Fast Calculation of Digitally Reconstructed Radiographs using Parallelism

Cobus Carstens

Department of Medical Radiation
iThemba LABS
Faure
cobus.carstens@gmail.com

Neil Muller

Division of Applied Mathematics
Department of Mathematical Sciences
University of Stellenbosch
neil@dip.sun.ac.za

Abstract

This study takes a look at different DRR generation methods for use in intensity based 2D/3D Image Registration. A specific emphasis is placed on whether the methods are trivially parallelisable with the aim to take advantage of hardware systems with more than one CPU. The ray casting algorithm and light field rendering are found to be suitable for this purpose. A discussion on the parallelisation of algorithms using OpenMP is presented and followed by performance measurements for the ray casting algorithm. A significant performance increase in the ray casting algorithm is achieved.

1. Introduction

Radiotherapy requires that patients be positioned accurately for treatment. An intensity based 2D/3D Image Registration (IR) process can be used as a final step in the positioning of the patient. An X-ray of the patient, called a portal radiograph (PR), is acquired and compared to numerous Digitally Reconstructed Radiographs (DRRs). Figure 1 shows a typical DRR. DRRs are generated from data obtained during a CT scan. The data is referred to as the *CT cube* [1]. The DRRs generated by the IR process represents multiple orientations of the CT data relative to the X-ray imaging system used to take the PR. The IR process searches for a rigid transform that produces a DRR most similar to the PR. The optimal transformation is used to align the CT coordinate system with that of the operating room. The time required for IR is dependent on

1. the time it takes for one objective function evaluation, which is dependant on the time it takes to calculate a DRR and the time it takes to measure how well the DRR and PR match.
2. the number of objective function evaluations required by the minimisation algorithm.

Van der Bijl [1] showed that the number of objective function evaluations (and therefore DRR calculations) required can be a few hundred. This study aims to reduce the time required to generate DRRs whilst maintaining an acceptable level of accuracy. A reduction in time reduces the chances of the patient moving and shortens any discomfort that might be experienced.

2. Discussion

2.1. DRR generation methods

A graphical illustration of DRR generation is given in Figure 2. Various methods to generate DRRs have been investigated:

[1] It does not always have the dimensions of a cube.

Figure 1: *A Digitally Reconstructed Radiograph generated using the ray casting method.*

Ray casting Let $\rho(i,j,k)$ denote the voxel density or attenuation coefficient in a 3-dimensional CT volume and $l(i,j,k)$ the length of the intersection of an X-ray with that voxel, then the radiological path is defined as

$$d = \sum_i \sum_j \sum_k l(i,j,k)\rho(i,j,k)$$

The radiological path is an approximation of the physics involved when an X-ray image is generated. Computing DRRs using the radiological path definition is $O(n^3)$ and very inefficient. Only a few voxels actually contribute to a path, since most $l(i,j,k)$ values will be zero. Siddon [2] proposed viewing CT voxels as the intersection of equally spaced, parallel planes. The intersection of the ray with the planes is then calculated, rather than the intersection of the ray with the different voxels. Determining the intersection of a ray with equally spaced, parallel planes is a simple problem. One needs to calculate the intersection with the first plane and the rest follows at fixed intervals because the planes are equally spaced.

An optimised version of Siddon's algorithm was proposed in [3]. The new algorithm reduces computation

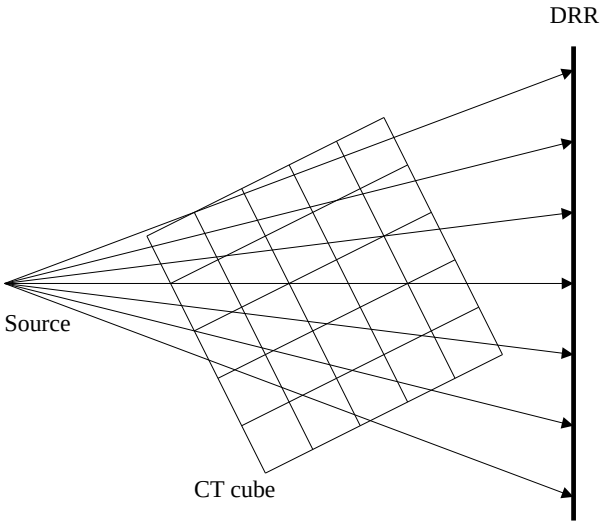

Figure 2: *A 2D view of DRR generation. The DRR is the set of values for the radiological paths from the source to the pixels on the image plane.*

