Implementation of a backprojection algorithm on CELL

Mario Koerner

March 17, 2006

1 Introduction

X-ray imaging is one of the most important imaging technologies in medical applications. It allows to look into the human body by exhibiting it to radiation that is emitted from an X-ray source and measuring intensities with a detector on the other side of the object. When using a so-called C-arm device for image acquisition, source and detector are placed on an arm that allows the rotation of both around the object. If enough images are collected, a 3D volume can be reconstructed from these 2D projections.

Reconstruction from cone beam projections that are measured on a circular trajectory can be done with the Feldkamp algorithm which is used in many practical applications due to its simplicity.

2 Practical implementation of 3D backprojection

The Feldkamp algorithm requires a weighting of the measured projection images and filtering with a special filtering kernel. These steps are assumed to be completed, before the backprojection step starts.

The backprojection basically iterates over all projections P_j and all voxels v_i of the volume to be reconstructed and performs the following operations:

- Compute the coordinates p_{ij} of voxel v_i in projection P_j using the projection geometry from the measurement.
- Get the projection value at this position (nearest neighbour, bilinear interpolation)
- Accumulate the weighted value to the reconstruction volume

2.1 Coordinate computation using projection matrices

The relationship between voxels in the reconstruction volume and pixels in the projection images can be described as a perspective projection with the X-ray source as the projection center and the distance from the source to the center of rotation as focal length.

To find the projection matrix, we first rotate the fan around the z-axis by an angle of $-\beta$ and shift it to the center of the object coordinate system. This can be done by applying the matrix R to the voxel position in homogeneous coordinates:

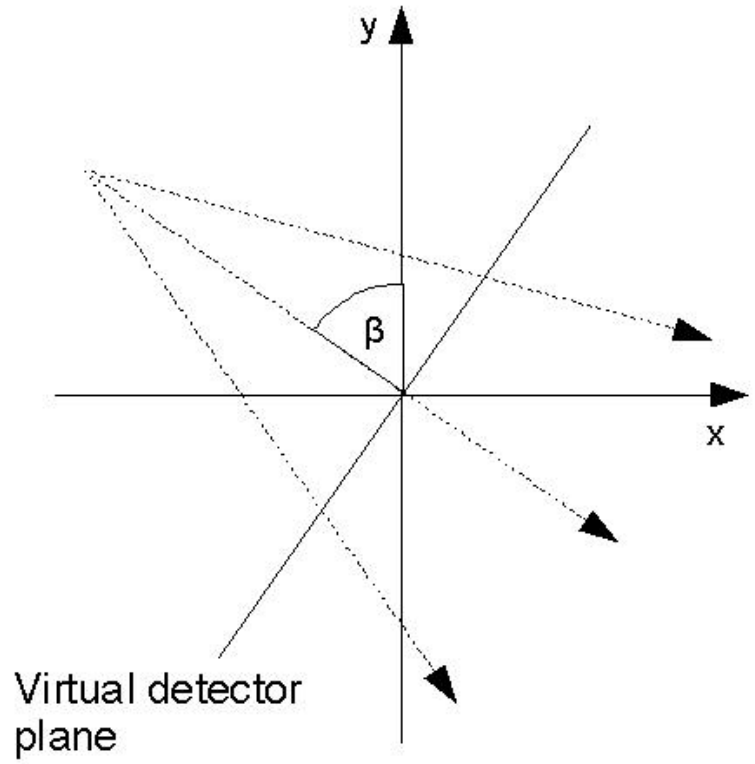

Figure 1: Fan rotation in the x-y-plane

$$R = \begin{pmatrix} \cos(-\beta) & -\sin(-\beta) & 0 & 0 \\ \sin(-\beta) & \cos(-\beta) & 0 & -D \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{1}$$

Now we can apply the projection matrix P:

$$P = \begin{pmatrix} D & 0 & 0 & 0 \\ 0 & 0 & D & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix} \tag{2}$$

Which results in

$$PR = \begin{pmatrix} D\cos(\beta) & D\sin(\beta) & 0 & 0 \\ 0 & 0 & D & 0 \\ \sin(\beta) & -\cos(\beta) & 0 & D \end{pmatrix} \tag{3}$$

Applying this matrix to a voxel $(x, y, z, 1)^T$ gives the formulas for the corresponding pixel coordinates from the Feldkamp algorithm:

$$a(x, y, \beta) = D\frac{x\cos\beta + y\sin\beta}{D + x\sin\beta - y\cos\beta} \tag{4}$$

$$b(x, y, z, \beta) = \frac{zD}{D + x\sin\beta - y\cos\beta} \tag{5}$$

Because the mapping between the voxel coordinates and the pixel coordinates in a specific projection is linear (in homogeneous coordinates), the columns of the projection matrix can be used as increments, when moving in x-, y-, or z-direction through the volume.

