Fast and Exact Convolution with Polygonal Filters

1. INTRODUCTION

Convolving an image with a polygonal shaped kernel has many applications: for example simulating defocus from a camera with a polygonal iris, computing shadows [Soler and Sillion 1998], approximating more complex filters using the fact that many shapes can be approximated well by polygons, or simply for artistic effect. By using the technique known as summed area tables or running sums it is possible to convolve a digital signal with a box filter in constant time, independently of the size of the box [Crow 1984]. These techniques have a number of applications including fast image smoothing and feature detection [Lewis 1995] as well as anti-aliasing texture-mapping. These methods can be generalised to a variety of kernel types by making use of repeated integration [Heckbert 1986]. [Sun 2003] generalises to filters defined by shapes other than axis aligned boxes such as parallelograms and hexagons. However, for polygon edges that are not at zero or 45° angles to an axis this method is not a true convolution due to the presence of what that paper calls gap pixels. This paper presents an exact method for convolving with arbitrary convex integral polygons in such a way that scaling the polygon by an integer size has no impact on the number of operations performed per pixel.

Our approach will be to show how to derive algorithms from *transfer functions* for filters. (These functions are also known in the engineering literature as *z-transforms* and in the mathematics literature as *generating functions* [Wilf 1994].) We will then show how to derive transfer functions for *cones* and then use Brion's Theorem [Brion 1988] to assemble the transfer function for the polygon filter from the transfer functions of its *tangent cones*. (We will define the new terminology later.)

2. TRANSFER FUNCTIONS

We will initially work in d dimensions and later specialise to $d = 2$. We will use bold face to represent d-tuples: $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)$ and use tuples as exponents: $\boldsymbol{x}^{\boldsymbol{n}} = x_1^{n_1} x_2^{n_2} \ldots x_d^{n_d}$. In one dimension we drop the redundant indices and assume $\boldsymbol{x} = (x)$. We consider digital images and filter kernels to be functions $\mathbb{Z}^d \to \mathbb{R}$. Given such a function $f\colon \mathbb{Z}^d \to \mathbb{R}$ define the linear operators Z_i by

$$(Z_i f)(x_1, \ldots, x_i, \ldots, x_d) = f(x_1, \ldots, x_i + 1, \ldots, x_d)$$

(Similarly in one dimension we define $(Zf)(x) = f(x+1)$). Using the tuple notation above we can write expressions such as:

$$(\boldsymbol{Z}^{\boldsymbol{y}}) f(x_1, \ldots, x_d) = (Z_1^{y_1} \ldots Z_d^{y_d} f)(x_1, \ldots, x_d) = f(x_1 + y_1, \ldots, x_d + y_d)$$

for integer y_i.

The *discrete convolution* of f with g, $f * g$, is defined by

$$(f * g)(\boldsymbol{x}') = \sum_{\boldsymbol{x} \in \mathbb{Z}^d} f(\boldsymbol{x}' - \boldsymbol{x}) g(\boldsymbol{x})$$

(We will only use the discrete convolution in this paper so we henceforth omit the word *discrete*.)

We can write this as

$$(f * g)(\boldsymbol{x}') = \Big(\sum_{\boldsymbol{x} \in \mathbb{Z}^d} g(\boldsymbol{x}) \boldsymbol{Z}^{-\boldsymbol{x}} \Big) f(\boldsymbol{x}')$$

So we can replace the problem of evaluating the convolution of f and g with that of computing the linear operator

$$\sum_{\boldsymbol{x}\in\mathbb{Z}^d} g(\boldsymbol{x})\boldsymbol{Z}^{-\boldsymbol{x}}$$

and then applying that operator to f. If g has finite support, i.e. it is non-zero at only a finite number of locations, then this expression is a polynomial in the linear operators Z_i. For most of this paper we are concerned with the structure of the polynomial itself rather than the linear operator so we will work with polynomials in the unknowns z_i rather than polynomials in the linear operators Z_i.

