3. Image-Based Reconstruction Error Comparison

In this chapter, images produced with three reconstruction filters are compared to reference images rendered without reconstruction. An image error metric is defined to quantify the comparison results. The data used to produce the test and reference images are designed to highlight differences in reconstruction accuracy by incorporating features of varying *scale*. The concept of features existing in spatial and scale dimensions is borrowed from the vision literature and summarized in the next section (§3.1). The image error measurements in this chapter are reinforced in chapter four with analytical reconstruction error bounds expressed as a function of feature scale. Numerical error analysis expresses error bounds as a polynomial function of the original signal, its derivatives, and the sample interval. Expressing the error bound in terms of feature scale and the sample interval is shown to be an alternative representation.

3.1. Feature Scale

While the Fourier domain conveys information about signals as a whole, it is not very intuitive or descriptive for relating the properties of a local region of a signal. A domain more descriptive of local signal properties is that of *scale space*. Signal regions are classified as *features* of a particular *scale* based on the response of an operator with some aperture, positioned over that region. A useful operator is the Laplacian of the Gaussian, $A = \nabla^2(G(\sigma, d))$, where

$$G(\sigma, d) = (2\pi\sigma^2)^{-1/2} \exp(-d^2/ 2\sigma^2) \qquad (3.1)$$

with the operator aperture defined as σ. Figure 3.1 illustrates a one-dimensional signal f(x) and the responses of operators A of three different apertures. At point p the response is highest for the operator with aperture σ_1. In the Fourier domain we say a signal has an energy spectrum over a range of frequencies. The operator response at different apertures is analogously the energy spectrum over a range of scales - but in this case the spectrum describes only a local region of the signal, not the whole thing. A single feature may cause an isolated peak at a small scale level and also be part of a larger structure at a higher scale level. A multiscale hierarchy is used in some advanced interactive segmentation techniques [Pizer88] that have been applied to volume rendering [Yoo+91].

If scale spectra were known for the whole data set, it might be used to adaptively sample the data to maintain a tolerable error. A similar idea for adaptive sampling is applied to volumes by David Laur and Pat Hanrahan [Laur91]. Their approach measures error by comparing the true values of a region of points with their approximate representation at different levels of hierarchy. The highest level (i.e.: least-detailed) representation allowed by a variable error bound is used to render the region. The utilization of feature scale information to determine an error metric and corresponding sample rate has not

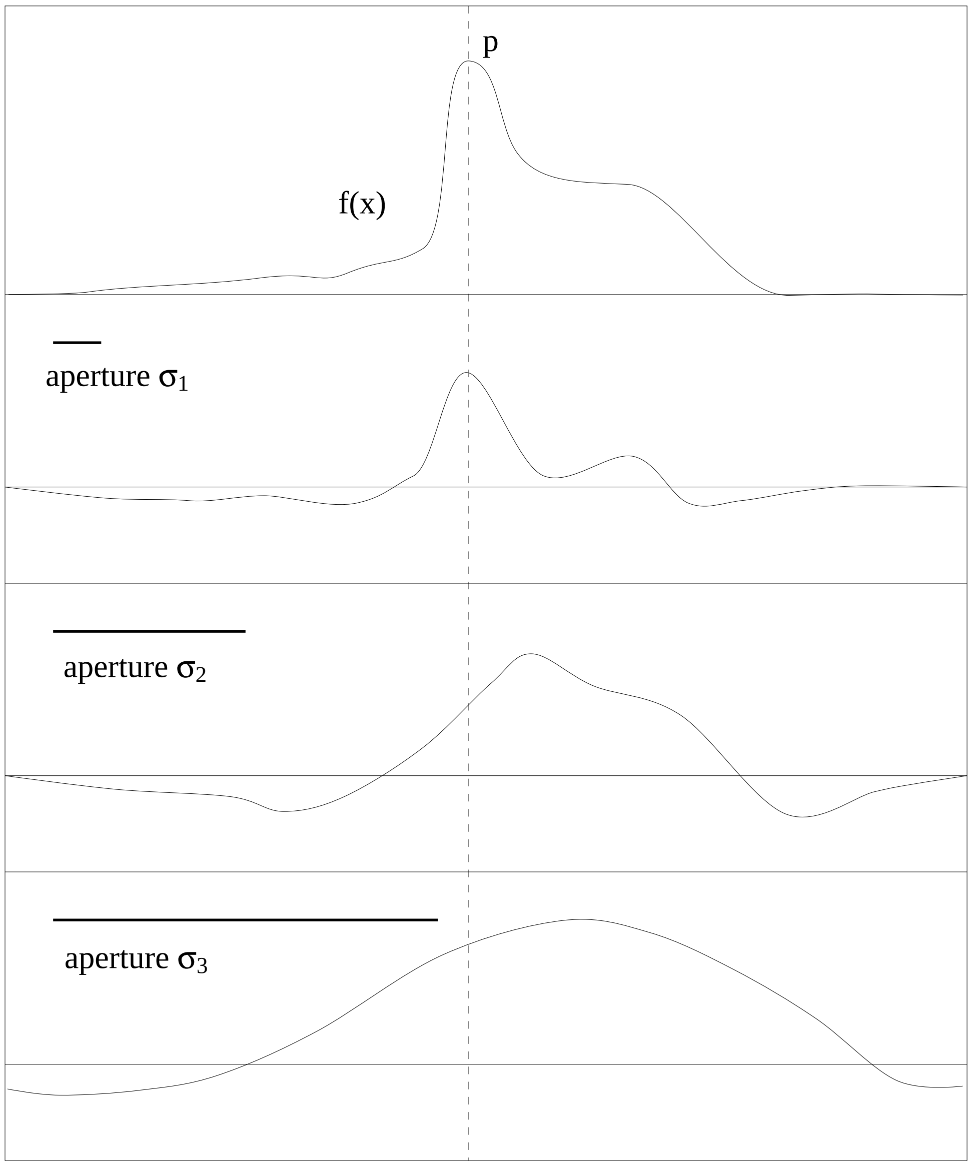


Fig. 3.1 - Multiscale operator response function

been tried. This problem is outside the scope of this work, but is an area for future exploration.

3.2. Test Data

Volume data, for the purposes of this research, will be a scalar field in $\Re^3$. The field function may be encoded or represented in a variety of ways. For example, it could be described by an explicit function $f(x, y, z)$ or an array of samples $F_{i, j, k}$. The array of samples representation is most commonly encountered when the data is acquired from the real world or a simulation of the real world. Medical images or results from a finite element analysis are examples of volumes represented by arrays of samples. The explicit function representation is useful in modeling. One approach to modeling objects in a volume is to use multiple field generator primitives [Blinn82] [Muraki91]. The scalar field at any point in the volume is the sum of the effects of all the generators. To produce an image from either data format, we must be able to sample the data on the image space lattice. The accuracy of this sampling process is greatly influenced by the representation. Only the explicit function may be sampled anywhere with arbitrary precision. Practical considerations limit the sampling accuracy when using the array-of-samples representation. Perfect reconstruction is theoretically possible with an infinite-extent filter kernel. In practice, limited-extent filters are used to make computation tractable.

3.2.1. Explicit Volume Data

In this study an explicit representation of volume data is created as a standard against which to compare reconstruction methods. It is based on Gaussian point and line generators. A point generates a radially symmetric field G(σ, d) parameterized by σ, the feature scale, which determines the distribution of the field about its origin. A small σ corresponds to a tightly packed distribution while a large σ yields a broadly distributed field. For any σ value the total energy of the field is normalized so that the path integral through the center point always yields unity. One test volume (called "*points*") has an array of forty-eight point generators organized as six rows with eight points per row. Within any row σ is constant while each row has a different σ. The point coordinates ⟨i, j, k⟩ are given in figure 3.2. They are defined to occupy a volume of dimension 64×64×64. Figure 3.4a is an image of the *points* data when viewed down the k-axis.

A second test data set (called "*mixed*") consists of line segments and points. The line segments generate a field with Gaussian decay as a function of the minimum distance to the segment. Figure 3.3 lists the lines and points in the *mixed* data while figure 3.4b is an image of it viewed down the k-axis. Only one coordinate is listed for line segments since they all share a common endpoint at ⟨32, 32, 32⟩. The total volume occupied by the *mixed* data is 64×64×64 units.

σ = 0.5	**σ = 0.7**	**σ = 0.9**	**σ = 1.1**	**σ = 1.3**	**σ = 1.5**
9.0, 7.0, 9.0	9.0, 16.0, 14.0	9.0, 25.0, 19.0	9.0, 34.0, 24.0	9.0, 43.0, 29.0	9.0, 52.0, 34.0
12.0, 7.34, 10.4	12.0, 16.3, 14.7	12.0, 25.3, 19.7	12.0, 34.3, 24.7	12.0, 43.3, 29.7	12.0, 52.3, 34.7
15.6 7.68, 10.4	15.6, 16.6, 15.4	15.6, 25.6, 20.4	15.6, 34.6, 25.4	15.6, 43.6, 30.4	15.6, 52.6, 35.4
20.0, 8.0, 11.2	20.0, 17.0, 16.2	20.0, 26.0, 21.2	20.0, 35.0, 26.2	20.0, 44.0, 31.2	20.0, 53.0, 36.2
25.1 8.36 11.9	25.1, 17.3, 16.9	25.1, 26.3, 21.9	25.1, 35.3, 26.9	25.1, 44.3, 31.9	25.1, 53.3, 36.9
30.8, 8.7, 12.7	30.8, 17.7, 17.7	30.8, 26.7, 22.7	30.8, 35.7, 27.7	30.8, 44.7, 32.7	30.8, 53.7, 47.7
37.2, 9.04, 13.4	37.2, 18.0, 18.4	37.2, 27.0, 23.4	37.2, 36.0, 28.4	37.2, 45.0, 33.4	37.2, 54.0, 38.4
44.4, 9.38, 14.2	44.4, 18.3, 19.2	44.4, 27.3, 24.2	44.4, 36.3, 29.2	44.4, 45.3, 34.2	44.4, 54.3, 39.2

