EUROGRAPHICS 2013 / I. Navazo, P. Poulin
(Guest Editors)

Volume 32 (2013), Number 2

Analytic Rasterization of Curves with Polynomial Filters

Josiah Manson and Scott Schaefer

Texas A&M University

Figure 1: *Vector graphic art of butterflies represented by cubic curves, scaled by the golden ratio. The images were analytically rasterized using our method with a radial filter of radius three.*

Abstract

We present a method of analytically rasterizing shapes that have curved boundaries and linear color gradients using piecewise polynomial prefilters. By transforming the convolution of filters with the image from an integral over area into a boundary integral, we find closed-form expressions for rasterizing shapes. We show that a polynomial expression can be used to rasterize any combination of polynomial curves and filters. Our rasterizer also handles rational quadratic boundaries, which allows us to evaluate circles and ellipses. We apply our technique to rasterizing vector graphics and show that our derivation gives an efficient implementation as a scanline rasterizer.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

There are two basic ways of representing images: raster graphics and vector graphics. Raster graphics are images that are stored as an array of pixel values, whereas vector graphics represent images as collections of geometric shapes, such as lines, curves, circles, and polygons. Vector graphics are used in many applications in computer graphics. In 3D, almost all geometry is represented in vector form as polygons or smooth parametric shapes. Even in two dimensions, vector images are extremely common. For example, nearly all text is drawn with vector fonts, where letters are stored as quadratic curves that indicate the transition between the inside and outside of the letter. Vector images are also used in maps, signs, and logos to give a crisp, clear appearance.

Although many shapes are stored in vector format, displays are typically raster devices. This means that vector images must be converted to pixel intensities to be displayed. A major benefit of vector images is that they can be drawn accurately at different resolutions. For example, text appears the same when printed at 600 dpi on paper or on a 72 dpi screen and the butterflies in Figure 1 are easily drawn at different sizes. Recently, mobile devices have become another important medium and have generated a renewed interest in scalable graphics. In particular, websites are often viewed both on computers with screens thousands of pixels wide and on phones with screens hundreds of pixels across. Therefore text, images, and icons must scale gracefully.

There has been a trend to improve the quality of images

rather than the number of pixels that are drawn. In part, this trend exists because the rate at which mathematical operations are performed is increasing faster than the speed of memory. The biggest improvement to pixel quality comes from reducing aliasing. Aliasing refers to any artifact that results from sampling a signal at a frequency less than twice the highest frequency in the signal. Without changing the sampling frequency, the only way to reduce aliasing is to filter the signal to remove frequencies that are higher than half of the sampling frequency. Multisampling, anisotropic texture-filtering, mipmapping, and motion blur are all filtering techniques commonly used to reduce aliasing.

When rasterizing curves, another form of aliasing is present. Most rasterizers only draw polygons and approximate curved boundaries as many-sided polygons. Thus, aliasing appears both when sampling points on the curve to form a polygon and from polygon rasterization. Our paper reduces both forms of aliasing simultaneously by directly rasterizing curved boundaries using high-order filters. We show that analytic solutions for pixel values can be found using polynomial filters to rasterize curved shapes with linear color gradients. We rasterize shapes with a variety of different curves and have closed-form solutions for Bézier curves of arbitrary degree as well as rational quadratic Bézier curves, which allows us to rasterize exact circles and ellipses. In practice, this set of shapes encompasses most primitives that make up vector images.

1.1. Related Work

The simplest form of rasterization is point sampling, which simply determines if a point is inside of a polygon or not. Calculating which polygon, if any, contains an arbitrary sample can be done using ray casting [App68]. One can also recursively cut the image into quadrants, subdividing around polygon edges [War69]. More typically, pixel values are calculated a scanline at a time [WREE67].

The solution to reducing aliasing artifacts is to convolve the signal with a filter that removes frequencies that are higher than half the sampling rate. This is equivalent to assigning pixel values by integrating the product of the image and the filter centered at the pixel. Approximating this integral by quadrature is called supersampling, and as more samples are used for each pixel, the quality of the rasterization improves. If subsamples are also on a regular grid, aliasing artifacts can still occur for very high-frequency images. In ray casting there is significant freedom in sample placement, and several early papers [DW85, Coo86, Mit87] analyze image sampling patterns. These works show that samples should be random but have uniform density, and there is renewed interest in such sampling patterns [JC99, HC06, Kal07, GM09, Wei10, Fat11, EDP*11]. There is also research on how to optimize the sampling pattern and weights simultaneously [LA06].

Calculating prefiltered pixel values requires an integral over areas of an image, and accurate approximations require numerous point samples, regardless of the sampling pattern. Calculating area integrals is complicated, especially when taking occlusion into account, but calculating line integrals is more tractable. Several papers [GT96, JP00, GBAM11] have described methods of approximating area integrals by calculating a one dimensional quadrature over line samples that are computed analytically. This approach reduces the dimensionality of the problem by one, and yields a rasterization with fewer artifacts from the same number of samples.

One can also evaluate area integrals exactly. One of the first methods to do so [Cat78] solves both the visibility and integration problems simultaneously. The method clips polygons to pixels, and then against each other, so that the remaining list of polygons in a pixel are all completely visible. It then sums the areas of the polygons in the pixel times their color, which is equivalent to sampling with a box filter. Some other methods simplify cutting to pixels by first cutting polygons into trapezoids aligned with scanlines [GT92]. These trapezoids are easier to cut into pixels than polygons and one can easily find the area of a trapezoid.

Duff uses trapezoid decomposition to evaluate polynomial filters over polygons [Duf89]. Our method is more general, because we are able to rasterize curved boundaries in addition to polygonal boundaries. Our derivation of closed-form rasterization equations also leads to a different rasterization algorithm, because Duff integrates over areas perpendicular to the scanline, whereas we integrate over the boundary along the scanline.

