Exact 2-D Integration inside Quadrilateral Boundaries

Jack Tumblin, Computer Science Department, Northwestern University, Evanston IL 60201 {jet@cs,j-tumblin}@northwestern.edu

Abstract

This short paper shows how to shift, add, scale and multiply a few small matrices to compute the integral of any 2-D polynomial \( f(x,y) \) within any specified quadrilateral boundaries, including non-convex chevrons and triangles. The solution is exact, general, and might be suitable for parallel implementation on graphics hardware. It can aid in many graphical tasks, including compositing, anisotropic texture filtering, high-contrast imagery, or high-quality antialiased rendering. Extending the same idea to tensors may provide integrals of higher-dimensional polynomials within a limited class of curved boundaries as well.

1 Introduction

The integral of a function over a bounded interval, area or volume is often cumbersome and messy to compute because the limits of integration are variable. Traditional solutions using symbolic manipulation and substitutions are tough to implement in procedural languages such as C, C++ and Java. Unfortunately, these are precisely the sorts of integrals we solve in anisotropic texture mapping, antialiased rendering of richly shaded objects with motion blur, accurate compositing of high contrast images or depth maps, form factors, mass, volume, centroids, and higher order moments. These 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, lookup tables and even radial basis functions.

No single best answer exists, but there is another choice that may prove useful in applications where accuracy and quality are more important than speed. This paper shows how a simple combination of unusual matrix operations can directly compute the integral of any 2-D polynomial within any quadrilateral bounded area, including chevrons and triangles. The result is not an approximation; it is the traditional symbolic solution fitted into matrix expressions, and its accuracy it set by machine precision. The method is straightforward to write as a computer program, because it requires only an orderly series of shift, add, scale, and multiplies of small matrices. This paper gives a detailed description of the method and briefly discusses extensions from matrices to tensors. Tensor forms can integrate higher-dimensional polynomials and even a limited class of curved boundaries.

2 Polynomial Forms

The most familiar expressions for polynomials can be difficult for computers to manipulate. Instead, this paper uses polynomial forms shown in Equations 1-3, and a few invented notations of its own. Any 2D polynomial \( f(x,y) \) can be written either as the sum-of-products in Equation 1, or the polynomial form \( Y^T F X \), where \( F \) is the coefficient matrix and the column vectors \( X \) and \( Y \) each hold sequential integer powers of the variables \( x, y \) respectively:

\[
\begin{align*}
  f(x,y) &= f_{00} + f_{01} x + f_{02} x^2 + \ldots \\
  &= \sum_{i,j} f_{ij} x^i y^j \quad (1)
\end{align*}
\]

\[
\begin{align*}
  F = \begin{bmatrix} f_{00} & f_{01} & f_{02} & \ldots \\
          f_{10} & f_{11} & f_{12} & \ldots \\
          f_{20} & f_{21} & f_{22} & \ldots \\
          \vdots & \vdots & \vdots & \ddots 
\end{bmatrix} \quad \begin{bmatrix} 1 \\
          y \\
          y^2 \\
          \vdots
\end{bmatrix} = \begin{bmatrix} x \\
          x^2 \\
          \vdots
\end{bmatrix} = Y^T F X \quad (2)
\end{align*}
\]

An \( N \times M \)-degree polynomial \( f(x,y) \) has a coefficient matrix \( F \) of at least \( M \times N \) elements, even if many are zero. For example, the \( 4^{th} \times 3^{rd} \) degree polynomial \( f(x,y) = x^3 y^3 \) needs an \( F \) of \( 3 \times 4 \) or larger, but only element \( f_{33} \) 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 their right side does not change its meaning, and makes it easy to combine polynomial forms of different sizes.

Of course, the coefficient matrix \( F \) alone is enough to completely specify the polynomial \( f(x,y) \). Though the \( X, Y \) vectors are mathematically necessary, they convey little added
meaning: the element \( f_{rc} \) found at row \( r \), column \( c \) of \( F \) is always the coefficient for the term \( x^r y^c \) in the polynomial. To help separate the coefficient matrix from the fixed parts of the form, I borrowed the rather obscure symbol \( \otimes \) to indicate an element-by-element multiply of two matrices of the same size. Written “\( \ast \) big dot” in MATLAB \([3]\), the symbol usually denotes a circle-making operator (see [1]):

\[
f(x,y) = \sum_{\text{all elements}} (YX^T) \otimes F
\]

\[
= \sum_{\text{all elements}} \begin{bmatrix}
1 & x & x^2 & \ldots \\
y & xy & x^2y & \ldots \\
y^2 & y^2x & x^2y^2 & \ldots \\
\vdots & \vdots & \vdots & \ddots
\end{bmatrix}
\otimes
\begin{bmatrix}
f_{00} & f_{01} & f_{02} & \ldots \\
f_{10} & f_{11} & f_{12} & \ldots \\
f_{20} & f_{21} & f_{22} & \ldots \\
f_{30} & \ldots & \ldots & \ldots
\end{bmatrix}; (3)
\]