Fig. 3.2 - *Points* test data generator coordinates

Point coords	σ	Point coords	σ
20.7, 42.3, 32.5	2.44	24.4, 29.4, 27.4	1.86
37.5, 35.4, 40.9	1.28	41.7, 24.4, 23.2	1.77
38.7, 43.0, 30.4	0.64	26.4, 38.6, 41.0	0.76
22.0, 28.8, 29.4	3.07	37.5, 35.5, 29.9	2.16
24.6, 22.9, 35.6	1.48	26.0, 22.8, 42.3	0.56
25.6, 30.4, 42.2	2.51	24.0, 22.8, 42.3	0.56
31.3, 26.0, 28.9	1.03	38.8, 27.1, 26.8	2.93
40.9, 35.3, 23.3	3.32	32.7, 30.6, 34.2	0.59
Line endpoint	**σ**	**Line endpoint**	**σ**
12.9, 25.2, 53.0	0.89	12.6, 45.0, 37.3	0.85
18.6, 13.7, 31.4	0.77	33.2, 40.8, 27.0	0.75
26.0, 34.0, 52.7	0.92	17.4, 44.0, 35.4	0.79
53.0, 15.0, 47.7	0.91	33.3, 26.5, 13.9	0.61
45.9, 36.1, 31.1	0.58	51.1, 23.0, 51.7	0.55
33.6, 28.4, 23.4	0.64	48.9, 47.8, 36.8	0.99
27.8, 21.9, 20.1	0.97	16.4, 18.4, 16.5	0.76

Fig. 3.3 - *Mixed* test data generator coordinates

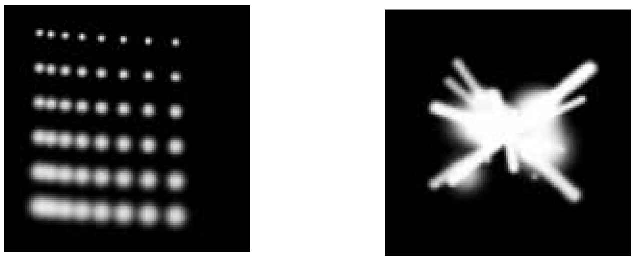

a - *points* b - *mixed*

Fig. 3.4 - Test data sets viewed down the k-axis

3.2.2. Sampled Volume Data and Prealiasing

The test volumes defined above may also be represented by arrays of samples. Point sampling the explicit functions on a 64×64×64 regular grid in $\Re^3$ produces the sampled data sets that we will consider. Sampling theory dictates that such samples capture spectral features of the original function that are below one-half the sampling frequency (the Nyquist limit); higher frequency components will cause prealiasing. This section shows what proportion of a feature's spectrum causes prealiasing. The proportion of the spectral content above the Nyquist limit is called the *prealiasing error*. A comparison can be made between the reconstruction error and prealiasing error. In chapter four the reconstruction error is shown to be larger than the prealiasing error; this demonstrates that the reconstruction filter is generating error in addition to that present in the samples themselves.

The Fourier spectrum of a Gaussian is known to be

$$\mathbf{F}(G(\sigma, d)) = \exp(-2(\pi\sigma\nu)^2) \qquad (3.2)$$

Rewriting this in the form of another Gaussian and normalizing it for unit total energy we obtain

$$\mathbf{F}(G(\sigma, d)) = ((2\pi)^{1/2}\,\sigma)^{-1} \exp(-(2\pi\sigma^2\nu)^2\,(2\sigma^2)^{-1}) \qquad (3.3)$$
$$= G(\sigma, 2\pi\sigma^2\nu) \qquad (3.4)$$

This shows that the Fourier spectrum of a Gaussian can be obtained by simply scaling the horizontal axis. We determine the prealiasing error P(σ) by using the erf(z) function to evaluate the integral of a portion of a Gaussian.

$$P(\sigma) = 1 - \mathrm{erf}(z) = 1 - 2\,(\pi^{-1/2}) \int_0^z \exp(t^2)\,dt \qquad (3.5)$$

where

$$t = z / (2^{1/2}\,\sigma) \qquad (3.6)$$
$$z = 2\pi\sigma^2\nu \qquad (3.7)$$

The sample frequency is 1.0 if we use the sample grid for spatial measurement. The Nyquist frequency is then ν = 0.5. The energy above the Nyquist frequency is

$$P(\sigma) = 1 - \mathrm{erf}(\pi\sigma^2) \qquad (3.8)$$

A graph of this prealiasing error is shown in Fig. 3.5. For all but the smallest feature, the test-data features have σ ≥ 0.7, and their prealiasing error is very low. It is important to note that prealiasing error does not derive from sample error, each sample value is accurate. Prealiasing error manifests itself during reconstruction. Even by applying an arbitrarily-good reconstruction filter, the prealiasing error limits the reconstruction

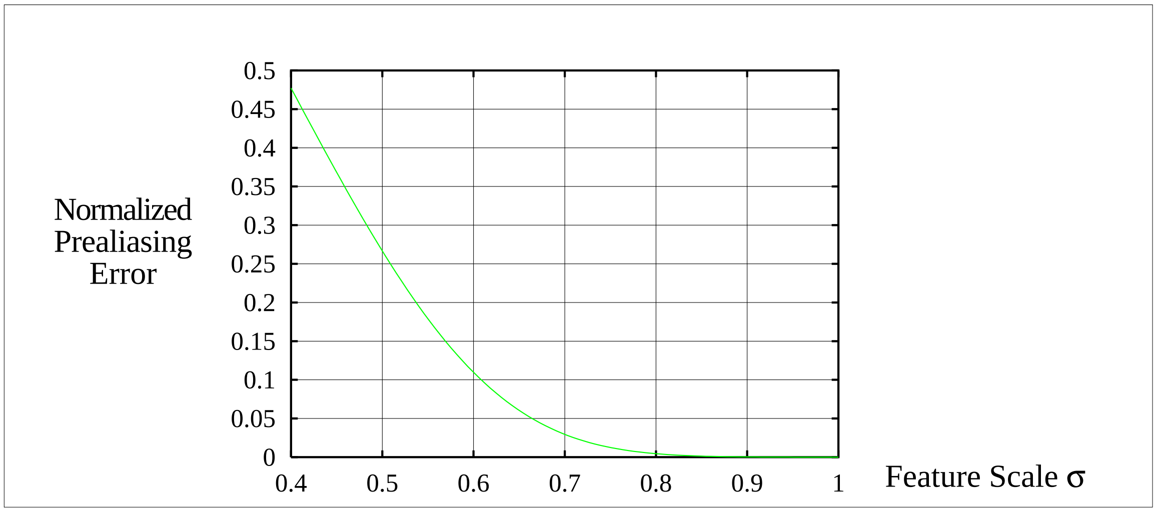


Fig. 3.5 - Prealiasing error P(σ)

accuracy.

3.3. Image Comparison Metric

One approach to evaluating reconstruction methods is to compare rendered images against a standard. We have an explicit form of volume data that may be precisely evaluated anywhere in the volume and the sampled form that requires reconstruction. Images rendered using the reconstructed data are compared against images rendered using the explicit form. If all steps but the resampling are the same, any differences in the images must be due to errors in reconstruction.

The question arises as to how to measure error between two volume rendered images. While this is ultimately an observer-perception issue and perceptual error is difficult to quantify, there is a rationale for employing a simple metric. The observer normally does not view the reconstructed volume directly - it is usually classified and shaded by an arbitrarily-nonlinear process. Because of this nonlinearity, it is important that reconstruction produce low numerical error. The assumption is that greater numerical error is more likely to produce more perceptually-significant artifacts. Minimizing the numerical error should minimize perceived errors in the final image. Since the shading and classification functions are omitted during test image rendering, the image differences are a measure of the reconstruction errors only. A quantitative metric of the difference between images is obtained by treating each image as a single vector and normalizing its magnitude; the inner product of two normalized image-vectors produces a scaler that is a measure of how closely the images match. Each pixel's attributes are an element of the image vector. In addition to pixel intensity, the gradient vector computed with a 3×3 Sobel kernel is also used as a pixel attribute. Since the human vision system is sensitive to intensity gradients, it it logical to include this attribute in the comparison metric. Image are compared by computing the inner product of the intensity, gradient, and both attributes of each pixel. Let A and B each be normalized image-vectors with p pixels. The inner product IP is a scalar defined as

$$IP = A_0 \cdot B_0 + A_1 \cdot B_1 + ... + A_{p-1} \cdot B_{p-1} \qquad (3.9)$$

When only the pixel-intensity attributes are considered, elements A_j and B_j are scalar intensity values. When the pixel-gradient attributes are used, A_j and B_j are tuples whose elements are the x and y-components of the gradient vector: $A_j = \langle A_{j\partial x}, A_{j\partial y} \rangle$ and $B_j = \langle B_{j\partial x}, B_{j\partial y} \rangle$. When using the pixel-gradient attributes the IP product-terms are computed as

$$A_j B_j = A_{j\partial x} \cdot B_{j\partial x} + A_{j\partial y} \cdot B_{j\partial y}. \qquad (3.10)$$

Similarly, A_j and B_j may be three-tuples containing the pixel intensity and gradient components: $A_j = \langle A_{ji}, A_{j\partial x}, A_{j\partial y}\rangle$ and $B_j = \langle B_{ji}, B_{j\partial x}, B_{j\partial y}\rangle$. In this case the IP product-terms are computed as

$$A_jB_j = A_{ji}\cdot B_{ji} + A_{j\partial x}\cdot B_{j\partial x} + A_{j\partial y}\cdot B_{j\partial y}. \tag{3.11}$$

Admittedly, this inner product approach by itself is not a strong evaluator of the reconstruction methods. Any single measure of how much difference exists between the pixel attributes ignores the spatial distribution of that error. Viewing the images can compensate for this by enabling our visual systems to assess the distribution, but observations are subjective, and a more rigorous measure of quality is desirable. Therefore, the feature scale approach which gives an error bound is employed in chapter four. Those results are intended to complement the image based approach. An error bound alone does not fully evaluate a reconstruction method either, the distribution of error is also important. The image comparisons provide a measure of the average error since many reconstructed values contribute to each pixel of the images.

