Optimized Evaluation of Box Splines via the Inverse FFT

Michael D. McCool

Computer Graphics Laboratory
Department of Computer Science, University of Waterloo, Waterloo, Ontario, N2L 3G1
mmccool@watcgl.uwaterloo.ca http://www.cgl.uwaterloo.ca/~mmccool/
Tel: (519) 888 4567 x4422

Abstract

Box splines are a multivariate extension of uniform univariate B-splines. Direct evaluation of a box spline basis function can be difficult, but they have a relatively simple Fourier transform and can therefore be evaluated with an inverse FFT. Symmetry, recursive evaluation of the coefficients, and parallelization can be used to optimize performance. A windowing function can also be used to reduce truncation artifacts. We explore all these options in the context of a high-performance parallel implementation. Our goal is the provision of an empirical touchstone for the inverse FFT evaluation of box spline basis functions.

Keywords: Box spline evaluation. B-splines. Fast Fourier Transforms (FFT). Parallelism.

Multivariate box splines [6] have several properties that would be useful in the context of a splat-based volume renderer [13, 15], but one: they lack an efficient and general evaluation technique.

This paper is a detailed analysis of the optimization of the inverse FFT evaluation technique, which is one possible but underexploited approach to solving this problem. Since the Fourier transform of a box spline basis function is known and has a simple closed form, samples of it can be evaluated on a grid and a multivariate inverse FFT can be used to evaluate a grid of samples of the spline. The inverse FFT algorithm is not difficult in concept. Achieving adequate performance, however, requires attention to detail and it is this detail that we are attempting to present in this paper. We optimize both the inverse FFT and the evaluation of the coefficients to achieve an absolute evaluation rate compatible with interactive volume rendering. We also employ Parzen windowing to avoid negative values and ringing artifacts in the output.

Our implementation runs in parallel on a 16-processor shared memory multiprocessor. We can draw some useful qualitative conclusions from the behaviour of our implementation. The most important observed property of the inverse FFT algorithm is the fact that it is insensitive to high multiplicies in the box spline. This is potentially very useful in the context of splat volume rendering, where the box spline is formed from the projection of a tensor product B-spline reconstruction kernel.

This document is organized as follows: first we review basic B-spline and box spline theory, so the paper will be self-contained. Then, we compare the inverse FFT evaluation technique with other existing evaluation methods. Many evaluation techniques have been developed for particular box splines as used in computer aided geometric design (CAGD), but some of these techniques do not apply to the rendering application we have in mind. After presenting the naïve algorithm, we give the details of the symmetry optimizations, windowing, and coefficient recurrence optimizations. Finally, we present empirical results from our implementation.

Box Splines and B-Splines

Box splines are one possible multivariate generalization of the univariate B-splines [4, 5, 7, 3]. The uniform univariate B-spline basis functions of order n,

$B_n(x)$, can be defined by recursive convolution:

$$\begin{aligned} B_0(x) &= \delta(x), \\ B_n(x) &= \int_{[0,1)} B_{n-1}(x-t)\,dt. \end{aligned}$$

Here $\delta(x)$ is the Dirac delta distribution defined by $\int \phi(x-t)\delta(t)\,dt = \phi(x)$ for any continuous $\phi(x)$.

The support of $B_n(x)$ is given by $[0,n)$; this asymmetry can be a nuisance, so the centered spline $B_{nc}(x) = B_n(x+n/2)$ with support on $[-n/2, n/2)$ is often very useful.

Also pertinent to this discussion is the Fourier transform of $B_n(x)$,

$$\widehat{B}_n(\omega) = \left(\frac{1-\exp(-\mathrm{i}\omega)}{\mathrm{i}\omega}\right)^n,$$

where $\mathrm{i}^2 = -1$. The Fourier transform of the centered spline $B_{nc}(x)$ is simpler:

$$\widehat{B}_{nc}(\omega) = \operatorname{sinc}^n(\omega/2),$$

where $\operatorname{sinc}(\nu) = \sin(\nu)/\nu$ for $\nu \neq 0$ and $\operatorname{sinc}(0) = 1$. Because the centered spline is symmetric, its Fourier transform is real. Because either spline basis is real, both Fourier transforms possess (Hermetian) symmetry.

Box spline basis functions can be defined similarily, but because they are multivariate we first need to introduce some extra structure. The $m \times n$ *direction vector matrix* Ξ is defined as an ordered sequence of m-dimensional column vectors $\boldsymbol{\xi}_i$: $\Xi = [\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \ldots, \boldsymbol{\xi}_n]$. If a direction vector $\boldsymbol{\xi}_i$ is repeated μ times, we say it has *multiplicity* μ.

The box spline basis function $M_n(\mathbf{x}|\Xi)$, with $\mathbf{x} \in \mathbb{R}^m$, can then be defined recursively by

$$\begin{aligned} M_0(\mathbf{x}|\emptyset) &= \delta(x), \\ M_n(\mathbf{x}|\Xi) &= \int_{[0,1)} M_{n-1}(\mathbf{x} - t\boldsymbol{\xi}_n | \Xi \backslash \boldsymbol{\xi}_n)\,dt. \end{aligned}$$

Here $\Xi\backslash\boldsymbol{\xi}_n$ means the vector $\boldsymbol{\xi}_n$ is removed from Ξ. Box splines have a multivariate Fourier transform given by

$$\widehat{M}_n(\boldsymbol{\omega}|\Xi) = \prod_{i=1}^{n} \frac{1-\exp(-\mathrm{i}\boldsymbol{\omega}\cdot\boldsymbol{\xi}_i)}{\mathrm{i}\boldsymbol{\omega}\cdot\boldsymbol{\xi}_i}. \qquad (1)$$