More formally, the expression \( F \otimes G = H \) is true and defined if-and-only-if \( F, G, \) and \( H \) are matrices with matching numbers of rows and columns, and for every \( i, j \)-th element we have \( h_{ij} = f_{ij} g_{ij} \). Matrix operations on the forms in Equations 2 and 3 are easy to program, and we can combine add, multiply, and shift operations to perform integration.

### 3 Add, Multiply and Integrate

**Add:** Given two 2-D polynomials \( f(x,y) \) and \( g(x,y) \) described by an \( N \times N \) coefficient matrix \( F \) and an \( M \times M \) matrix \( G \) respectively, simply add their matrices: thus \( f(x,y) + g(x,y) = h(x,y) \) is implemented by \( F + G = H \).

**Multiply:** In the conventional notation of Equations 1, the polynomial product \( f(x,y) g(x,y) = h(x,y) \) is a mess, but in matrix form it reduces to discrete convolution, written \( F \ast G = H \). 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 polynomials. To understand it, first define a shift \((Q,a,b)\) matrix function that offsets the contents of a matrix \( Q \) downwards and rightwards by augmenting \( Q \) on top with \( a \) rows of zeros and on the right side with \( b \) columns of zeros. The product \( h(x,y) = f(x,y) g(x,y) \) is given by a \((N+M-1) \times (N+M-1)\) coefficient matrix \( H \) made by adding shifted copies of the \( G \) matrix that were scaled by an \( F \) matrix elements. The convolution of matrices \( H \) and \( G \) is given by:

\[
H = F \ast G \equiv \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \text{shift}(f_{ij}, i, j) \quad \text{(4)}
\]

or equivalently,

\[
h_{ij} = \sum_{m=0}^{M-1} \sum_{n=0}^{M-1} f_{mi} g_{n-j} \quad \text{(5)}
\]

Equation 5 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 less familiar to many readers, Equation 4 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 a coefficient for \( x^i y^j \), 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 let’s indicate \( f^k(x,y) \) by writing the coefficient matrix \( F \) with an exponent of \( *k\):

\[
f^k(x,y) = \sum_{\text{all elements}} (YX^T) \otimes F^{*k} \quad \text{where:} (6)
\]

\[
F^{*0} = \begin{bmatrix}
1 & 0 & 0 & \ldots \\
0 & 0 & 0 & \ldots \\
0 & 0 & 0 & \ldots \\
\vdots & \vdots & \vdots & \ddots
\end{bmatrix};
F^{*1} = F; \quad F^{*2} = F \ast F; \quad F^{*3} = F \ast F \ast F; \quad (7)
\]

\[
F^{*4} = F \ast F \ast F \ast F; \quad \ldots
\]

**Integrals:** Indefinite integrals of \( f(x,y) \) are computed with \( S \), a matrix with inverse off-diagonal terms:

\[
\int f(x,y) dx = Y^T FSX;
\]

\[
\int f(x,y) dy = Y^T S^T FX;
\]

\[
\int \int f(x,y) dxdy = Y^T S^T FSX; \quad (9)
\]

Notice how easily the definite integrals of a polynomial \( f(x,y) \) can be solved if the integration limits are \([0,1]\). Setting \( x = 0 \) or \( y = 0 \) sets every element of vectors \( Y^T S^T \) or \( SX \) to zero, and when \( x = 1 \) or \( y = 1 \), all \( X \) or \( Y \) elements are also 1. Accordingly, we can integrate the function \( f(x,y) \) over the area of the unit square almost by inspection. Integrating first in the \( x \) dimension:

\[
\int_0^1 f(x,y) dx = Y^T FSX \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \end{bmatrix}; \quad (10)
\]

and then in the \( y \) dimension:

\[
\int_0^1 \int_0^1 f(x,y) dxdy = \begin{bmatrix} 1 & 1 & 1 & \ldots \end{bmatrix} S^T FS \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \end{bmatrix} \quad (11)
\]
\[ = \sum_{\text{all \ elem.}} S^T F S = \sum_{\text{all \ elem.}} (\hat{S}^T \hat{S}) \bigcap F \]  

(12)

where \( \hat{S}^T \equiv [ 1 \ 1 \ 1 \ 1 \ 1 \ ... ] \).