3.4. Rendering Methods

Before presenting the comparison results, some description of the volume renderers used for these tests is appropriate. Several functions are common to any rendering approach. Primarily these have to do with creating view matrices. We will adopt the convention that these matrices are 3×4 arrays and points are four-element column vectors. All transformations will be affine for simplicity's sake. Two coordinate systems are of interest, object space and image space. Object space has right-hand coordinates ⟨i, j, k⟩ where point samples lie on integer lattice points. Screen space has right-hand coordinates ⟨x, y, z⟩ where pixels fall on integer coordinate points in the x,y plane.

For any axis of rotation *n*, defined by α, β, and γ, [Rogers+76] and angle of rotation θ about *n*, a rotation matrix [Rot] can be constructed (Fig. 3.6). For volume data to be rotated about its center, a translation matrix [Trans] is constructed that will shift the

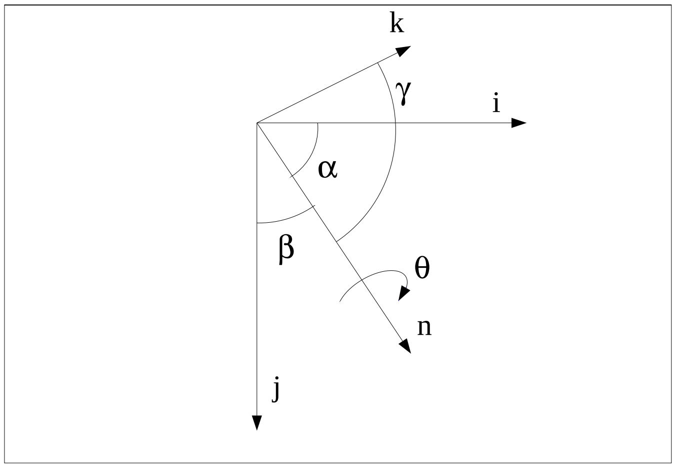


Fig. 3.6 - Specifying a view (adapted from Rogers and Adams pp. 56)

origin to the center of the volume. A matrix [Screen] is also created that transforms a rotated object space lattice to image lattice coordinates; this is at most a scaling and translation. For each of these matrices, an inverse is also computed. Concatenated as

$$[\mathrm{Mat}] = [\mathrm{Screen}]\,[\mathrm{Trans}]^{-1}\,[\mathrm{Rot}]\,[\mathrm{Trans}] \tag{3.12}$$

[Mat] will transform points from object space into image coordinates. When concatenated as

$$[\mathrm{Mat}]^{-1} = [\mathrm{Trans}]^{-1}\,[\mathrm{Rot}]^{-1}\,[\mathrm{Trans}]\,[\mathrm{Screen}]^{-1} \tag{3.13}$$

$[\mathrm{Mat}]^{-1}$ will transform points from image coordinates into object space.

Regardless of reconstruction method, all renderers must composite their resampled data values in view order. The common practice of using eight-bit integer representations of segment values leads to large relative errors when compositing [Wilhelms+91]. In the test implementations, all representations and computations use single-precision floating point format which is more than sufficient to maintain accuracy during compositing.

3.5. Experimental Results

The explicit data representation is used to render images that require no reconstruction. These images, shown in figure 3.7, are 128×128 pixels in size and considered to be references for making comparisons in the next sections. All test images are computed for a view specified by $\alpha = 6.75°$, $\beta = 16.5°$, $\gamma = 3.375°$, and $\theta = 18.0°$. A sequence of images with varying views of the *points* data set are also produced for visually illustrating view dependencies.

3.5.1. Ray Casting with Pyramid Filter

Ray casting is an *image order* rendering method which resamples reconstructed data at points along rays cast into the volume. Sample values are computed by trilinear interpolation of the eight nearest object lattice values. It is possible to decrease the

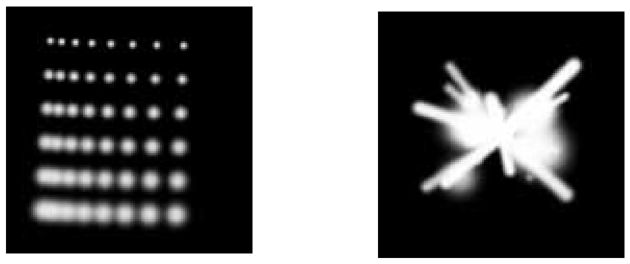


Fig. 3.7 - Reference images

amount of computation by subsampling, thus creating a coarser image lattice. Using a step size of two and casting rays through alternate pixels doubles the spacing between image lattice points and reduces the number of samples computed by a factor of eight. Images of the test data sets rendered with sample step sizes of one, two, four, and six along the screen space z-axis, and rays cast every 1×1, 2×2, and 4×4 pixels, are shown in figure 3.8 and 3.9. Pixels between rays are bilinearly interpolated.

By taking the inner product of the test images with the reference images, we produce the graphs shown in figure 3.10. These graphs show that the images in figures 3.8 and 3.9 that were obtained by sampling more densely than the object lattice show only slight variations in their correlations to the reference images (i.e.: the cases with 1×1 or 2×2 pixels per ray and a step size of 1 or 2). The differences are not visually apparent in figures 3.8 and 3.9. In this case, resampling at densities above the object lattice density is not worth the effort. This is also the case for the other two rendering tests in the remainder of this chapter. The graphs in figure 3.10 also indicate that trilinear

