# Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware

*Brian Cabral, Nancy Cam, and Jim Foran*
Silicon Graphics Computer Systems*

## Abstract

Volume rendering and reconstruction centers around solving two related integral equations: a volume rendering integral (a generalized Radon transform) and a filtered back projection integral (the inverse Radon transform). Both of these equations are of the same mathematical form and can be dimensionally decomposed and approximated using Riemann sums over a series of resampled images. When viewed as a form of texture mapping and frame buffer accumulation, enormous hardware enabled performance acceleration is possible.

## 1 Introduction

Volume Visualization encompasses not only the viewing but also the construction of the volumetric data set from the more basic projection data obtained from sensor sources. Most volumes used in rendering are derived from such sensor data. A primary example being Computer Aided Tomographic (CAT) x-ray data. This data is usually a series of two dimensional projections of a three dimensional volume. The process of converting this projection data back into a volume is called *tomographic reconstruction*.[1] Once a volume is tomographically reconstructed it can be visualized using volume rendering techniques.[5, 12, 9, 1, 6, 18]

These two operations have traditionally been decoupled, being handled by two separate algorithms. It is, however, highly beneficial to view these two operations as having the same mathematical and algorithmic form. Traditional volume rendering techniques can be reformulated into equivalent algorithms using hardware texture mapping and summing buffer. Similarly, the Filtered Back Projection CT algorithm can be reformulated into an algorithm which also uses texture mapping in combination with an accumulation or summing buffer.

The mathematical and algorithmic similarity of these two operations, when reformulated in terms of texture mapping and accumulation, is significant. It means that existing high performance computer graphics and imaging computers can be used to both render and reconstruct volumes at rates of 100 to 1000 times faster than CPU based techniques.

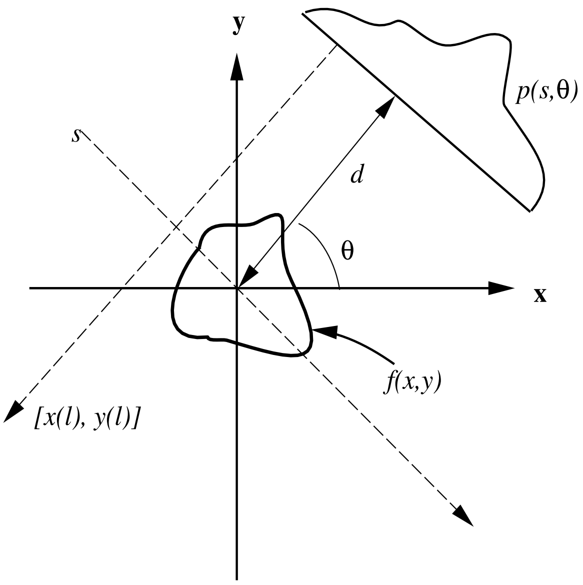


Figure 1: The Radon transform represents a generalized line integral projection of a 2-D (or 3-D) function $f(x, y, z)$ onto a line or plane.

## 2 Background: The Radon and Inverse Radon Transform

We begin by developing the mathematical basis of volume rendering and reconstruction. The most fundamental of which is the Radon transform and its inverse. We will show that volume rendering, as described in [5, 9, 1, 6, 18], is a generalized form of the Radon transform. Finally, we will demonstrate efficient hardware texture mapping based implementations of both volume rendering and it's inverse: volume reconstruction.

The Radon transform defines a mapping between the physical object space $(x, y)$ and its projection space $(s, \theta)$, as illustrated in figure 1. The object is defined in a Cartesian coordinate system by $f(x, y)$, which describes the x-ray absorption or attenuation at the point $(x, y)$ in the object at a fixed $z$-plane. Since the attenuation is directly proportional to the volumetric density of the object at that spatial position, a reconstructed image of $f(x, y)$ portrays a two dimensional non-negative density distribution.

The Radon transform can be thought of as an operator on the

*2011 N. Shoreline Blvd., Mountain View, CA 94043

[1]The term tomographic reconstruction or Computed Tomography (CT)[15] is used to differentiate it from signal reconstruction: the rebuilding of a continuous function (signal) from a discrete sampling of that function.

function $f$ producing function $p$, and is given by:

$$p(s,\theta) = \int_{-\infty}^{\infty} f(x(l),y(l))dl \quad (1)$$

The function $p(s,\theta)$ represents the line integral projection of $f(x(l),y(l))$, at angle $\theta$, along the ray $[x(l),y(l)]$, where:

$$x(l) = d\cos\theta + s\sin\theta + l\cos\theta$$
$$y(l) = -d\sin\theta + s\cos\theta + l\sin\theta$$

Note that this assumes that the projection, $p$, is $d$ distance away from the object coordinate system's origin and that all the projection rays are parallel. This later assumption can be relaxed as we will see below.

