Volume Graphics (2003)
I. Fujishiro, K. Mueller, A. Kaufman (Editors)

Rapid Emission Tomography Reconstruction

Ken Chidlow† Torsten Möller‡

Department of Computer Science, Simon Fraser University, Burnaby, B.C., Canada

Abstract

We present new implementations of the Maximum Likelihood Expectation Maximization (EM) algorithm and the related Ordered Subset EM (OSEM) algorithm. Our implementation is based on modern graphics hardware and achieves speedups of over eight times current software implementation, while reducing the RAM required to practical amounts for today's PC's. This is significant as it will make this algorithm practical for clinical use. In order to achieve a large speed up, we present bit splitting over different color channels as an accumulation strategy. We also present a novel hardware implementation for volume rendering emission data without loss of accuracy. Improved results are achieved through incorporation of attenuation correction with only a small speed penalty.

Categories and Subject Descriptors (according to ACM CCS): I.4.5 [Image Processing and Computer Vision]: ReconstructionTransform methods

1. Introduction

Since the introduction of Computed Tomography (CT) into clinical use in 1972, medical images have allowed radiologists to view the patients anatomy without the need of invasive surgery. The strength of Transmission Tomography, such as CT images, is the clarity of the anatomical structure. On the other hand, in Emission Tomography, the goal is to image the anatomy's function. Positron Emission Tomography (PET) and Single Photon Emission Computed Tomography (SPECT) are common techniques in nuclear medicine for imaging anatomical function.

A typical SPECT setup is shown in Fig. 1. The subject has been injected with a radioactive tracer, that radiates photons. A gamma camera rotates around the subject counting photons that are emitted. The collimator ensures that the camera only counts photons that are coming perpendicular to it.

The Radon transform is the line integral projection of a 3D function onto a 2D plane. In medical imaging, a set of x-ray images from different angles is the Radon transform of the subject. The problem then is to reconstruct the 3D object by performing an inverse Radon transform. Filtered backprojection (FBP) is the correct analytical solution to the Radon transform with no noise, which is the case in transmission tomography. However, Emission Tomography uses the attenuated Radon transform, for which FBP is only an approximate solution. Emission Tomography also has a high amount of noise, so FBP becomes less accurate. Since Emission Tomography is fundamentally different than transmission tomography, a reconstruction algorithm is needed that models its physics.

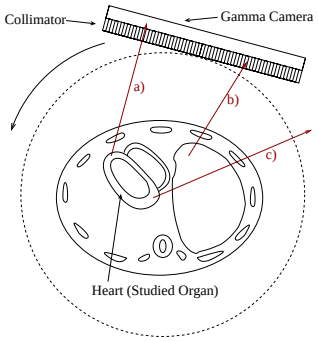


Figure 1: *SPECT setup. Photons emitted: a) passes through the collimator to the camera, b) is absorbed by the collimator, c) misses the camera entirely.*

A family of reconstruction techniques exist that can cor-

† chidlow@sfu.ca
‡ torsten@cs.sfu.ca

rectly modeling the physics of Emission Tomography. These techniques are iterative and have a high computational cost, but yield more accurate reconstructions than FBP.

Methods such as the Algebraic Reconstruction Technique (ART) and Maximum Likelihood Expectation Maximization (ML-EM, or EM) solve the reconstruction problem by iteratively measuring how close the 3D estimation simulates the known 2D projections followed by correcting the 3D estimate. The measuring step is done by projecting the estimation to the same angle as an original camera angle, and then comparing the projected image to the original image. This gives us a correction image that we now backproject to find out how to change our estimate. The problem with iterative techniques is their high computational cost. Computing a single iteration may take many minutes, and these iterative techniques have slow convergence rates. The computational cost has been so high that iterative techniques are not used clinically for SPECT.

In this work we present an implementation of the EM algorithm, that enables the use of commodity graphics hardware. This achieves an eight fold decrease in reconstruction time for the same level of accuracy. In Section 2 we discuss the previous work in accelerated reconstruction techniques. Section 3 describes the methods we used, and in Section 4 we give the implementation details of our work. These sections are followed by results and conclusions in Sections 5 and 6, respectively.

2. Previous Work

Dempster et al.[2] introduced the EM algorithm for solving incomplete data problems in 1977, but it was not until Shepp and Vardi [10] 1982, that the EM algorithm was applied to Emission Tomography. There have been many adaptations and modifications of the EM algorithm that are used in research in Emission Tomography.

Due to the huge computational cost of the EM algorithm, a lot of research has gone into accelerating it. One of the most significant modifications is the Ordered Subsets EM (OSEM) algorithm introduced by Hudson and Larkin [3]. OSEM has better convergence rates than the EM algorithm, which results in a reduction of the number of iterations needed.

In research of hardware acceleration, the different approaches can be divided into three types. The first type is through specialized hardware. Custom electronic boards are made with arrays of very large scale integration (VLSI) chips [4] which perform the necessary calculations for the projection and backprojection. The second approach is based on distributed computing where the algorithm is broken up into parallelizable chunks and distributed across many processors [12]. This works well as the EM algorithm is highly parallelizable. The drawback of both of these methods, is the cost of the hardware involved. In the third method hardware is used to accelerate reconstruction is via the graphics card. Cabral et al. [1] showed how to implement the backprojection step of the FBP algorithm using graphics hardware. More recently, Mueller and Yagel [8] used graphics hardware to reconstruct transmission tomography with the Algebraic Reconstruction Technique (ART).

The difference between the EM algorithm and ART is small. Both are iterative techniques that project the estimate volume against the original camera image to get a correction image, which is backprojected to correct the estimate. The difference is in how the correction image is calculated. In ART, the projection image is subtracted from the original image, yielding a correction image that is the distance between the two. In EM the original image is divided by the projection image. Mueller's work was with transmission tomography, while ours is dealing with emission tomography. The difference leads to different projection and backprojection steps. Our projection adds an attenuation correction step, explained in Section 3.5. Our backprojection step is essentially the same as Cabral's [1] method for FBP, but for implementation details as explained in Section 4.3.