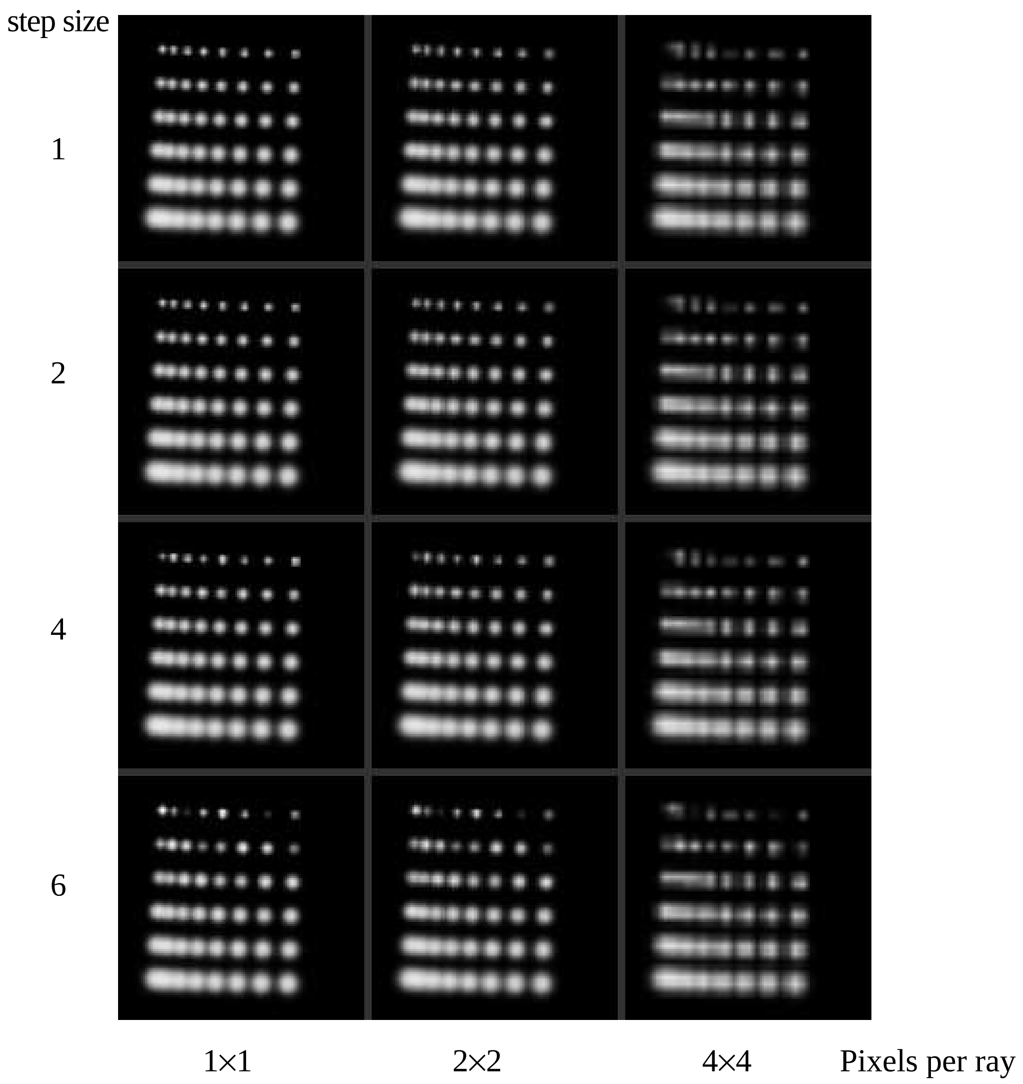


Fig. 3.8 - Ray casting with pyramid filter reconstruction

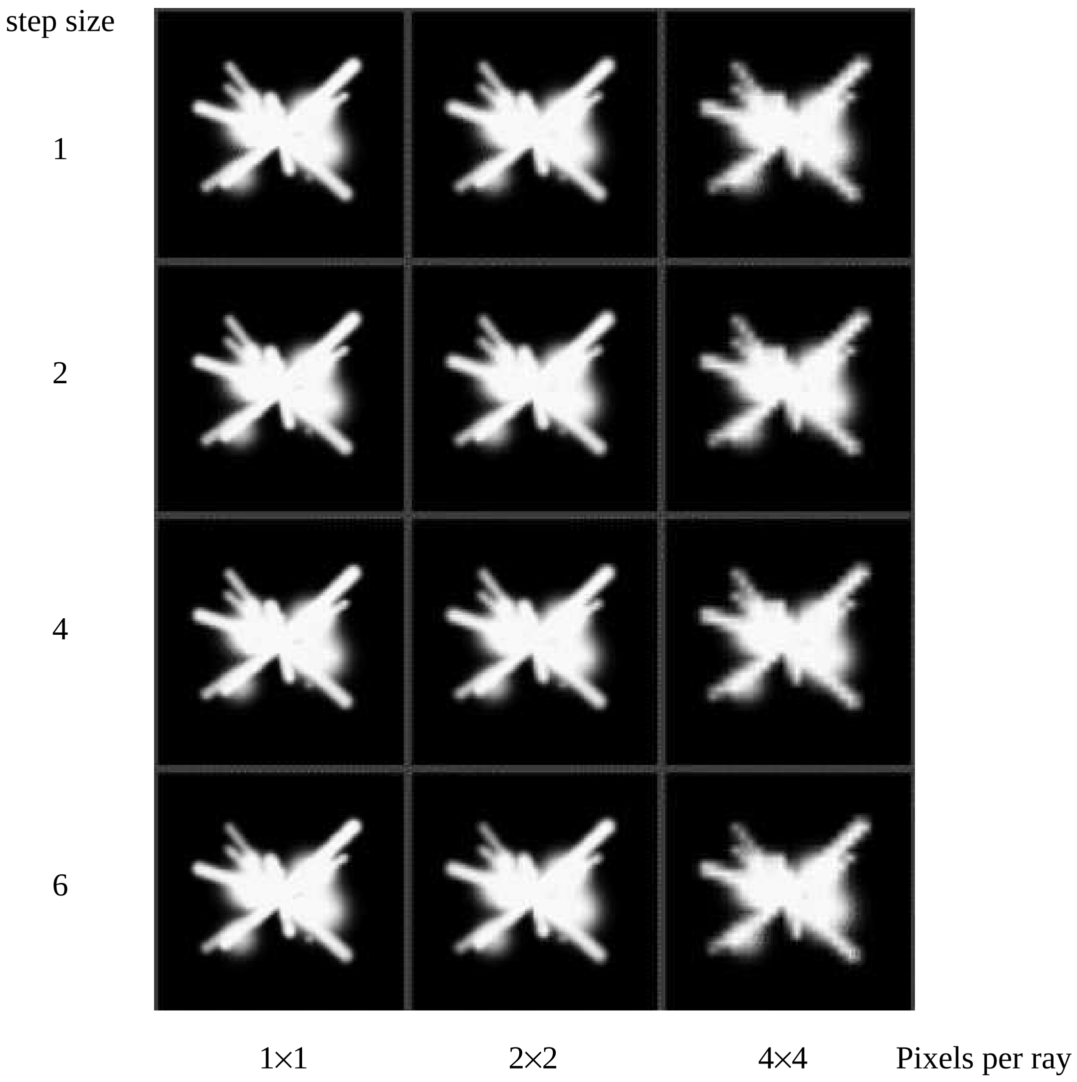
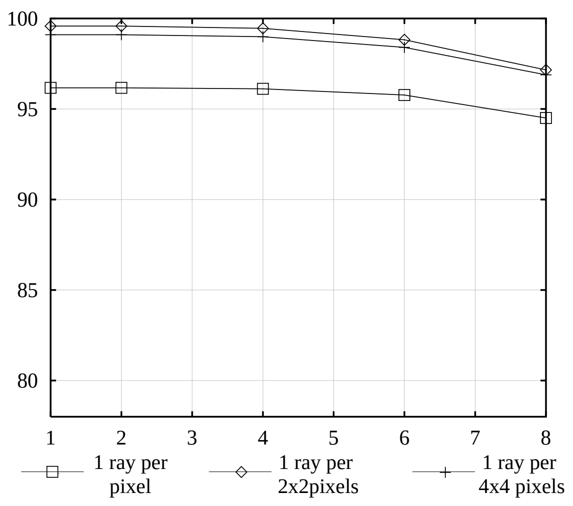


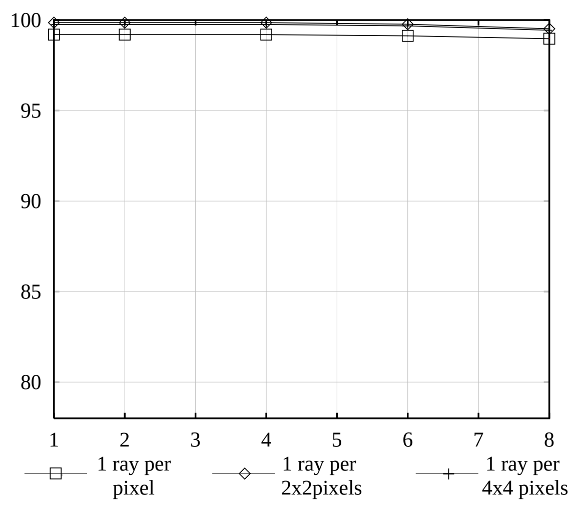
Fig. 3.9 - Ray casting with pyramid filter reconstruction

interpolation produces low overall error when the image lattice is at least as dense as the object lattice. The low inner product measured for sparse image lattice densities (4×4 pixels per ray, or step size > 2) is not due to reconstruction error, but rather due to subsampling of the reconstructed signal which is a source of postaliasing. It is interesting to note the anisotropy in subsampling along different axes. The image error measure is less sensitive to subsampling the z-dimension than the x and y-dimensions. This is due to the integration along the z-axis which effectively averages the errors and thereby minimizes their effect. We can exploit this characteristic by using a low image lattice density in the z-dimension without causing artifacts in the image.

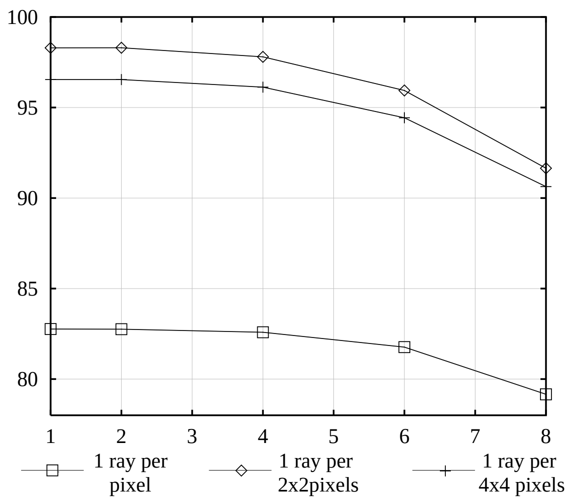
Figure 3.11 shows a reference animation sequence of eight images rendered using the explicit data form and no reconstruction. Figure 3.12 shows the same sequence rendered by ray casting with trilinear interpolation using a high density (128^3) image lattice. The absence of visually-apparent view-dependant artifacts offers some assurance that the test results are generalizable to view transformations other than the one used in these tests.



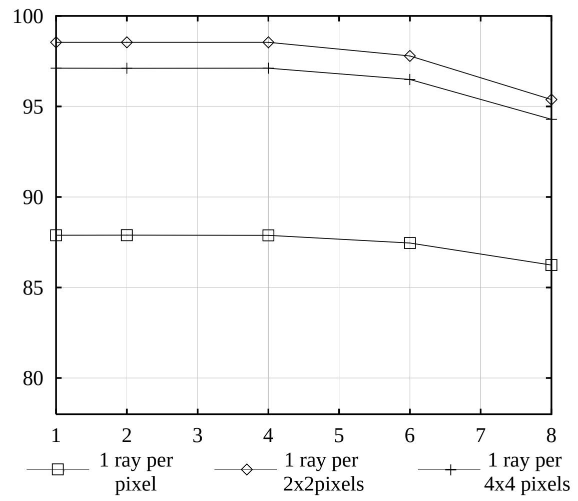
(a) - Intensity IP *(Points)*



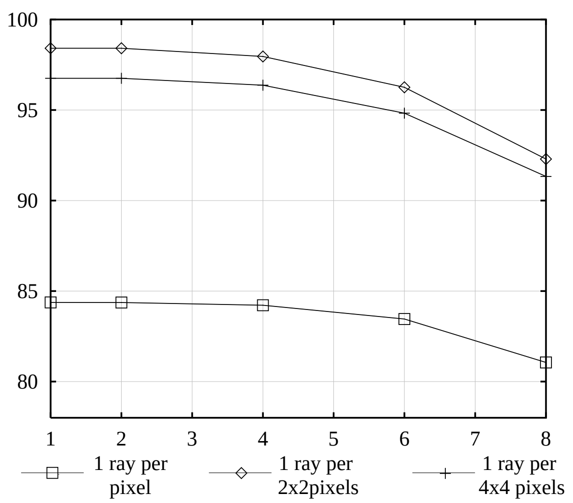
(d) - Intensity IP *(Mixed)*



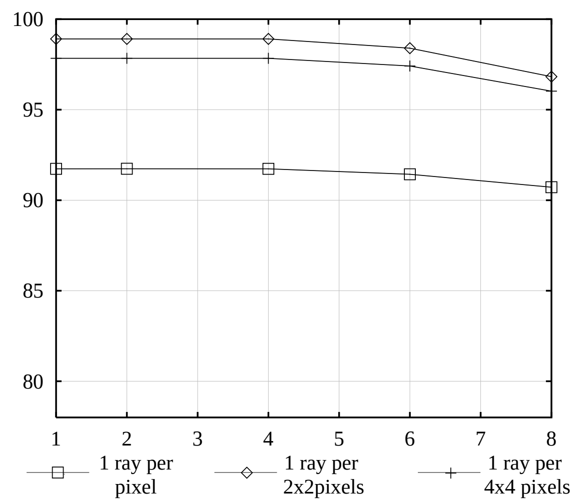
(b) - Gradient IP *(Points)*



(e) - Gradient IP *(Mixed)*



(c) - Intensity, Gradient IP *(Points)*



(f) - Intensity, Gradient IP *(Mixed)*

Fig. 3.10 - Inner Product of references with test images made with pyramid filter reconstruction
Horizontal axis is ray z-step size (data taken for step size = 1, 2, 4, 6, and 8)

