Exact 2-D Integration inside Quadrilateral Boundaries

Jack Tumblin, Computer Science Department, Northwestern University,
Evanston IL 60201 jet@cs.northwestern.edu

Abstract

This paper shows how shift, add, scale and multiply operations on a few small matrices can compute the integral of any 2-D polynomial $f(x,y)$ within any specified quadrilateral boundaries, including non-convex chevrons, bow-ties and triangles. For applications such as antialiased rendering, compositing, anisotropic texture filtering, and high-contrast imagery, such quad-bounded integrals are usually approximated by sampling or dicing into small fragments, but the method presented here is exact. It may be suitable for hardware implementation, but is practical only for low-degree polynomials (*e.g.* $N,M < 5$) due to machine-precision limits and high cost $O(N^3M^3)$. Sample C++ source code is provided online. Extending the same method to tensors may be useful for higher-dimensional polynomials within a limited class of curved boundaries as well.

1 Introduction

Computing the integral of a function over an arbitrary bounded area or volume is often messy and cumbersome. The integration limits usually include variables, and conventional evaluations apply symbolic manipulations that are tough to implement in procedural languages such as C, C++ and Java. Yet these are exactly the sorts of integrals we need to evaluate for image warping and re-sampling, accurate compositing, anisotropic texture mapping, global illumination, and antialiased rendering of richly shaded objects with motion blur. These integration problems are usually solved by a wide range of approximation methods that include MIP-maps, jittered supersampling, trapezoidal area calculations, Monte-Carlo integration, EWA filters, and lookup tables.

No single best answer exists, but there is another choice that may prove useful in applications where accuracy is more important than speed. This paper presents a simple sequence of matrix operations that can directly compute the integral of any 2-D polynomial within any quadrilateral bounded area, including chevrons, bow-ties and triangles. Like Mirtich's related method using Green's Theorem [4], the result is not an approximation; it is the traditional symbolic solution rearranged into matrix expressions, and machine precision sets its accuracy.

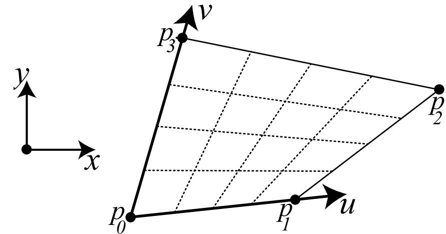

Figure 1: **To integrate any 2-D polynomial $f(x,y)$ over this convex quadrilateral, first make a bilinear map from the x,y corner points p_0,p_1,p_2,p_3 to the unit square in u,v space, then integrate $f(x(u,v),y(u,v))$ for $0 \le u,v < 1$ using matrix convolution.**

The method is straightforward to write as a computer program, because it requires only an orderly series of shift, add, scale, and multiply operations. The central ideas are: (a) A 2D polynomial can be expressed in matrix form, (b) integration within the unit square with corners at $(0,0)$,$(1,0)$,$(1,1)$ and $(0,1)$ is easy to evaluate, and (c) we can warp any quadrilateral to the unit square to perform this evaluation.

2 Polynomial Forms

The most familiar expressions for 2D polynomials can be written as matrix multiplies, and helped along by introducing a few more notations. A 2-D polynomial $f(x,y)$ is usually written as the sum-of-products (Equation 1), but the polynomial form Y^TFX shown in Equation 2 organizes $f(x,y)$ into a coefficient matrix F and column vectors X and Y that each hold sequential integer powers of the variables x,y respectively:

$$f(x,y) = \begin{matrix} f_{00} & +f_{01}x & +f_{02}x^2 & +\dots \\ f_{10}y & +f_{11}xy & +f_{12}x^2y & +\dots \\ f_{20}y^2 & +f_{21}xy^2 & +f_{22}x^2y^2 & +\dots \\ f_{30}y^3 & +\dots & +\dots & +\dots \end{matrix} = \sum_{i,j} f_{ij}x^jy^i \quad (1)$$

$$
= \underbrace{\left[\begin{array}{llll} 1 & y & y^2 & \dots \end{array}\right]}_{Y^T} \underbrace{\left[\begin{array}{llll} f_{00} & f_{01} & f_{02} & \cdots \\ f_{10} & f_{11} & f_{12} & \\ f_{20} & f_{21} & f_{22} & \cdots \\ \vdots & & \vdots & \end{array}\right]}_{F} \underbrace{\left[\begin{array}{l} 1 \\ x \\ x^2 \\ \vdots \end{array}\right]}_{X} = Y^T F X \tag{2}
$$