As with the B-splines, we can define a centered form. Let $\mathbf{c} = \frac{1}{2}\sum_i \boldsymbol{\xi}_i$; then define $M_{nc}(\mathbf{x}|\Xi) = M_n(\mathbf{x}+\mathbf{c}|\Xi)$. The Fourier transform of the centered box spline is

$$\widehat{M}_{nc}(\boldsymbol{\omega}|\Xi) = \prod_{i=1}^{n} \operatorname{sinc}(\boldsymbol{\omega}\cdot\boldsymbol{\xi}_i/2). \qquad (2)$$

If we only have r unique direction vectors $\boldsymbol{\xi}_j$, with multiplicities denoted by μ_j, then

$$\widehat{M}_{nc}(\boldsymbol{\omega}|\Xi) = \prod_{j=1}^{r} \operatorname{sinc}^{\mu_j}(\boldsymbol{\omega}\cdot\boldsymbol{\xi}_j/2). \qquad (3)$$

For CAGD purposes, the integral of the box spline is often normalized to 1 by dividing by $|\det \Xi|$. Since Ξ is rectangular, evaluation of this determinant requires some thought [6]. The vectors $\boldsymbol{\xi}_i$ contained in Ξ are also often limited to $\mathbb{Z}^m$, so the basis functions can be used in grids to form spline surfaces. For rendering purposes, neither of these limitations are mandatory, and so we will not mention them further.

Some properties of box splines are useful. First, they reduce to tensor product B-splines if the number of unique direction vectors r is equal to m, the dimension of the parameter space. Second, a box spline with r unique direction vectors is the projection of a tensor product B-spline basis function in r dimensions. Third, the convolution of two box spline basis functions can be found by simply concatenating their direction vector matrices. Finally, box splines satisfy recurrence and subdivision relationships that are very similar to those of the uniform univariate B-splines.

Taken together, some interesting graphics and signal processing applications can be envisioned, particularly if we also consider the connection of the univariate B-splines to digital signal processing [12]. Many CAGD applications have already been considered in the literature; some examples are given in [8, 6, 3]. Our work, however, is primarily motivated by the potential application of box splines to volume rendering, particularly "splat" volume rendering [13, 14, 15] with tensor-product B-spline reconstruction kernels. The primary difficulty with this application is the lack of a truly efficient evaluation technique for box splines. In the next section we review some evaluation techniques, and point out some of the properties of the inverse FFT algorithm which are particularly appealing.

Evaluation Taxonomy

Box spline evaluation is, in general, more difficult than univariate B-spline evaluation. This is mostly due to the increased flexibility that box splines provide. With care, efficient algorithms can be obtained for useful subclasses of box splines that possess adequate structure. General evaluation algorithms can exploit one or more of the following properties of box splines: (1) recurrence (possibly preceeded by a multivariate truncated power function decomposition), (2) two-scale subdivision, or (3) the Fourier transform. Other specialized techniques for CAGD applications, such as Bézier patch decomposition [9], are not germane to this dicussion because they depend on a static configuration of direction vectors. In rendering applications this may not be true, as can be seen from the comparison given in Figure 1.

Of the relevant evaluation techniques, only recurrence is exact. Unfortunately, it can be very expensive. For completely general box splines the cost increases combinatorically with the number of direction vectors. With care, on restricted classes of box splines, this explosion can be contained somewhat, but a large amount of arithmetic is still needed for every sample.

Subdivision leads to an approximate iterative technique which converges quadratically. The continuous inverse Fourier transform of the box spline can only be approximated with a discrete inverse Fourier transform, so inverse FFT evaluation is also an approximation. Both of these techniques share the property that a large number of samples are computed at once, and in fact they have similar asymptotic complexity measures.

Strangely, inverse FFT evaluation has not been looked at seriously in the literature [6]. The resulting evaluation algorithm is, however, asymptotically relatively efficient for high multiplicity box splines and is appropriate for some computer architectures, i.e. parallel machines or machines with a DSP coprocessor. Since parallel machines are often used for volume rendering, and high multiplicity box splines are potentially useful in that application, it is worthwhile to investigate the inverse FFT approach.

Certain optimizations can be made based on the real and symmetric nature of both the Fourier transform and the centered box spline that can reduce the overall computational load by a factor of four. We also derive a recursive technique to evaluate the Fourier coefficients, the starting point of the evaluation technique, with very few trigonometric evaluations.

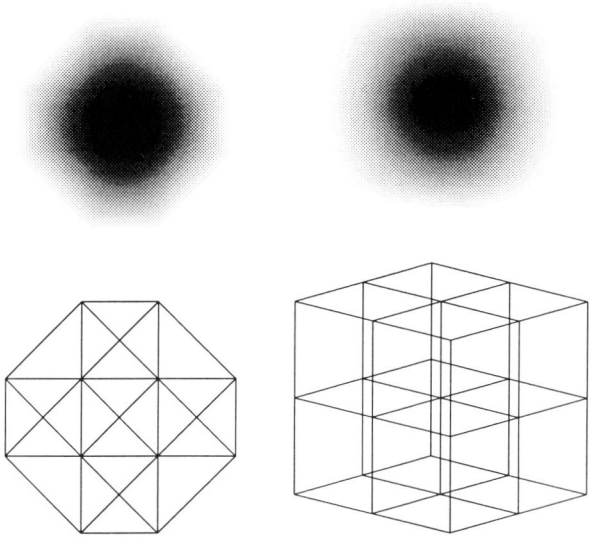


Figure 1: *Box splines for volume rendering have to be more flexible than those used in geometric design. On the left is the Zwart-Powell element often used in CAGD and finite element analysis; on the right, one instance of the projection of a tensor product of three tent functions (second order univariate B-splines), as would be used for a trilinear reconstruction kernel in volume rendering. Also shown are the discontinuity maps of these splines.*