3. Methods

3.1. Setup

The data collected by the gamma camera comes as a set of projection images p, with the number of projections in the set being λ. There is a total of I pixels over all projections in the set. A single pixel from the projection data is indexed by p_i, $1 \leq i \leq I$, which contains the counts of photons recorded. We now reconstruct a volume v with J voxels, indexed by v_j, $1 \leq j \leq J$, that are the expected number of photon emissions. The weight matrix w_{ij}, gives the probability of a photon emitted from voxel v_j being recorded at detector p_i.

The data set we use is a 3D mathematical cardiac-torso (MCAT) phantom which models the anatomical structures of the thorax, developed by Tsui et al. [11].

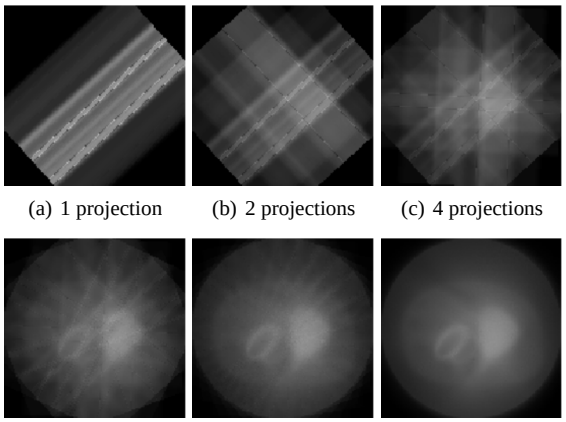
3.2. Filtered Backprojection

Filtered backprojection is almost exclusively used to reconstruct medical images from CT data. FBP is also used to reconstruct PET and SPECT despite its shortcomings mentioned in Section 1. The FBP algorithm can be separated into two steps. First the projection data p is filtered, then the filtered data p' is backprojected across the object space. The principle of filtered backprojection can be captured in the following formula:

$$v_j = \frac{\sum_{i=1}^{I} p'_i w_{ij}}{\lambda} \tag{1}$$

The backprojection step is simply a smearing of a 2D projection image across the 3D object space. This smearing can be seen in Fig. 2, where we are looking at a single 2D slice

of the 3D object space. Once an angle of projection data has been smeared across the object space, it is averaged with the other smeared angles. To average the smeared projections, we add them all together, and divide by the number of projections.



(a) 1 projection (b) 2 projections (c) 4 projections

(d) 8 projections (e) 16 projections (f) 64 projections

Figure 2: *Example of a backprojection after (a) 1, (b) 2, (c) 4, (d) 8, (e) 16 and (f) 64 projections. This figure is of a slice of the MCAT dataset being reconstructed. These projections were not filtered.*

3.3. Expectation Maximization

Expectation Maximization has been used to reconstruct Emission Tomography since Shepp and Vardi [10], although EM is not the primary reconstruction technique used in clinical applications.

Since this is an iterative algorithm, the iterations are indexed by k. The next estimate volume $v^{(k+1)}$ is dependent on our current estimate volume $v^{(k)}$. We start with $v^{(1)}$ being uniform and positive, and in all our examples, each voxel starts with value 1.0.

$$v_j^{(k+1)} = v_j^{(k)} \frac{\sum_{i=1}^{I} \left(\frac{p_i w_{ij}}{\sum_{m=1}^{J} w_{im} v_m^{(k)}} \right)}{\sum_{i=1}^{I} w_{ij}} \tag{2}$$

We now divide this equation up into four parts that are easier to manipulate.

3.3.1. Projection

The projection step is how we go from having a 3D volume to having a series of 2D images that correspond to the projection images captured by the gamma camera. For each pixel i, we define our projection estimate β_i:

$$\beta_i = \sum_{j=1}^{J} v_j w_{ij} \tag{3}$$

That is for each pixel in a projection image, we sum up the value of all the voxels that effect that detector. As seen in

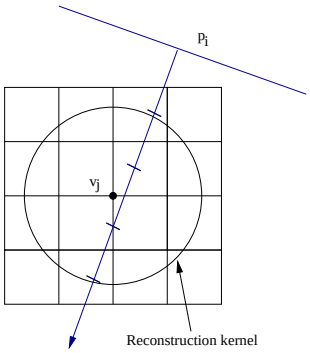


Figure 3: *Weight w_{ij} is calculated by uniformly sampling the ray from pixel i through voxel j's reconstruction kernel.*

Fig. 3, the weight w_{ij} is the integral along the ray from pixel p_i through voxel v_j's reconstruction kernel. This is typically approximated by uniformly sampling the ray.

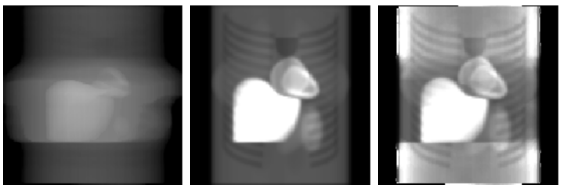
Computing the projection is effectively taking an x-ray of our volume. A projection of the volume after just one iteration can be seen in Fig. 4 (a).

3.3.2. Correction Images

The next step in the EM algorithm is calculating the correction images, Ω. This is done by taking the original projection images that the gamma camera took and dividing them by our calculated projection images, on a pixel by pixel basis.

$$\Omega_i = p_i / \beta_i \tag{4}$$

Equation 4 shows the computation of the correction images, while Fig. 4 shows an example. As shown in Fig.4 a, we see



(a) β (b) p (c) Ω

Figure 4: *Correction Image Calculation. (a) is a projection after one iteration, (b) is an original projection, and (c) is a correction image.*

that after 1 iteration, our volume is still quite blurry. When we create a correction image, Fig. 4 (c), dark parts of the image indicate that the projection was too bright, and light parts of the image show where the projection was too dark. Notice that in Equation 4 it is possible to have a denominator β_i of zero. In this case we set Ω_i to be zero, which will darken the

estimate volume in the corresponding voxels, explaining the dark sides of Fig. 4 (c).