Define the *transfer function* G of g by

$$G(\boldsymbol{z}) = \sum_{\boldsymbol{x}\in\mathbb{Z}^d} g(\boldsymbol{x})\boldsymbol{z}^{-\boldsymbol{x}}$$

For a given polygon P we will be interested in the kernel defined by

$$g(\boldsymbol{x}) = \begin{cases} 1 & \boldsymbol{x}\in P \\ 0 & \text{otherwise} \end{cases}$$

and hence our transfer functions will be of the form:

$$G(\boldsymbol{z}) = \sum_{\boldsymbol{x}\in P\cap\mathbb{Z}^d} \boldsymbol{z}^{-\boldsymbol{x}}$$

(Note the minus sign in the exponent. These will be ubiquitous in this paper.)

Our strategy will be to find a simple representation of these transfer functions and show how these can be applied to produce operators that implement a rapid convolution with g.

Consider an example in one dimension, convolution with the box filter given by

$$g(x) = \begin{cases} 1 & x \geq 0 \text{ and } x < n \\ 0 & \text{otherwise} \end{cases}$$

(Remember that in one dimension we drop the redundant suffix and just write x instead of x_1.)

The transfer function for this filter is

$$G(z) = \sum_{i=0}^{n-1} z^{-i} = \frac{1-z^{-n}}{1-z^{-1}}$$

This is a product of two transfer functions: $1-z^{-n}$ and $1/(1-z^{-1})$. We can interpret this as meaning that the filter is the composition of two filters, one corresponding to each of these functions. (See, for example, Chapter 5 [Woods 2006].) The first of these is straightforward: it is the filter that maps f to $(1-Z^{-n})f$, i.e. $f(x)-f(x-n)$. We can implement this in a C-like programming language as follows:

```
for (all x) {
    g[x] = f[x]-f[x-n];
}
```

The result is stored in a new array, g. If we were to loop over decreasing x we could perform the operation in place but we choose not to do so here for clarity. At stage x we read from f at both x and $x-n$ so we may have to read beyond the limits of the source data. When we do this we will assume that the source data is padded with zeroes. We may also have to write data over a larger range too. We will say more about this shortly.

Now our task is to find an interpretation of $1/(1-Z^{-1})$.

If $h = \frac{1}{1-Z^{-1}}g$ then $g = (1-Z^{-1})h$ so $g(x) = h(x) - h(x-1)$ and $h(x) = g(x) + h(x-1)$. So we can evaluate each value of $h(x)$ sequentially by feeding back the previously computed $h(x-1)$ at each stage. (Such a filter is known as a *recursive filter*.) A loop in C might look like

```
for (x increasing) {
    h[x] = g[x]+h[x-1];
}
```

This has a direct interpretation: h is the running sum of the values of g. We implement the loop with increasing x so that when we read $h(x-1)$ we are reading the previously updated value. So $1/(1-Z^{-1})$ is the running sum operator. $(1-Z^{-n})/(1-Z^{-1})$ computes its result by taking suitable differences between running sums to rapidly compute sums of n terms.

We can now return to the question we raised earlier of which values of x we need to loop over. Suppose we wish to compute the values of the N pixels for x in the range $[0, N-1]$. The second loop above reads values for x in the range $[-1, N-1]$. In order to compute those values the previous loop must have ranged over these values and would have read data from f over the range $[-n-1, N-1]$. If any part of this falls outside of the domain of the original image we pad with zeroes. In general it is straightforward to compute the domains over which we must work using this kind of analysis so we will omit it from further consideration.

Consider the case in two dimensions:

$$g(\boldsymbol{x}) = \begin{cases} 1 & x_1 \geq 0 \text{ and } x_1 < n \text{ and } x_2 \geq 0 \text{ and } x_2 < n \\ 0 & \text{otherwise} \end{cases}$$

The transfer function is

$$\begin{aligned} G(\boldsymbol{z}) = \sum_{x_1=0}^{n-1}\sum_{x_2=0}^{n-1} z_1^{-x_1} z_2^{-x_2} &= \sum_{x_1=0}^{n-1} z_1^{-x_1} \sum_{x_2=0}^{n-1} z_2^{-x_2} \\ &= \frac{(1-z_1^{-n})(1-z_2^{-n})}{(1-z_1^{-1})(1-z_2^{-1})} \\ &= \frac{1-z_1^{-n}-z_2^{-n}+z_1^{-n}z_2^{-n}}{1-z_1^{-1}-z_2^{-1}+z_1^{-1}z_2^{-1}} \end{aligned}$$