The Inverse FFT Algorithm

The Fourier transform of an uncentered box spline (1) is in general complex but exhibits conjugate symmetry as $M_n(\mathbf{x}|\Xi)$ is real. The Fourier transform of the centered box spline (2,3) is even simpler: $\widehat{M}_{nc}(\boldsymbol{\omega}|\Xi)$ is not only symmetric about $\boldsymbol{\omega} = 0$, but entirely real. The naïve FFT evaluation algorithm proceeds as follows:

1. Choose spatial sample spacings Δx_ℓ and a bounding box containing the support of the box spline. Let X_ℓ be the length of the bounding

box, Δx_ℓ the sample spacing, and N_ℓ the number of samples in the ℓ^{th} dimension. For a radix-2 FFT algorithm, N_ℓ should be a power of two. Compute the spacing of the frequency samples: $\Delta\omega_\ell = 2\pi/X_\ell$ for all ℓ.

2. Create a periodized version $\widehat{M}_{nc\phi}(\boldsymbol{\omega}|\Xi)$ of $\widehat{M}_{nc}(\boldsymbol{\omega}|\Xi)$ by replicating a base period in the frequency domain, $\Omega = \times_\ell[-\Omega_\ell/2, \Omega_\ell/2)$, with $\Omega_\ell = N_\ell \Delta\omega_\ell$. Any energy in $\widehat{M}_{nc}(\boldsymbol{\omega}|\Xi)$ outside of the base period will contribute to truncation error.

3. Create $\widehat{M}_{nc\phi}[\mathbf{k}|\Xi]$ by sampling $\widehat{M}_{nc\phi}(\boldsymbol{\omega}|\Xi)$ at $\mathbf{k}\Delta\boldsymbol{\omega} = (k_1\Delta\omega_1, k_2\Delta\omega_2, \ldots, k_s\Delta\omega_s)$ with $0 \leq k_\ell \leq N_\ell - 1$.

4. Window if necessary to avoid ringing artifacts: $\widehat{M}_{ncw\phi}[\mathbf{k}|\Xi] = \widehat{w}[\mathbf{k}]\widehat{M}_{nc\phi}[\mathbf{k}|\Xi]$.

5. Perform an inverse m-dimensional FFT on $\widehat{M}_{ncw\phi}[\mathbf{k}|\Xi]$ to obtain $M_{ncw\phi}[\mathbf{j}|\Xi]$.

6. Unpack the data from the FFT by permuting quadrants and shifting the center by $\mathbf{c}_\Xi$, if required:

$$M_n(\mathbf{j}\Delta\mathbf{x} - \mathbf{c}_\Xi|\Xi) \approx M_{ncw\phi}[(\mathbf{j} + \mathbf{N}/2) \bmod \mathbf{N}|\Xi).$$

Basically, the infinite support of the Fourier transform of a box spline is truncated and discretized, possibly weighted with a window function, and then inverted.

We can easily compute samples of $\widehat{M}_{nc}(\boldsymbol{\omega}|\Xi)$; however, these are samples of the continuous Fourier transform, not the discrete Fourier transform (DFT) which is what the inverse FFT algorithm expects. Sampling in space and frequency implies periodicity in both space and frequency for the DFT. Therefore, a true DFT is periodic as well as sampled, and the inverse will also be periodic. Given one period of a DFT, the FFT computes one period of the result.

Fortunately, box splines and their transforms go to zero, or nearly to zero, towards the edges of their domains. The box spline $M_{nc}(\mathbf{x}|\Xi)$ goes to zero exactly outside its support (which can be computed in linear time), while the transform $\widehat{M}_{nc}(\boldsymbol{\omega}|\Xi)$ goes to zero as $\prod_{j=1}^{r}(\boldsymbol{\omega}\cdot\boldsymbol{\xi}_j)^{-\mu_j}$. To approximate a Fourier transform by a DFT, one has to window a part of the sampled integral Fourier transform and use it as a base period which is then implicitly replicated.

The simplest window is rectangular, i.e. just a finite number of raw samples. Truncation will actually give the best least-squares approximation [1], but this fact doesn't guarantee that the approximation will be free of objectionable artifacts for all applications.

Because of periodicity in space, one must be careful to separate spatial duplicates to avoid having non-zero samples from one period overlap with non-zero samples of another. This can be ensured by using a spatial period which contains a bounding box around the support of the box spline. Radix 2 FFT algorithms also require data dimensions which are perfect powers of two, and so the dimensions of the bounding box, N_ℓ, must be rounded up to the nearest power of two. More general FFT algorithms are possible, but at the very least each N_ℓ should be highly composite, i.e. a product of a few small integer factors. Consequently, when evaluating the amortized cost of each computed sample with the support of a box spline, the proportion of samples wasted within this bounding box will be important. If the minimal bounding box containing the support of a two-dimensional box spline covers, for example, a 257×129 square, a 512×512 radix-2 FFT will be required. At least 3/4 of the computation will be wasted, although the extra samples in frequency will contribute somewhat to the accuracy. Depending on the shape of the support, it may be possible to pack the periods more tightly together than the minimax bounding box would suggest, although the conditions for this situation can be so easily violated that it is better to be conservative.