The function $p(s,\theta)$ when viewed as a two dimensional image is known as a *sinogram.* (see image 1) In this representation, $s$, is mapped onto horizontal axis and $\theta$ is the vertical axis. It is known as a sinogram because a fixed point $x, y$ in the object, $f$, will trace a sinusodial path through the projection space, $p$.

The Radon transform has two notable properties. The first is that $p$ is periodic in $\theta$: $p(s,\theta) = p(s,\theta+\pi)$, which means that a full reconstruction of $f$ can occur with projection from $0$ to $\pi$ instead of $0$ to $2\pi$. The second and more important property relates the spatial representation of the object, $f$, and it's projection, $p$, to Fourier space equivalent forms. This property, known as the *Fourier Slice Projection Theorem*, states that the inverse Fourier transform of a radial cross section of the Fourier transform, $F$, of the object function at angle $\omega$, is equal to $p$:

$$\tilde{p}(s,\theta) = \int_{-\infty}^{\infty} P(\omega,\theta)e^{j2\pi\omega s}|\omega|d\omega \quad (2)$$
$$= \mathcal{F}^{-1}(|\omega|) * p(s,\theta) \quad (3)$$
$$f(x,y) = \int_{0}^{\pi} \tilde{p}(s,\theta)d\theta \quad (4)$$

Where $P$ is the Fourier transform of the projection $p$ and $\tilde{p}$, represents a filtered version of the projection $p$. The filtering is basically a one dimensional convolution operator in the spatial domain, but is usually performed in Fourier domain to minimize computations.[2] More importantly, note that $f$, has a form identical to that found in equation (1), that of a single integral of a scalar function. These two integral transforms can be represented in discrete form by:

$$p(s,\theta) \approx \Delta l \sum_{k=1}^{M} f(x(l_k),y(l_k)) \quad (5)$$
$$f(x,y) \approx \frac{\pi}{N}\sum_{k=1}^{N} \tilde{p}(s_k,\theta_k) \quad (6)$$

Note that both of these equations imply that the summands are resampled as the summation is taking place. This resampling is inherent in the coordinate system transformation implied by the projection operation of the Radon transform and its inverse. The resampling can be made more explicit by rewriting equations (5) and (6)

$$p(s_i,\theta_j) \approx \Delta l \sum_{k=1}^{M} f(x_k,y_k) \quad (7)$$
$$f(x_i,y_j) \approx \frac{\pi}{N}\sum_{k=1}^{N} \tilde{p}(s_k,\theta_k) \quad (8)$$

where

$$x_k = d\cos\theta_j + s_i\sin\theta_j + l_k\cos\theta_j \quad (9)$$
$$y_k = -d\sin\theta_j + s_i\cos\theta_j + l_k\cos\theta_j \quad (10)$$
$$s_k = d\tan(\tan^{-1}(\frac{y_j}{x_i}) - \theta_k) \quad (11)$$

In particular note that $f$ and $\tilde{p}$ in equations (7) and (8) respectively, are being discretely resampled as a function of the input parameters. Traditionally it has been this resampling and summation that takes the majority of time in the reconstruction and rendering process.[7, 17] A problem that is exacerbated when handling the full three dimensional case.

## 2.1 Orthographic volume rendering and the generalized Radon transform

We can generalize the discrete form of the Radon transform equation (7) to handle all single scattering volume rendering techniques.[12, 4] This is done by multiplying the summand by a weighting term, $c_k$.

$$p(s_i,\theta_j) \approx \Delta l \sum_{i=k}^{M} c_k f(x_k,y_k) \quad (12)$$

Here $s$ represents a pixel in a scan line and $\theta$ the viewing position about the volume, $f$. The variations between different volume rendering algorithms focus on determining these weighting values and choosing an efficient mechanism for computing equation (12). A simple weighting scheme is to make $c_k$ be proportional to $l_k$ or $l_k^2$. This technique is widely used in computer graphics and is more commonly known as depth queueing. It has been used for volume rendering by [1]. However, most volume renderers, use a more complicated data dependent weighting scheme. The most common of which is defined by the **over** operator described in [10] and used in volume rendering by [5]. $c_k$ can be defined recursively from the definition of the **over** operator as:

$$\begin{aligned} c_0 &= \alpha_0 \\ c_1 &= \alpha_0(1-\alpha_1) \\ c_2 &= c_1(1-\alpha_2) \\ &\vdots \\ c_k &= c_{k-1}(1-\alpha_k) \end{aligned}$$