Some methods rasterize polygons with radial filters. An early method [Cat84] clips polygons to pixels and adds each edge's contribution using a radial filter. If an edge forms a triangle with the center of the filter, that triangle is split into two right triangles. Each right triangle is parameterized by two values that index a lookup table to find the edge's contribution. A more recent algorithm [LCSW05b] removes the need of clipping polygons to pixels by taking the modulo of final pixel values. The method also has analytic solutions for polynomial filters, but cannot use filters with negative values because of the modulo operation. The authors published a paper describing positive filters that are suitable for their method [LCSW05a].

There are also some rasterizers that are difficult to classify. One method approximates the rasterization of self-intersecting polygons [Doa04]. In this method, the contours within pixels that contain intersections are simplified and clipped against other contours. Another method calculates analytic rasterizations of shapes filtered by box splines [McC95]. Filters must be positive, and the method only rasterizes triangles. Auzinger et al. [AGJ12] analytically rasterize antialiased triangles and tetrahedra with linear data defined over the simplices. We also handle linear gradients, but we analytically rasterize complex, curved, two-dimensional shapes. A process similar to polygon rasteri-

zation has also been used to calculate surface irradiance in a raytracer by using Stokes' theorem to evaluate incoming light from polygonal light sources [CA00, CA01].

The main contribution of our paper is prefiltered, antialiased rasterization of shapes with curved boundaries. There has been some work in directly rasterizing shapes with curved boundaries, but most rasterizers simply approximate curved boundaries by subdividing the curves into many sided polygons [Cat74]. One of the first papers to rasterize curved shapes [CJ88] accurately determined if pixels were inside or outside of the region bounded by the curves, but only performed point sampling for rasterization.

Most antialiased rasterization methods estimate the distance to the boundary in some way [FF97, LB05, QMK06, QMK08, NH08] and approximate a radial filter by setting pixel values based on the distance to the boundary. Using distance to approximate a radial filter is only exact for line segments when no vertices are in the filter's support and works especially poorly between curves that are within a filter diameter of one another.

Another method [MS11] calculates exact filtering of shapes bounded by Bézier curves. The method is reminiscent of the recursive algorithm of Warnock, because the method calculates wavelet coefficients over a quad-tree to produce a filtered rasterization of the shape. If the scaling function approximates a low-pass filter, such as a box filter, the method performs antialiasing. Unfortunately, this method only works with wavelet filters, and most common filters other than the box filter are not wavelets. Our method combines the best aspects from Duff's work [Duf89] and wavelet rasterization [MS11] because our method has closed-form solutions to polynomial curves sampled using polynomial filters. In addition, we show how our derivation can rasterize shapes with rational quadratic boundaries such as circles and shapes with linear color gradients in their interior.

2. Rasterizer

Rasterization is the process of sampling an image $I(x,y)$ defined by a set of closed shapes, where each shape M_i is defined by a set of boundary curves ∂M_i. We assume that the input does not contain overlapping shapes. If this is not the case, we can preprocess the input to remove overlaps [HW07]. We define $I(x,y) = \sum_i I_i(x,y)$ as the sum of images of each shape $I_i(x,y)$, where each shape has color $c_i(x,y)$ inside of M_i and is zero outside. To remove aliasing, we prefilter the image by convolving $I(x,y)$ with a low-pass filter $h(x,y)$ prior to point sampling, which is equivalent to taking the inner product of the image $I(x,y)$ with the filter centered at each pixel. We explain how to rasterize the image of a single shape and drop the shape index i in the remainder of the paper because the final image is just a sum of shape images. We calculate the value of a pixel located at ℓ_x, ℓ_y by the area integral

$$\iint_{\mathbb{R}^2} I(x,y)h(x-\ell_x, y-\ell_y)\,dx\,dy.$$

We can simplify the expression by changing the domain of integration to be only over the interior of M because the image is zero outside of the boundary.

$$\iint_{x,y\in M} c(x,y)h(x-\ell_x, y-\ell_y)\,dx\,dy \tag{1}$$

The divergence theorem relates an integral over the boundary of a domain to an integral over the domain by

$$\oint_{p(s)\in\partial M} F(p(s))\cdot n(s)\,ds = \iint_{x,y\in M} \nabla\cdot F(x,y)\,dx\,dy,$$

where the unit normal of the shape is given by $n(s)$ and the domain's boundary is given by the arc-length parameterized curve $p(s)$. Therefore, if we find a vector function $F(x,y)$ whose divergence is equal to $f(x,y) = c(x,y)h(x-\ell_x, y-\ell_y)$, then we can evaluate Equation 1 as a boundary integral.

Because divergence is a sum of differentials, we find $F(x,y)$ by integrating $f(x,y)$ and we parameterize solutions for $F(x,y)$ by α, such that

$$F(x,y) = \begin{pmatrix} (1-\alpha)\int_{-\infty}^{x} f(u,y)\,du \\ \alpha\int_{-\infty}^{y} f(x,u)\,du \end{pmatrix}.$$

Thus, we have converted the rasterization problem from integration over an unknown interior to integration over a known boundary. Wavelet rasterization [MS11] has a similar derivation, but uses the inner product to calculate projections onto a wavelet basis. In the case of wavelets, it is possible to choose α so that $F(x,y)$ has compact support, but compact support is impossible for arbitrary polynomial filters. However, if we choose $\alpha = 0$, then $F(x,y)$ has infinite support only in the direction of the scanline, which we will exploit during rasterization in Section 2.1.