The required inverse FFT is of course multivariate for the evaluation of multivariate splines. A multivariate FFT can be computed directly using a specialized FFT algorithm. The kernel of the multivariate DFT is separable, so a multivariate FFT can also be performed by univariate FFTs applied to each dimension in turn. The multiple univariate transforms within a dimension can be computed independently, as can the generation of the coefficients, and so this algorithm is naturally highly parallel. Some communication cost is incurred when the data needs to be transposed for transforms across the various dimensions.

Symmetry Optimizations

The symmetry of $\widehat{M}_{nc}(\boldsymbol{\omega}|\Xi)$ means that we can immediately halve our initial work by only computing half the samples in $\widehat{M}_{nc\phi}[\mathbf{k}|\Xi]$, then infer the rest by reflection. The symmetry condition for DFT coefficients in the standard FFT order is $\widehat{M}_{nc\phi}[\mathbf{k}|\Xi] = \widehat{M}^{*}_{nc\phi}[\mathbf{N}-\mathbf{k}|\Xi]$, for real $M_{nc\phi}[\mathbf{j}|\Xi]$. Symmetry also allows us to make some savings in the computation of the FFT itself.

In the following we assume $m = 2$ for simplicity, and also because this will be the most common case. Assume we compute a two dimensional inverse FFT by computing all the column inverse transforms followed by all the row inverses. The algorithm would proceed by computing real values of $\widehat{M}_{nc\phi}[k_1, k_2|\Xi]$ located at all $(k_1, k_2) \in \{0, 1, \ldots, N_1/2\} \times \{0, 1, \ldots, N_2 - 1\}$, then would compute all the inverse column transforms to derive half of an intermediate transform $b[k_1, k_2]$. Columns $k_1 = \{N_1/2 + 1, \ldots, N_1 - 1\}$ in $\widehat{M}_{nc}[k_1, k_2|\Xi]$ are reflections of columns $k_1 = \{1, \ldots, N_1/2 - 1\}$ in $\widehat{M}_{nc}[k_1, k_2|\Xi]$; the discrete version of the symmetry noted above is $\widehat{M}_{nc}[k_1, k_2|\Xi] = \widehat{M}_{nc}[N_1 - k_1, N_2 - k_2|\Xi]$. This implies that the corresponding coefficients in columns $\{N_1/2 + 1, \ldots, N_1 - 1\}$ of $b[k_1, k_2]$ are the complex conjugates of those for their corresponding computed columns: $b[N_1 - k_1, k_2] = b^{*}[k_1, k_2]$. Equivalently, since the samples in $\widehat{M}_{nc\phi}[k_1, k_2|\Xi]$ are real and therefore their Fourier transforms are even, $b[N_1 - k_1, N_2 - k_2] = b[k_1, k_2]$. In the second phase of the algorithm, we need to perform row inverse FFT's to obtain the values of the centered box spline. Since the centered box spline is also even in a two-dimensional sense, we only need to compute $N_2/2 + 1$ row inverses, then use the symmetry of the box spline itself to fill in the remaining values.

Exploitation of symmetry in this way gains a factor of two over the naïve algorithm. Another factor of two can be gained by exploiting the fact that both the centered box spline and its transform are real, while the FFT calculates with complex arithmetic. By packing one column of coefficients in the real part and one in the imaginary part of the data passed to an FFT routine, then using even and odd properties to unpack the result, we can double our throughput. Likewise, for the second phase we can set up the coefficients so that two rows can be computed at once, doubling throughput at that stage as well. In the following we describe these optimizations more precisely.

In the first, column transformation stage, create a set of $N_1/4+1$ complex coefficient columns as follows:

$$\widehat{a}[\ell_1, k_2] = \begin{cases} \widehat{M}_{nc\phi}[2\ell_1, k_2|\Xi] + \mathrm{i}\widehat{M}_{nc\phi}[2\ell_1 + 1, k_2|\Xi] & \\ \qquad\qquad (\text{for } \ell_1 < N_1/4); & \\ \widehat{M}_{nc\phi}[2\ell_1, k_2|\Xi] \qquad (\text{for } \ell_1 = N_1/4). & \end{cases}$$

Since all values of $\widehat{M}_{nc\phi}[k_1, k_2|\Xi]$ are real,

$$\begin{aligned} \widehat{M}_{nc\phi}[2\ell_1, k_2|\Xi] &= \Re\{\widehat{a}[\ell_1, k_2]\}, \\ \widehat{M}_{nc\phi}[2\ell_1 + 1, k_2|\Xi] &= \Im\{\widehat{a}[\ell_1, k_2]\}. \end{aligned}$$