The numerator is again straightforward to interpret:

```
for (all x1) {
    for (all x2) {
        g[x1][x2] = f[x1][x2]-f[x1-n][x2]-
            f[x1][x2-n]+f[x1-n][x2-n];
    }
}
```

Considering the denominator, write

$$h = \frac{1}{(1-Z_1^{-1})(1-Z_2^{-1})}g.$$

Then

$$g = (1 - Z_1^{-1} - Z_2^{-1} + Z_1^{-1}Z_2^{-1})h.$$

So

$$g(x_1,x_2) = h(x_1,x_2) - h(x_1-1,x_2) - h(x_1,x_2-1) + h(x_1-1,x_2-1).$$

We can now rewrite this as

$$h(x_1,x_2) = g(x_1,x_2) + h(x_1-1,x_2) + h(x_1,x_2-1) - h(x_1-1,x_2-1),$$

which can be implemented as:

```
for (x1 increasing) {
    for (x2 increasing) {
        h[x1][x2] = g[x1][x2]+h[x1-1][x2]+
            h[x1][x2-1]-h[x1-1][x2-1];
    }
}
```

Translating the numerator to an algorithm is a direct translation of the polynomial into code that sums values at various locations. Handling denominators in full generality is slightly more complex. When we apply the corresponding filter to the points in the grid we will apply them in some order, typically sweeping through one row or column at a time. We need to choose an ordering and a term in the denominator (preferably 1) such that the term chosen always corresponds to a grid point that comes later than that corresponding to the other terms. We then move this term to the left hand side and leave the other terms on the right hand side. For example in the case above we have moved the constant term 1 corresponding to $g(x_1,x_2)$ to the left hand side ensuring that the terms on the right hand side, $g(x_1-1,x_2)$, $g(x_1,x_2-1)$ and $g(x_1-1,x_2-1)$, have already been updated. If there is no constant term available then we divide both numerator and denominator by a monomial chosen to ensure that there is one. It should be apparent that this update rule for g gives the usual algorithm for constructing summed area tables given by Crow [Crow 1984].

Our goal in this paper will be to generalise this method from axis-aligned boxes to more general polygonal shapes.

3. RATIONAL CONES AND POLYTOPES

A *rational convex polyhedron* in d dimensions is a (possibly unbounded) subset, P, of $\mathbb{R}^d$ defined by

$$P = \left\{ \boldsymbol{x} \in \mathbb{R}^d : \boldsymbol{c}_i \cdot \boldsymbol{x} \leq \beta_i : i = 1, \ldots m \right\}, \text{ where } \boldsymbol{c}_i \in \mathbb{Z}^d \text{ and } \boldsymbol{\beta}_i \in \mathbb{Z}.$$

In other words, it is a finite intersection of half-spaces, each given by an equation with integer coefficients.

If the set is bounded it is called a *polytope* and if all of its vertices are in $\mathbb{Z}^d$ then it is called an *integer* polytope. We shall refer to two dimensional polytopes as *polygons*. We call the polytope P, scaled by a factor of n (with scaling based at the origin), the nth dilate of P denoted nP.

Suppose $\boldsymbol{v}$ is a vertex of a convex polyhedron. The polygon is the intersection of a collection of half spaces, i.e. the sets

$$H_i = \{ \boldsymbol{x} : \boldsymbol{c}_i \cdot \boldsymbol{x} \leq \beta_i \}$$

The point $\boldsymbol{v}$ will lie on the boundary of some of these half-spaces, i.e. those H_i for which $\boldsymbol{c}_i \cdot \boldsymbol{v} = \beta_i$. We define the tangent cone of P at $\boldsymbol{v}$, cone$(P,\boldsymbol{v})$, to be the intersection of just those half-spaces. (See [Brion and Vergne 1997] for