Since $c_k$ depends on all previous $c$'s most volume rendering algorithms perform this calculation iteratively front to back, or visa versa, computing $c_k$ implicitly as the value of $p$ get summed. By looking at volume rendering as a generalized Radon transform we keep the mathematical form of both the forward and inverse Radon transform the same; implying that the same basic hardware primitives can be used to solve either problem with equal ease. Before describing how one might create these algorithms it is important to describe two other types of generalizations. The first is fan beam

[2] the shape of this filter in the Fourier domain is such that it has infinite spatial extent and, hence, makes spatial implementations computationally impractical. We will not study filter implementations in this paper. Instead, we will focus on the backprojection operation, which is the most demanding part of the reconstruction.

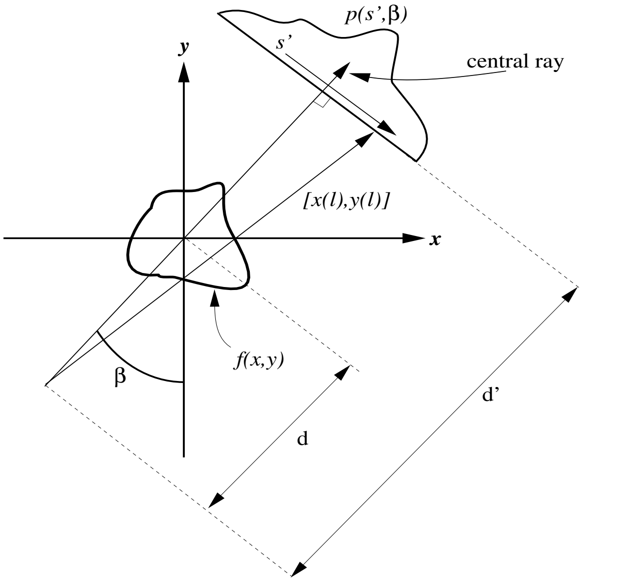


Figure 2: Fan beam geometric arrangement.

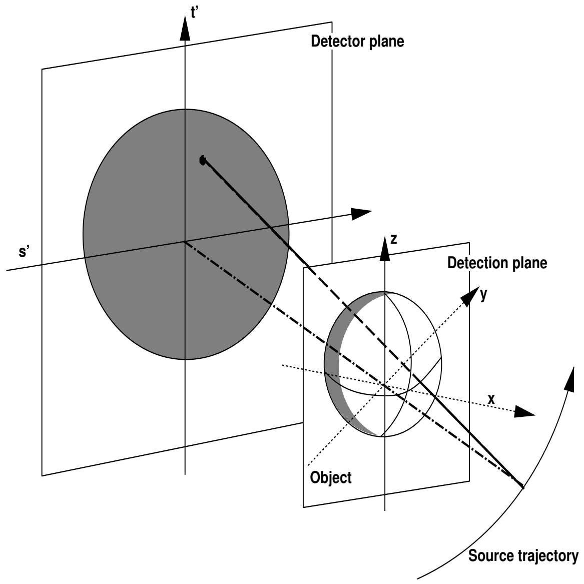


Figure 3: The three dimensional configuration of a planar sensor array with a point source x-ray

reconstruction. The second is the three dimensional generalization of the fan beam reconstruction known as cone beam reconstruction, which is particularly interesting because it is isomorphic to the perspective form of volume rendering.

## 2.2 Fan beam reconstruction

Parallel ray CT can be generalized to the fan beam case where there is a single source of the x-rays which go through the object onto a linear array of sensors as before.[3] (see Figure 2) There are two ways one can go about performing reconstruction using a fan beam data. The first and conceptually the simplest, is to take all the rays from all $M$ projections and sort them into the appropriate parallel projection. This approach has a very serious draw back because constraints must be placed on the sampling angles and sensor angles so that the individual rays get sorted into parallel projections which are exactly parallel to the corresponding fan beam ray. Even if these constraints are met the resulting parallel projection data will not be evenly sampled in $s$, introducing additional artifacts in the resampling process.