3.3.3. Backprojection

Now that we have our correction images Ω, notice that the backprojection equation, Eq. 5 of the EM algorithm is the same process as that of FBP, Eq. 1. Here we are just backprojecting the correction images instead of the filtered projection images, and we do not divide by λ, which is compensated for when we update the volume. Backprojecting the correction image results in a volume of scaling factors Ψ. A similar smearing results from the following equation as that seen in Fig. 2.

$$\Psi_j = \sum_{i=1}^{I} \Omega_i w_{ij} \tag{5}$$

3.3.4. Update Volume

Once the backprojection is done, we have a volume of correction factors Ψ and we multiply voxel by voxel with our estimate volume $v^{(k)}$ to yield our new estimate $v^{(k+1)}$. Here, the sum of the weights is pre-computed, and in simple cases it is the number of projections λ for each j. This is because the weight of every voxel is one on each projection.

$$v_j^{(k+1)} = \frac{v_j^{(k)} \Psi_j}{\sum_{i=1}^{I} w_{ij}} = \frac{v_j^{(k)} \Psi_j}{\lambda} \tag{6}$$

3.4. Ordered Subset EM

In the projection step of the EM algorithm, we project the volume at each projection angle in p. Then all correction images are calculated and all Ω's are backprojected. This is done to obtain a scaling factor for the estimate volume. To improve the convergence rate, we can divide the projections into sets, and do a sub-iteration on each set. Using subsets allows us to calculate a scaling factor at a smaller cost, so we can compute scaling factors more often. This is the Ordered Subsets EM algorithm from Hudson and Larkin [3].

Once the number of subsets have been chosen, the projections are divided evenly into the subsets. If we have M subsets, then the first subset gets projection $1, M+1, 2M+1$ etc. Although it is possible to change the number of subsets after each iteration, and change the way that the projections are divided into subsets, our implementation does not currently handle these cases.

This greatly decreases the number of iterations required for reconstruction. As seen in the results section, we work with both OSEM and EM for different purposes. When we run EM, our goal is to closely examine convergence rates. OSEM is much faster than EM, so we use it for speed timings.

3.5. Attenuation Correction

Attenuation correction is the modeling of the probability of an emitted photon from a voxel attenuating through material and reaching the gamma camera. For example, it is much more likely for bone to attenuate a photon than flesh. To model this, we require information regarding the location of materials in the volume. We combine our knowledge of the attenuation coefficients at each voxel to create a volume called an attenuation map. The best attenuation maps come from CT scans which measure the transmission properties of the subject.

Updating Equation 2 to include attenuation correction requires a change in both the projection and backprojection stages.

$$A_j = e^{-(\mu_j l)} \tag{7}$$

where μ_j is the attenuation coefficient of voxel v_j and l is the length of a voxel, yielding A_j, the attenuation factor of voxel v_j. Let ray r_i extending from pixel p_i have sample points $S_1..S_n$ in order from p_i to v_j, as seen in Fig. 3. Now the projection Equation 3 is updated to be:

$$\beta_i = \sum_{j=1}^{J} v_j (\Pi_{m=1}^{n} A_{S_m}) w_{ij} \tag{8}$$

Likewise, the backprojection equation, Eq. 5 becomes:

$$\Psi_j = \sum_{i=1}^{I} \Omega_i (\Pi_{m=1}^{n} A_{S_m}) w_{ij} \tag{9}$$

Consider a small example where a detector p_i is directly in line with 3 voxels, v_1, v_2 and v_3, with v_1 being the closest to the detector and v_3 the farthest, as seen in Fig. 5. Each of these voxels have corresponding attenuation factors A_1, A_2, A_3 and weights w_{i1}, w_{i2}, w_{i3} are all 1, since these voxels are directly in line with p_i. Without attenuation correction the projection step simulates transmission tomography, where $\beta_{no-AC} = v_1 + v_2 + v_3$. When accounting for attenuation, $\beta_{AC} = v_1 A_1 + v_2 A_2 A_1 + v_3 A_3 A_2 A_1$. As a result attenuation corrected projections are effected more by material that is close to the camera.

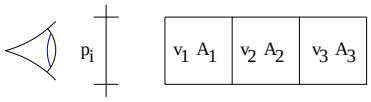


Figure 5: *Example setup with pixel p_i directly in line with three voxels, each with a value and attenuation factor.*

In the backprojection step, a correction value Ω_i is distributed over the voxels of the correction volume Ψ_j weighted by w_{ij}. In the three voxel example, without attenuation correction we get $\Psi_1 = \Psi_2 = \Psi_3 = \Omega_i$. With attenuation correction the backprojection equations are $\Psi_1 = \Omega_i A_1$, $\Psi_2 = \Omega_i A_1 A_2$ and $\Psi_3 = \Omega_i A_1 A_2 A_3$. Our implementation

does not do this step. We have chosen to only use the attenuation map in the projection step. The motivation for doing this is to reduce the computational burden while achieving a similar convergence. Unmatched projector / backprojector pairs have been used in image reconstruction before, sometimes yielding an increase in the rate of convergence [6]. See Zeng and Gullberg [14] for a detailed unmatched projector / backprojector pairs comparison.

4. Implementation

4.1. Projection

The projection step in the EM algorithm is a classic problem in scientific visualization. Taking a volume and projecting it to a 2D image can be done via a number of methods including ray casting [9], 3D texture slicing [1], splatting [13], and shear warp [5]. Since we are trying to use hardware to accelerate reconstruction we will focus on methods that utilize the graphics cards texture mapping capabilities. In particular, we will focus on 3D texture slicing for its speed and other properties.

One advantage of using a 3D texture slicer is that the volume data is transferred to the graphics card once, and then sliced as many times as needed. As data transfer between the graphics card and main memory is relatively slow, texture slicing becomes advantageous. In 3D texture slicing, we place the estimate volume in texture memory on the graphics card, and then take a series of 2D slices of the texture and accumulate or blend the slices in the frame buffer. This pipeline resides almost entirely in hardware, leaving only the calculation of the slicing coordinates to be done in software.