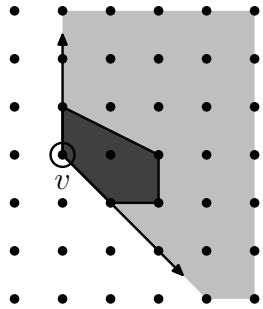


Fig. 1. The tangent cone (light shading) of a polygon (dark shading)

further details.) Any convex polyhedron is the intersection of its tangent cones:

$$P = \bigcap_{\boldsymbol{v} \in \mathrm{Vert}(P)} \mathrm{cone}(P, \boldsymbol{v}) \tag{1}$$

A cone, K, is said to be generated by the k linearly independent vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k$ if

$$K = \{\lambda_1 \boldsymbol{u}_1 + \cdots + \lambda_k \boldsymbol{u}_k \text{ where } \lambda_i \geq 0 \text{ for all } i = 1, \ldots, k\}$$

In two dimensions the tangent cones of a polytope are each translates of cones generated by a pair of vectors, one along each edge from the vertex of the cone to its neighbour (see Figure 1). In higher dimensions the cone will only be generated in this way if the tangent cone is formed as the intersection of exactly d half-spaces. If this is not the case then the polytope can be decomposed into suitable cones in various ways (e.g. [Barvinok and Pommersheim 1997]).

Before tackling polytopes we first compute the transfer function of a tangent cone. If the vertex of the cone is at the point P with vertices $(x_1, \ldots, x_n)$ then the transfer function is given by $\boldsymbol{z}^{-\boldsymbol{x}} G(\boldsymbol{z})$ where G is the transfer function of the cone whose vertex is translated to the origin. So without loss of generality we work with cones based at the origin.

Let K be the cone generated by the vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k$ with integer components. Then

$$\Pi = \{\lambda_1 \boldsymbol{u}_1 + \cdots + \lambda_k \boldsymbol{u}_k \text{ for all } 0 \leq \lambda_1, \ldots, \lambda_k < 1\}$$

is said to be the *fundamental parallelepiped* or *fundamental domain* generated by the $\boldsymbol{u}_i$. It can be seen that the cone K is tiled by copies of the fundamental parallelepiped as in Figure 2. If the $\boldsymbol{u}_i$ are all integral then given a point $x \in K \cap \mathbb{Z}^d$ we can write it uniquely in the form

$$\boldsymbol{x} = \boldsymbol{m} + a_1 \boldsymbol{u}_1 + \ldots + a_k \boldsymbol{u}_k$$

with $\boldsymbol{m} \in \Pi$ and the a_i all non-negative (see [Stanley 1999], Ch.4). In other words, every integer point in the cone can be expressed in exactly one way as a sum of a point in the fundamental parallelepiped and positive integer combinations of the generators. This is just another way of saying that the cone is a disjoint union of translates of fundamental parallelepipeds. In Figure 2 illustrates the cone generated by $(2,-1)$ and $(1,1)$. The fundamental parallelepiped has vertices $(0,0)$, $(2,-1)$, $(3,0)$ and $(1,1)$ and contains 3 integer points.

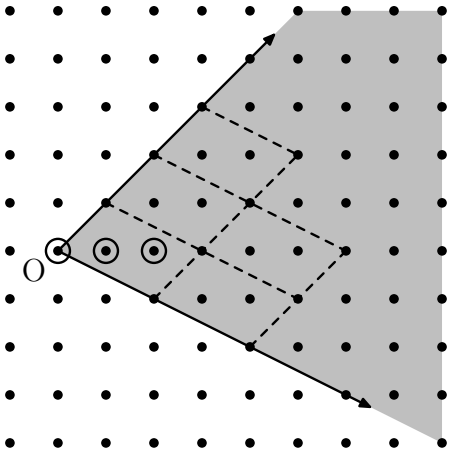


Fig. 2. A cone with its fundamental parallelepiped

Fix a value of $\boldsymbol{m}$. Then the transfer function for the set $S = \{\boldsymbol{m} + \sum_i a_i \boldsymbol{u}_i$ such that $a_i \in \mathbb{Z}, a_i \geq 0\}$ is