The second approach is to utilize a direct method; let the detectors be located linearly in $s'$ at distance $d'$ from the source as shown in figure 2. First, we will normalize for $d'$, relocating the detector array to the origin by scaling: $s = s'\frac{d}{d'}$. The fan beam projection data must also be multiplied by a path length difference term, $d/\sqrt{d^2+s^2}$. After making these adjustments, the weighted projection data are convolved as before to yield $\tilde{p}(s,\beta)$.[4]

There must also be an inverse projective distance squared normalization term. This will be explained later. It is given by $\frac{1}{r^2}$, where $r$ is the distance relative to d along the central ray for point $x, y$ in the object. After making these adjustments the fan beam equations are:

$$f(x,y) \quad = \quad \int_0^{2\pi} \frac{1}{r^2}\tilde{p}(s,\beta)d\beta \tag{13}$$

[3] Fan beam reconstructions usually come in two flavors: the curved equi-angular sensor configuration; or the equi-spaced sensor configuration. It is the later which is the focus of this paper.

[4] The equivalence of the filtering used in both the parallel and the fan beam cases can be found in [7, 17].

$$f(x_i,y_j) \quad \approx \quad \Delta\beta \sum_{k=1}^{N} \frac{1}{r^2}\tilde{p}(s_k,\beta_k) \tag{14}$$

# 3 Three dimensional reconstruction and rendering

There are two primary ways in which one can reconstruct a three dimensional volume from CT data depending on how the data was acquired. Both the parallel and fan beam approaches described above assume that the sensor array is one dimensional, residing in a constant $z$-plane. Reconstructing a three dimensional volume can be done one $z$-plane at a time until all the slices are reconstructed. Likewise, one could volume render in this manner one scan line at a time using an inverse form of the fan beam formula. However, the more compelling case for both volume rendering and reconstruction occurs if the sensor or pixel array is a two dimensional plane. For volume rendering this is a more natural approach since the output of any volume renderer is necessarily a two dimensional array of pixels. For reconstruction it means that data can be gathered at a much higher rate.

## 3.1 Cone Beam Reconstruction

Consider the case of an X-Ray source which projects through a region of interest onto a rectangular grid of detectors. Assume that the detector array and X-Ray source are fixed on a rotating gantry which allows a cylindrical region of interest to be swept out. The projection data $p(s',t',\beta)$ which can be measured by X-Ray detectors is:

$$p(s',t',\beta) \quad = \quad \int_0^{\sqrt{d'^2+s'^2+t'^2}} f(x(l),y(l),z(l))dl \tag{15}$$

Here, $s'$ and $t'$ represent the detector coordinate variables in the detector plane $d'$ from the source, at angle $\beta$ about the object $f(x,y,z)$; $l$ is the parametric distance along the ray, $[x(l),y(l),z(l)]$. (see Figure 3) The cone beam can be thought of as fan beam tilted up or down in $z$ by an angle $\gamma$. As with the fan beam, we will first normalize for detector distance by scaling $s'$ and $t'$ by $d/d'$. The formulation for the cone beam is that of the fan beam, with exception that the fan beam rays now contain a $z$-component and the fan beam path length difference term becomes $d/\sqrt{d^2+s^2+t^2}$

After convolution, the projection data are used to calculate the three dimensional volume data in Cartesian coordinates by the operation of weighted back projection. The weighting factor is determined by inspecting the relationship of the three dimensional pixel grid $x, y, z$ to the projective coordinate system of an individual projection. This coordinate system is composed of the $s$ and $t$ indexes of the normalized projection data set and the perpendicular coordinate $r$ which represents the direction from the source to the detector array. These three form a projective coordinate system in that rays emanating from the source to the detector are considered to be at a constant $s, t$ position. The projective coordinate system can be transformed into a Cartesian coordinate system by multiplying the s and t coordinates by $r$ to yield $(rs, rt, r)$.[14] This Cartesian system can be transformed to the three dimensional volume grid by a standard three dimensional transformation matrix.

It is now easy to see that the volumetric differential in the projective coordinate system of the beam is related to the three dimensional volumetric differential by:

$$dxdydz = r^2 dsdtdr$$

Furthermore, it is also easy to see that the weighting factor for the fan beam is also $1/r^2$ because the fan beam is simply the cone beam with t=z=0; the differential is unchanged!