The unit normal of the curve is defined as $n(s) = p^{\perp}(s)/\|p'(s)\|$, where the direction perpendicular to the curve is

$$p^{\perp}(s) = \begin{pmatrix} p'_y(s) \\ -p'_x(s) \end{pmatrix}.$$

We use the notation that $p'(s)$ is the derivative of $p(s)$ with respect to s, while x and y subscripts, respectively, refer to the first and second components of a vector quantity. Changing variables from an arc-length parameterization ds to a uniform parameterization dt weights the differential by $ds = \|p'(t)\|\,dt$, which simplifies our expression because $n(s)\,ds = \frac{p^{\perp}(t)\|p'(t)\|\,dt}{\|p'(t)\|} = p^{\perp}(t)\,dt$. The dot product between vector functions also simplifies to a scalar product,

$$\oint F(p(t))\cdot p^{\perp}(t)\,dt = \oint F_x(p(t))p'_y(t)\,dt, \tag{2}$$

that uses only the x-component, $F_x(x,y)$, of $F(x,y)$ because the y-component, $F_y(x,y)$, is zero when $\alpha = 0$. Note that

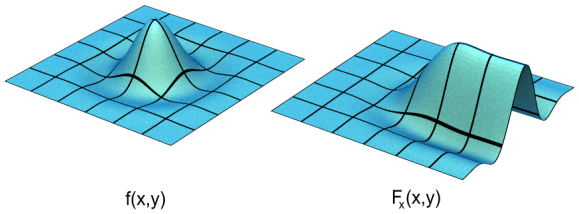

Figure 2: *A filter $f(x,y)$ is shown on the left and its integral in the x-direction $F_x(x,y)$ is shown on the right. The graphs are plotted over the filter's support. In $F_x(x,y)$, values remain constant in the direction of integration beyond $f(x,y)$'s support.*

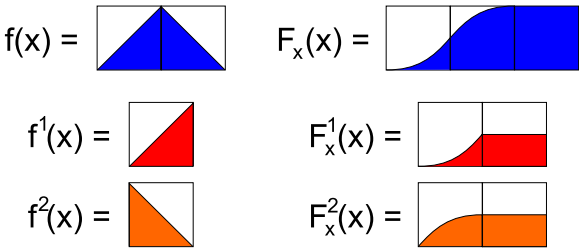

Figure 3: *We decompose a filter $f(x)$ into pixel sized pieces and evaluate each piece separately. The integral over $f(x)$ is equal to one, but the integral over each piece is not.*

most boundary curves $p(t)$ are defined piecewise, and it is trivial to sum the integral of each piece of the curve to evaluate Equation 2. Differentiation, integration, multiplication, and function application are closed under polynomials; so if $h(x,y)$, $c(x,y)$, and $p(t)$ are polynomial, the entire expression evaluates to a polynomial. An example of $f(x,y)$ and its integral $F_x(x,y)$ are shown in Figure 2. In this figure, it is clear that the support of $F_x(x,y)$ extends beyond the support of $f(x,y)$ in the positive x-direction only, so that the value of a pixel depends on the boundary to the right of the pixel.

There are a variety of filters $h(x,y)$ to choose from. Some filters, such as the box, tent, and Gaussian filters, blur the image to remove aliasing. There are also frequency bandpass filters that approximate the *sinc* function. The *sinc* function has too large of a support to be practical, so *sinc* is typically windowed to produce tensor-product filters such as the Lánczos and Mitchell-Netravali [MN88] filters. In two-dimensions, a circular bandpass results in a radial *jinc* filter. If a filter is not already polynomial, we can approximate it by a piecewise polynomial. We approximate windowed *jinc* and Lánczos filters by piecewise cubic and bicubic polynomials respectively with C^0 continuity aligned to pixel boundaries. This approximation produces images that are visually indistinguishable from images of the original filters.

2.1. Scanline Algorithm

To rasterize an image, we evaluate Equation 2 using the curves that are in the support of $F_x(x,y)$, which is finite in y, but is infinite in the positive x-direction. This directionality of infinite support leads to a scanline rasterization algorithm that evaluates pixels from right to left (although changing the direction of integration yields a traditional left to right rasterization algorithm). We assume that the filter is represented as a piecewise polynomial where the pieces align to the pixel grid. Since the polynomial pieces do not overlap, we can write the filter as a sum of polynomial pieces. In Figure 3, we show the decomposition of a one-dimensional tent filter into multiple pieces. We show the full filter and its integral on the top row of the figure, below which, we show the decomposition of the filter into pixel-sized pieces and the integrals of the pieces. Each piece $h^{(i,j)}(x,y)$ is defined over the $(x,y) \in [0,1)^2$ domain and is indexed by i,j so that

$$h(x,y) = \sum_i \sum_j h^{(i,j)}(x-i, y-j). \tag{3}$$

We then define the colored filter pieces over the same domain as $f^{(i,j)}(x,y) = c(x+\ell_x, y+\ell_y)h^{(i,j)}(x,y)$, with integrals $F_x^{(i,j)}(x,y) = \int_{-\infty}^{x} f^{(i,j)}(u,y)\,du$. Because our filter is piecewise polynomial with polynomial pieces that align to the pixel grid, we cut boundary curves to the pixel grid. With proper care, solving for the points to cut polynomials is robust even for degenerate curves [Bli05, Bli07]. We index cut curves by the cell λ_x, λ_y so that each curve segment

$$p^{(\lambda_x,\lambda_y)}(t) = p(t) \cap \left([0,1)^2 + (\lambda_x,\lambda_y)\right) - (\lambda_x,\lambda_y)$$

is within the same $[0,1)^2$ domain as the filter pieces. We can evaluate each pixel sized piece of the filter independently and sum the results to evaluate a pixel with indices ℓ_x, ℓ_y by

$$\sum_i \sum_j \int_0^1 F_x^{(i,j)}\left(p^{(\ell_x+i,\ell_y+j)}(t)\right) p_y'^{(\ell_x+i,\ell_y+j)}(t)\,dt. \tag{4}$$