A major disadvantage of using graphics hardware in the reconstruction process is the lack of precision of the hardware. The standard frame buffer bit depth for PC based graphics cards is 8-bits for each of the 4 channels, RGBα.

The volume we are reconstructing is kept as a float and is non negative. When the volume is put into texture memory, it gets stored as an 8-bit number, resulting in a loss of precision. Although we have a method that accounts for numbers with an integer component larger than 255 (see Section 4.5), the 8-bit maximum, we have not prevented loss of precision in the fractional part. We simply round each value to the nearest integer.

3D texture slicing is used for our projection step because it is doing exactly what would be done in software. In the software version, when the projection angle is not axis aligned, voxels are bilinearly interpolated, and then added along the ray. The texture slicing does tri-linear interpolation in hardware, then composites the whole slice into the frame buffer. Although the order of operations is different, the same work is done. When a projection is completed, the frame buffer image is read to main memory.

4.1.1. Projection with Attenuation Correction

The projection stage requires extra steps when accounting for attenuation correction. In software this is done by computing an angular dependent attenuation volume for each angle in the projection set. This is done to optimize the code for speed, as the attenuation coefficient multiplications are done in a pre-computation step. In the MCAT dataset, this requires 64 volumes of size 128^3 with each voxel being a 32-bit floating point number, implying that this approach trades memory use for reduction in computation. With a separate attenuation volume for each projection angle, the software now only needs to multiply each voxel in the estimate volume to the corresponding attenuation volume voxel, and add the results together. We could take the same approach for our hardware implementation, but the time required for the amount of data transferred between main memory and the graphics card would negate much of the performance gain from using the graphics hardware.

Instead we will utilize the blending operations available, setting the blending state to yield the same calculation. Besides needing a blending equation that implements Equation 8, we also need the ability to read the values in the frame buffer into main memory where we can accumulate without loss of precision. To solve this problem we slice the 3D texture from front to back using a four step procedure: slice, blend, read and clear.

We keep the product of the A_j's encountered along the pixel's ray in the α channel of the Frame Buffer (FB_α). When a new slice is taken, its color component can be multiplied by its own A_j and by FB_α and then added to the frame buffer color channel FB_{color}. When we need to read out the frame buffer to main memory, we only clear the FB_{color} and leave the FB_α intact.

This procedure is accomplished by first setting the 3D texture color channel to the product of the estimate volume and the attenuation volume, $v_j A_j$. The textures α channel is set to be the attenuation volume A. Computing the 3D texture and transferring the texture to the graphics card occurs once for each set of projections. When blending a slice into the frame buffer, we need different blending equations for the α channel and the color channel. The slice's α channel, α_{slice} needs to be multiplied by the value in FB_α, which keeps the product of the A_j's encountered. The slice's color channel C_{slice} needs to be multiplied by FB_α and then added to FB_{color}. This way, FB_{color} can be read and cleared as often as needed, but does not need to be read and cleared after every single slice that is blended into the frame buffer.

This rendering procedure is described in the pseudocode of Figure 6. In the pseudocode, the variable "limit" is the input parameter and is the maximum number of slices that can be accumulated in the frame buffer without overflow.

```
Slice3DTexture(limit)
  projection[] = 0;
  glBlendColor(1,1,1,0);
  glBlendFunc(GL_DST_ALPHA,
              GL_CONSTANT_COLOR);
  glClear(frame_buffer);
  for (i = 1; i <= total_slices; i++)
  {
    if (i % limit == 0)
    {
      projection[] += glReadPixels(screen);
      glBlendColor(0,0,0,1);
      glBlendFunc(GL_ZERO,
                  GL_CONSTANT_COLOR);
      draw_full_screen_polygon();
      glBlendColor(1,1,1,0);
      glBlendFunc(GL_DST_ALPHA,
                  GL_CONSTANT_COLOR);
    }
    texture_2d = slice_3Dtexture(i);
    render_slice(texture_2d);
  }
  projection[] += glReadPixels(screen);
return;
```

Figure 6: *3D texture slicing pseudocode with attenuation correction.*

4.2. Correction Image

Once we have our computed projections β_i which are integers, we calculate the correction images. The original projections, p_i are also integers as they are simply detector counts. The correction image calculation, Equation 4, gives us Ω_i. The problem with doing this in graphics hardware is the range of Ω_i, which is a non negative real number, $[0 \ldots \infty)$. Accordingly, this step is done in software.

With the large data range of Ω_i, backprojecting becomes tricky. First we must scale our data into an integer range, backproject it, and then scale it back. Before scaling Ω_i, we clamp the correction image data to a more reasonably sized range. The histogram for the correction images should be tightly centered around 1.0, especially as we iterate and become close to a solution. This can be seen in the histogram, Fig. 7, where in the first iteration the values are not centered around 1.0, but after ten iterations they are. As the histogram indicates few values greater than 2, we have chosen to clamp Ω_i to the range $[0 \ldots 2]$, leaving us with Ω_i'. Now that our Ω_i' are in the range $[0 \ldots 2]$, scaling the correction images into 8-bit integers is done with a scaling factor of $\delta = 127$. The scaling can be written as $\Omega_i^\delta = \Omega_i' \delta$.

The clamping of the data range has very little effect on the convergence properties, as the theoretical range maximum rarely occurs. What we limit is the algorithms potential to increase the intensity of the voxels to twice the previous intensity per iteration. With this limitation, assuming a limit of 12-bit data, it could take up to 12 iterations for the maximum intensity to be reached. Our volume data is actually much smaller than the projection data in values, as a projection pixel is the sum of the voxels that the pixel beam intersects.