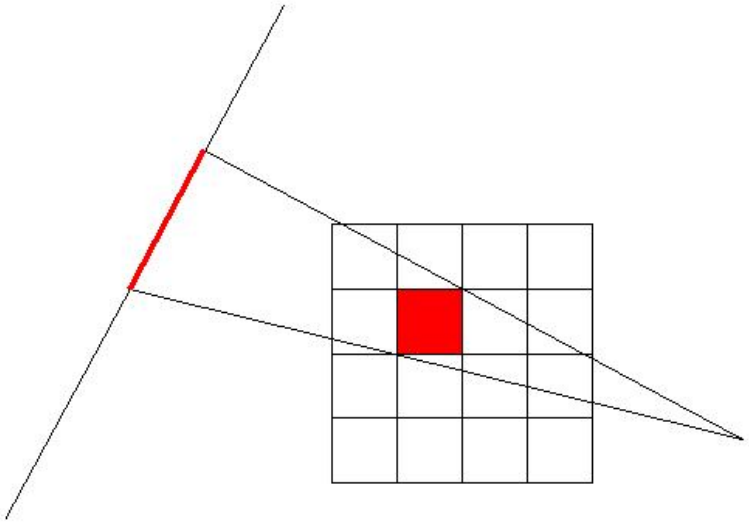

Figure 2: Dependency between input data (the projections) and output data (the reconstruction volume)

2.2 Computation of the backprojection weights

In the Feldkamp algorithm, a voxel increment is weighted with a factor $1/U^2$, where

$$U(x, y, \beta) = \frac{D + x \sin\beta - y \cos\beta}{D} = \frac{\sin\beta}{D} x - \frac{\cos\beta}{D} y + 1 \tag{6}$$

So the weight is also linear in the voxel coordinates. Furthermore, it is identical to the third homogeneous coordinate of the pixel, if the projection matrix in equation (3) is scaled by a factor of $1/D$.

Because the reciprocal of the third homogeneous coordinate has to be computed anyway to get the cartesian coordinates of the pixel, the result can be reused. This saves one division operation for each voxel of the reconstruction volume.

3 Porting backprojection to CELL

The backprojection algorithm has to deal with a huge amount of data. E.g. a volume of size $(512)^3$ stored with 32 bit floating point numbers requires 512 MB of memory; 500 projections with $(1024)^2$ pixels need 2 GB.

In order to perform the backprojection on the CELL SPUs, data must be partitioned to fit into the 256K local storage. Because data is sampled in the output space (meaning we go over all voxels and project them into the projection images), it is intuitive to partition the volume into small subcubes and partition the input data (the projections) according to the data dependencies implied by the projection geometry. See figure 2 for a sketch.

The piece of the projection data required to reconstruct a special subvolume is computed by projecting all 8 corners and finding the bounding box around them. Due to restrictions of the DMA controllers, the lines must be aligned to a multiple of 4 pixels (16 bytes) and must have a length of a multiple of 4 pixels.

A basic reconstruction task performed by a SPU then consists of a subvolume and a projection. It receives the task description through a structure in main memory, loads the required data, performs the actual backprojection and writes the results back.

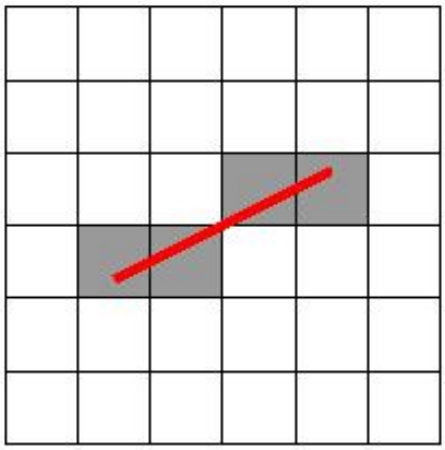

Figure 3: Pixels in the projection image corresponding to 4 subsequent voxels in the reconstruction volume)

The current implementation is PPU centric, so the PPU is responsible for scheduling the tasks to the SPUs. Furthermore, the PPU performs all I/O operations to load the data into main memory and computes the projection shadows.