Because summation is commutative, the order in which we evaluate Equation 4 does not matter and allows us to evaluate each curve, scanline, and filter piece in parallel. Given a curve $p^{(\lambda_x,\lambda_y)}(t)$ and filter piece $f^{(i,j)}(x,y)$, we can see from Equation 4 that $p^{(\lambda_x,\lambda_y)}(t) = p^{(\ell_x+i,\ell_y+j)}(t)$, and we therefore add the contribution from curve $p^{(\lambda_x,\lambda_y)}(t)$ to the pixel $(\lambda_x - i, \lambda_y - j)$. Our only constraint is that we must evaluate each filter piece in scanline order due to the infinite support of $F_x^{(i,j)}(x,y)$.

Figure 4 illustrates the process of rasterizing a single piece of $f^{(i,j)}(x,y)$. We show a scanline consisting of four pixels, where each row shows how we evaluate the pixel drawn in orange with the extended support of $F_x^{(i,j)}(x,y)$ drawn in red. The figure shows evaluation of a single curve that we draw in blue, with normals to indicate the curve's orientation. When the curve is within the support of the filter piece (top row), we evaluate of the curve integral and immediately update the

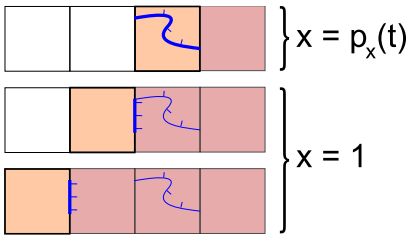

Figure 4: *The thick box outlines show which pixel is being evaluated for a filter piece $f^{(i,j)}(x,y)$. Evaluating a curve has two possible values: one within the filter piece; and one for all pieces to the left, found by setting the curve's x-coordinate equal to one.*

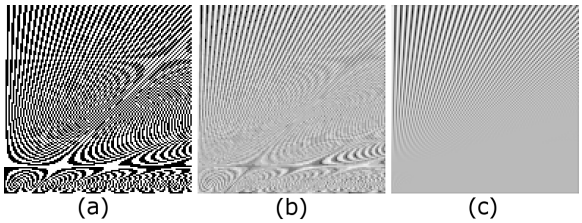

Figure 5: *Examples of (a) point sampling, (b) 16x MSAA tent filtering using an ATI Radeon HD 5700, (c) analytic tent filtering.*

pixel value. All pixels left of the curve (bottom rows) will have the same value added because $F_x^{(i,j)}(x,y)$ is constant with respect to x in the red region. We efficiently propagate this constant contribution by evaluating the curve once to update an accumulator that we add to remaining pixels in the scanline. Furthermore, we can simplify Equation 2 for propagated values by treating the curve as a line segment that connects the curve's end points and has x-coordinates equal to one. Before processing a scanline, the accumulator is initialized to zero outside of the shape. The orientation of a curve automatically adds color as the filter moves inside the shape's boundary and subtracts color as the filter moves outside of the shape.

Rasterizing shapes with linear color gradients is almost the same as rasterizing shapes with constant color. The only difference is that in Equation 4, $F_x^{(i,j)}(x,y)$ depends on the pixel coordinates ℓ_x, ℓ_y when $c(x,y)$ is linear. We define the color of the shape as $c(x,y) = C_0 + C_1 x + C_2 y$ in the shape's reference coordinate system. This means that the difference in value when moving from a pixel (ℓ_x, ℓ_y) to $(\ell_x - 1, \ell_y)$ is

$$-C_1 \int_{-\infty}^{x} h^{(i,j)}(u,y)\, du. \tag{5}$$

The integral of $h^{(i,j)}(x,y)$ is constant to the right of the filter piece, so we augment the accumulation buffer with a linear term that stores the value of Equation 5. The modification to our algorithm is simply that after processing a pixel, we add the linear accumulation term to the constant accumulation term. This process is easily extended to general polynomial color functions, where a quadratic term is accumulated into a linear term, and so on.

2.2. Implementation

We have described a general framework for a rasterization algorithm. The key idea is that the equations we use for rasterization naturally lead to efficient evaluation by rasterizing in scanline order. Rasterization using our method is easily parallelizable. Cutting all of the curves to pixels is done as a batch process prior to rasterization, where curves are distributed between processors. Each scanline is then evaluated in parallel, as is each polynomial piece of a filter. Note that this parallel implementation processes each curve a number of times equal to the area of support of the filter. It is also possible to write an efficient serial implementation that scans over the entire image once but maintains an array of accumulation values, one for each polynomial piece, and processes each curve once.

3. Results

Closed-form solutions are available for a wide variety of interesting filters and curves and we show that high-order filters reduce aliasing artifacts in several test images. We also show that calculating closed-form solutions of the rasterization equation is competitive in speed with approximate solutions while generating higher-quality images by comparing our implementation to other rasterizers.

The advantages of using high-order filters are most apparent in images with high-frequency details. We show rasterizations of test patterns that are prone to aliasing in Figures 5 and 6, where we rasterize the images using different filters. Figure 5 shows lines that are at a frequency of 1/4 lines per pixel at the top to 2 lines per pixel at the bottom. This figure illustrates the difference between point sampling and supersampling with a GPU (ATI Radeon HD 5700) versus analytically rasterizing images. Image (a) is point sampled and aliasing is clearly visible. Both (b) and (c) are sampled using a tent filter, but (b) is supersampled with 16 points per pixel and has obvious aliasing, whereas exact evaluation of (c) using our method suppresses most aliasing.