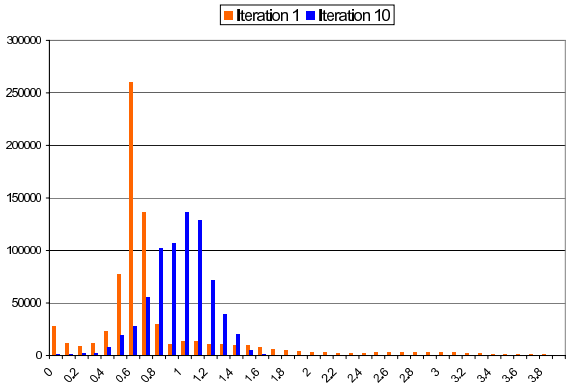


Figure 7: *Histogram of correction image. As we iterate, the histogram becomes more centered around 1.0.*

In EM reconstruction it is common to do many iterations, often upwards of 80. Since the number of iterations required to correct for the range clamping is an order of magnitude less than the number of iterations run, this range clamping does not effect the overall convergence.

4.3. Backprojection

The backprojection step is done by taking one row of each scaled correction image and smearing it across the frame buffer, resulting in a 2D slice of our volume. This is then done for each row, constructing a correction volume slice by slice. Our backprojection implementation slightly differs from Cabral's [1] as ours is implemented on PC graphics cards that do not have an accumulation buffer. We accumulate in the frame buffer via blending with the OpenGL blend operator blendFunc(GL_ONE,GL_ONE). Unfortunately the frame buffer only has 8-bits per channel so we need to be careful not to allow our frame buffer to overflow. The major bottleneck in this process becomes the data transfer between the graphics card and main memory, where we can accumulate. To minimize the data going to the card, we use 1D textures, and use the graphics hardware to stretch the 1D texture over the 2D frame buffer. A discussion of minimizing frame buffer reads into main memory is presented in Section 4.5. After the backprojection step is complete, we have a correction volume Ψ^δ that is still scaled up by δ.

4.4. Update Volume

Now that we have our correction volume Ψ^δ as integers, we need to scale it back to get Ψ as floating point numbers. The modified equation to get our new estimate volume is now:

$$v_j^{(k+1)} = \frac{v_j^{(k)} \Psi_j^\delta}{\delta \lambda} \tag{10}$$

We do this step without use of the graphics hardware because of the precision required. The estimate volume voxels v_j are 32-bit non-negative floating point numbers, as are Ψ_j.

4.5. Frame Buffer Extension

In practice, we need more bits for the *RGB*α channels than the standard 8 bits per channel available on most PC graphics cards. Our data often uses more than 8 bits per voxel, so no single slice fits in the frame buffer. Second, we would like to be able to accumulate in the frame buffer without the possibility of overflow. The more we can accumulate in the frame buffer, the less we need to read the frame buffer to main memory. As graphics cards have not been optimized for data transfer from the card to main memory, the frame buffer reads are time consuming. Mueller [8] also needed to extend the frame buffer, using 16-bit CT data on SGI machines with 12 bit frame buffer channels.

For this example, we will assume that the input data is 16-bit, even though SPECT data is usually 12-bit or less. We demonstrate this technique with a simple example of adding two numbers, 1243 plus 869 equals 2112.

By breaking up our 16-bit value into four pieces that are four bits each, we can utilize all four channels in the frame buffer. This can be seen in the example shown in Fig. 8 where two numbers are divided up into four pieces and added together.

α	B	G	R
0000	0100	1101	1011
0000	0011	0110	0101
0000 0000	0000 0111	0001 0011	0001 0000

Figure 8: *Virtual frame buffer extension. This example shows the decimal numbers 1243 and 869 being added using all four color channels.*

Now we can add 17 values into the frame buffer without any overflow. When we read the frame buffer back into main memory we must shift our results to get the original value back. This is shown in Fig. 9 which starts with the result of the adding stage shown in Fig. 8.

α	0000	0000			
B		0000	0111		
G			0001	0011	
R				0001	0000
	0000	0000	1000	0100	0000

Figure 9: *Combining the four channels to get the result. In this example, the result in decimal is 2112*

We achieve only a 4/17ths reduction in data transfer, as we only read every 17 additions, but when we read, we have data in all 4 channels instead of just 1. However the graphics hardware used passes all four channels simultaneously. This means that we get a full 1/17th reduction in time taken for the data transfer.

As mentioned earlier, SPECT volume data is often significantly less than 16 bit, so we can distribute data bits over multiple channels as needed. We do not actually need to know the final number of bits needed for the volume data, as we can check after each iteration what the volume's maximum value is, and dynamically decide the bit distribution, and the resulting number of frame buffer reads required. We propose an improvement in Section 5.1.1 to the case where we split the data unevenly across the color channels over previous methods.

4.6. Other Applications

Projecting an emission volume is a problem encountered by Max [7]. Part of his work showed volume rendering of clouds and gases which have emission properties. A problem that we have in common with Max is that the range of the accumulation far exceeds the range of the graphics hardware. To deal with this, Max uses a very small constant scaling factor so that the accumulation does not exceed the available hardware range. This has the adverse result of clouds that are "too tenuous at the edges". We propose an accurate solution to this problem in Section 4.1.1.

5. Results

We use Tsui et al. [11]'s 3D mathematical cardiac-torso (MCAT) phantom, which models the anatomical structures of the thorax. The MCAT phantom is of size 128^3 and has 64 projections, with 128^2 pixels each.

To measure the accuracy of our reconstruction we compare the reconstructed volumes to the true MCAT phantom volume, T. Our primary measure of accuracy is the average Relative Error (RE) to T, defined as follows:

$$RE = \frac{1}{J}\sum_{j=1}^{J}\frac{|T_j - v_j|}{T_j} \tag{11}$$

We also use the Peak Signal to Noise Ratio (PSNR) which is based on the Root Mean Square Error (RMSE) metric.

$$RMSE = \sqrt{\frac{1}{J}\sum_{j=1}^{J}(T_j - v_j)^2} \tag{12}$$

$$PSNR = 20log_{10}\left(\frac{max(T)}{RMSE}\right) \tag{13}$$

PC Timings were obtained on an AMD MP 2000+, with 2GB of RAM, and an NVidia GeForce 4 Ti 4600, running

Linux and Windows. The software version timings were performed under Linux, and the hardware timings under Windows. Both the software and hardware versions run under both operating systems, and the timings are reported on the operating system that optimal performance was achieved on. Note that if the system had 1GB or less RAM, it would not be able to run the software version as quickly, as memory paging dramatically slows down the PC.