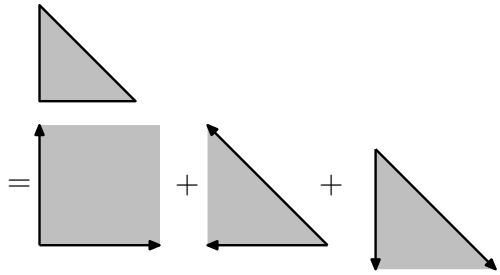
$$\begin{aligned}
\sum_{x \in S} \boldsymbol{z}^{-\boldsymbol{x}} &= \sum_{n \in \mathbb{Z}^k, n_i \geq 0} \boldsymbol{z}^{(-\boldsymbol{m} + n_1 \boldsymbol{u}_1 \ldots + n_k \boldsymbol{u}_k)} \\
&= \boldsymbol{z}^{-\boldsymbol{m}} \sum_{n_i \in \mathbb{Z}, n_i \geq 0} \prod_{i=1}^{k} \boldsymbol{z}^{-n_i \boldsymbol{u}_i} \\
&= \boldsymbol{z}^{-\boldsymbol{m}} \prod_{i=1}^{k} \sum_{n=0}^{\infty} \boldsymbol{z}^{-n \boldsymbol{u}_i} \\
&= \boldsymbol{z}^{-\boldsymbol{m}} \prod_{i=1}^{k} \frac{1}{1 - \boldsymbol{z}^{-\boldsymbol{u}_i}}
\end{aligned}$$

We can now state a remarkable theorem from combinatorics [Brion 1988]: **Brion's Theorem**

$$G(\boldsymbol{z}) = \sum_{\boldsymbol{v} \in \mathrm{Vert}(P)} F(\mathrm{cone}(P, \boldsymbol{v}); \boldsymbol{z})$$

where $F(K, \boldsymbol{z})$ is the transfer function of the cone K. Despite the deceptive similarity to Equation 1 this is in fact a non-trivial result originally proved using techniques from algebraic geometry.

Consider the example of the triangle with vertices $(0,0)$, $(2,0)$ and $(0,2)$. Pictorially Brion's theorem becomes:



and algebraically it is:

$$1+x_1^{-1}+x_1^{-2}+x_2^{-1}+x_1^{-1}x_2^{-1}+x_2^{-2}= \tag{2}$$

$$\frac{1}{(1-x_1^{-1})(1-x_2^{-1})}+\frac{x_1^{-2}}{(1-x_1)(1-x_1x_2^{-1})}+\frac{x_2^{-2}}{(1-x_2)(1-x_1^{-1}x_2)} \tag{3}$$

For the triangle wth vertices $(0,0)$, $(n,0)$ and $(0,n)$ we replace x_1^2 and x_2^2 with x_1^n and x_2^n. The left hand side becomes a larger sum but the complexity of the expression on the right hand side remains the same. This is the crucial point of this paper: The transfer function of the nth dilate of a polygon is of the form

$$G_n(\boldsymbol{z}) = \sum_{\boldsymbol{v}\in \mathrm{Vert}(P)} \boldsymbol{z}^{-n\boldsymbol{v}} F_{\boldsymbol{v}}(\boldsymbol{z})$$

(where the $F_{\boldsymbol{v}}$ are rational functions) which means that as n increases the complexity of the transfer function remains constant. In other words the algorithm to compute the convolution takes constant time per pixel as a function of n.

An important aspect of this approach is that we can consider the convolution coming from the denominator as a pre-processing step producing a generalised summed area table. We may now apply the numerator to extract sums for each polygon from the table in a random access manner. Our new approach allows us to extract the sum for any integral polygon whose edges have the same gradients making it suitable as an alternative to the usual summed area table for filtering textures.

4. A FULLY WORKED EXAMPLE: A HEXAGONAL FILTER

To illustrate the generality of the method we will consider not just an integer dilate of a fixed polygon but a hexagon, $H_{a,b,c}$, with opposite sides parallel and of the same length, and with edges defined by vectors $(a,0)$, $(b,2b)$ and $(-c,2c)$. The vertices lie at $(0,0)$, $(a,0)$, $(a+b,2b)$, $(a+b-c,2b+2c)$, $(b-c,2b+2c)$ and $(-c,2c)$. The choice $(a,b,c)=(2,1,1)$ gives an approximate regular hexagon. We will denote the transfer function by $G_{a,b,c}$. (If we use $(a,b,c)=(2n,n,n)$, say, we will have a situation identical to that discussed earlier.)