Figure 6 shows differences between analytically evaluated filters of increasing quality. The top row shows examples using linear curves, while the bottom shapes are made of quadratic curves. As the order of the filter increases, the transition between detailed and blurred regions becomes sharper and aliasing is reduced. The box filter (a) is clearly the worst filter, because it does not remove high frequencies very well, which results in obvious aliasing patterns. The tent filter (b) is noticeably better than the box filter. Although some high

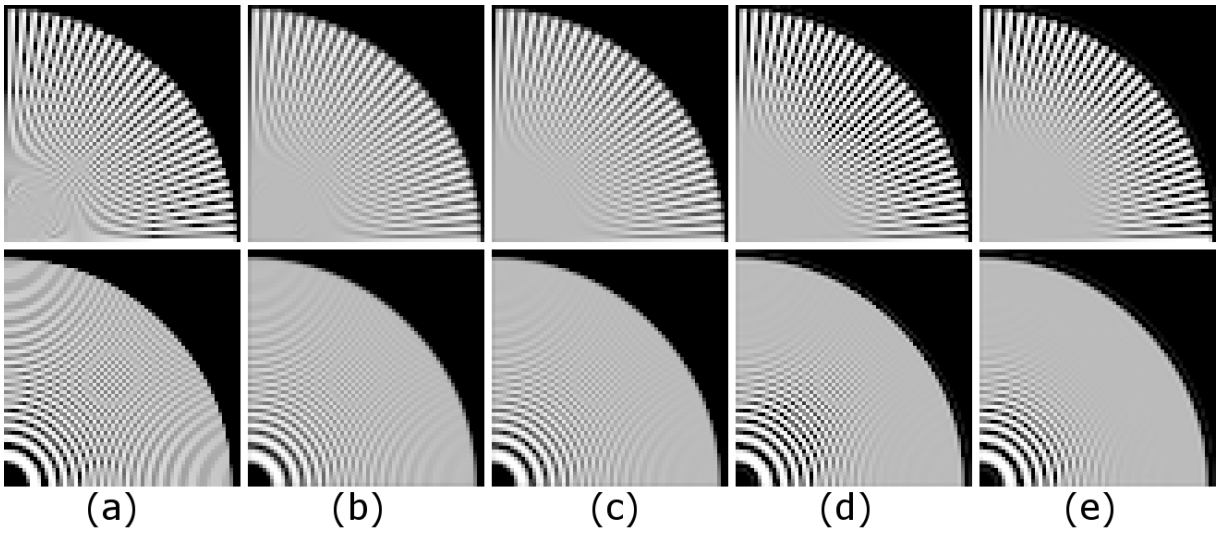

Figure 6: *Analytic rasterizations using different filters are shown. The first row uses radial triangles and the second row uses rings of quadratic curves. The columns show the filters: (a) box, (b) tent, (c) Mitchell-Netravali, (d) Lánczos 3, and (e) radial 3.*

	Fig. 5	Fig. 6 (top)	Fig. 6 (bottom)
AGG	2.73	0.25	**0.35**
Cairo	5.35	0.57	**2.34**
Wavelet	3.84	0.42	3.14
Box	1.23 *(0.60)*	0.14 *(0.07)*	0.37 *(0.17)*
Tent	2.36 *(0.83)*	0.27 *(0.12)*	0.75 *(0.26)*
Q. B-spline	9.60 *(3.29)*	1.02 *(0.31)*	4.45 *(1.26)*
Mitchell	21.2 *(6.90)*	2.26 *(0.79)*	37.0 *(11.4)*
Lánczos 2	26.3 *(7.64)*	2.71 *(0.96)*	27.6 *(7.54)*
Lánczos 3	48.7 *(16.0)*	5.37 *(1.88)*	115. *(37.7)*
Radial 2	21.2 *(5.70)*	2.31 *(0.78)*	17.1 *(5.02)*
Radial 3	42.9 *(14.1)*	4.64 *(1.68)*	45.8 *(15.6)*

Table 1: *Times to rasterize various images, measured in milliseconds. AGG, Cairo, and wavelet rasterization all use a box filter. Serial times are followed by parallel times using four processors in parentheses.*

frequencies still pass in the top image, the center of the circle is much closer to a uniform color. A disadvantage of the tent filter is that low frequencies are attenuated too much, which is visible as blurriness. The Mitchell-Netravali filter (c) appears slightly better than the tent filter. The Lánczos 3 (d) and radial (e) filters both have a radius of 3, but (e) is the only filter that is not a tensor product. In (d) it is easy to see the square fall-off of aliasing in the bottom picture compared to the circular fall-off in (e).

Our method is not restricted to polynomial curves and can theoretically be applied to any boundary defined by parametric curves. Closed-form expressions are not guaranteed for all curves but do exist for rational quadratic Bézier curves.

Figure 7: *We demonstrate rasterization of rational Bézier curves of varying sizes in a* 128×64 *pixel image.*

These are an important class of curves because they include all circular and elliptical arcs. We show an Apollonian gasket composed of rational quadratic Bézier curves that we evaluate exactly using a box filter in Figure 7.