In the reconstructions from both the software and hardware methods, the data occupies a small range. Without attenuation correction, the reconstructions have the range $[0\ldots16]$, while attenuation corrected reconstructions are in the range $[0\ldots55]$. Although radiologists are trained for viewing gray scaled images, we found it useful to examine colored images which can emphasize small changes. Both color and gray scaled images (scaled up by 4.5) can be seen in Fig. 13 and Fig. 14 respectively.

5.1. Expectation Maximization

In EM reconstruction, we have surpassed the speed of the software implementation by more than eight times, without any loss of accuracy. The error analysis and timings can be seen in Table 1 and the reconstructed images are in Fig. 11.

No Attenuation	*EM*		*OSEM*	
Correction	SW	HW	SW	HW
Time (sec)	851	98	130	30.8
Speed up factor	8.7		4.2	
RAM (MB)	250	66	250	78
Avg iteration (sec)	10.6	1.2	13.0	3.1
Relative error	.256	.260	.256	.259
PSNR	17.6	17.6	17.6	17.6

Table 1: *Overall comparison table between software (SW) and hardware (HW) methods without attenuation for 80 EM iterations and 10 OSEM iterations.*

The projection step does not use the frame buffer extension with the MCAT dataset, as the volume values are small enough to allow accumulation of multiple slices. The only error induced here is from rounding the 32-bit floating point volume voxels into 8-bit integers to fit into texture memory. Our implementation of 3D texture slicing achieves 82.3 frames per second (FPS) while accumulating 17 slices in the frame buffer per read without use of the frame buffer extension. Our implementation changes the number of slices it can accumulate in the frame buffer according to the values in the volume, making sure that no overflow can occur. After each iteration is completed, the maximum value in the volume is found and used to determine the number slices per frame buffer read used in the projection step. This means that the first few iterations are very fast, but the projection step slows down as it converges. In the MCAT phantom, the convergence point has 17 slices being accumulated per read. The number of reads per projection has an inverse linear relationship with the time required for a projection.

In the backprojection step, the 8-bit scaled correction data Ω^{δ} is broken into four 2-bit chunks and placed in texture memory in the RGBα channels. Backprojecting the 64 correction images on a slice-by-slice basis, reconstructing a correction volume Ψ^{δ} of size 128^3 is done in 0.26 seconds.

5.1.1. Error Analysis

When using texture mapping hardware, the result of any calculation is rounded. If the texture contains our data split across multiple channels, the round off error in a channel containing upper bits will result in a large error when the value is combined.

The MCAT dataset and the real datasets that we have obtained all have small enough values that the frame buffer extension is not needed in the projection step. However, the frame buffer extension is used in the backprojection step. To quantify the error introduced by this technique, the following experiment was conducted. We backprojected 8 bit correction data without the extension, and read out the frame buffer after each correction image was backprojected. This was then used as the true backprojection that the frame buffer extension based backprojection techniques were compared to. The results of the experiment are listed in Table 2.

Input Bits in αBGR	*FB reads*	*Time*	*RE*	*PSNR*
0008	64	4.2	-	-
0053	8	.93	.0037	63.0
0035	8	.92	.0089	54.9
0044	4	.60	.0057	58.9
2222	1	.26	.0159	49.9

Table 2: *Error comparison on frame buffer extension techniques against no extension (top line) in backprojection. The α channels holds the highest bits while the Red channels gets the lowest.*

The results of this test appear to indicate that the error is relatively small. Although the error is small per iteration, over many iterations, it accumulates, resulting in noisy images (see Fig. 11).

5.2. Ordered Subset Expectation Maximization

In our OSEM experiment, we used 8 subsets of size 8, with numeric analysis show in Table 1, and visual comparison available in Fig. 12. The more frequent transfer of data required in OSEM between the graphics card and main memory has significantly cut down the performance gain of hard-

ware over software. However, our graphics hardware implementation is still almost 4 times faster than software. There is also a slight decrease in error in our OSEM implementation over our EM, as the error from the frame buffer extension is minimized as mentioned in the following section.

5.2.1. Ordered Subsets in Frame Buffer Extension

Using ordered subsets for the reconstruction does not effect the projection step, except for the number of projections made at a time. The backprojection step however can be altered, as it uses the virtual frame buffer extension based on the number of correction images that are being backprojected. In the MCAT dataset, we have 64 projections. To read out the frame buffer only once to main memory, 2-bits are put in each of the RGBα channels. Remember that the correction images are scaled, so we know that all 8-bits are used in the value that are being backprojected. When using Ordered Subsets, the size of the subsets determines the bit allocations. For example, if our subset size is 16, then we can use two color channels, putting 4 bits in each, and we only need to read out once. With a subset size of 8, we can put 5-bits in the upper channel (Green), and 3-bits in the lower (Red), and accumulate all 8 images in the frame buffer while minimizing the error. This is a different approach than Mueller's [8] frame buffer extension. By placing the upper 5-bits in the green channel and the lower 3-bits in the red channel, we reduce the size of any round off error caused by the hardware. Previously, uneven division of bits has had fewer upper bits and more lower bits. Note the improvement on error in Table 2 (second vs. third line) from this change. Mueller's approach has an advantage if the values are often small enough that they do not get divided. We know the average size of our scaled values, and know that we do not have many values small enough for this division style to be advantageous.

5.3. Attenuation Correction

The hardware based reconstruction algorithm only uses the attenuation map in the projection step. The software uses the attenuation map in both the projection and the backprojection step. This results in higher error in the hardware reconstruction, and means that a direct comparison is unfair. In terms of speed, the hardware method does a full OSEM reconstruction in less time than the software takes just for the pre-computation step. Using the hardware reconstructed volume as an initial estimate for the software reconstruction becomes a viable option, that can cut down on the number of software iterations necessary. Looking at Table 3 we see that the hardware method was over 5 times faster than software for EM and over 5 times faster using OSEM. Also note the huge memory consumption for the software version. This implies that the software version has a scalability problem, as either increasing the dimensions of the reconstruction, or the number of projections may require more RAM than reasonable even on high end PC's.