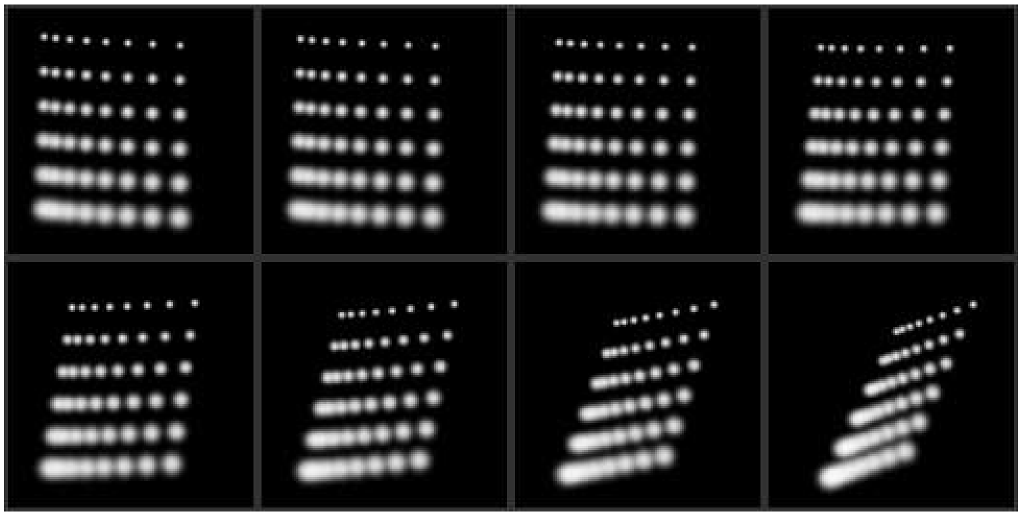


Fig. 3.11 - Reference animation of *points*

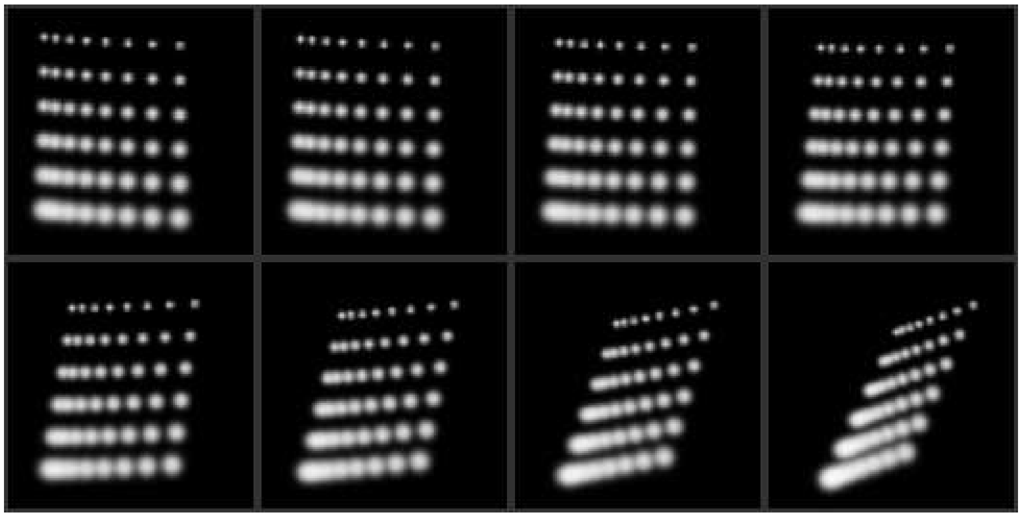


Fig. 3.12 - *Points* animation using pyramid filter reconstruction

3.5.2. Splatting with Projected Gaussian

Splatting is an object-order rendering method. In principle, it is a 3D filtering method, but for efficiency it has only been used in a 2D form as described in section 2.1.2. Points within a slice of the data are filtered with a Gaussian to reconstruct a 2D image of the slice. The slice is projected under the view matrix and resampled at each pixel. Successive slices are composited to produce a final image. Zero-valued data points need not be filtered since they do not contribute to the image. The amount of computation may be decreased by using a coarser image lattice. Since splatting does not resample along the image z-axis, we may only decrease the lattice density in the x and y-dimensions. Figure 3.13 shows images of the *points* data for various filter kernel sizes with lattice samples every 1×1, 2×2, and 4×4 pixels. Figure 3.14 shows similar images for the *mixed* data. Pixels between lattice samples are bilinearly interpolated. As in the pyramid filter test, resampling more densely than the object lattice has a relatively small effect on image quality compared to the rather large increase in the number of resampling points and the computation required to compute their values.

The inner products of the test and reference images are shown in figure 3.15. The graphs suggest that the lowest error is produced with a filter σ between 0.5 and 0.6. The animation sequence in figure 3.16 is produced with a filter σ = 0.5 and 128×128×64 image lattice samples.