We present rasterization timings for the patterns in Figure 5 (128^2 pixels) and Figure 6 (64^2 pixels) in Table 1. Figure 5 contains 1024 lines in 256 quads, Figure 6 (top) contains 96 lines in 32 triangles, and Figure 6 (bottom) contains 32 concentric rings made out of a total of 128 non-rational quadratic curves and 64 radial lines. We ran tests on an Intel Core i7 870. The first row contains the times taken by Anti-Grain Geometry (AGG), which is a highly optimized, high-quality software rasterizer. The second contains timings for Cairo, which is a rasterizer used in several large software projects. Both AGG and Cairo use box filtering and approximate curved boundaries by polygons, for which we used their default tolerances to subdivide curves. The third

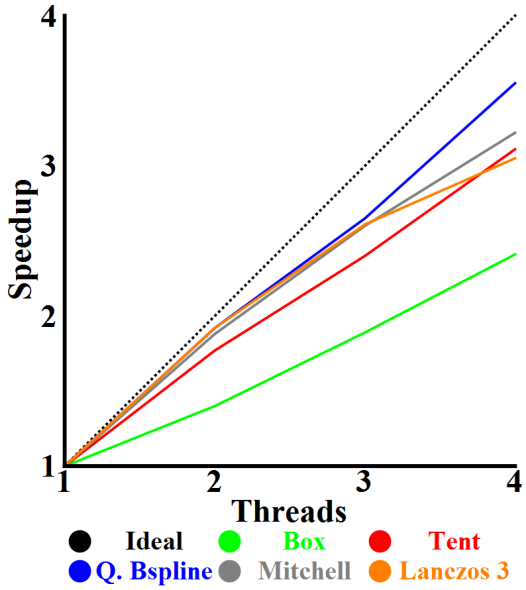

Figure 8: *Speedup for different numbers of threads on a four core Intel Core i7 870. The ideal speedup is shown as a dashed black line, and the speedup for different filters is shown as solid lines of different colors.*

row contains timings for wavelet rasterization [MS11] using the authors' implementation, which analytically rasterizes piecewise polynomial curves using a box filter. The remaining times are from our method using different filters. We use a serial implementation of our algorithm to compare with the other methods and include times for our parallel implementation running on four processors in parentheses. The Mitchell-Netravali and Lánczos 2 filters are both cubic tensor product filters, but have a different number of zero terms. The Radial 2 filter has the same support but has fewer coefficients compared to the cubic tensor product filters because we use a filter of cubic total degree.

The other methods that we compare against rasterize polygons using a box filter, so our box filter produces the same output as them for Figure 5 and Figure 6 (top). AGG and Cairo approximate quadratic and cubic curves as polygons, so we show the time for these approximate rasterizations in bold. Comparing against AGG, we see that the times to rasterize shapes are similar, even though we compute an analytic rasterization of quadratic curves rather than a polygonal approximation. Cairo is slower than AGG and our method, because it is more generic and is designed for more complicated rendering operations.

Unlike the other three methods, our technique can analytically filter images using higher order filters. As the order of the filter increases, so does the computation time, which is approximately linear in the area sampled by the filter. The tent filter is interesting because it provides a significant improvement in image quality, but takes only twice as long to compute as a box filter and remains competitive in time with AGG, which uses a box filter. High-order filters are expensive to compute, but may be acceptably fast for final, production renders.

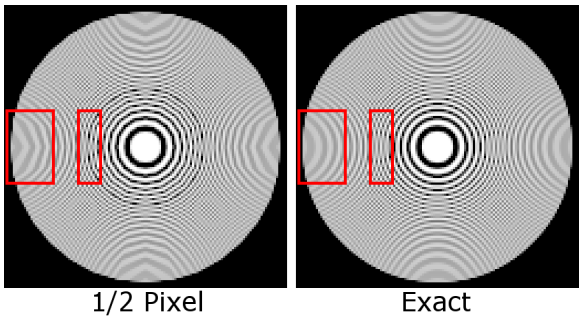

Figure 9: *In the image on the left, curves are approximated by polygons within a tolerance of $\frac{1}{2}$ of a pixel, while the image on the right is calculated exactly.*

Figure 8 shows the speedups we achieve rendering Figure 11 with a parallel implementation of our algorithm using OpenMP to split calculations between different numbers of threads on a four core system. The image contains 1,577 cubic curves that we rasterized into a 316×613 pixel grid. Speedups vary based on filter width. Some filters like the box filter are simple enough that the overhead of parallelization prevents ideal scaling. Nevertheless, with four cores we still achieve a $2.4\times$ speedup. As the size of the filter increases, Equation 2 requires more operations and our speedup approaches the perfect scaling relationship. In the case of a quadratic B-spline filter, we achieve a $3.5\times$ speedup over our serial implementation.

A common approach to handling curved boundaries is to approximate curves by line segments. However, this form of approximation produces its own aliasing artifacts, even when performing analytic filtering. For example, Figure 9 is composed of quadratic curves. On the left, we subdivided curves into line segments so that they are within 0.5 pixels of the actual curve. On the right, we show curves that are rasterized with exact formulas for quadratics. In both cases, we use our method for analytic rasterization so that the differences are only due to approximating curves by line segments. Notice that a black ring appears in the left image and that the aliasing patterns appear polygonal rather than round. It is also possible that a topological problem can occur when subdividing curves into line segments because, even if input curves do not intersect, the line segments approximating those curves may intersect. Determining the level of subdivision required to prevent line segments from intersecting can be complicated and expensive.