With Attenuation Correction	*EM*		*OSEM*	
	SW	HW	SW	HW
Time (sec)	897	174.6	205	47.8
Pre-comp (sec)	55	0	55	0
Speed up factor	5.5		5.4	
RAM (MB)	1330	66	1330	78
Avg iteration (sec)	11.2	2.2	20.5	4.8
Relative error	.109	.146	.109	.131
PSNR	29.3	28.4	29.3	29.0

Table 3: *Comparing software (SW) and hardware (HW) methods with attenuation for 80 EM iterations and 10 OSEM iterations.*

5.4. Convergence

As seen in Fig. 10, the convergence of EM without attenuation correction is the same for both the hardware and the software version. The hardware version does not seem to be effected by the correction image range clamping mentioned in Section 4.2. However, the range clamping might be the reason why the convergence of the hardware version is not the same as software when using the attenuation map. The slope of the error curve is not as steep in the first few iterations for hardware, but seems to be caught up by 15 iterations. Not having a backprojection step that utilizes the attenuation map is why there is still a small difference between the hardware and software methods.

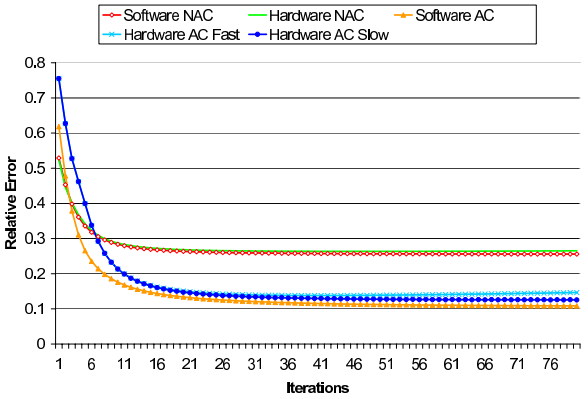


Figure 10: *Relative Error vs Iterations for EM with attenuation correction (AC) and without (NAC) for both software and hardware.*

A slow hardware method is shown here and in Figures 11, 13 and 14. This version of the hardware method does not use the frame buffer extension from Section 4.5 in the backprojection step instead we read the frame buffer after every projection. As a result, it has lower error, produces smoother images but does take longer to reconstruct.

We show these images to demonstrate the potential that the hardware accelerated methods have.

6. Conclusions

The results of our work indicate that graphics hardware can be used for iterative reconstruction via the EM algorithm, with the same level of accuracy and a significant increase in speed. We achieve a large increase in speed with minimal error through our new bit splitting method. The error analysis on bit splitting shows our method has a significant reduction in error over previous methods. We incorporated the ordered subset modification for the EM algorithm to achieve an increase in convergence rate. We also incorporated the use of attenuation maps to properly compute the inverse of the attenuated radon transform. We demonstrated how to accurately implement the hardware accelerated rendering of volumes with emission properties.

7. Future Work

In the near future, graphics hardware will become available with a higher resolution pipeline. Until now the standard graphics card pipeline has been 8-bits per channel; however, two leading graphics processor manufacturers, ATI and NVidia, are now developing floating point pipelines with up to 32-bits per channel. With this new technology the errors added in the methods we presented can be removed from the reconstruction process, and the reconstructions should be indistinguishable from those constructed in software. The new technology will permit a speed up via removal of the current data transfer bottleneck. Without the need to accumulate in main memory, the data transfer will be lowered, and the hardware-based reconstruction will be faster than that presented here.

Our future work includes reconstructing clinical SPECT datasets. We hope to obtain data of larger dimensions than 128^3 and data with more than 64 projections. Since it is uncommon for SPECT data to be higher resolution than 128^3, we plan on obtaining data from other modalities. We will also implement attenuation correction in the backprojection step in software and hardware to do a proper error analysis on reconstruction with attenuation correction. We plan on hardware reconstruction using a full 3D version of the EM algorithm. The software version of this algorithm takes over 35 minutes per iteration on the PC system described earlier.

8. Acknowledgments

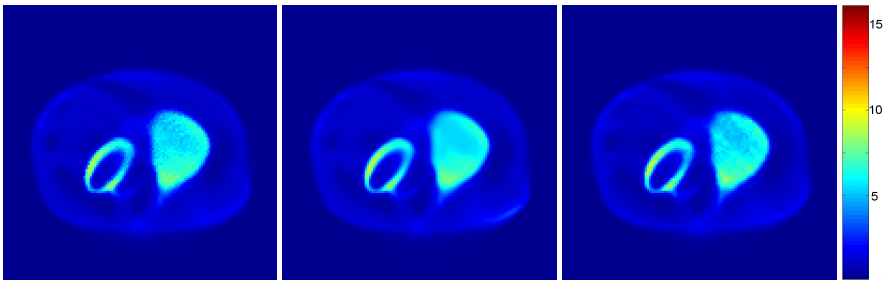
The authors would like to thank Klaus Mueller for useful advice, Anna Celler, and Stephan Blinder for medical explanations and datasets, and Troy Farncombe for guidance, encouragement and datasets! Special thanks to Steve Kilthau for fruitful discussions, and all the members of GrUVi lab.

The authors would also like to acknowledge the National Science and Engineering Council of Canada (NSERC), the Advanced Systems Institute (ASI) of British Columbia and the Canada Foundation of Innovation (CFI) who partially funded this research.