by eliminating the need to explicitly compute the voxel indices for every interval. Also, it removes the need to allocate memory for the different arrays containing the intersection points.

Ray casting is classified as in image-order algorithm as it iterates over the pixels in the image that is generated.

The ray casting method is used as a benchmark of DRR quality, but it is too slow for real-time computations [4] and in its standard form slower than most other algorithms [4][5][6].

Splatting and voxel projection algorithms Splatting algorithms form part of the object-order class of algorithms. It operates by iterating over the voxels in the CT data. This class of algorithm computes the contribution of a voxel to the image. It uses a filter that distributes the voxel's value to a set of pixels. Splatters are also called forward projection algorithms since voxels are projected directly into the image, in the same direction as the light rays. Accurately computing the voxel footprint[2] and the resampling weights is expensive because it is view dependent. The footprint must be scaled, rotated and transformed depending on the viewing transformation. In a perspective projection every voxel has its own footprint. In theory, the splatter can produce exactly the same image as a ray caster. However, it is difficult to implement an algorithm that is both efficient and that produces high quality results [6].

Shear-warp factorisation Shear-warp factorisation is another object-order rendering algorithm. It has the advantage that rows of voxels in the data volume are aligned in parallel with rows of pixels in an intermediate image. This step is called shearing. An intermediate image is created using a scanline based algorithm that traverses the data volume and intermediate image in synchrony. The intermediate image then gets warped to form the final image.

[2]The polygon that is formed when projecting the voxel onto the image plane.

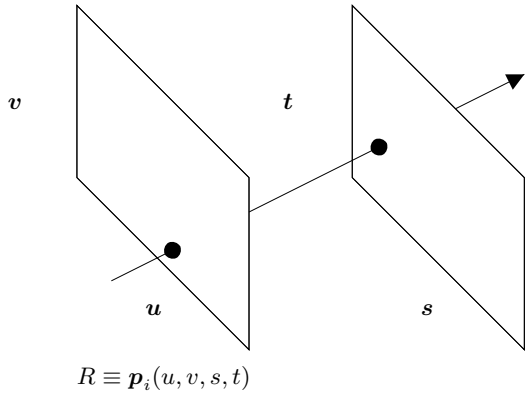

Figure 3: *The light field coordinate system.*

The arbitrary nature of the transformation from object space to image space complicates efficient projection in object-order volume rendering algorithms. This complication can be avoided if the data volume is transformed to an intermediate coordinate system for which there is a very simple mapping from the original (object) coordinate system and which allows efficient projection. Lacroute and Levoy [6] called this intermediate coordinate system the *sheared object space*. By construction all viewing rays in sheared object space are parallel to the third coordinate axis.

A simple algorithm based on the shear-warp factorisation operates as follows:

1. Transform the volume data to sheared object space. This is done by translating and resampling each slice. For perspective transformations each slice is also scaled.
2. Composite the resampled slices in front-to-back order using the "over" operator. This generates an intermediate image in sheared object space.
3. Transform the intermediate image to image space by warping it. This produces the final image.

Cylindrical harmonics Wang, Davids and Vemouri [7] proposed an algorithm based on the construction of projections of each harmonic in a cylindrical harmonic expansion of the CT data. After constructing a DRR from an arbitrary projection point, rotated DRRs can be efficiently computed. Only the weighted superpositions of the basis projections needs to be recomputed [8].