We show examples of SVG files that contain cubic curves

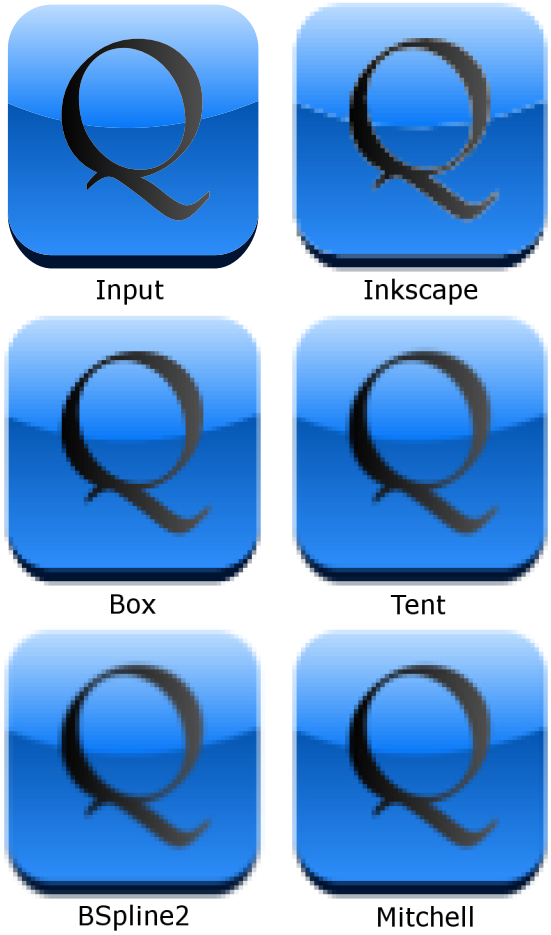

Figure 10: *A vector graphic image of an icon with linear gradients and cubic Bézier curves. On top we show the vector image, with rasterizations at 64^2 resolution using box, tent, quadratic B-spline, and Mitchell-Netravali filters below.*

in Figures 1, 10, and 11. Figure 1 shows a vector image that is scaled by irrational values. There are 60 butterflies, and each butterfly contains 452 cubic curves that form two colored shapes so that there are 27,120 cubic curves in total. Figure 10 shows an example of an icon made from 108 cubic curves that incorporates linear gradients and text. We show the input vector image on the top left, and show a 64^2 pixel rasterization from Inkscape and from our implementation of box, tent, quadratic B-spline, and Mitchell-Netravali filters. Note that many renderers, including Adobe Acrobat, have problems rendering the curved boundary between the linear color gradients correctly. This problem is visible in the Inkscape image as a bright halo. Hence, even the vector input may appear to have artifacts depending on your choice of viewing software. Finally, Figure 11 shows the effect of

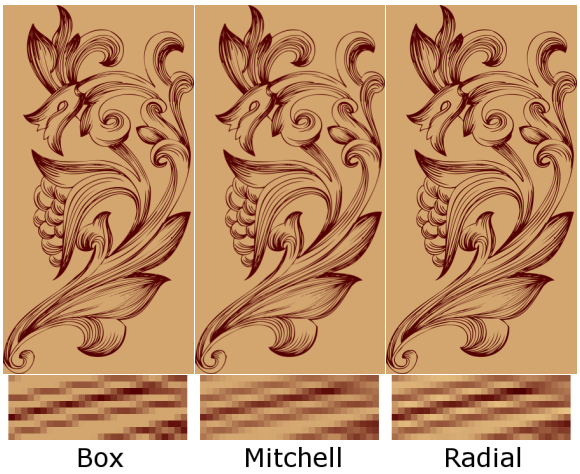

Figure 11: *An image rasterized using a box filter (left), Mitchell-Netravali filter (center), and radial filter of radius 3 (right). A zoomed-in section of the image is shown below each high-resolution image to show differences in pixel values.*

different filters on a detailed vector image without Moiré artifacts that is made of 1,577 cubic curves.

4. Conclusions and Future Work

With the processing power of video cards and the variety of display devices increasing in tandem, accurate rasterization of shapes is both more desirable and achievable. In this paper, we have revisited the theory behind rasterization, and described a new approach for finding analytic solutions for pixel colors using a variety of filters when rasterizing several types of curves that are commonly used in vector graphics.

Rasterization is a complex subject and remains an interesting topic for research. There are a couple of extensions to our method that may be useful. The curves and color gradients described in our paper are closed under affine transformations, but rasterizing three-dimensional scenes adds complicated occlusions and requires a projective transformation. For example, Gouraud shading defines linear color functions over triangles, but Gouraud shaded triangles have color functions that are rational with respect to screen coordinates after projection. Closed-form expressions of Equation 1 may exist for rational functions, but are certainly not polynomial. Rasterization of textured triangles is even more difficult. It could also be useful to apply our method to higher dimensions, for example, to calculate motion blur by integrating over a three-dimensional domain.

Acknowledgements

Work was supported by NSF CAREER award IIS 1148976 and the NSF Graduate Research Fellowship Program.

References

[AGJ12] AUZINGER T., GUTHE M., JESCHKE S.: Analytic antialiasing of linear functions on polytopes. *Computer Graphics Forum (Proceedings of EUROGRAPHICS) 31*, 2 (2012), 335–344. 2

[App68] APPEL A.: Some techniques for shading machine renderings of solids. In *AFIPS* (1968), pp. 37–45. 2

[Bli05] BLINN J. F.: How to solve a quadratic equation? *IEEE Comput. Graph. Appl. 25*, 6 (2005), 76–79. 4

[Bli07] BLINN J. F.: How to solve a cubic equation, part 5: Back to numerics. *IEEE Comput. Graph. Appl. 27*, 3 (2007), 78–89. 4

[CA00] CHEN M., ARVO J.: A closed-form solution for the irradiance due to linearly-varying luminaries. In *Eurographics Workshop on Rendering Techniques* (2000), pp. 137–148. 3

[CA01] CHEN M., ARVO J.: Simulating non-lambertian phenomena involving linearly-varying luminaires. In *Eurographics Workshop on Rendering Techniques* (2001), pp. 25–38. 3

[Cat74] CATMULL E. E.: *A subdivision algorithm for computer display of curved surfaces.* PhD thesis, 1974. The University of Utah. 3

[Cat78] CATMULL E.: A hidden-surface algorithm with antialiasing. In *SIGGRAPH* (1978), pp. 6–11. 2

[Cat84] CATMULL E.: An analytic visible surface algorithm for independent pixel processing. In *SIGGRAPH* (1984), pp. 109–115. 2

