary on the influence of the tolerance on the efficiency and quality of the results and on the application of similar methods to filtered curve drawing and rational formulations.

Acknowledgements

We acknowledge the support of NSERC through operating grants and an equipment grant which considerably facilitated this research. The support of the University of British Columbia in establishing a computer graphics laboratory in our department and IBM Canada in establishing GraFiC is greatly appreciated. Pierre Poulin, Bob Lewis and Paul Lalonde, of Imager, helped in various aspects of this work. Bill Reeves of Pixar and Dave Forsey of the University of British Columbia made available their sets of parametric patches, and spent more time than they had to spare helping us understand their formats and run our statistics, and we are

grateful. Finally we thank Vaughan Pratt, who, when the first author told him many years ago what he was looking for, said "Chebyshev".

References

1. J.T. Kajiya, ‘‘Ray Tracing Parametric Patches,’’ *Computer Graphics (Siggraph ’82 Proceedings)*, vol. 16, no. 3, pp. 245-254, July 1982.
2. T.W. Sederberg and D.C. Anderson, ‘‘Ray Tracing of Steiner Patches,’’ *Computer Graphics (Siggraph ’84 Proceedings)*, vol. 18, no. 3, pp. 159-164, July 1984.
3. E.E. Catmull, *A Subdivision Algorithm for Computer Display of Curved Surfaces,* PhD Thesis University of Utah, 1974.
4. J. H. Clark, ‘‘A Fast Scan-Line Algorithm for Rendering Parametric Surfaces,’’ in *SIGGRAPH ’79 Conference Proceedings*, vol. 13, August, 1979. Supplement to proceedings.
5. J.M. Lane, L.C. Carpenter, T. Whitted, and J.F. Blinn, ‘‘Scan Line Methods for Displaying Parametrically Defined Surfaces,’’ *Comm. of the ACM*, vol. 23, no. 1, pp. 23-34, January 1980.
6. T. Nishita, T. W. Sederberg, and M. Kakimoto, ‘‘Ray Tracing Trimmed Rational Surface Patches,’’ *Computer Graphics (Siggraph ’90 Proceedings)*, vol. 24, no. 4, pp. 337-345, August 1990.
7. H. Weghorst, G. Hooper, and D.P. Greenberg, ‘‘Improved Computational Methods for Ray Tracing,’’ *ACM Trans. on Graphics*, vol. 3, no. 1, pp. 52-69, January 1984.
8. M.A.J. Sweeney and R.H. Bartels, ‘‘Ray Tracing Free-Form B-Spline Surfaces,’’ *IEEE Computer Graphics and Applications*, vol. 6, no. 2, pp. 41-49, February 1986.
9. C. G. Yang, ‘‘On speeding up ray tracing of B-spline surfaces,’’ *Computer Aided Design*, vol. 19, no. 3, pp. 122-130, April 1987.
10. in *An Introduction to Ray Tracing*, ed. A.S. Glassner, Academic Press, 1989.
11. D.L. Toth, ‘‘On Ray Tracing Parametric Surfaces,’’ *Computer Graphics (Siggraph ’85 Proceedings)*, vol. 19, no. 3, pp. 171-179, July 1985.
12. T. J. Rivlin, *The Chebyshev Polynomials,* John Wiley & Sons, 1974.
13. I.E. Sutherland, R.F. Sproull, and R.A. Schumacker, ‘‘A Characterization of Ten Hidden-Surface Algorithms,’’ *Computing Surveys*, vol. 6, no. 1, pp. 1-55, March 1974.
14. J. Buchanan and A. Fournier, ‘‘Drawing Parametric Curves Using Chebyshev Polynomials,’’ *Proceedings of Graphics Interface ’91*, pp. 16-23, 1991.
15. D. R. Forsey and R. V. Klassen, ‘‘An Adaptive Subdivision Algorithm for Crack Prevention in the Display of Parametric Surfaces,’’ *Proceedings of Graphics Interface ’90*, pp. 1-8, May 1990.
16. D. R. Forsey and R. H. Bartels, ‘‘Hierarchical B-Spline Refinement,’’ *Computer Graphics (Siggraph ’88 Proceedings)*, vol. 22, no. 4, pp. 205-212, 1988.
17. J. Amanatides and A. Woo, ‘‘Optik Users’ Manual,’’ DGP Technical Report 1987-2, University of Toronto, August 1987.