The cone beam equations are thus:

$$f(x,y,z) = \int_0^{\pi} \frac{1}{r^2}\tilde{p}(s,t,\beta)d\beta \quad (16)$$

$$f(x_i,y_j,z_k) \approx \frac{2\pi}{N}\beta \sum_{k=1}^{N} \frac{1}{r^2}\tilde{p}(s_k,t_k,\beta_k) \quad (17)$$

### 3.2 Perspective volume rendering using the generalized Radon transform

The generalized Radon transform can be further generalized to the three dimensional perspective case. Like the cone beam one has to take into account both the perspective weighting due to the off center traversal of $[x(l), y(l), z(l)]$ and the $1/r^2$ volumetric differential term. With this in mind equation (1) can be rewritten as:

$$p(s,t) = \frac{d}{\sqrt{s^2+t^2+d^2}} \int_0^{l_{max}} \frac{c(l)f(x(l),y(l),z(l))}{r^2} dl \quad (18)$$

Where, $c(l)$ is the continuous form of the weighting term introduced in section 2.1 and $1/r^2$ is the same projective weighting as described above. $p$ is the image plane for a projective $(s,t,r)$ view from any arbitrary direction. The perspective weighting term is necessary because the sensor/image plane resides in a Cartesian coordinate system. Since the projective weighting term is constant with respect to $s$ and $t$ it can be performed as a post multiply on the projection $p$ and is given by:

$$p(s,t) \approx \frac{d}{\sqrt{s^2+t^2+d^2}} \frac{d'}{M} \sum_{k=1}^{M} \frac{1}{r^2} c_k f(x_k,y_k,z_k) \quad (19)$$

Notice that mathematical equations (17) and (19) are of the same form: that of a weighted sum. In particular both sums are being weighted by a projection term. This similarity means that both techniques are ideally suited for implementation on high performance true three dimensional texture rendering hardware. We will now show that the difficulty of this problem merits hardware acceleration.

## 4 Computational complexity

The computational complexity implied by the discrete form of the Radon transform and the filtered back projection is sufficiently large that most CT scanners include specialized hardware to perform these tasks. However, two trends are rapidly changing CT medical imaging. One is that increasing pressure to reduce costs is driving a transition from special purpose hardware to general purpose "off the shelf" hardware. The second is that technological advances are increasing the quantity and quality of the data which can be collected for use in diagnosis.

These trends call for a computer system architecture which addresses data visualization and reconstruction in a general purpose way. This is not, however, the same as saying that the problem must be solved by a Von Neuman type processor of sufficient spec mark rating. Although that may someday be possible it will never represent an optimal solution for this type of problem.

### 4.1 FFT and filtering complexity

A typical modern generation CT scanner has on the order of 512 detectors and is capable of taking 1024 projections per second. For a fan beam reconstruction each data set must undergo Fast Fourier Transformation (FFT), a multiply, and inverse FFT to accomplish the "bow tie", $\omega$, convolution operation. To prevent aliasing the data set must be extended to 1024 [7, 17]. The FFT will then consist of 10 sets of 512 butterfly calculations each of which is a complex multiply and two complex adds or four simple multiplies and six simple additions. Thus the FFT will require $10*10*512$ floating point operations or 50 KFLOPs. This is followed by a complex multiplication of 6 KFLOPs and an inverse transform of 50 KFLOPs for a total of 106 KFLOPs for a single projection. Multiplying this result by the 1024 projections yields 106 MFLOPs. Generally, it is desirable to perform reconstruction in the time required to acquire the scan data, typically one second.

### 4.2 Back Projection and Radon transform complexity

Although modern floating point processors can now handle such magnitude of calculations there is little spare performance available for the additional required operations. Convolved projection data must be back projected to create an image. To create a 512 squared image the contribution to each pixel of each convolved projection must be calculated and summed. It is this operation which is the most costly in the reconstruction process.

For each projection each pixel's projection onto the detector data set must be calculated. The data must then be resampled (usually using linear interpolation) and summed to accumulate the total for that pixel. Since we have 256 K pixels, these operations will need to be performed 256 million times per second. Each operation requires several operations as well as memory fetches.

In the backprojection operation, for each projection, each pixel must be projected into the source data set and then the data set must be interpolated. To generate the projected pixel, two variables must be iterated followed by a division of one by another. The interpolation, in addition to the data fetches, consists of a subtraction, two multiplications and an addition. This data must then be weighted by the square of another iterated variable before being summed with the prior value of that pixel. This brings our total operations to five additions, three multiplications and one division (i.e. nine operations) per pixel per projection. Multiplying 256 million * 9 operations yields 2.3GFlops.