Note that because $N_1/2+1$ is odd, one of the packed columns in $\widehat{a}$ has a zero imaginary part. Let $a[p_1, k_2]$ contain the inverse univariate FFT of each of the columns in $\widehat{a}[\ell_1, k_2]$. To unpack each column, we exploit the facts that FFTs of real signals are even, FFTs of imaginary signals are odd, superposition holds, an inverse FFT can be computed by taking the conjugate of the FFT of a conjugated signal, and symmetry properties hold [2] to derive the formulae

$$b[j_1, k_2] = \begin{cases} (a[j_1/2, k_2] + a^{*}[j_1/2, N_2 - k_2])/2 & \\ \qquad (\text{for } j_1 \bmod 2 = 0, j_1 < N_1/2); & \\ -\mathrm{i}\,(a[j_1/2, k_2] - a^{*}[j_1/2, N_2 - k_2])/2 & \\ \qquad (\text{for } j_1 \bmod 2 = 1, j_1 < N_1/2); & \\ a[N_1/4] \qquad (\text{for } j_1 = N_1/2); & \\ b^{*}[N_1 - j_1, k_2] \qquad (\text{for } j_1 > N_1). & \end{cases}$$

We will now be taking the inverse FFTs of the first $N_2/2 + 1$ rows of $b[j_1, k_2]$ to determine the first $N_2/2 + 1$ rows of the box spline; the rest can be inferred by symmetry. Note that we have expressed the symmetry in b above so the rows $k_2 > N_2/2$ never have to be calculated. Since we know the result will be real, we would like to pack the coefficients so we can compute two rows at once. Again, we can use the odd/even relationship to derive new packed row coefficients $\widehat{c}[j_1, \ell_2]$ that will have the appropriate properties:

$$\widehat{c}[j_1, \ell_2] = \begin{cases} b[j_1, 2\ell_2] + \mathrm{i}b[j_1, 2\ell_2 + 1] & (\text{for } \ell_2 < N_2/4); \\ b[j_1, 2\ell_2] & (\text{for } \ell_2 = N_2/4). \end{cases}$$

Note again that since $N_2/4+1$ is odd by assumption the last transform only computes the inverse transform of one signal. Since b is complex, however, all rows will have generally nonzero real and imaginary parts. The row-wise inverse FFT c of $\widehat{c}$ will be the desired box spline values, in FFT order and packed into the real and imaginary parts:

$$M_{nc\phi}[j_1, j_2|\Xi] = \begin{cases} \Re\{c[j_1, j_2/2]\} \text{ (for } j_2 \bmod 2 = 0); \\ \Im\{c[j_1, j_2/2]\} \text{ (for } j_2 \bmod 2 = 1). \end{cases}$$

The refined algorithm, including all the above symmetry considerations for two dimensions, is given below. Similar optimizations could be made for higher dimensions.

1. Center the box spline and choose spatial sample spacings, bounding box, etc.
2. Model $\widehat{M}_{nc}(\boldsymbol{\omega}|\Xi)$ by $\widehat{M}_{nc\phi}(\boldsymbol{\omega}|\Xi)$.
3. Create $\widehat{M}_{nc\phi}[\mathbf{k}|\Xi]$ by sampling $\widehat{M}_{nc\phi}(\boldsymbol{\omega}|\Xi)$ at $\mathbf{k}\Delta\boldsymbol{\omega} = (k_1\Delta\omega_1, k_2\Delta\omega_2)$ with $k_1 \in \{0, 1 \ldots N_1/2\}$ and $k_2 \in \{0, 1, \ldots N_2 - 1\}$.
4. Window if necessary to create $M_{ncw\phi}[\mathbf{k}|\Xi]$.
5. Create $\widehat{a}$ by packing the sampled coefficients.
6. Take inverse FFTs of all columns in $\widehat{a}$ to create a.
7. Convert a to $\widehat{c}$, i.e. unpack and repack using the formulas for b. Only the non-redundant entries in b need to be computed.
8. Take inverse FFTs of all rows in $\widehat{c}$ to create c.
9. Unpack the real and imaginary parts of c and reflect and shift as necessary to obtain the values of the box spline in natural order. Shift the center back to its correct location if an uncentered box spline was required.

In summary, by exploiting symmetries we have reduced the overall FFT computation cost by nearly a factor of four, while reducing memory cost by nearly a factor of two. The extra overhead incurred is limited to a slightly more complicated transposition that involves some addition.

Windowing

Windowing functions are used to modify the effect bandlimiting has on the FFT approximation. Essentially, a window function is a filter that smooths a signal in the spatial domain so that it may be better approximated by an FFT. By a standard result, the lowest RMS error is actually achieved if a rectangular window (truncation) is used [1], but the RMS metric may be inappropriate for applications that require local error bounds or conditions such as nonnegativity.

Because box splines are already fairly smooth for most configurations of direction vectors, a window function may be unnecessary. However the Parzen window is easy to implement and may be useful in some circumstances. The univariate Parzen window,

$$\begin{aligned} \widehat{w}_p(\omega) &= (1 - |\omega/\omega_N|)\, \chi_{[-\omega_N, \omega_N)}(\omega) \\ &= \chi_{[-\omega_N/2, \omega_N/2)}(\omega) * \chi_{[-\omega_N/2, \omega_N/2)}(\omega), \end{aligned}$$

is a "tent" function which decreases from 1 at $\omega = 0$ to 0 at the Nyquist frequency $\omega_N = \omega_s/2$, half the sampling frequency, in both directions. A multivariate window function would be formed with a tensor product. This window and its inverse Fourier transform is shown in Figure 2.