The degree of $f(x,y)$ sets the minimum matrix size: an $N \times M$-degree polynomial $f(x,y)$ has a coefficient matrix F of at least $(M+1) \times (N+1)$ elements, even if many are zero. For example, the $4^{th} \times 3^{rd}$ degree polynomial $f(x,y) = x^4 y^3$ needs an F of at least 4×5, but only element f_{34} is non-zero. The coefficient matrix may also use 'zero-padding' as needed: augmenting the F matrix by rows of zeros on the bottom or columns of zeros on the right side does not change its meaning, and makes it easier to combine polynomial forms of different sizes.

Several useful operators on polynomials such as add, multiply, exponentiate, and integrate are simple matrix procedures on the polynomial forms shown in Equations 2 :

Add: To add two 2-D polynomials $f(x,y)$ and $g(x,y)$, we add pairs of coefficients: thus $f(x,y) + g(x,y) = h(x,y)$ is implemented as $F + G = H$. If the F and G matrix sizes do not match, 'zero-padding' the smaller matrix can enlarge it without changing its meaning.

Multiply: In the conventional notation of Equation 1, the polynomial product $f(x,y)\,g(x,y) = h(x,y)$ is a mess, but in matrix form it becomes a discrete 2-D convolution of coefficients, written $F * G = H$. Convolution is a fundamental tool of digital signal processing (see [5]) found in many software libraries such as MATLAB [3], but it is not commonly associated with polynomial multiplication (although a textbook on Fourier transforms [1] has a nice 1-D example). To understand it, first define a $\mathit{shift}(P,a,b)$ matrix function that offsets the contents of a matrix P downwards and rightwards by augmenting P on top with a rows of zeros and on the left side with b columns of zeros. The product $h(x,y) = f(x,y)\,g(x,y)$ is given by the $(M_F + M_G - 1) \times (N_F + N_G - 1)$ coefficient matrix H made by adding shifted copies of the G matrix that are each scaled by an F matrix element. The convolution of matrices F and G is given by:

$$
H = F * G \equiv \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \mathit{shift}(f_{ij}\,G, i, j) \tag{3}
$$

or equivalently,

$$
h_{ij} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f_{ij}\, g_{(m-i),(n-j)} \tag{4}
$$

Equation 4 computes each element h_{ij} separately, but relies on zero values for elements with negative or out-of-range indices of F and G. Though possibly less familiar to some readers, Equation 3 is simpler and more efficient to implement.

Convolution works as an orderly, automatic way to expand and collect terms in a polynomial multiplication. Each element f_{ij} is the coefficient for $x^j y^i$, and to shift the matrix G by i,j and scale it by f_{ij} is equivalent to computing the product of $g(x,y)$ and the i,j term of $f(x,y)$. Adding together the results of this operation for every term in $f(x,y)$ will then compute the product $f(x,y)g(x,y)$.