Present day CPUs are nowhere near being able to attain these performance requirements. However, a general purpose imaging and graphics architecture which is available today can implement these algorithms in the one to two second range. The $RealityEngine^{TM}$ system is such an architecture [8]. It is truly general purpose, because it can be used in a wide variety of situations permitting economic benefit of the economy of scale required to amortize continuing evolutionary development and practical manufacturability. The same architecture can be economically used for visual simulation applications as well as photogrammetry, reconnaissance imaging, and advanced design. The RealityEngine comprises a large source image memory connected to the frame buffer through a high performance resampling engine. The resampling engine can perform all the operations required for back projection at 80 to 320 million pixels per second.

## 5 A texture map based reconstruction algorithm

The discrete form of the inverse radon transform suggests a spatial implementation of this equation known as *filtered back projection*. An algorithm to implement this concept is broken up into two separate passes. The first pass filters, $p$, to produce, $\tilde{p}$. The second pass performs the back projection operation. This is accomplished by smearing $\tilde{p}$ back across object space $(x, y)$, at an angle $\theta$. This process is evident in the images found in image 2 and the following pseudo-code implementation:

$\tilde{p} \leftarrow$ Filter($p$)
**For** $\forall i$ and $j$ pixels
    $f(x_i, y_j) \leftarrow 0$
    **For** $\forall k$ angles in $\theta$
        $f(x_i, y_j) \leftarrow f(x_i, y_j) + \tilde{p}(s_k, \theta_k)$
    $f(x_i, y_j) \leftarrow f(x_i, y_j) * \Delta\theta$

After each back projection the values are summed to complete the calculation of equation (4). Traditionally the back projection took orders of magnitude longer to compute than the filtering of $p$. By viewing the heart of back projection as a resampling problem, one can see how to recast this process into texture mapping. If we treat $\tilde{p}(s, \theta)$ as a two dimensional texture and reverse the order of the loops in the above algorithm. The back projection algorithm becomes a texture mapping of a circle rotated by $\theta$ followed by an accumulation.

$\tilde{p} \leftarrow$ Filter($p$)
**For** $\forall k$ angles in $\theta$
    Setup the $\theta_k$ rotation in Z
    Render a texture mapped circle with texture, $\tilde{p}(\omega, \theta_k)$
    Accumulate frame buffer
Return Accumulation and scale by $\Delta\theta$[5]

The fan beam case is implemented using a similar technique. The major differences being the filtering as described in section 2.2, the $\frac{1}{r^2}$ attenuation term, and the fan beam geometry itself. It turns out these differences will not adversly affect the performance the texture mapping implementation. This is because we can use the z-depth cueing to handle the $\frac{1}{r^2}$ behavior at no extra rendering time cost and because the triangle shape of the fan is no harder to render than the circle used for the parallel case. Given these changes the above algorithm becomes:

$\tilde{p} \leftarrow$ Filter(weighted $p$)
Setup depth queueing for the fan beam
Setup perspective texture coordinate matrix
**For** $\forall k$ angles in $\beta$
    Setup the $\beta_k$ rotation in Z
    Render a texture mapped circle, $p(s, \beta_k)$
    Accumulate frame buffer
Return Accumulation and scale by $\Delta\beta$

In all three cases both texture mapping and frame buffer accumulation is hardware accelerated on existing state-of-the-art graphics systems. These new algorithms are significantly faster (see section 7) than previous algorithms and rival specialized hardware designed explicitly to perform these tasks.

The cone beam algorithm is a modification of the fan beam algorithm. For the cone beam case one a circular slice in $z$ (constant $t$) is reconstructed at a time. This circular slice is projected onto the $s, t$ projection image using homogeneous texture coordinates, see [14]. If the distances from the source, object, and sensor plane remain constant with respect to $\beta$, then the projection onto the sensor plane is constant. Thus the $s, t, r$ texture coordinates and projection matrix is computed once for each slice. The circular slice is then rendered once for each projection, summed, and normalized producing the reconstructed slice. So the above algorithm becomes:

**For** $\forall z$ slices in $t$
    $\tilde{p} \leftarrow$ Filter(weighted $p$)
    Setup depth queueing for the fan beam
    Setup perspective texture coordinate matrix
    **For** $\forall k$ angles in $\beta$
        Setup the $\beta_k$ rotation in Z
        Render a texture mapped circle, $p(s, \beta_k)$
        Accumulate frame buffer
    Return Accumulation and scale by $\Delta\beta$

## 6 Texture mapped volume rendering

Most volume rendering algorithms come in one of two flavors: either backwards projecting or forwards projecting. The backwards projecting or ray tracing approach solves equation (18) by iterating over each $s$ and $t$ and summing along the ray $[x(l), y(l), z(l)]$. A forwards projecting technique loops over all values of $f$ in back to front order finding the appropriate $s$ and $t$ to sum into. Our approach is a hybrid of these two.