[CJ88] CORTHOUT M., JONKERS H.: A new point containment algorithm for B-regions in the discrete plane. In *Theoretical foundations of computer graphics and CAD* (1988), pp. 297–306. 3

[Coo86] COOK R. L.: Stochastic sampling in computer graphics. *ACM Transactions on Graphics 5*, 1 (1986), 51–72. 2

[Doa04] DOAN K.: Antialiased rendering of self-intersecting polygons using polygon decomposition. In *Pacific Graphics* (2004), pp. 383–391. 2

[Duf89] DUFF T.: Polygon scan conversion by exact convolution. In *International Conference On Raster Imaging and Digital Typography* (1989), pp. 154–168. 2, 3

[DW85] DIPPÉ M. A. Z., WOLD E. H.: Antialiasing through stochastic sampling. In *SIGGRAPH* (1985), pp. 69–78. 2

[EDP*11] EBEIDA M. S., DAVIDSON A. A., PATNEY A., KNUPP P. M., MITCHELL S. A., OWENS J. D.: Efficient maximal poisson-disk sampling. *ACM Transactions on Graphics 30*, 4 (2011), 49:1–49:12. 2

[Fat11] FATTAL R.: Blue-noise point sampling using kernel density model. *ACM Transactions on Graphics 28*, 3 (2011), 48:1–48:10. 2

[FF97] FABRIS A. E., FORREST A. R.: Antialiasing of curves by discrete pre-filtering. In *SIGGRAPH* (1997), pp. 317–326. 3

[GBAM11] GRIBEL C. J., BARRINGER R., AKENINE-MÖLLER T.: High-quality spatio-temporal rendering using semi-analytical visibility. In *SIGGRAPH* (2011), pp. 54:1–54:12. 2

[GM09] GAMITO M. N., MADDOCK S. C.: Accurate multidimensional poisson-disk sampling. *ACM Transactions on Graphics 29*, 1 (2009), 8:1–8:19. 2

[GT92] GISH W., TANNER A.: Hardware antialiasing of lines and polygons. In *Symposium on Interactive 3D graphics* (1992), pp. 75–86. 2

[GT96] GUENTER B., TUMBLIN J.: Quadrature prefiltering for high quality antialiasing. *ACM Transactions on Graphics 15*, 4 (1996), 332–353. 2

[HC06] HUANG R., CHAE S.-I.: Implementation of an openvg rasterizer with configurable anti-aliasing and multi-window scissoring. In *International Conference on Computer and Information Technology* (2006), pp. 179–185. 2

[HW07] HANNIEL I., WEIN R.: An exact, complete and efficient computation of arrangements of bézier curves. In *Symposium on Solid and Physical Modeling* (2007), pp. 253–263. 3

[JC99] JOUPPI N. P., CHANG C.-F.: Z3: An economical hardware technique for high-quality antialiasing and transparency. In *SIGGRAPH/Eurographics Workshop on Graphics Hardware* (1999), pp. 85–93. 2

[JP00] JONES T. R., PERRY R. N.: Antialiasing with line samples. In *Eurographics Workshop on Rendering Techniques* (2000), pp. 197–206. 2

[Kal07] KALLIO K.: Scanline edge-flag algorithm for antialiasing. In *Theory and Practice of Computer Graphics* (2007), pp. 81–88. 2

[LA06] LAINE S., AILA T.: A weighted error metric and optimization method for antialiasing patterns. *Computer Graphics Forum 25*, 1 (2006), 83–94. 2

[LB05] LOOP C., BLINN J.: Resolution independent curve rendering using programmable graphics hardware. *ACM Transactions on Graphics 24*, 3 (2005), 1000–1009. 3

[LCSW05a] LIN Z., CHEN H.-T., SHUM H.-Y., WANG J.: Optimal polynomial filters. *Journal of Graphics Tools 10*, 1 (2005), 27–38. 2

[LCSW05b] LIN Z., CHEN H.-T., SHUM H.-Y., WANG J.: Prefiltering two-dimensional polygons without clipping. *Journal of graphics, GPU, and game tools 10*, 1 (2005), 17–26. 2

[McC95] MCCOOL M. D.: Analytic antialiasing with prism splines. In *SIGGRAPH* (1995), pp. 429–436. 2

[Mit87] MITCHELL D. P.: Generating antialiased images at low sampling densities. In *SIGGRAPH* (1987), pp. 65–72. 2

[MN88] MITCHELL D. P., NETRAVALI A. N.: Reconstruction filters in computer-graphics. *ACM Computer Graphics 22*, 4 (1988), 221–228. 4

[MS11] MANSON J., SCHAEFER S.: Wavelet rasterization. *Computer Graphics Forum 30*, 2 (2011), 395–404. 3, 7

[NH08] NEHAB D., HOPPE H.: Random-access rendering of general vector graphics. In *SIGGRAPH Asia* (2008), pp. 135:1–135:10. 3

[QMK06] QIN Z., MCCOOL M. D., KAPLAN C. S.: Real-time texturemapped vector glyphs. In *Symposium on Interactive 3D Graphics and Games* (2006), pp. 125–132. 3

[QMK08] QIN Z., MCCOOL M. D., KAPLAN C.: Precise vector textures for real-time 3D rendering. In *Symposium on Interactive 3D Graphics and Games* (2008), pp. 199–206. 3

[War69] WARNOCK J. E.: *A hidden surface algorithm for computer generated halftone pictures*. PhD thesis, 1969. The University of Utah. 2

[Wei10] WEI L.-Y.: Multi-class blue noise sampling. *ACM Transactions on Graphics 29*, 4 (2010), 79:1–79:8. 2

[WREE67] WYLIE C., ROMNEY G., EVANS D., ERDAHL A.: Half-tone perspective drawings by computer. In *AFIPS* (1967), pp. 49–58. 2