Figure 3 illustrates the 6 vertices, their corresponding tangent cones, their fundamental parallelogram and the integer points lying within them. Let F_i be the transfer function for the cone at vertex i. For example the fundamental parallelogram at vertex 3 has integer points at $(0,0)$, $(-1,1)$, $(-1,0)$ and $(-1,1)$, relative to the vertex, and this gives rise to the $1+z_1^{-1}z_2^{-1}+z_1^{-1}+z_1^{-1}z_2$ numerator in F_3. The tangent cone at vertex 3 has generators $(-1,2)$ and $(-1,-2)$ and this gives rise to the denominator of F_3, $(1-z_1^{-1}z_2^2)(1-z_1^{-1}z_2^{-2})$. Transfer vertices for the other 5 cones can be constructed similarly:

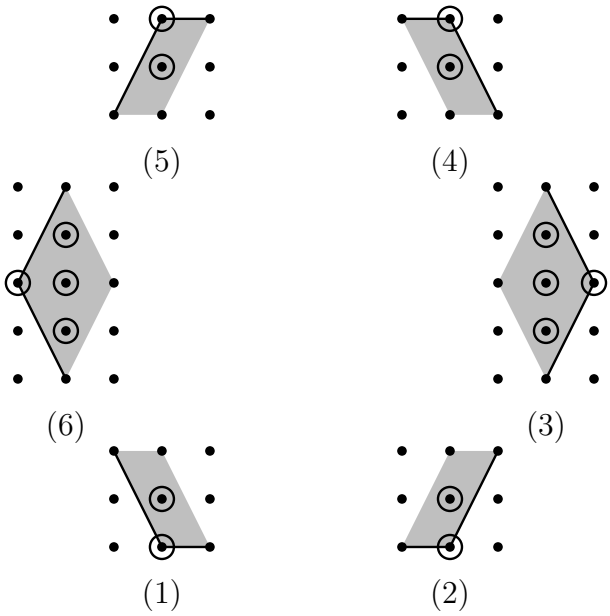


Fig. 3. Six vertices of a hexagon with fundamental domains shaded

$$F_1(z_1,z_2) = \frac{1+z_2}{(1-z_1)(1-z_1^{-1}z_2^2)}$$
$$F_2(z_1,z_2) = \frac{1+z_2}{(1-z_1^{-1})(1-z_1z_2^2)}$$
$$F_3(z_1,z_2) = \frac{1+z_1^{-1}(1+z_2+z_2^{-1})}{(1-z_1^{-1}z_2^2)(1-z_1^{-1}z_2^{-2})}$$
$$F_4(z_1,z_2) = \frac{1+z_2^{-1}}{(1-z_1^{-1})(1-z_1z_2^{-2})}$$
$$F_5(z_1,z_2) = \frac{1+z_2^{-1}}{(1-z_1)(1-z_1^{-1}z_2^{-2})}$$
$$F_6(z_1,z_2) = \frac{1+z_1(1+z_2+z_2^{-1})}{(1-z_1z_2^2)(1-z_1z_2^{-2})}$$

Brion's theorem gives:

$$
\begin{aligned}
G_{a,b,c}(z_1,z_2) = (&z_1 - z_1^{2+a} + z_1z_2 - z_1^{2+a}z_2 - \\
&z_1^2z_2^2 + z_1^{1+a}z_2^2 - z_1^2z_2^3 + z_1^{1+a}z_2^3 - \\
&z_1^{1+a+b}z_2^{1+2b} + z_1^{2+a+b}z_2^{1+2b} - z_1^{1+a+b}z_2^{2+2b} + \\
&z_1^{3+a+b}z_2^{2+2b} - z_1^{1+a+b}z_2^{3+2b} + z_1^{2+a+b}z_2^{3+2b} - \\
&z_1^{1-c}z_2^{1+2c} + z_1^{2-c}z_2^{1+2c} + z_1^{2-c}z_2^{2+2c} - \\
&z_1^{-c}z_2^{2+2c} - z_1^{1-c}z_2^{3+2c} + z_1^{2-c}z_2^{3+2c} - \\
&z_1^{2+b-c}z_2^{1+2b+2c} + z_1^{1+a+b-c}z_2^{1+2b+2c} - \\
&z_1^{2+b-c}z_2^{2+2b+2c} + z_1^{1+a+b-c}z_2^{2+2b+2c} + \\
&z_1^{1+b-c}z_2^{3+2b+2c} - z_1^{2+a+b-c}z_2^{3+2b+2c} + \\
&z_1^{1+b-c}z_2^{4+2b+2c} - z_1^{2+a+b-c}z_2^{4+2b+2c}) / \\
&(z_1 - z_1^2 - z_2^2 + z_1z_2^2 - z_1^2z_2^2 + z_1^3z_2^2 + z_1z_2^4 - z_1^2z_2^4)
\end{aligned}
$$