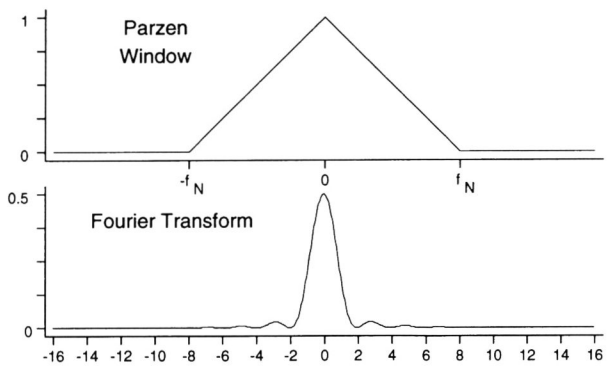


Figure 2: *The Parzen (tent) window function, along with its inverse Fourier transform.*

We call the inverse Fourier transform the *impulse response* of the window function, since it is the image that would result from convolution with an impulse; the Fourier transform of an impulse is a constant at all frequencies. The effect of multipling by a window function in the frequency domain is, according

to the duality principle, equivalent to convolving by the window function's impulse response in the spatial domain. Hence, the shape of the impulse responses of these functions gives us an idea of their spatial properties.

Since the Parzen window can be recognized as a second order uniform B-spline basis function, which is the convolution of two box functions, its impulse response is a squared sinc: $w_p(x) = \text{sinc}^2(x/4)$, which has multiple modes and decreases at infinity by $1/x^2$ but is always positive. The function $w_p(x)$ is sometimes called the Fejér kernel and its positivity ensures the absence of overshoot. In particular, no negative values can occur in the output since the spline is also totally positive.

Extensions to the Parzen window include the univariate B-spline basis functions, which converge to the Gaussian window function. Using B-spline windows requires dividing the domain into more than two sections, which is slightly inconvenient. Fourier inverses of even order B-splines will share the positivity of the Fejér kernel, since their transforms will be of the form $\text{sinc}^{2k}(x/2k)$ with $k \in \mathbb{N}$.

Coefficient Evaluation

In addition to computing the inverse FFT, we also need to compute the input coefficients. Each sample of the box spline's Fourier transform $\widehat{M}_{nc}(\mathbf{x}|\Xi)$ requires r sine evaluations, divisions, and exponentiations if Ξ contains r unique direction vectors. Trigonometric functions and divisions can be fairly expensive. Trigonometric functions consume a budget of 10 to 50 floating-point multiplications on a range of high-performance machines [10]. Division, while not as expensive, cannot usually be pipelined.

Some optimizations are possible. Since we need to compute so many equally spaced samples we can take advantage of trigonometric recurrences to incrementally compute the sine values. The exponentiation can be done in sublinear time by iterating multiplications. For common multiplicity values, exponentiation can be performed at a cost of only a few multiplications which can be written explicitly to maximize pipelining. For example, exponentiation by four, probably the maximum multiplicity in practice, requires only two multiplications.

The following recurrence can be used to compute sequential trigonometric coefficient values:

$$\sin(\Delta\omega j) = \quad (4)$$
$$2\cos(\Delta\omega)\sin(\Delta\omega(j-1)) - \sin(\Delta\omega(j-2)).$$

The value $2\cos(\Delta\omega)$ is a constant which only needs to be computed once during preparation. In addition to this constant, initialization of the recurrence requires the evaluation of two samples of the sine function, which can be used going both forwards and backwards from 0. In a parallel implementation, initializing and scanning each row still leaves a large number of independent work packets. After initialization, this recurrence requires only a single multiplication and an addition per sine sample. This can be accomplished in a multiply-accumulate (MAC) operation, which many processors have as a single instruction and which typically maximizes floating-point performance.

Error for Recursive Sine Evaluation			
Iteration	ω	max rel err:	max abs err:
1000	165.3	7.7×10^{-14}	9.4×10^{-13}
2000	330.5	1.4×10^{-13}	2.4×10^{-10}
3000	495.8	2.0×10^{-13}	2.4×10^{-10}
Error for Recursive Sinc Evaluation			
Iteration	ω	max rel err:	max abs err:
1000	165.3	4.9×10^{-16}	9.2×10^{-13}
2000	330.5	4.9×10^{-16}	2.4×10^{-10}
3000	495.8	4.9×10^{-16}	2.4×10^{-10}

Figure 3: *Empirical error of sine and sinc evaluation via recurrence.*

Since the sine recurrence in (5) is only marginally stable, we need to evaluate how accurate it is in practice. In Figure 3 we empirically compare direct evaluation of sine and sinc functions to the incremental approach, using IEEE double-precision floating point. Note that the error increases slightly the farther we stray from the initialization; however, if we start at low frequency values and work up to the less important[1] high frequencies, the effect of the error can be masked. The error is further masked by the division, which damps the amplitude of the coef-

[1]Perceptually speaking; of course the truth of this statement depends on the application.

ficients with the highest error. Even after 3000 samples (far more than we will typically need in practice) we still maintain 10 digits of precision.

The division in the sinc introduces a singularity which must be dealt with to maintain the robustness of the algorithm. In the context of a two-dimensional grid of coefficients, this singularity will not necessarily occur at the zero frequency of the row, since it occurs along a line which cuts across the row being evaluated. The most efficient and robust approach to dealing with the singularity is to compute its position for each sine factor and work outwards from it. This results in an uncluttered inner loop that can be effectively optimized. The windowing function should be evaluated and applied in a separate step.

An Implementation

An implementation of the FFT evaluation algorithm was performed on a shared memory multiprocessing machine: the Pulsus G2 from ISG Technologies. This machine had 16 RISC processors (M88110's) connected via a hierarchy of buses to a large global memory. Every four processors also shared 512K of fast static memory. The implementation used the symmetry optimizations outlined above, as well as selectable direct or recursive coefficient evaluation. Parallelism was performed on a row-column basis, with automatic load balancing using a work packet server. Two barriers were needed to synchronize the processors after the coefficient computation and for the transition between row and column FFT's. Parzen windowing was implemented. Some sample evaluations are shown in Figure 5.