For $\forall k$ screen parallel image planes in distance $l_k$
    Intersect slicing plane $l_k$ and trilinearly interpolate $f$
    Blend the texture mapped slice into framebuffer, $p$
    (included in the blend is $\Delta l$ weighting)
Attenuate $p$ by $d/\sqrt{s^2 + t^2 + d^2}$

The final attenuation is necessary, as was mentioned above to

[5] Implementation note: Since the accumulation buffer is a fixed point 24 bit buffer, data moving in and out of this buffer must be appropriately scaled and biased beyond the normal scaling by $\Delta\theta$ specified in equation (9).

handle the path length difference for off center pixels. However, this term tends to negligible in all cases except when there is extreme perspective over a large range in $r$.

It is worth noting that a number of researchers have independently implemented this rendering algorithm using 3 dimensional texture mapping on the RealityEngine system [13, 3]. The significant difference here is not in the particular implementation but rather in presenting the reconstruction and rendering algorithms in similar form and thus showing the inherent similarity.

## 7 Performance results

The hardware accelerated algorithms for volume rendering and reconstruction result in formidable performance increases. All timing measurements were performed on a four Raster Manager (RM) RealityEngine Onyx with a 150Mhz R4400 CPU. Since both the reconstruction and volume rendering algorithms are a linear function of the fill rate, removing RMs affects performance in linear manner. All images are screen captures of the actual test codes used to implement the algorithms described here.

All three texture mapped based reconstruction algorithms have been implemented and tested. This algorithm has shown roughly 100 times speed improvement over a CPU based implementation. Images 3 and 4 illustrates the quality of the reconstruction of a 512x804 sinogram into a 512x512 reconstruction space. This normally takes 7+ minutes to reconstruct using an optimized CPU based algorithm. As indicated in the caption the hardware implementation performs the same task in 2.1 seconds. Plate 1 shows the results of applying the cone beam reconstruction algorithm on a 3 dimensional version of the Shepp-Logan phantom [17]. The reconstruction time for $128^3$ volume is 3.5 seconds.

These times do not include the time it takes to filter $p$. Traditionally the time to filter $p$ was overwhelmed by the reconstruction time and thus was not viewed as a bottleneck. The filtering time becomes preponderant when performing hardware accelerated back projection. It is, however, well known how to implement high performance parallel and hardware accelerated FFT filtering.[16]

We have implemented both the perspective and parallel plane volume rendering algorithm. This algorithm can render a 512x512x64 8bit volume into 512x512 window in 0.1seconds. This time includes performing the trilinear interpolation implied by the arbitrary slicing of the volume by the sampling plane. It also includes on-the-fly density to RGB$\alpha$ color look up transfer functions. Since the planar sampling described in section 6 is arbitrary and oblique with respect to the overall volume, it is trivial to provide for arbitrary half-plane slicing as seen in Plate 3. When coupled with arbitrary hardware clipping planes as provided by the graphics hardware it is possible to do both volume rendering and slicing at the same time with no performance degradation. The RealityEngine supports up to six concurrent arbitrary clipping planes (in addition to the six viewing frustum clipping planes). Specific, implementation details are essentially the same as that of [13, 3].

## 8 Future directions and conclusion

The algorithmic unification of the volume rendering and reconstruction in the spatial domain means that a single hardware accelerated solution is possible. Future directions include handling curvilinear coordinate systems and domain iterative solutions[7]. A domain iterative solution is a powerful technique for handling cases where only a limited number of projections are gathered. This approach iterates over a cycle of filtering followed by a back projection followed by a radon transform (volume rendering). Since both the back projection and volume rendering steps can be accelerated with a single piece of hardware, such domain iterative solutions become practical.

While the visual results presented are compelling and comparable to previous work [13, 5, 9, 1, 6, 3] a good quantitative comparison is essential to determining whether this or any other approach can be deemed good. Unfortunately, very little work has been done in the area of quantitatively comparing various volume rendering and reconstruction algorithms. Such work is a formidable task and beyond the scope of this paper and represents an important area of future research.