References

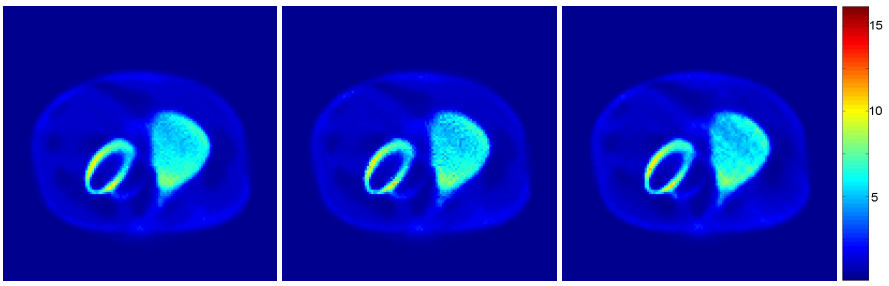
1. B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Symposium on Volume Visualization*, pages 91–98, 1994.
2. A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. In *J. Royal Stat. Soc.*, volume Ser.B, 39, pages 1–38, 1977.
3. H. M. Hudson and R. Larkin. Accelerated image reconstruction using odered subsets of projection data. In *IEEE Transactions on Medical Imaging*, pages 100–108, 1994.
4. W. Jones, L. Byars, and M. Casey. Positron emission tomographic images and expectation maximization: A vlsi architexture for multiple iterations per second. In *IEEE Transactions on Nuclear Science*, volume 35, No. 1, pages 620–624, 1988.
5. P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Computer Graphics (Proceedings of SIGGRAPH 1994)*, pages 451–458, 1994.
6. D. Lalush and B. Tsui. Improving convergence of iterative filtered backprojection algorithm. In *Medical Physics*, volume 21, pages 1283–1286, 1994.
7. N. Max. Optical models for direct volume rendering. In *IEEE Transactions on Visualization and Computer Graphics*, volume 1, No. 2, pages 99–108, 1995.
8. K. Mueller and R. Yagel. Rapid 3-d cone-beam reconstruction with the simultaneous algebraic reconstruction technique (sart) using 2-d texture mapping hardware. In *IEEE Transactions on Medical Imaging*, volume 19, No. 12, pages 1227–1237, 2000.
9. P. Sabella. A rendering algorithm for visualizing 3d scalar fields. In *Computer Graphics (Proceedings of SIGGRAPH 1988)*, volume 22 (4), pages 51–58, 1988.
10. L. Shepp and Y. Vardi. Maximum likelihood reconstruction for emission tomography. In *IEEE Transactions on Medical Imaging*, volume MI-2, pages 113–122, 1982.
11. B. Tsui, X. Zhao, G. Gregoriou, D. Lalush, E. Frey, R. Johnston, and W. McCartney. Quantitative cardiac spect reconstruction with reduced image degradation due to patient anatomy. In *IEEE Transactions on Nuclear Science*, volume 41, No. 6, pages 2838–2844, 1994.
12. S. Vollmar, C. Michel, J. Treffert, D. Newport, M. Casey, C. Knöss, X. L. K. Weinhard, M. Defrise, and W. Heiss. Heinzelcluster: accelerated reconstruction for fore and osem3d. In *IEEE Transactions on Medical Imaging*, volume M9C, No. 14, pages 1766–1770, 2001.
13. L. Westover. Footprint evaluation for volume rendering. In *Proceedings of SIGGRAPH 1990*, pages 367–376. ACM, 1990.

14. G. Zeng and G. Gullberg. Unmatched projector/backprojector pairs in an iterative reconstruction algorithm. In *IEEE Transactions on Medical Imaging*, volume 19, No. 5, pages 548–555, 2000.



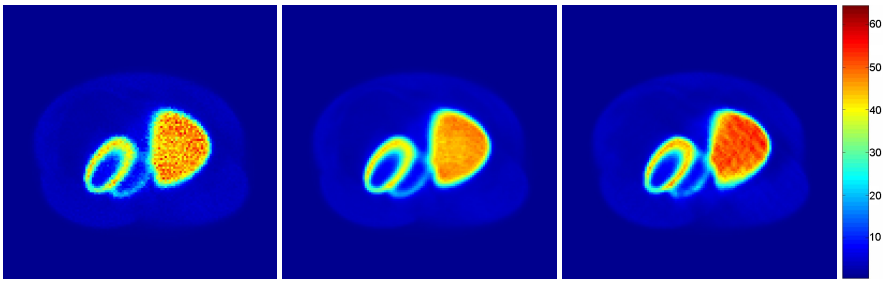
(a) (b) (c)

Figure 11: *EM, 25 iterations: (a) hardware with Frame Buffer Extension (FBE), (b) hardware without FBE,(c) software.*



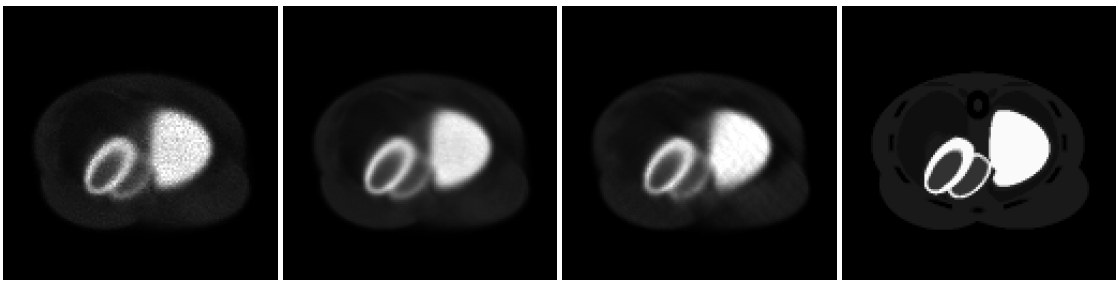
(a) (b) (c)

Figure 12: *OSEM, 5 iterations: (a) hardware using 5-3 split FBE, (b) hardware using 2-2-2-2 split FBE, (c) software.*



(a) (b) (c)

Figure 13: *Attenuation Correction EM, 25 iterations: (a) hardware using FBE, (b) hardware without FBE, (c) software*



(a) (b) (c) (d)

Figure 14: *Attenuation Corrected EM, 25 iterations no classification function: (a) hardware using FBE, (b) hardware without FBE, (c) software, (d) Phantom*