The two one-filled vectors in equation 11 mean that the answer is just the sum of all the elements in the matrix \( S^T F S \). But examining the \( S \) matrix shows that \( S^T F S \) will add an unnecessary row and column of zeros to the result. Equation 12 uses an inverse-counting vector \( \hat{S} \) to express the same answer more compactly, and is easy to verify from Equation 1.

Equation 12 holds a key observation in matrix form: we can compute the volume under the unit square for any polynomial \( f(x,y) \) by computing a constant-weighted sum of its coefficients. The next section generalizes this result to all possible quadrilaterals by using a bilinear map and a change of variables.

4 Bilinear Map

Suppose we choose a convex quadrilateral in the \( x,y \) plane by naming its corner points \( p_0 = (x_0,y_0), p_1 = (x_1,y_1), p_2 = (x_2,y_2) \) and \( p_3 = (x_3,y_3) \) in counter-clockwise order, as shown in Figure 1 (see Section 6 for non-convex cases). If we also define these points as the corners of the unit square in \( u,v \) coordinates, then we can link the \( u,v \) space to \( x,y \) space using a bilinear mapping. Let \( p_0 \) be the origin of \( u,v \) and then trace around the \( u,v \) square in counter-clockwise order to define \( p_1 \) as \( (u,v) = (1,0) \), point \( p_1 \) as \( (u,v) = (1,0) \), \( p_2 \) as \( (u,v) = (1,1) \), and \( p_3 \) as \( (u,v) = (0,1) \). The bilinear mapping from \( u,v \) to \( x,y \) space is then described by a \( 2 \times 2 \) matrix \( B \) made of scalar constants \( a,b,c,...,h \):

\[
\begin{bmatrix}
  x(u,v) \\
  y(u,v)
\end{bmatrix} = \begin{bmatrix} a & b & c & d \\
  e & f & g & h \\
\end{bmatrix} \begin{bmatrix} u \\
  v \\
\end{bmatrix} = BW
\]

(13)

where:

\[
B = \begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\
  y_0 & y_1 & y_2 & y_3 \\
\end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 & 1 \\
  0 & 1 & 0 & -1 \\
  0 & 0 & 0 & 1 \\
  0 & 0 & 1 & -1 \\
\end{bmatrix}
\]

(14)

5 Change of Variables

Rather than integrate \( f(x,y) \) over the convex quadrilateral between points \( p_0,p_1,p_2 \) and \( p_3 \), we will instead use the bilinear map of Equation 13 to transform the problem to \( u,v \) coordinates, and then integrate over the unit square using Equation 12. Standard calculus texts such as [2] show that an integral’s change-of-variables requires only a change in its limits, multiplication by the determinant of the Jacobian, and rewritting \( f(x,y) \) in \( u,v \) terms:

\[
\int_{y_0}^{y_1} \int_{x_0}^{x_1} f(x,y) \, dx \, dy = \int_{u_0}^{u_1} \int_{v_0}^{v_1} f(u,v) \, du \, dv
\]

The bilinear map of Equation 13 converts the integration limits to \((0,1)\), and its Jacobian determinant is:

\[
\frac{\partial x}{\partial u} = b + dv; \quad \frac{\partial x}{\partial v} = c + du; \quad \frac{\partial y}{\partial u} = f + hv; \quad \frac{\partial y}{\partial v} = g + hu;
\]

\[
\begin{vmatrix}
  \frac{\partial x}{\partial u} & \frac{\partial x}{\partial v} \\
  \frac{\partial y}{\partial u} & \frac{\partial y}{\partial v}
\end{vmatrix} = [1 \ v] V T [b & c & f & g]
\]

(16)

Like \( X \) and \( Y \), the column vectors \( U \) and \( V \) adjust to hold any needed higher integer powers of \( u,v \) respectively (e.g. \( 1,u,u^2,u^3,... \)). To convert \( f(x,y) \) to \( u,v \) terms we begin with Equation 13 to write both \( x \) and \( y \) as \( 2 \times 2 \) polynomial forms:

\[
x(u,v) = \begin{bmatrix} 1 \\
  v \\
\end{bmatrix} \begin{bmatrix} a & b \\
  c & d \\
\end{bmatrix} \begin{bmatrix} 1 \\
  u \\
\end{bmatrix} = V T B_x U
\]

(17)

\[
y(u,v) = \begin{bmatrix} 1 \\
  v \\
\end{bmatrix} \begin{bmatrix} e & f \\
  g & h \\
\end{bmatrix} \begin{bmatrix} 1 \\
  u \\
\end{bmatrix} = V T B_y U
\]

(18)

Next, apply Equation 6 to find the \( i \)-th and \( j \)-th powers of \( x(u,v) \) and \( y(u,v) \) from the \( B_x \) and \( B_y \) matrices:

\[
x^i y^j = V T \left( B_x^i \ast B_y^j \right) U;
\]