Exponents: Repeatedly convolving F with itself will compute the integer powers of $f(x,y)$, so we describe $f^k(x,y)$ by writing the coefficient matrix F with an exponent of $*k$:

$$
f^k(x,y) = Y^T F^{*k} X \quad \text{where:} \tag{5}
$$

$$
F^{*0} = \left[\begin{array}{llll} 1 & 0 & 0 & \dots \\ 0 & 0 & 0 & \\ 0 & 0 & 0 & \dots \\ \vdots & & \vdots & \end{array}\right]; \quad \begin{array}{l} F^{*1} = F; \\ F^{*2} = F * F; \\ F^{*3} = F * F * F; \\ F^{*4} = F * F * F * F; \\ \dots \end{array} \tag{6}
$$

Integrals: Indefinite integrals of $f(x,y)$ are computed with S, a matrix that is zero-valued except for inverse counting on its first upper diagonal. (Zero-pad matrix F to add one row and one column of zeros to keep X and Y large enough):

$$
\underbrace{\left[\begin{array}{cccccc} 0 & 1 & 0 & 0 & 0 & \dots \\ 0 & 0 & \frac{1}{2} & 0 & 0 & \\ 0 & 0 & 0 & \frac{1}{3} & 0 & \\ 0 & 0 & 0 & 0 & \frac{1}{4} & \dots \\ \vdots & & & & \vdots & \end{array}\right]}_{S} \underbrace{\left[\begin{array}{c} 1 \\ x \\ x^2 \\ x^3 \\ \vdots \end{array}\right]}_{X} = \underbrace{\left[\begin{array}{c} x \\ \frac{1}{2}x^2 \\ \frac{1}{3}x^3 \\ \frac{1}{4}x^4 \\ \vdots \end{array}\right]}_{= \int X\,dx}; \tag{7}
$$

$$
\begin{aligned}
\int f(x,y)dx &= Y^T F(SX); \\
\int f(x,y)dy &= (SY)^T F X; \\
\int\!\!\int f(x,y)dx\,dy &= (SY)^T F(SX);
\end{aligned} \tag{8}
$$

Notice how easily we can evaluate these integrals of $f(x,y)$ when the limits are $\{0,1\}$. When $y=0$ or $x=0$, every element is zero in the Y^T or X vectors, and when $y=1$ or $x=1$ every element is one in these vectors. If we define Q as a column vector of 1-valued elements, then we can integrate f over the unit square (opposite corners at $0,0$ and $1,1$) and evaluate it almost by inspection. First in the x direction:

$$
\int_0^1 f(x,y)dx = Y^T F S X \Big|_0^1 = Y^T F(SQ) \tag{9}
$$

and then in the y direction:

$$
\int_0^1 \int_0^1 f(x,y)\,dx\,dy = (SQ)^T F(SQ) \tag{10}
$$

$$
\text{where } (SQ)^T \equiv \left[\begin{array}{ccccc} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \dots \end{array}\right]. \tag{11}
$$

Equation 11 holds a key observation in matrix form: we can evaluate the integral of any polynomial $f(x,y)$ within unit-square limits by computing a constant-weighted sum of its coefficients. The next three sections extend this idea to arbitrary quadrilateral limits. Section 3 shows how to bilinearly warp x,y space so that the given quadrilateral maps to a unit square in u,v space, Section 4 then shows how to properly integrate $\hat{f}(u,v) = f(x(u,v), y(u,v))$ to get the desired x,y space result, and Section 6 explains why this result is true even for non-convex quadrilaterals.

3 Bilinear Map

A properly chosen bilinear map can convert a convex quadrilateral in the x,y plane to a unit square in the (u,v) plane. As shown in Figure 1, we have named the (x,y) corner points in counter-clockwise order as $p_0 = (x_0,y_0)$, $p_1 = (x_1,y_1)$, $p_2 = (x_2,y_2)$ and $p_3 = (x_3,y_3)$. If we also define these points as the corners of the unit square in u,v coordinates, then a bilinear mapping can link each point in the u,v space to a unique point in x,y space. Let p_0 be the origin of u,v, and then trace around the u,v square in counter-clockwise order to define point p_1 as $(u,v) = (1,0)$, p_2 as $(u,v) = (1,1)$, and p_3 as $(u,v) = (0,1)$. To describe the bilinear mapping from u,v to x,y space by a matrix of scalar constants $a,b,c,\ldots,h$:

$$\begin{bmatrix} x(u,v) \\ y(u,v) \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \end{bmatrix} \begin{bmatrix} 1 \\ u \\ v \\ uv \end{bmatrix} \tag{12}$$