## 9 Acknowledgements

Thanks to John Airey for many insightful discussions on texture mapping and volume rendering. Also thanks to Chris Tanner who wrote the original software which inspired the volume rendering part of this work. Steve Azevedo of Lawrence Livermore National Laboratory and University of California, Davis deserves special thanks for insightful discussions on CT and for his exceptionally written Ph.D. thesis on the subject matter. Also thanks to Bob Drebin, Benjamin Zhu, and Carl Korobkin of Silicon Graphics for discussions and helpful suggestions; and thanks to Amber Denker for providing color printing assistance.

## References

[1] Akeley, Kurt. “RealityEngine Graphics. Rendering.” Proceedings of SIGGRAPH ’93 (Anaheim, California, August 1-6, 1993). In Computer Graphics Proceedings, Annual Conference Series, 1993, ACM SIGGRAPH, New York, 1993, pp. 109-116.

[2] Azevedo. Stephen. “Model-based Computed Tomography for Nondestructive Evaluation.” *Ph.D. thesis. UCRL-LR-106884*.

[3] Cullip U., Neumann, U. “Accelerating Volume Reconstruction with 3D Texture Hardware.” *UNC Tech Report TR93-0027*. (1993).

[4] Dorf, Richard. “The Electrical Engineering Handbook.” *CRC Press*. (1993), 1510, 1516.

[5] Drebin, Robert, Carpenter, Loren and Hanrahan, Pat. “Volume Rendering.” Proceedings of SIGGRAPH ’88 (Atlanta, Georgia, August 1-5, 1988). In Computer Graphics 22, 4 (august 1988), 65-74.

[6] Haeberli, Paul and Akeley, Kurt. “The Accumulation Buffer: Hardware Support for High-Quality Rendering.” Proceedings of SIGGRAPH ’90 (Dallas, Texas, August 6-10, 1990). In Computer Graphics 24, 4 (August 1990), 309-318.

[7] Kajiya, James and Von Herzen, Brian. “Ray Tracing Volume Densities.” Proceedings of SIGGRAPH ’84 (Minneapolis, Minnesota, July 23-27, 1984). In Computer Graphics 18, 3 (July 1984), 165-174.

[8] Kajiya, James, “The Rendering Equation.” Proceedings of SIGGRAPH ’86 (Dallas, Texas, August 18-22, 1986). In Computer Graphics 20, 4 (August 1986), 143-150.

[9] Kak, Avinash and Slaney, Marc. “Principles of Computed Tomographic Imaging.” *IEEE Press*. 1988.

[10] Krueger, Wolfgang. “The Application of Transport Theory to the Visualization of 3-D Scalar Fields.” *Computers in Physics*. (April 1991). 397-406.

[11] Porter, Thomas and Duff, Tom. “Compositing Digital Images.” Proceedings of SIGGRAPH ’84 (Minneapolis, Minnesota, July 23-27, 1984). In Computer Graphics 18, 3 (July 1984), 253-259.

[12] Russ, John. “The Image processing Handbook.” *CRC Press*. (1992), 339-365.

[13] Sabella, Paolo. “A Rendering Algorithm for Visualizing 3D Scalar Fields.” Proceedings of SIGGRAPH ’88 (Atlanta, Georgia, August 1-5, 1988). In Computer Graphics 22, 4 (August 1988), 51-58.

[14] Segal, Mark, Korobkin, Carl, van Widenfelt, Rolf, Foran, Jim and Haeberli, Paul. “Fast Shadows and Lighting Effects Using Texture Mapping.” Proceedings of SIGGRAPH ’92 (Chicago, Illinois, July 7-26, 1992). In Computer Graphics 26, 2 (July 1992), 249-252.

[15] Totsuka, Takashi and Levoy, Marc. “Frequency Domain Volume Rendering.” Proceedings of SIGGRAPH ’93 (Anaheim, California, August 1-6, 1993). In Computer Graphics Proceedings, Annual Conference Series, 1993, ACM SIGGRAPH, New York, 1993, pp. 271-278.

[16] Upson, Craig and Keeler, Michael. “V-BUFFER: Visible Volume Rendering.” Proceedings of SIGGRAPH ’88 (Atlanta, Georgia, August 1-5, 1988). In Computer Graphics 22, 4 (August 1988), 59-64.

[17] Westover, Lee. “Footprint Evaluation for Volume Rendering.” Proceedings of SIGGRAPH ’90 (Dallas, Texas, August 6-10, 1990). In Computer Graphics 24, 4 (August 1990), 367-376.

[18] Wilson O., Van Gelder A. and Wilhelms Jane. “Direct Volume Rendering via 3D Textures”. To be published in the Visualization ’94 Conference Proceedings (October 1994).