As described in Section 2 we need a 1 in the denominator so as to move a suitable term to the left hand side. We divide the numerator and denominator by z_1 to achieve this and we can now translate to pseudo-code.

The denominator gives this block of pseudo code:

```
for (x2 decreasing) {
    for (x1 increasing) {
        F[x1][x2] = G[x1][x2]+F[x1+1][x2]+
            F[x1-1][x2+2]-F[x1][x2+2]+
            F[x1+1][x2+2]-F[x1+2][x2+2]-
            F[x1][x2+4]+F[x1+1][x2+4];
    }
}
```

From the numerator we may now deduce that for any a, b and c we may now find the sum of the pixels within the hexagon $H_{a,b,c}$ translated to (x_1,x_2) with the following constant time per pixel computation:

```
H[x1][x2] =
    F[x1][x2]-F[x1+1+a][x2]+F[x1][x2+1]-
    F[x1+1+a][x2+1]-F[x1+1][x2+2]+
    F[x1+a][x2+2]-F[x1+1][x2+3]+F[x1+a][x2+3]-
    F[x1+a+b][x2+1+2*b]+F[x1+1+a+b][x2+1+2*b]-
    F[x1+a+b][x2+2+2*b]+F[x1+2+a+b][x2+2+2*b]-
    F[x1+a+b][x2+3+2*b]+F[x1+1+a+b][x2+3+2*b]-
    F[x1-c][x2+1+2*c]+F[x1+1-c][x2+1+2*c]+
    F[x1+1-c][x2+2+2*c]-F[x1-1-c][x2+2+2*c]-
    F[x1-c][x2+3+2*c]+F[x1+1-c][x2+3+2*c]-
    F[x1+1+b-c][x2+1+2*b+2*c]+
    F[x1+a+b-c][x2+1+2*b+2*c]-
```

(a) Unprocessed high dynamic range image

(b) Image processed with $(a,b,c) = (45,45,45)$

(c) Image processed with $(a,b,c) = (2,45,2)$

(d) Image with processed varying (a,b,c)

Fig. 4. High dynamic range images processed with the proposed algorithm

```
F[x1+1+b-c][x2+2+2*b+2*c]+
F[x1+a+b-c][x2+2+2*b+2*c]+
F[x1+b-c][x2+3+2*b+2*c]-
F[x1+1+a+b-c][x2+3+2*b+2*c]+
F[x1+b-c][x2+4+2*b+2*c]-
F[x1+1+a+b-c][x2+4+2*b+2*c];
```

We show some example processed images in Figure 4. In each case, after the generalised summed area table was computed, each pixel required reading precisely 28 pixels from the source image regardless of the size of the kernel.

5. DISCUSSION

There are some ways to simplify the transfer functions. For example, if all of the coordinates of the vertices of a polygon are divisible by an integer n then the algorithm proposed here allows us to view convolution with this polygon as convolution with the nth dilate of a smaller and simpler polygon. We described above how the tangent cone is generators by vectors along the polygon edges. But if both components of one of these generators are divisible by n

then we can divide it by n and still generate the same cone. In effect we're just tiling the same cone with smaller tiles. Large fundamental domains can result in complicated expressions so it is advantageous to divide each of these vectors by the highest common factor of its two components. In general, the complexity of computing the transfer function as a rational function grows polynomially with the complexity of the polygon for suitably defined notions of complexity. See [Barvinok and Pommersheim 1997] for details. So although for each integral convex polygon we have shown that there is a corresponding fast filter, an algorithm that takes as input both an image and a general integral convex polygon and image and returns the convolution may have high complexity.