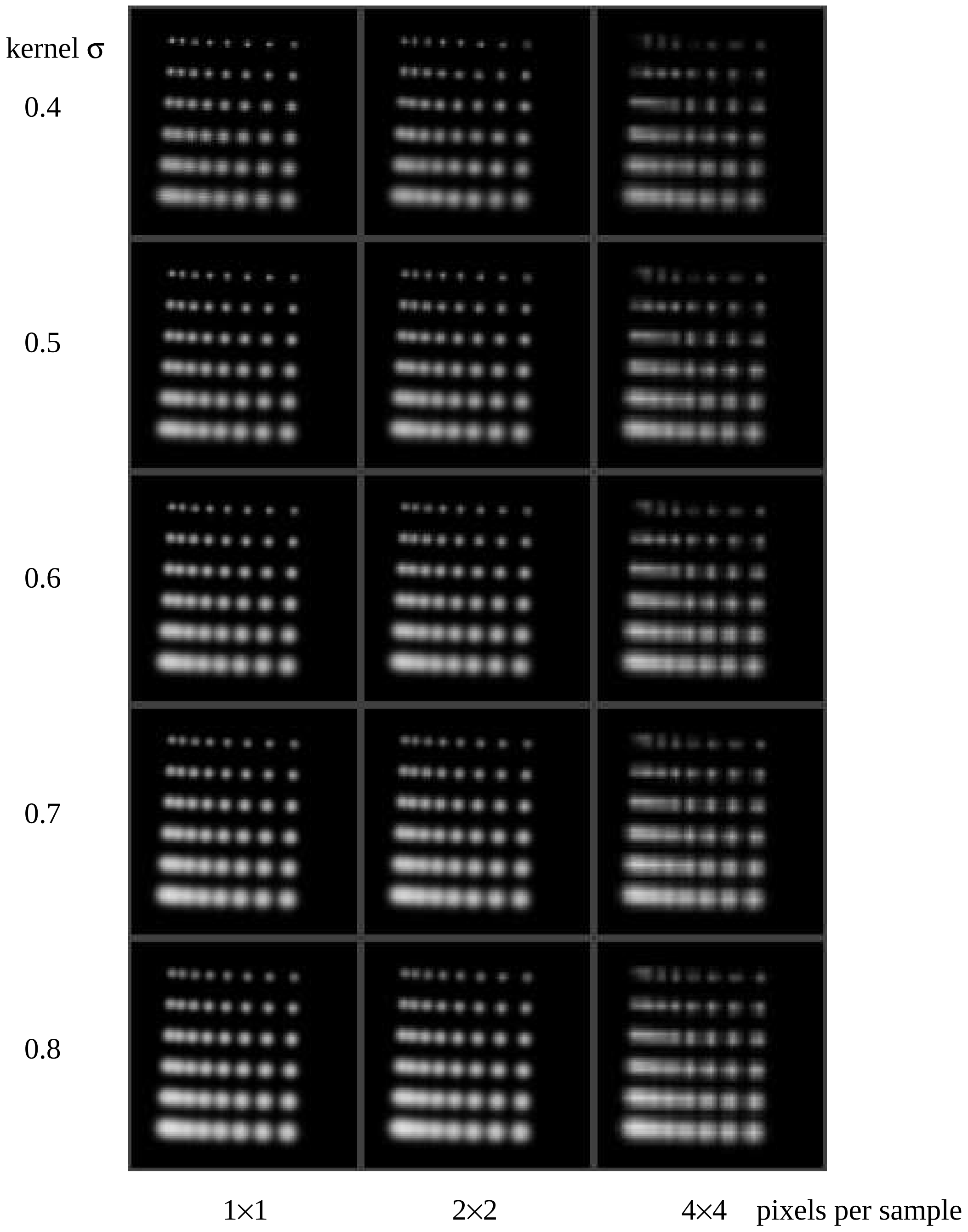


Fig. 3.13 - Splatting with Gaussian filter reconstruction

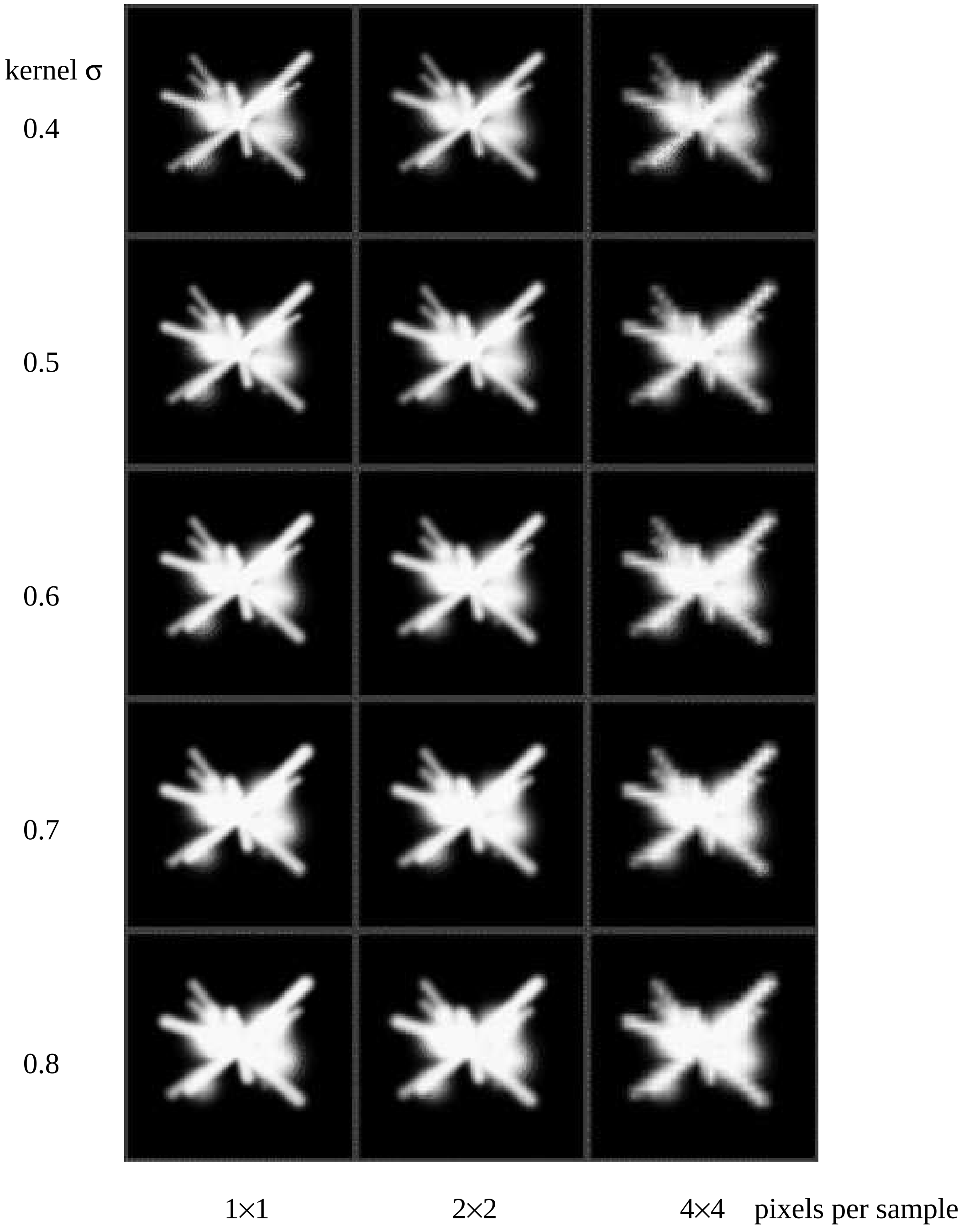
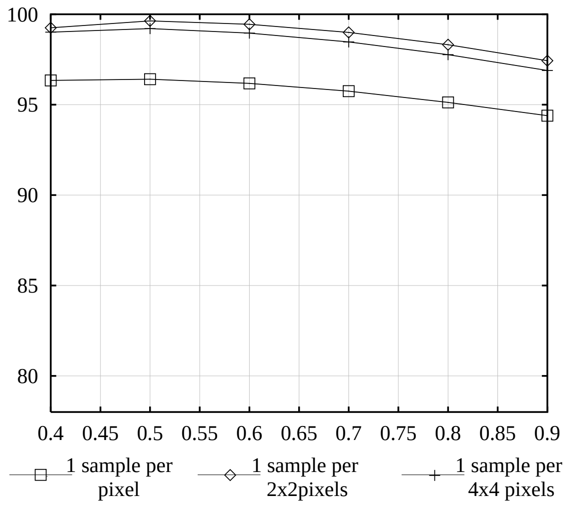
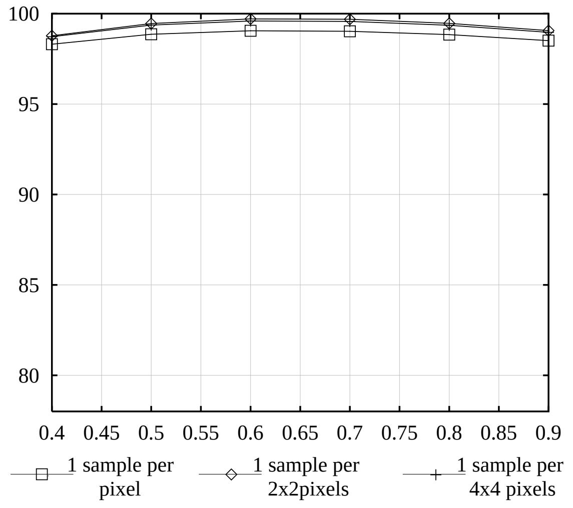


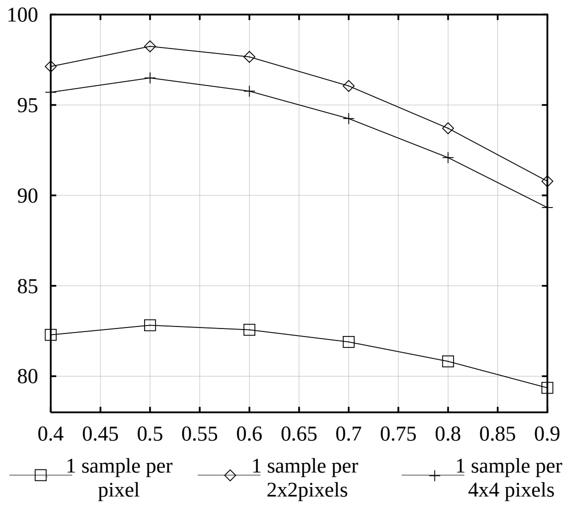
Fig. 3.14 - Splatting with Gaussian filter reconstruction



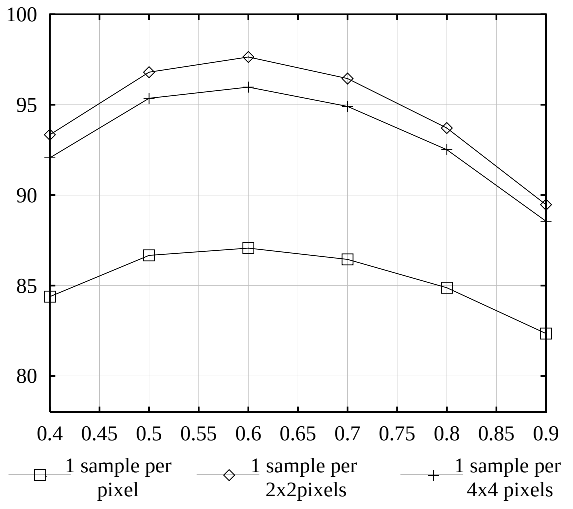
(a) - Intensity IP *(Points)*



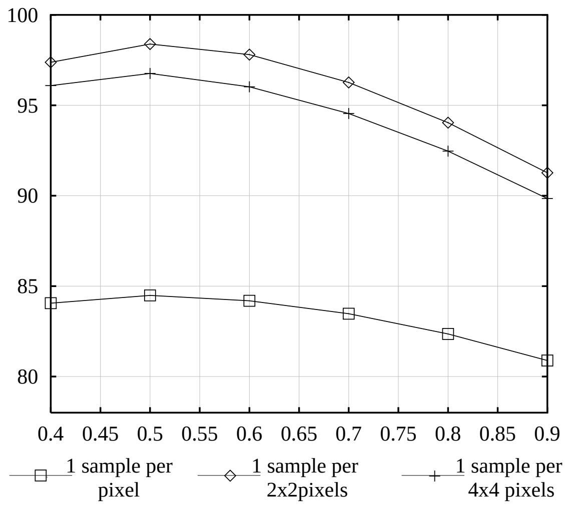
(d) - Intensity IP *(Mixed)*



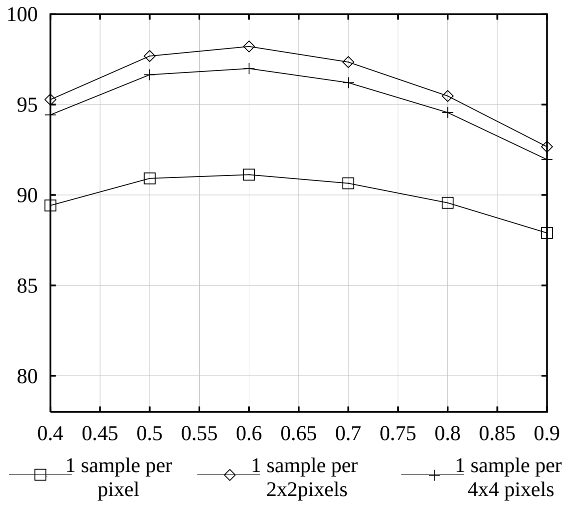
(b) - Gradient IP *(Points)*



(e) - Gradient IP *(Mixed)*



(c) - Intensity, Gradient IP *(Points)*



(f) - Intensity, Gradient IP *(Mixed)*

Fig. 3.15 - Inner Product of references with test images made with a Gaussian filter
Horizontal axis is Gaussian kernel σ (data taken at $\sigma = 0.4, 0.5, 0.6, 0.7, 0.8$, and 0.9)

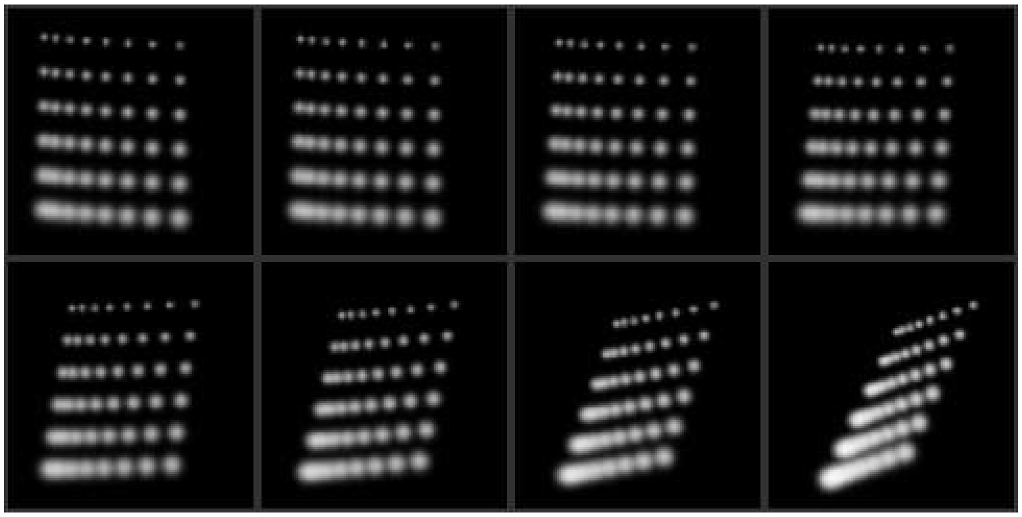


Fig. 3.16 - *Points* animation rendered by splatting with Gaussian filter $\sigma = 0.5$

3.5.3. Volume Shearing with Separable Cubic Filter

The images in this section are produced by volume shearing in two passes as opposed to the usual three pass method. Reconstruction and resampling is done along the image x and y-axis only; omitting the third pass saves over one-third of the computation time so it is worth doing. As in splatting, no resampling along the z-axis is performed. In fact, this method is similar to splatting in that two-dimensional images are reconstructed on the object lattice and projected onto the screen, but here a separable cubic filter is used instead of a Gaussian. The integration along the z-axis compensates for the view-dependent distance between samples by using the "K" coefficient that is computed from the view matrix and normally used during the third (z-axis) pass [Hanrahan90]. Chapter four details the computation savings for this method.