(19)

Then use these results in Equations 1 and 3 to get:

\[
f(x(u,v),y(u,v)) = \sum_{i,j} \hat{F} \bigcap (V U T) \]

(20)

where \( \hat{F} \) is:

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

Assemble Equations 12, 15 and 20 for the final result:

\[
\int_{y_0}^{y_1} \int_{x_0}^{x_1} f(x,y) \, dx \, dy = \int_{u_0}^{u_1} \int_{v_0}^{v_1} f(x(u,v),y(u,v)) \, du \, dv
\]

(21)

Equation 21 is a tidy new way to express the solution to quadrilateral-bounded integration of any polynomial, and immediately suggests a straightforward implementation. Using this new expression we can compute the volume under an \( N^2 \) degree 2-D polynomial \( f(x,y) \) within convex quadrilateral bounds by following these steps:

- Construct the \( 2 \times 2 \) matrices \( B_x \), \( B_y \) and \( J \) from Equations 14 and 16. The cost is only 6 scalar multiplies and 8 scalar adds.
- Construct the \( (2N+2) \times (2N+2) \) matrix \( \hat{F} \) from Equations 4, 6 and 20. Though the cost is order \( N^4 \) multiply-accumulate operations, \( N \) is rarely larger than 3 (bicubic) for most practical graphics applications.
Convexity Test:

Figure 2: Detect convexity with lines $L_{02}$ and $L_{13}$, and construct chevron and triangle areas as the difference of two convex areas

- Convolve $\hat{F}$ and $J$ matrices, and find the sum of its elements weighted by the constant matrix $\hat{S}S^T$. The cost is $(2N + 3) \times (2N + 3)$ multiply-accumulate operations.

6 Chevrons and Triangles

Though Equation 21 requires a convex quadrilateral region to define its bilinear map (Equation 13) unambiguously, it can still be used to integrate non-convex quadrilaterals as well. These shapes fall into two classes known as chevrons and triangles, and both can be constructed from the difference of two convex quadrilaterals, as shown in Figure 2.

To detect these non-convex cases that require decomposition, group together opposing points $(p_0, p_2)$ and $(p_1, p_3)$ as ‘point pairs’, and construct two corresponding lines $L_{02}$ and $L_{13}$ that pass through these point pairs. A quadrilateral is convex if and only if each point pair straddles its opposing pairs’ line. It is a chevron if both points in one pair are on the same side of the opposing pair’s line, and a triangle if one point is collinear with an opposing pairs’ line. The quadrilateral is degenerate and has zero area if two points are collinear with the opposing pairs’ line. I also reject the ambiguous ‘bow-tie’ case where both point pairs are on the same side of the opposing pair’s line because its interior and exterior cannot be defined by traversing its four corner points.

To integrate within a chevron, add a new point $p_4$ on the opposite side of the chevron’s same-side point pair. Substituting $p_4$ for one of the same-side points will produce two convex quadrilaterals whose difference is the chevron, as shown in Figure 2. Triangle integration has a similar solution: lengthen one triangle side to a new point $p_5$, and construct a parallelogram with the adjacent triangle side to define point $p_4$. Merge the two shapes to form a new, convex quadrilateral, and subtract the parallelogram to recover the original triangle.

7 Conclusions and Extensions

The derived result in Equation 21 provides a straightforward, compact and efficient way to compute the exact result of an entire class of definite integrals. Software implementations of Equation 21 can accept any 2-D polynomial of any degree, and any quadrilateral region of that polynomial specified by an ordered list of its corner coordinates.

Self-convolution to create the $\hat{F}$ matrix of Equation 20 accounts for the bulk of the computational cost in this method, and its symmetry and repetitive processes suggest that a more careful analysis could yield substantial improvements in computing speed, especially for fixed-degree integration needed for most image-space applications such as texture filtering, and antialiased rendering of shaded polygon fragments. Hardware implementations of Equation 21, could offer substantial quality improvements that might make its added cost justifiable.

For example, an $N = 3$ implementation in hardware could achieve the excellent performance of the tunable piece-wise bicubic filter of [4].

For larger polynomials, the order $4N^3$ efficiency due to the repeated convolutions in Equation 20 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, notice that applying higher-order mappings from $u, v$ to $x, y$ such as biquadratics or bicubics, would allow users to integrate within quadrilaterals with polynomially curved sides. Similarly, just as a 2-D polynomial $f(x, y)$ is completely described by 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 we could store as a tensor. Combining these observations, it may be possible to convolve, add, and $\bigodot$ such tensors to compute definite integrals of polynomials of any dimension over curved, hyper-cube-like boundaries!

References