4 Optimizing the backprojection on CELL

4.1 Optimizations on the SPU

The instruction set for the SPUs mainly consists of vector instructions. This means that each operation is always applied to 4 data elements in parallel (SIMD). This also means, that it is very inefficient to execute scalar code, because the functional units are not used in an optimal way. Furthermore, writing a scalar to local storage requires reading the old vector data, merging the scalar into it and writing it back to memory. So code optimization for the SPUs should start with rewriting the scalar code to use vector instructions.

The inner loop of the backprojection only allows limited vectorization:

- The computation of the coordinates for the pixel corresponding to a specific voxel can be done in parallel for 4 voxels, because all voxels may be handled independently.
- The memory access for loading the value of the selected pixel cannot be vectorized, because the pixels will not lie linearly in memory. This is depicted in figure 3.
- Computation of the weight and accumulating the projection values onto the reconstruction volume can be done with vector instructions, if the projection data is copied to a vector before.

One difficulty is the lack of a division with full 32 bit floating point accuracy on the SPUs. The hardware just supports an estimate for the reciprocal of a floating point number that is 12 bit accurate. Fortunately the SDK library provides two versions of a division function that handle the problem.

The first one is marked to be IEEE compliant. This version gives full 32 bit float accuracy, rounds the result consistently with the rounding mode of the processor (truncate towards zero) and even handles exceptional cases as underflows and overflows.

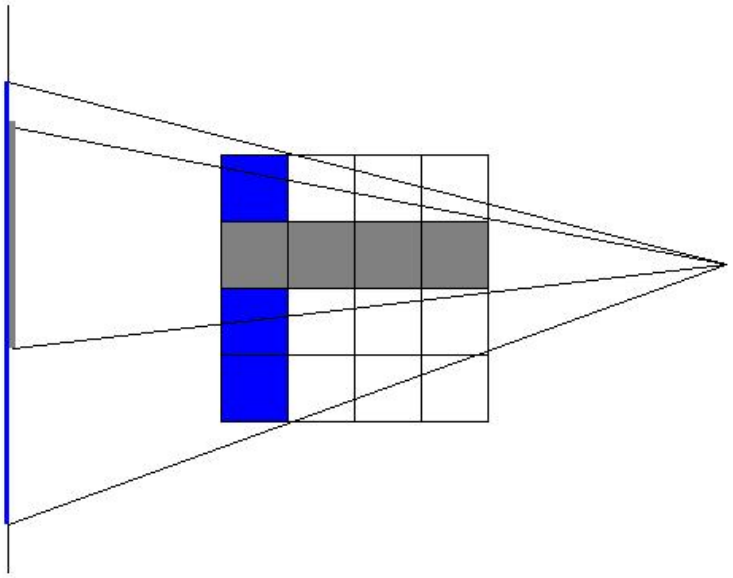

Figure 4: Relationship between subvolume shape and projection shadow size

The second version just adds a Newton iteration to the hardware estimate to achieve 32 bit float accuracy. This version is faster than the IEEE compliant version and should be sufficient for the purposes of the backprojection (a formal verification has still to be done).

4.2 Optimizations of the data transfer

The DMA units on the SPEs allow overlapping of computation and communication. So the double buffering technique can be used to hide latencies when accessing main memory. Double Buffering can be used easily for reading the next piece of projection data.

For the volume data, things are a little more complicated, because the data has to be written back to main memory after the computation. Using a third buffer would be a bad solution because of the memory shortage on the SPUs. Two buffers are sufficient, if the fenced version of the 'get' command is used for reading the next subcube. The fence makes sure, that the 'put' command for the previous subvolume is completed before the 'get' command for the next one starts.

Transferring small pieces of memory to the SPUs results in very poor performance. The memory bandwidth is only used optimally, if chunks with a size of more than 128 bytes are used. This implies that the size of the subvolumes must not be chosen too small in x direction, if the volume is stored line-by-line, plane-by-plane in main memory.

4.3 Optimizing the scheduling of subvolumes

Because different subvolume shapes result in projection shadows of different size (see figure 4), the amount of data to be transferred varies with the chosen shape and the considered projection.

A good scheduling strategy is required to make sure that the projection shadows are small enough to fit into the local storage of the SPUs and the overall amount of data to be transferred is minimized.

5 Literature references

[CBEA] Cell Broadband Engine Architecture. IBM Corporation, 2005

[CBETut] Cell Broadband Engine Programming Tutorial. IBM Corporation, 2005

[Kak] A. C. Kak and Malcolm Slaney, Principles of Computerized Tomographic Imaging. Society of Industrial and Applied Mathematics, 2001

[Tur] Henrik Turbell, Cone-Beam Reconstruction using Filtered Backprojection. Dissertation, Linkping Studies in Science and Technology, 2001