The cubic filters used are of the two parameter family described in [Michell$^+$88] and section 4.3. The parameters are called B and C, and range over [0, 1]. Test images are computed for the nine possible combinations with B and C taking on the values {0, 0.5, 1.0}. Three sets of nine images are tested, corresponding to image lattice densities of 1×1, 2×2, and 4×4 pixels per sample point. Only the density of the x and y-axes of the image lattice are variable. The number of samples along the image lattice z-axis is fixed by the dimensions of the object lattice, which in this case is sixty-four points. As with splatting and ray casting, increasing the resampling density above the object-lattice density produces relatively minor increases in the IP results. Figure 3.17 shows images of the *points* data, figure 3.18 shows the *mixed* data. The inner products for the parameters producing the best and worst images are shown in figure 3.19. The remaining seven parameter combinations were omitted from the graphs for clarity. These graphs illustrate that the Catmull-Rom cubic spline with B = 0 and C = 0.5 has the lowest image error in all cases. This coincides with previous analysis of cubic filters [Keys81]

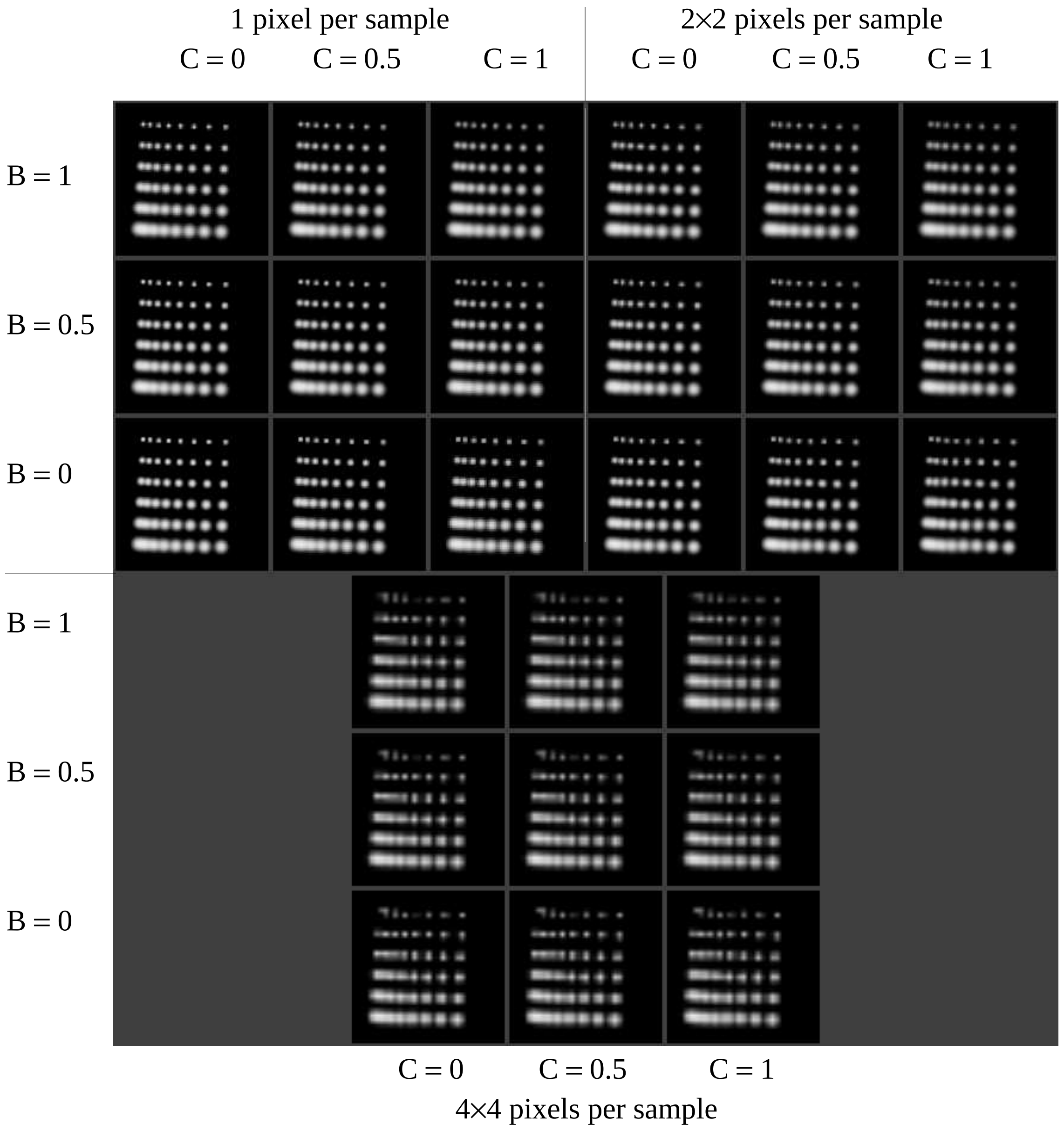


Fig. 3.17 - Volume shearing with separable cubic filter reconstruction

[Park+83] [Michell+88]. Figure 3.20 shows the eight frame animation of *points* data using the Catmull-Rom filter.

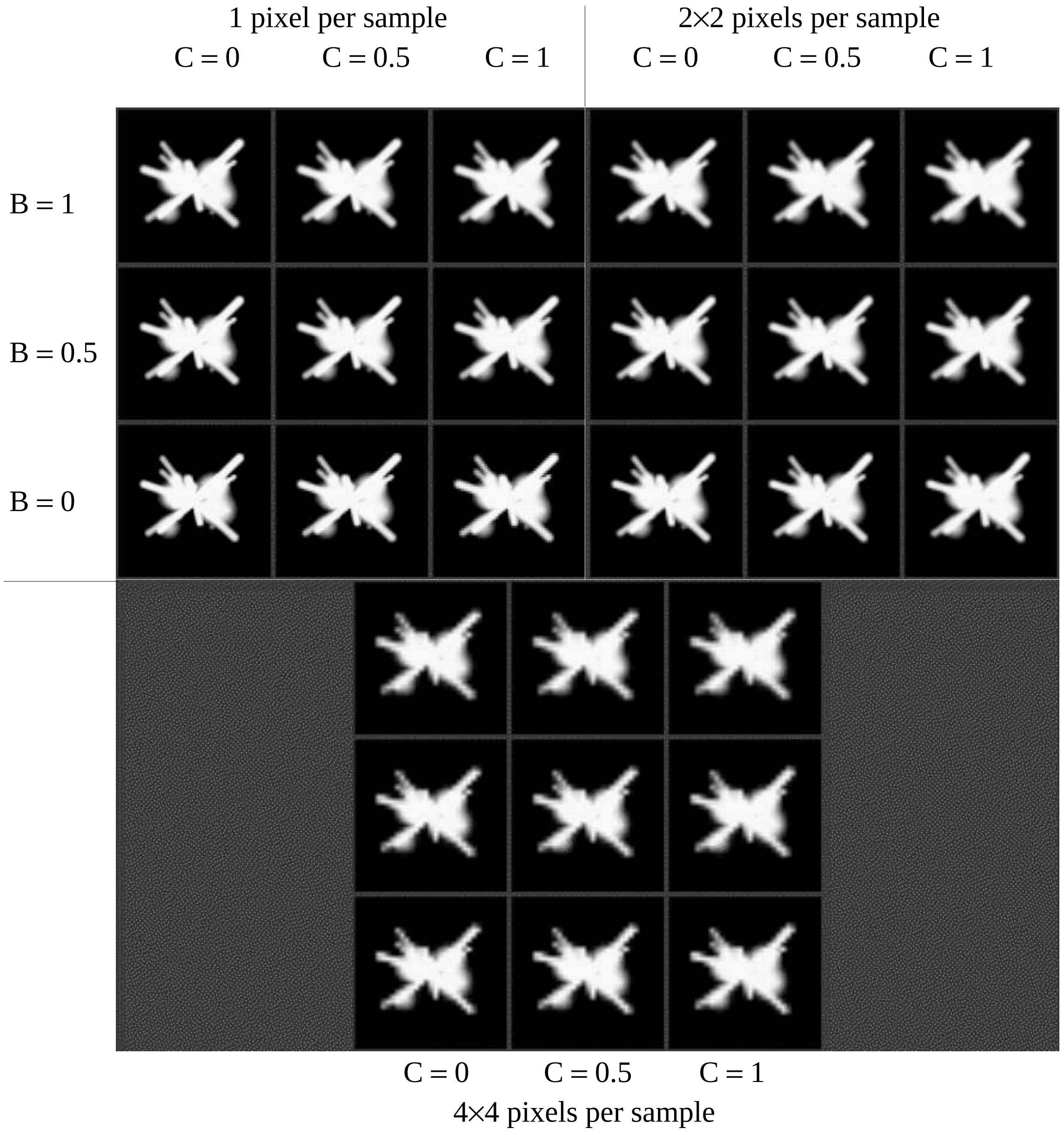
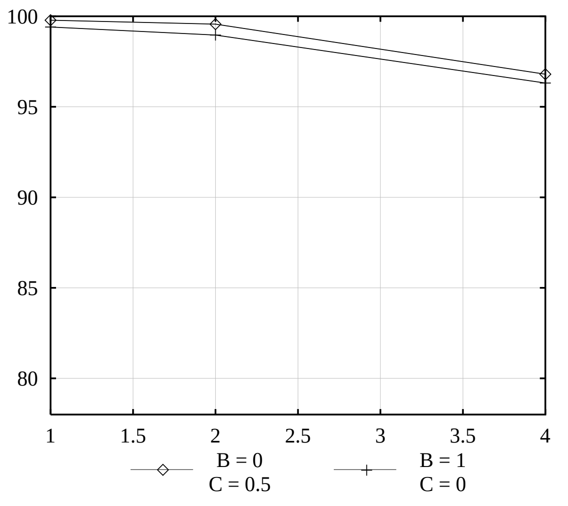
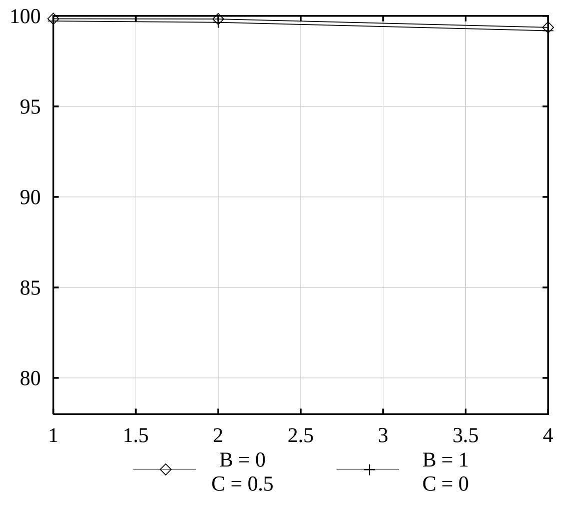