Values for $a,b,c,\ldots,h$ come from known points in (x,y) and (u,v). Replace the left-hand side of Equation 12 with a 2×4 matrix of corner positions in (x,y). On the right-hand side, replace the 4×1 vector with a 4×4 matrix whose columns hold values for the corner positions in u,v (values of 0 or 1). Post-multiply both sides by the inverse of this matrix to get:

$$\begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ y_0 & y_1 & y_2 & y_3 \end{bmatrix} \begin{bmatrix} 1 & -1 & -1 & 1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \end{bmatrix} \tag{13}$$

To convert $f(x,y)$ to u,v terms, use Equation 12 to write both x and y as 2×2 polynomial forms:

$$x(u,v) = \begin{bmatrix} 1 & v \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 1 \\ u \end{bmatrix} \equiv V^T B_x U \tag{14}$$

$$y(u,v) = \begin{bmatrix} 1 & v \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} \begin{bmatrix} 1 \\ u \end{bmatrix} \equiv V^T B_y U \tag{15}$$

In the next section, this bilinear mapping helps convert integration of $f(x,y)$ over quadrilateral limits into integration of $\hat{f}(u,v) = f(x(u,v), y(u,v))$ over unit-square limits.

4 Change of Variables

To properly evaluate the integral of $\hat{f}(u,v)$, we use a change of variables that follows the previous section's bilinear map (Equation 12). Standard calculus texts (*e.g.* [2]) explain why this change-of-variables requires only a change in integration limits, multiplication by the determinant of the Jacobian, and a re-writing of $f(x,y)$ in u,v terms:

$$\int_{y_0}^{y_1} \int_{x_0}^{x_1} f(x,y)\,dx\,dy =$$
$$\int_{y(u,v)\geq y_0}^{y(u,v)<y_1} \int_{x(u,v)\geq x_0}^{x(u,v)<x_1} f\left(x(u,v), y(u,v)\right) \begin{vmatrix} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} \end{vmatrix} du\,dv \tag{16}$$

The bilinear map of Equation 12 converts the integration limits to $(0,1)$, and its Jacobian determinant is:

$$\frac{\partial x}{\partial u} = b + vd; \quad \frac{\partial x}{\partial v} = c + ud; \quad \frac{\partial y}{\partial u} = f + vh; \quad \frac{\partial y}{\partial v} = g + uh;$$
$$\begin{vmatrix} \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v} \end{vmatrix} = \underbrace{\begin{bmatrix} 1 & v \end{bmatrix}}_{V^T} \underbrace{\begin{bmatrix} gb - fc & hb - fd \\ gd - hc & 0 \end{bmatrix}}_{J} \underbrace{\begin{bmatrix} 1 \\ u \end{bmatrix}}_{U} = V^T J U; \tag{17}$$

To write the Jacobian determinant in matrix form $V^T J U$, the column vectors U and V should adjust to hold any needed higher integer powers of u,v respectively (*e.g.* $1, u, u^2, u^3, \ldots$), just as we did earlier for X and Y in Equations 7 and 8,

Next, we need to re-write the polynomial $f(x,y)$ in u,v terms by replacing each x and y with $x(u,v)$ and $y(u,v)$. Equations 5 and 6 show us how to find the i-th and j-th power of $x(u,v)$ and $y(u,v)$ from the B_x and B_y matrices defined in Equations 14 and 15:

$$x^j y^i = V^T \left(B_x^{*j} * B_y^{*i}\right) U; \tag{18}$$

We can then find a matrix form for $f(x(u,v), y(u,v))$ by weighting each $x^i y^j$ term by the i,j-th coefficient of $f(x,y)$, or equivalently, by the element f_{ij} of matrix F. This weighted sum of matrices yields the coefficient matrix $\hat{F}$ for $\hat{f}(u,v)$:

$$f\left(x(u,v), y(u,v)\right) = \hat{f}(u,v) = V^T \hat{F} U \tag{19}$$

where $\hat{F}$ is:

$$\hat{F} \equiv \sum_{i,j} f_{ij} \left(B_x^{*j} * B_y^{*i}\right).$$

Assemble Equations 11, 16 and 19 for the final result:

$$\begin{aligned}\int_{p_0,p_1,p_2,p_3}\int f(x,y)dxdy &= \int_0^1\int_0^1 \hat{f}(u,v)V^T JU du dv \\ &= \int_0^1\int_0^1 (V^T\hat{F}U)(V^T JU)du dv \\ &= (SQ)^T\left(\hat{F}*J\right)SQ \qquad (20)\end{aligned}$$

5 Implementation

Using Equation 20 we can compute the definite integral of an $N\times M$ degree 2-D polynomial $f(x,y)$ within convex quadrilateral bounds. You may find the C++ source code submitted with this article helpful, or follow these steps to write your own:

- Apply Equation 13 to find $a,b,c,...,h$ values, and construct the B_x, B_y and J matrices from Equations 14, 15 and 17. The cost is only 6 scalar multiplies and 8 scalar adds.

- Compute two tables of matrices $T_x[j]=B_x^{*j}$ and $T_y[i]=B_y^{*i}$ to describe the powers of x and y in u,v terms. Recall that an $(M+1)\times(N+1)$ matrix F describes the $f(x,y)$ polynomial whose highest-degree term is $f_{ij}\cdot x^j\cdot y^i = f_{M,N}\cdot x^N\cdot y^M$. Table $T_x[j]$ describes the powers of x^j, and holds $(N+1)$ matrices indexed by $0\le j\le N$. Similarly, the $T_y[i]$ table holds $(M+1)$ matrices indexed by $0\le i\le M$ to describe y^i in u,v terms. Initialize the first two entries in both tables with known values: $T_x[0]=T_y[0]$, where both are a 1×1 matrix with value 1; the $T_x[1]$ table entry is matrix B_x, and the $T_y[1]$ entry is B_y from Equations 14 and 15. We fill the rest of the table with higher powers of x^j and y^i by repeated convolution: $T_x[j+1]=T_x[j]*B_x=B_x^{*(j+1)}$, and $T_y[i+1]=T_y[i]*B_y=B_y^{*(i+1)}$. Note that the size of the $T_x[j]$ matrix is $(j+1)\times(j+1)$ (Equations 3 and 6 describe convolution). Though the cost of each table is order N^4 multiply-accumulate operations, N is rarely larger than 3 (bicubic) for most practical graphics applications.

- Next, compute the $\hat{F}$ matrix as shown in Equation 19 from a weighted sum of table entries. For each $(0\le i\le M,\ 0\le j\le N)$ find the F matrix element f_{ij}. Convolve the j-th and i-th entries in the $T_x[j]$ and $T_y[i]$ matrix tables, and multiply each element in the result by f_{ij} (Equivalently, you may multiply the smaller matrix table entry's elements by f_{ij} before convolution for better efficiency). Add together the matrix results for each f_{ij} to construct the $\hat{F}$ matrix. Computing cost is $O(N^3M^3)$.

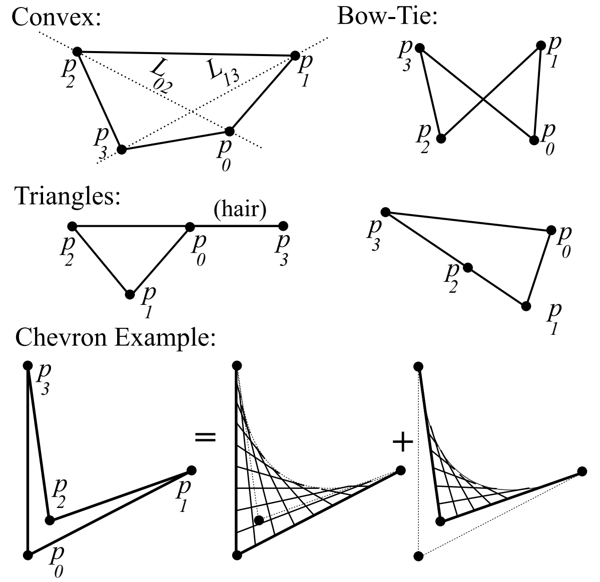

Figure 2: **All kinds of arbitrary quadrilaterals; Equation 20 can evaluate each of them correctly.**