We have written our transfer functions in the form $P(\boldsymbol{z})/Q(\boldsymbol{z})$ with P and Q polynomials, but they could be rearranged in many ways giving different algorithms at the expense of requiring more passes. A computer algebra package can be valuable in finding good representations. Typically the form we have been using requires implementations with the minimum number of passes, two, and hence is better behaved with resepct to cache coherence.

Note that when applying the running sum filter it is sometimes possible to obtain intermediate values that are large, exceeding the limits of the data type used. However, if we know that the final computed values for the filter are all smaller than a large integer, say N, then we can perform all of the computations modulo N. Although the intermediate values of the computation are known only modulo N final result is determined exactly. Typically we choose $N = 2^b$ where b is the number of bits in the word size. That way all integer additions and multiplications are automatically computed modulo N on most CPUs. (See the standard signal processing literature (e.g. [Lim 1989],[Dudgeon and Mersereau 1983]) for the theory behind the stability of filters when using finite-precision floating point numbers.)

We have been considering the implementation of filters in two passes corresponding to the numerator and denominator of our transfer functions. Once the pass from the denominator has been precomputed, the filter derived from the numerator requires, for each output pixel, a sum of a constant number of values from the image, making it well adapted to implementation on a GPU. This allows fast on-the-fly modification of the dilation size.

The methods described here can be extended to non-square pixels. For example we can derive a generalisation of summed area tables for hexagonal regions of hexagonal pixels. [Staunton 1999]

This method could also be used in a way similar to Crow's [Crow 1984] paper as an alternative to anti-aliasing texture maps with rectangular summed area tables.

It is surprising to find such a valuable connection between combinatorics and 2D graphics, but in fact many combinatorial problems give rise to rational generating functions (see [Stanley 1999], [Wilf 1994]) that could potentially be interpreted as transfer functions. It is interesting to speculate that maybe there are more useful theorems in the combinatorial literature awaiting translation into the language of graphics or image processing.

6. ACKNOWLEDGEMENTS

I'd like to thank J.P.Lewis and Greg Abbas for their helpful feedback during the writing of this paper.

REFERENCES

BARVINOK, A. AND POMMERSHEIM, J. 1996-1997. An algorithmic theory of lattice points in polyhedra. In *New Perspectives in Algebraic Combinatorics*. 91–147.

BRION, M. 1988. Points entiers dans les polyèdres convexes. *Ann. Sci. École Norm. Sup.* 21, 653–663.

BRION, M. AND VERGNE, M. 1997. Lattice points in simple polytopes. *Journal of the American Mathematical Society 10*, 371–392.

CROW, F. C. 1984. Summed-area tables for texture mapping. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*. ACM Press, 207–212.

DUDGEON, D. E. AND MERSEREAU, R. M. 1983. *Multidimensional Digital Signal Processing*. Prentice-Hall.

HECKBERT, P. S. 1986. Filtering by repeated integration. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*. ACM Press, 315–321.

LEWIS, J. 1995. Fast normalized cross-correlation. Vision Interface.

LIM, J. S. 1989. *Two-Dimensional Signal and Image Prcoessing*. Prentice Hall.

SOLER, C. AND SILLION, F. X. 1998. Fast calculation of soft shadow textures using convolution. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM Press, 321–332.

STANLEY, R. P. 1999. *Enumerative combinatorics. Vol. I.* Number 49 in Cambridge Studies in Advanced Mathematics. Cambridge University Press.

STAUNTON, R. 1999. *Hexagonal Sampling in Image Processing*. Vol. 107. Academic Press.

SUN, C. 2003. Diamond, hexagon, and general polygonal shaped window smoothing. In *Proc. VIIth Digital Image Computing: Techniques and Applications*, S. C., T. H., O. S., and A. T., Eds. Sydney.

WILF, H. S. 1994. *generatingfunctionology*, Second ed. Academic Press Inc., Boston.

WOODS, J. W. 2006. *Multidimensional Signal, Image, and Video Processing and Coding*. Academic Press, Inc., Orlando, FL, USA.