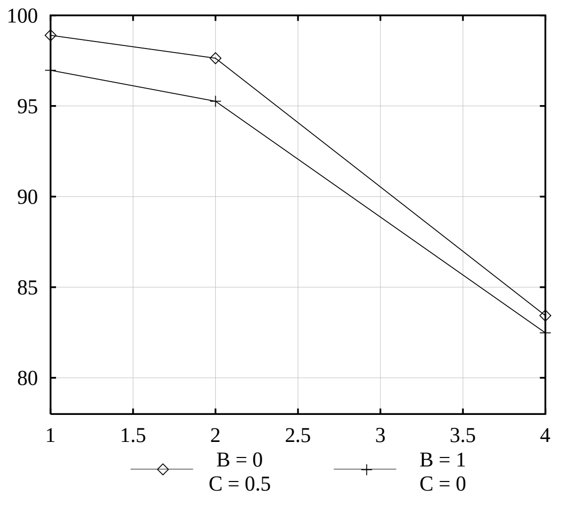
Fig. 3.18 - Volume shearing with separable cubic filter reconstruction



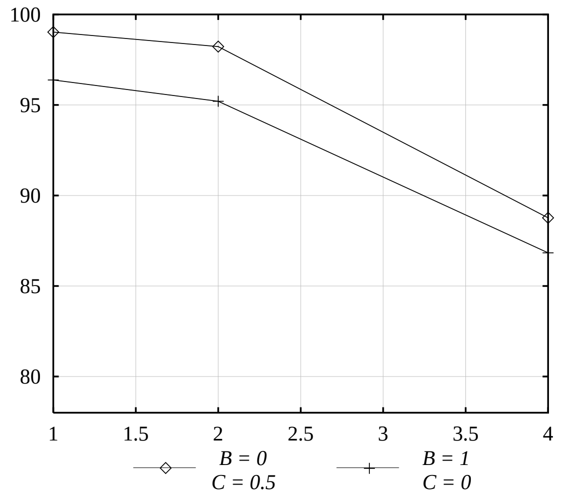
(a) - Intensity IP *(Points)*



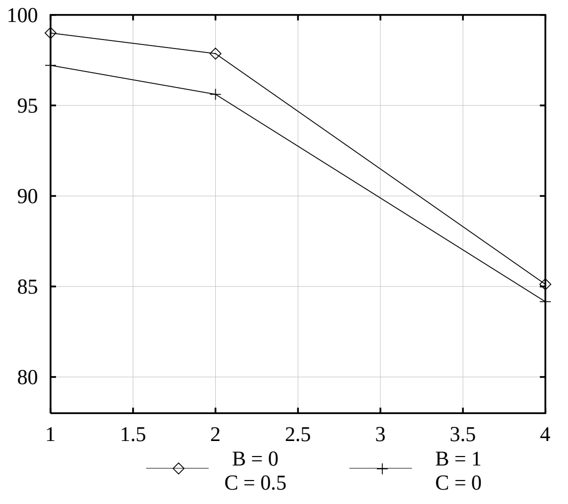
(d) - Intensity IP *(Mixed)*



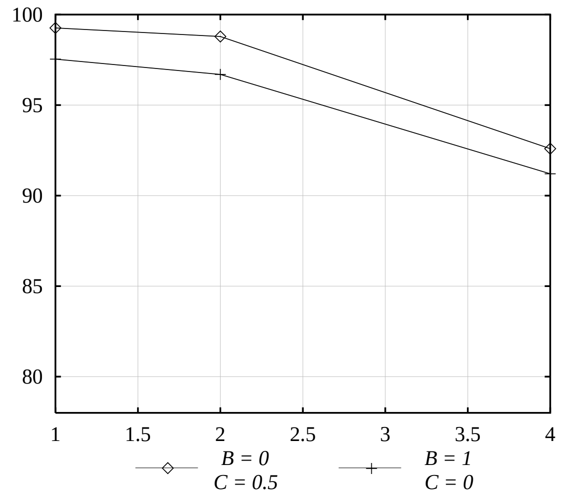
(b) - Gradient IP *(Points)*



(e) - Gradient IP *(Mixed)*



(c) - Intensity, Gradient IP *(Points)*



(f) - Intensity, Gradient IP *(Mixed)*

Fig. 3.19 - Inner product of references and test images made with separable cubic filters
Horizontal axis is number of pixels per sample points (data taken at 1×1, 2×2, and 4×4)

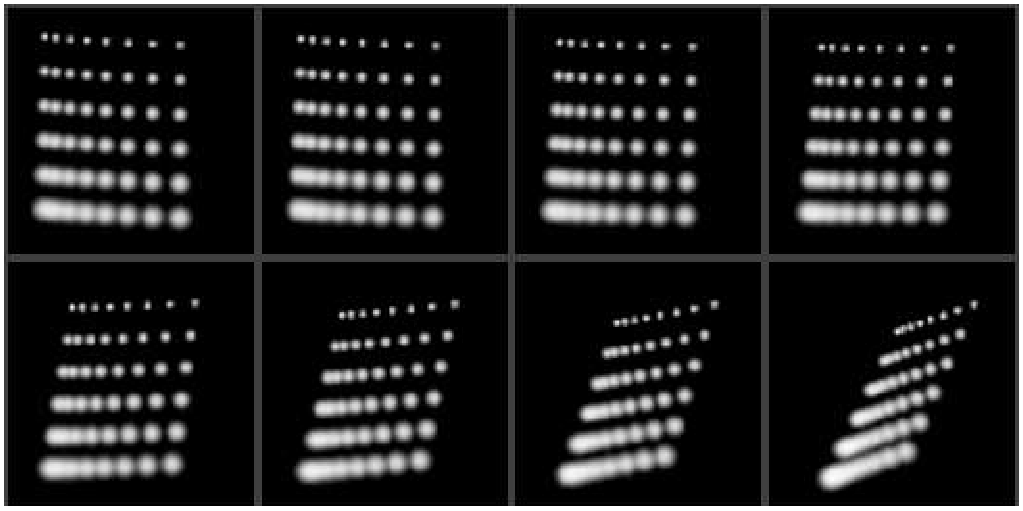


Fig. 3.20 - Animation sequence made with Catmull-Rom cubic reconstruction filter

3.5.4. Filter Comparison

A comparison of the inner product results is shown in figure 3.21 for 128×128 and 64×64 pixel images. The best IP results (using gradient and intensity) for each filter are averaged over the *mixed* and *points* data. The ray casting step size is fixed at two to equalize the image lattice density for all filters. The Gaussian splat kernel has σ = 0.6 and the cubic filter is the Catmull-Rom spline.

In the experiment described in this chapter, the rendering methods, the dimension of the filter, and the resampling point are all varied in addition to the filter kernels. It was implicitly assumed that the filter kernel was the dominant factor affecting image quality. Additional tests are run to verify this assumption. The splatting rendering method is used

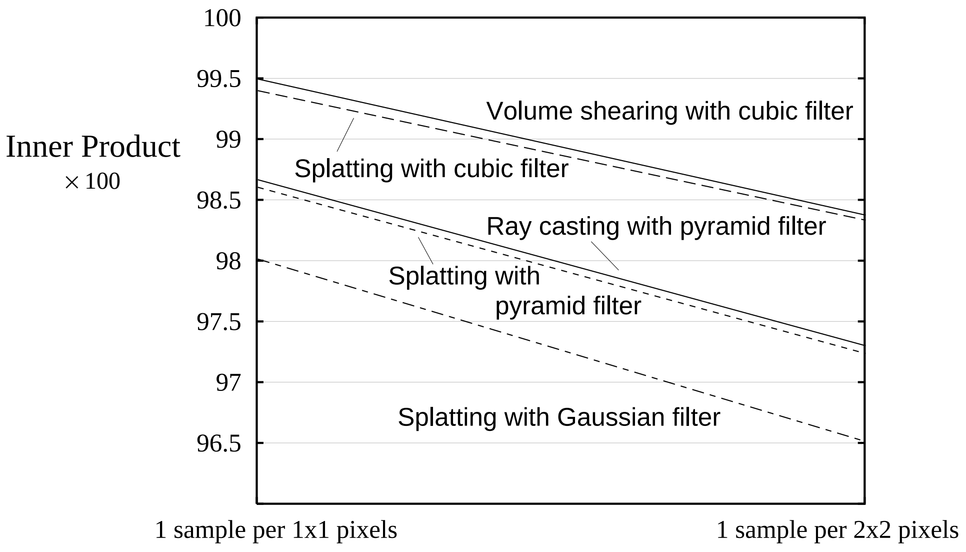


Fig. 3.21 - Inner product comparison

with a selectable 2D filter to render images with the three filters used in the previous tests: a) a 2D Gaussian filter with $\sigma = 0.6$, b) a 2D pyramid filter, c) a 2D Catmull-Rom cubic spline. This test varies *only* the choice of filter kernels - the rendering method, dimension of the filter, and location of resampling points are kept constant. The IP results (using gradient and intensity) for 128×128 and 64×64 images are averaged for the *mixed* and *points* data. The results are shown in figure 3.21 and confirm the assumption that the filter kernel is the major determinant of image quality. These results support the thesis statement, *"A separable cubic filter provides more accurate volume reconstruction than a pyramid filter or a Gaussian filter."* Further evidence of this is provided in chapter four.