- Convolve $\hat{F}$ and J matrices, then pre- and post-multiply by the constant vector (SQ) as shown in Equations 11 and 20. The cost is $O(N^2M^2)$ multiply-accumulate operations.

6 Chevrons, Bow-Ties and Triangles

As Figure 2 shows, many different kinds of quadrilaterals are possible. Arbitrary values for $p_0...p_3$ describe either a convex shape, a chevron, a bow-tie, a triangle, a triangle-with-a-hair, or a line. You may think we solved only the convex case, but Equation 16 is correct for *any* differentiable change of variables, and our method does indeed work for all quadrilaterals, and as our demonstration program confirms (see Section 8 for source code).

To improve your intuition, consider the chevron case at the bottom of Figure 2. Even though the gridded (u,v) unit square maps to areas in x,y well outside the chevron's boundaries, we integrate $f()$ over these areas twice, but with opposite sign. Viewed another way, the u,v surface that stretches between quadrilaterals boundaries turns upside-down here, it doubles back on itself. Equivalently, the edge-ordering x,y is both counter-clockwise and clockwise in the overlapped regions outside the chevron boundaries. Examining the edge-ordering for each lobe of the 'bow-tie' shape reveals they also use opposite signs in integration; the integral of $f(x,y)$ over one lobe is subtracted from the other in our result.

Figure 2 also shows how to identify quadrilateral shapes. Simply group together opposing points (p_0,p_2) and (p_1,p_3)

as "point pairs", construct lines L_{02} and L_{13} between them, and test point pairs against opposing lines. The demonstration program source code includes a shape identifier routine to help with testing.

7 Discussion

Convolutions to create the $\hat{F}$ matrix of Equation 19 account for the bulk of the computational cost in this method. For larger polynomials, the $O(M^3N^3)$ cost due to the repeated convolutions might also be improved by thoughtful use of the Fast Fourier Transform, because repeated matrix convolutions become repeated element-by-element multiplications in the frequency domain.

Finally, higher-order mappings from u,v to x,y such as bi-quadratics or bi-cubics are possible, and could evaluate integrals within quadrilaterals with polynomially curved sides. Similarly, just as a 2-D polynomial $f(x,y)$ fits neatly into a 2-D grid of numbers stored as a matrix, a 3-D polynomial $f(x,y,z)$ can be described by a 3-D grid of numbers and stored as a tensor. Combining tensors and similar arrangements to convolve, add, and multiply them might evaluate definite integrals of polynomials of any dimension over curved, hyper-cube-like boundaries.

8 Web Information

Sample C++ source code and demonstration programs are available online at:
http://www.acm.org/jgt/papers/Tumblin06.

References

[1] Ronald N. Bracewell. *The Fourier Transform and its Applications*, chapter 3, pages 30–35. Networks and Systems. McGraw-Hill, New York, 1989.

[2] Charles H. Edwards and David E. Penney. *Calculus and Analytic Geometry*, chapter 15, pages 748–756. Prentice-Hall, Inc., Englewood Cliffs, NJ 07632, 1982.

[3] MathWorks. Matlab. software product, 1984. www.mathworks.com, Natick, MA 01760.

[4] Brian Mirtich. Fast and accurate computation of polyhedral mass properties. In *Journal of Graphics Tools*. AK Peters.

[5] Alan V. Oppenheim and Ronald W. Schafer. *Digital Signal Processing*, chapter 2.3, pages 21–27. Signal Processing Series. Prentice Hall, Englewood Cliffs, NJ 07632, 1989.