Light fields Light fields is a method that was originally proposed by Levoy and Hanrahan [9]. It can be described as a way of parameterising the set of all rays that emanate from a static scene. Each ray is identified by its intersection with two arbitrary planes in space. It is convention that the coordinate system on the first plane is (u, v) and that this plane is called the *focal plane*. The second plane has a coordinate system (s, t) and is called the *image* plane. It follows that every ray in this space can be represented as a point or pixel value $\mathbf{p}_i = (u_i, v_i, s_i, t_i)$ in 4-dimensional space (see Figure 3).

A *light slab* is the shape that is created when the focal plane and the image plane are connected. This represents all the light that enters the restricted focal plane and exits the restricted image plane.

If one can generate infinitely many rays inside a light slab, one can recreate almost any image with a focal point inside the light slab. This is done by finding the associated rays and their corresponding pixel values (see Figure 4). In practice one cannot generate infinitely many rays and are thus constrained to generate a large number and compute the missing rays using some form of interpolation.

Russakoff et al ([5]) points out three principle drawbacks when using light fields:

1. The 4D space must be densely sampled to create good images.
2. The static scene must be free of occluding objects.
3. The illumination of the scene must be fixed.

When dealing with DRR generation, only the first drawback applies.

A pixel value in a general light field is an indication of the amount of light reflected off the first surface a ray intersects with. When evaluating DRRs, however, the pixel values are the sum of the CT attenuation coefficients that the ray encounters from the projection point to the image plane.

To accommodate the generation of DRRs, we can associate each point $\boldsymbol{p}_i = (u_i, v_i, s_i, t_i)$ with a scalar function $\boldsymbol{p}_i \mapsto q(\boldsymbol{p}_i)$ which maps a point to the sum of the CT attenuation coefficients encountered along the ray $R_{\boldsymbol{p}_i}$.

In order to trace a ray through the CT data and maintain the same parameterisation of rays in space as traditional light fields one must cast the rays beyond a *virtual focal plane* onto an *effective focal plane*. The values on the effective focal plane is used for the light field generation.

In traditional light field rendering as well as light field DRR generation, the generated image is a skewed perspective image. However, where in traditional light field rendering the effective image plane remains fixed between the scene and the focal plane, in DRR generation the virtual image plane remains fixed while the effective image plane can move and lies on the other side of the scene from the focal plane.

2.2. Choosing a DRR method for our application

Our application impose the following criteria:

- Perspective projections must be used.
- DRR generation must be as fast as possible, whilst still maintaining an acceptable level of accuracy.
- DRR generation will be constrained to a search space defined by ϵ_t and ϵ_θ, which are the maximum translation and rotation errors, respectively, that the initial patient position can differ from the expected patient position.

Many computers today have multicore central processing units (CPUs). Some algorithms can be adapted to run on more than one CPU in order to speed up computation time. This characteristic was also deemed important in the selection of an algorithm.

When adapting an algorithm to run on more than one CPU, certain parts of it must be *parallelisable*. This means that certain sections of program code must be able to execute concurrently with as little interaction or interference as possible. For example, the ray casting algorithm is trivially parallelisable. Each ray

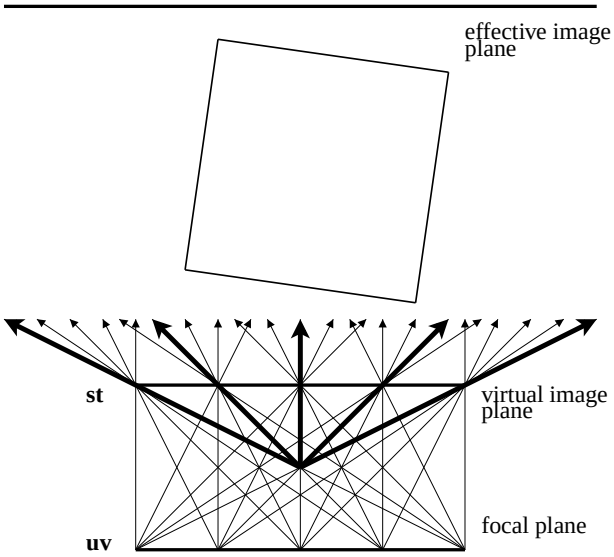

Figure 4: *A 2D view of a light slab, illustrating the view (in bold) generated for an arbitrary focal point.*

value can be computed without affecting any of the other rays. This is true for most image-ordered algorithms.