In this section we use this implementation as an empirical example to extract some qualitative properties of the FFT algorithm, in an attempt to characterize the niches for which it is suited. Our empirical results are contained in Figure 4.

Figure 4a compares the execution times for various problem sizes and various numbers of processors. Note that for small problem sizes, the processors saturate quickly and serial overhead quickly destroys efficiency. In the case of the smaller problem size execution time actually increases as the number of processors is increased. Larger problem sizes can more efficiently use a larger number of processors. This leads to our first observation: for a parallel implementation of the inverse FFT evaluation algorithm, larger resolutions are more efficient. This observation, which is a standard expectation for all parallel programs, is somewhat confused by the fact that on cache-based architectures, problem sizes larger than the cache size can affect efficiency as well. This particular machine has only a 4K data cache per processor, and cache misses result in expensive global memory accesses. On a more balanced architecture this would be less of a problem.

From the results in Figure 4b we see that the inverse FFT algorithm is relatively insensitive to multiplicities, an important characteristic that distinguishes it from the others we have mentioned (recurrence and subdivision). The difference in execution times between a 3rd and a 12th order spline is only 3%, if the order increases only by adding multiplicities. This leads to our second qualitative observation: multiplicities have little impact on performance. Note that in this implementation exponentiation was implemented using $\lceil \lg \mu_\ell \rceil$ multiplications.

Finally, in Figure 4c we compare direct evaluation of the Fourier coefficients to recursive evaluation. In direct evaluation a separate call to `sin()` is made for every sample. In recursive evaluation, there are only two calls to `sin()` and one to `cos()` per row per unique direction vector, with all other samples derived via a recurrence. A division is required for every sample in both cases. As is shown in the table, the performance increment is significant, at least for this machine which requires approximately 50 floating point operations for evaluation of the sine. Direct evaluation of the coefficients dominates the evaluation of the inverse FFT, while recursive evaluation is nearly negligible compared to the inverse FFT. On other processors with more efficient trigonometry the difference may be less significant, although we can still expect a factor of ten improvement. In all cases, evaluation of the coefficients via recurrence should be negligible. We should note that the implementation of the FFT that we use (from [11]) also uses a recurrence to avoid extraneous trigonometric evaluations. This is not a highly optimized version of the FFT, and a better implementation could probably improve the FFT timings. In particular, a radix-4 or radix-8 algorithm would reduce the memory to processor bandwidth by reducing the number of stages required in the FFT algorithm.

Conclusions

In summary, we have observed the following: (1) a parallel implementation of the inverse FFT algorithm can achieve significant speedup on a shared memory architecture; (2) larger grid resolutions are more efficient; (3) the FFT evaluation algorithm is relatively insensitive to multiplicities, growing only by $O(\lg \mu_\ell)$ but with a very small scale factor; and (4) direct evaluation of the coefficients consumes a significant fraction of the execution time, but recursive evaluation is effectively negligible relative to the cost of the inverse FFT.

These properties indicate that the inverse FFT algorithm is indeed suitable for some rendering applications, in particular splat-based volume rendering.

For two-dimensional splines, we can conclude from the asymptotic complexity of the FFT algorithm that this algorithm takes $O\left(r\lg\left(\sum_j \mu_j\right) + \lg N\right)$ time per sample, where r is the number of unique direction vectors, the μ_j are the multiplicities of each direction vector, and N is the maximum resolution. To achieve this, however, $O(N^2)$ samples need to be evaluated, and it should be remembered that the application of the FFT does require some small amount of approximation.

Acknowledgements

This research was performed while the author was a Ph.D. candidate at the Dynamic Graphics Project at the University of Toronto under the supervision of Eugene Fiume. The Dynamic Graphics project receives generous support from NSERC and the Information Technology Research Centre (ITRC). ISG Technologies kindly loaned DGP the parallel machine, the Pulsus G2, upon which this implementation was performed.

References

[1] R. N. Bracewell. *The Fourier Transform and Its Applications.* McGraw-Hill, 1978.

[2] E. O. Brigham. *The Fast Fourier Transform.* Prentice Hall, 1974.

[3] C. K. Chui. *Multivariate Splines.* SIAM, 1988.

[4] C. de Boor and K. Höllig. Recurrence relations for multivariate B-splines. *Proceedings of the American Mathematical Society*, 85(3):397–400, 1981.

[5] C. de Boor and K. Höllig. B-splines from parallelepipeds. *Journal d'Analyse Mathématique*, 42:99–115, 1983.

[6] C. de Boor, K. Höllig, and S. Riemenschneider. *Box Splines.* Academic Press, 1994.

[7] K. Höllig. Box splines. In *Collection: Approximation Theory V (College Station, Tex., 1986)*, pages 71–95. Academic Press, Boston, MA, 1986.

[8] K. Höllig. Box-spline surfaces. In *Collection: Mathematical methods in computer aided geometric design (Oslo, 1988)*, pages 385–402. Academic Press, Boston, MA, 1989.

[9] Ming Jun Lai. Fortran subroutines for B-nets of box splines on three- and four-directional meshes. *Numerical Algorithms*, 2(1):33–38, 1992.

[10] M. D. McCool. *Analytic Signal Processing for Computer Graphics using Multivariate Polyhedral Splines.* May 1995. University of Waterloo Department of Computer Science Tech Report CS-95-05.