The object-ordered algorithms traverse the volume data and update multiple image values. It is therefore not trivially parallelisable, as some sort of locking mechanism will be required to marshal access to the image values.

Ray casting was implemented. It is the most accurate method and used as a benchmark of DRR quality. Furthermore, it is trivially parallelisable and can generate DRRs from arbitrary viewpoints. It also lends itself to be easily adapted to use X-ray attenuation calibration curves and and make provision for beam hardening correction.

Light fields will also be implemented. It is trivially parallelisable and has the benefit that a lot of computation can be done pre-operatively, rather than intra-operatively. It also supports DRR generation from arbitrary viewpoints in the light slab. The light slab is constructed pre-operatively using the ray casting algorithm and a fast (parallelisable) interpolation method is used intra-operatively to compute the DRRs. The light slab dimensions are chosen to make provision for both ϵ_t and ϵ_θ.

Splatting and voxel projection are object-ordered algorithms and are therefore not trivially parallelisable.

Shear-warp factorisation is an object-ordered algorithm and also not trivially parallelisable. It is also better suited for volume rendering than DRR generation.

Cylindrical harmonics is not particularly suited for arbitrary perspective projections and therefore not considered for implementation.

3. Implementation

3.1. OpenMP

In its manual ([10]) OpenMP is described as a set of compiler directives, library routines, and environment variables that can be used to specify shared-memory parallelism in C, C++ and Fortran programs. It provides a model for parallel programming

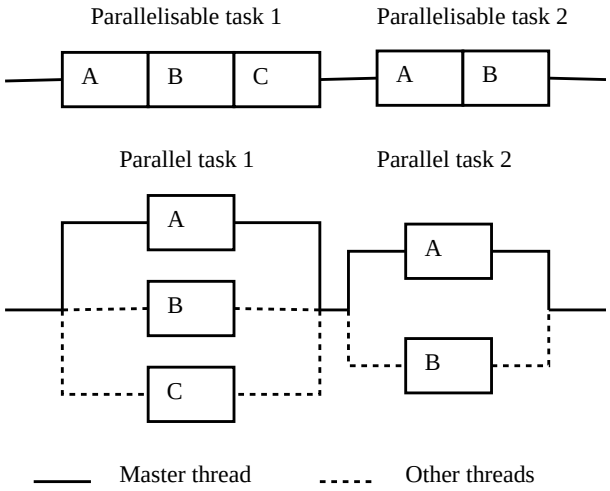

Figure 5: *A sequential but parallelisable algorithm (top) and the equivalent algorithm using fork-join parallelism (bottom).*

that is portable across shared memory architectures. OpenMP allow users to create and manage portable parallel programs. The OpenMP directives extend the C, C++ and Fortran languages with single program multiple data (SPMD) constructs, work-sharing constructs, and synchronisation constructs, and they provide support for the sharing and privatisation of data. The runtime environment can be controlled using the library routines and environment variables. Figure 5 illustrates how some algorithms can be parallelised.

Pros

- It is simple
- The data layout and decomposition is handled automatically by the directives
- No dramatic code change is needed as work can be done to one part of a program without affecting the rest
- A single code base for both serial and parallel applications: If OpenMP support is not available, the compiler treats the OpenMP directives as comments
- Serial code statements in general need not be modified when parallelised with OpenMP. This reduces the chance of inadvertently introducing bugs.
- Coarse-grained and fine-grained parallelism are possible

Cons

- It only runs efficiently on shared-memory multiprocessor platforms
- Requires a compiler that supports OpenMP
- Scalability is limited by memory architecture
- Reliable error handling is missing
- Cannot control thread to processor mapping
- Synchronisation between a subset of threads is not allowed

3.2. The programming language and compiler

The following compilers support OpenMP:

- Visual C++ 2005 (Professional and Team System editions)
- Intel x86 and IPF compilers
- Sun Studio (for Solaris OS)
- GCC 4.2 (although some distributions such as Fedora Core 5 have support for OpenMP in their GCC 4.1 based compilers)

The ray casting algorithm was implemented in C++ and compiled using GCC 4.1. An example of the trivial use of OpenMP is given by the following code snipit:

```
#ifdef _OPENMP
#pragma omp for
#endif
for (int j=0; j<height; j++)
{
#ifdef _OPENMP
#pragma omp for
#endif
    for (int i=0; i<width; i++)
        image(i,j)=raycast(focalPoint,
                           targetPoint(i,j),
                           volumeData);
}
```

The *omp for* directive specifies that the iterations of the associated loop will be executed in parallel. The iterations of the loop are distributed across different threads. If the *ifdef* guards are omitted and the compiler does not support OpenMP (or it was simply not enabled), the compiler should only generate a warning message and compile the sequential code.

4. Results

4.1. Expected performance

In the ideal case one expects an *N* times speedup when *N* processors are available. This is seldom the case for the following reasons:

- A large portion of the program may not be parallelised by OpenMP, which sets theoretical upper limit of speedup according to Amdahl's law. For the case of parallelisation, Amdahl's law states that if F is the fraction of a program or algorithm that is sequential (i.e. cannot benefit from parallelisation), and $(1 - F)$ is the fraction that can be parallelised, then the maximum speedup that can be achieved by using N processors is $\frac{1}{F+(1-F)/N}$. As N tends to infinity the maximum speedup tends to $1/F$. In practice the price/performance ratio falls rapidly as N is increased once $(1 - F)/N$ is small compared to F. Parallel computing is only useful when either a small number of processors is used, or when the problem has a low value of F [11].
- *N* processors may provide *N* times the computation power of a single processor, but the memory bandwidth usually does not scale with *N*. Quite often performance degradation may be observed when processors compete for shared memory bandwidth.
- Load balancing, synchronisation overhead and other common problems encountered in parallel computing also apply to OpenMP.

4.2. Measured performance

The tests were performed on a machine with the following specifications:

CPUs Two quad-core Intel Xeon CPUs running at 2GHz. This effectively translates to *eight* processors.

RAM 2GB

Operating System Oracle Enterprise Linux AS release 4 (based on Red Hat)

Compiler gcc version 4.1.0 20060515 (Red Hat 4.1.0-18)

The data volume used was synthetically constructed with a dimension of 256^3 and slice and pixel distances of $2mm$. The DRRs generated have a resolution of 512^2. Table 1 shows the ray casting performance times when using sequential compilation and when using OpenMP. Eight DRR images were generated. It must be noted that OpenMP by default dynamically manage its threads at run time to improve performance.

Table 1: *Ray casting performance.*

Image	Sequential	OpenMP
I_1	1.843s	0.308s
I_2	3.564s	0.748s
I_3	1.384s	0.251s
I_4	3.568s	0.744s
I_5	1.844s	0.307s
I_6	2.011s	0.395s
I_7	1.385s	0.253s
I_8	2.032s	0.393s
I_{avg}	2.204s	0.425s

To illustrate the performance gains as the number of CPUs increase, the number of threads used by OpenMP was constrained. It is assumed that each thread will run on a different CPU. This, however, is not always the case, but for illustrative purposes it suffices. Figure 6 shows the results. The dotted curve represents the *ideal* value V_1/N, where V_1 is I_{avg} computed using a single thread and N is the number of threads used. The solid curve represents the actual measured times. Since the test computer only have 8 processors, we do not expect to have any improvement when using more than 8 threads. Figure 6 confirms this as the DRR generation time does not decrease when more than 8 threads are used.