[11] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes.* Cambridge University Press, 1989.

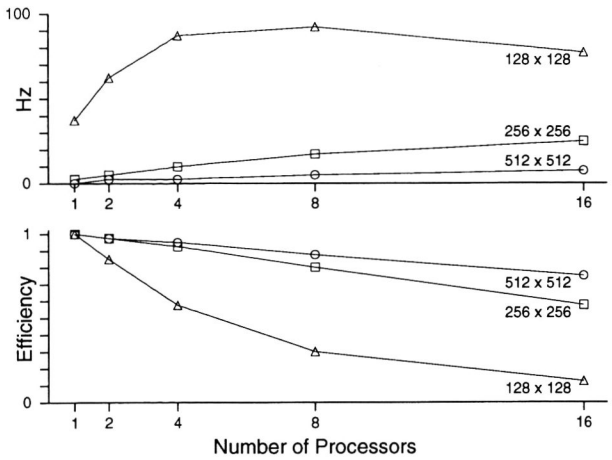
[12] M. Unser, A. Aldroubi, and M. Eden. B-spline signal processing. *IEEE Transactions on Signal Processing*, 41(2):821–848, February 1993.

[13] L. Westover. Interactive volume rendering. In *Chapel Hill Workshop on Volume Visualization*, pages 9–16, Chapel Hill, North Carolina, May 1989.

[14] L. Westover. Footprint evaluation for volume rendering. *Computer Graphics (SIGGRAPH '90 Proceedings)*, 24(4):367–376, August 1990.

[15] J. Wilhelms and A. Van Gelder. A coherent projection approach for direct volume rendering. *Computer Graphics (SIGGRAPH '91 Proceedings)*, 25(4):275–284, July 1991.

(a) Performance vs. Number of Processors



(b) Performance vs. Multiplicity			
Multiplicity	ms	Hz	ratio
1	128.39	7.79	1.00
2	130.50	7.66	0.98
3	132.93	7.52	0.97
4	132.34	7.56	0.97

(c) 2D FFT vs. Coefficient Evaluation			
Problem Size	FFT ms	Dir. ms	Rec. ms
128×128	16.05	25.72	2.96
256×256	32.98	51.44	5.87
512×512	108.55	213.81	19.52
	FFT/ Dir.	FFT/ Rec.	Dir./ Rec.
128×128	0.624	5.42	8.689
256×256	0.641	5.61	8.763
512×512	0.508	5.56	10.95

Figure 4: *(a) Averages over 100 runs, three direction box spline, 16 processors, uniform multiplicity 1. Efficiency is the ratio between the speedup (relative to a single processor) and the number of processors used. (b) Averages over 100 runs,* 512×512 *problem size, recursive coefficient evaluation, three direction box spline, 16 processors, uniform multiplicities. (c) Averages over 100 runs of the evaluation of a three direction box spline of uniform multiplicity 1.*

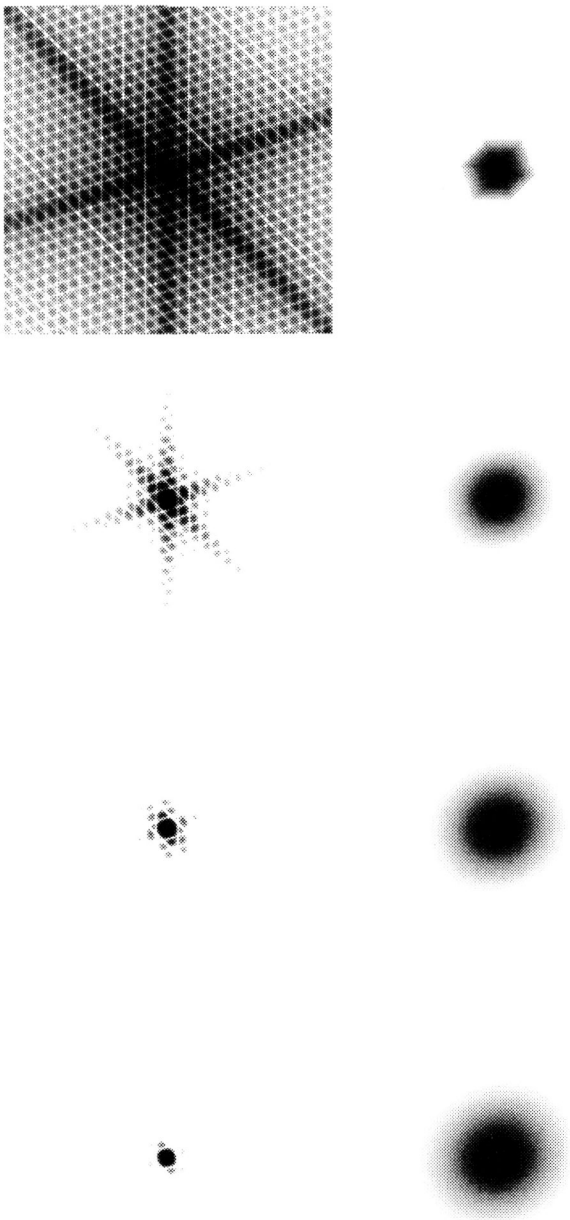


Figure 5: *FFT centered box spline evaluation examples. On the left are* 256×256 *unwindowed Fourier transforms, and on the right the resulting spline. Each spline has three unique direction vectors, with multiplicity increasing from top to bottom.*