5. Conclusions and future work

In this study we have shown that certain DRR generation algorithms are trivially parallelisable and that parallelisation can lead to significantly increased performance for the ray casting algorithm (approximately 5 to 6 fold for the hardware used). We expect similar improvements for the light fields method. Firstly, when generating the light slab pre-operatively, the parallelised ray casting algorithm will be used. This will significantly reduce the construction time. Secondly, the intra-operative part of light field rendering consists of interpolating light slab values. This is also a trivially parallelisable algorithm where we expect to see speedups similar to that seen for ray casting and, as with the ray casting implementation, adapting the light field algorithm will take very little effort.

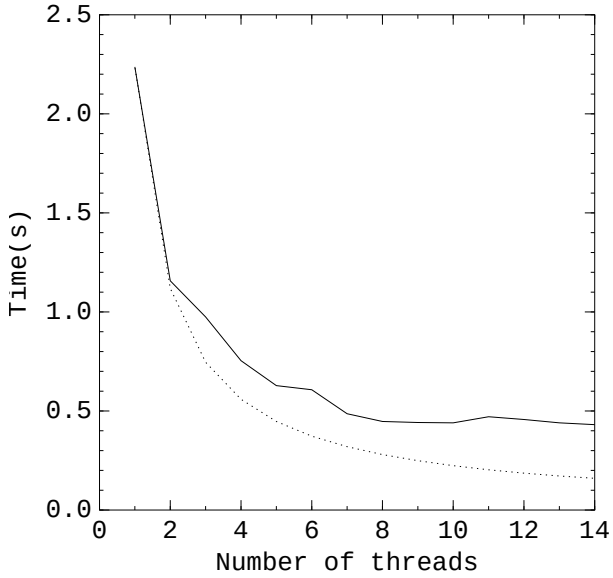

Figure 6: *The algorithm performance as the number of threads increase. The dotted curve represents the ideal time while the solid curve represents the actual measured time.*

6. References

[1] Leendert van der Bijl, "Verification of patient position for proton therapy using portal x-rays and digitally reconstructed radiographs," M.S. thesis, University of Stellenbosch, 2006.

[2] Robert L. Siddon, "Fast calculation of the exact radiological path for a three-dimensional CT array," *Medical Physics*, vol. 12, no. 2, pp. 252–255, March 1985.

[3] F. Jacobs, E. Sundermann, B. De Sutter, M. Christiaens, and I. Lemahieu, "A fast algorithm to calculate the exact radiological path through a pixel or voxel space," *Journal of computing and information technology*, vol. 6, no. 1, pp. 89–94, 3 1998.

[4] C.T. Metz, "Digitally reconstructed radiographs," M.S. thesis, Utrecht University, 2005.

[5] D. Russakoff, T. Rohlfing, D. Rueckert, R. Shahidi, D. Kim, and C. Maurer, "Fast calculation of digitally reconstructed radiographs using light fields," 2003.

[6] Philippe Lacroute and Marc Levoy, "Fast volume rendering using a shear-warp factorization of the viewing transformation," *Computer Graphics*, vol. 28, no. Annual Conference Series, pp. 451–458, 1994.

[7] Fei Wang, Thomas E. Davis, and Baba C. Vemuri, "Real-time drr generation using cylindrical harmonics," in *MICCAI '02: Proceedings of the 5th International Conference on Medical Image Computing and Computer-Assisted Intervention-Part II*, London, UK, 2002, pp. 671–678, Springer-Verlag.

[8] Michael Baumann, "Fast DRR generation with light fields," M.S. thesis, Universität Karlsruhe, 2004.

[9] Marc Levoy and Pat Hanrahan, "Light field rendering," in *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, New York, NY, USA, 1996, pp. 31–42, ACM Press.

[10] OpenMP Architecture Review Board, "Openmp application program interface," 2005.

[11] Gene Amdahl, "Validity of the single processor approach to achieving large-scale computing capabilities," in *AFIPS Conference Proceedings*, Atlantic City, N.I., April 1996, vol. 30, pp. 483–485, AFIPS Press, Reston, Va.