A Fast Ray-Tracing Technique for TCT and ECT Studies[1]

Guoping Han, Zhengrong Liang, and Jiangsheng You
Departments of Physics and Astronomy, Radiology, and Computer Science
State University of New York, Stony Brook, NY 11794

Abstract

In transmission computed tomography (TCT) and emission computed tomography (ECT) studies, a common geometric problem is to trace those voxels along a certain projection ray. It is a time consuming task due to enormous number of voxels on each ray and enormous number of rays involved for the tomographic studies. A straight-forward ray-tracing technique would require computing time that scales with the array size N^3. Siddon proposed a fast method to trace the rays whose computing time scales with $3N$. In this study, a refinement to Siddon's algorithm is investigated. In Siddon's algorithm, the index of each voxel along a ray and the intersecting length of that ray within that voxel are computed by four multiplications, which consumes 53% of the total computing time for a typical tomographic study involving an array of 21^3 points. By our new algorithm described in this article, three multiplications of Siddon's method for computing the voxel indices are replaced by an increment or decrement operation. The fourth multiplication for computing the voxel intersecting lengths is eliminated by carefully selecting a factor to parameterize the rays. Simulation studies on randomly generated projection rays in a N^3 voxel array of $N = 21$, 64, 128, 256, 384 and 512 showed a decrease of approximately $2/3$ in total computing time in tracing the rays.

I. Introduction

In transmission computed tomography (TCT) and emission computed tomography (ECT) studies, simulated projections are computed from known or assumed source and/or attenuation coefficient distributions, which may act as a projector in iterative image reconstruction. In TCT simulation studies, it is desirable to compute the radiological path defined by

$$d = \sum_{i,j,k} l(i,j,k)\ \mu(i,j,k) \tag{1}$$

where $\mu(i,j,k)$ is the density (or attenuation coefficient) distribution and $l(i,j,k)$ is the intersecting length of the path in the voxel (i,j,k). In single-photon ECT (SPECT) simulations, it is common to compute the sum along a projection ray

$$d = \sum_{i,j,k} l(i,j,k)\ s(i,j,k) \exp\left[-\sum_{i',j',k'} l(i',j',k')\ \mu(i',j',k')\right] \tag{2}$$

where $s(i,j,k)$ is the source (or radiotracer) distribution. In (2), the sum over i,j,k is evaluated over all voxels along a certain ray and the sum over i',j',k' is evaluated over those voxels along that ray from a starting point to an ending point on the ray. Although it is simple and straightforward to compute the above sums in principle, due to enormous number of rays to be traced in a study and enormous number of voxels on each ray, an elaborate algorithm and a significant amount of computing time are usually required.

II. Method

For a $N \times N \times N$ voxel array, direct evaluation of (1) or (2) entails an algorithm which scales with N^3, *i.e.*, the number of terms in the sums. Siddon proposed a fast algorithm which scales with $3N$ [1]. The idea of Siddon's method is the consideration of the voxels on a ray as the intersection of that ray with orthogonal sets of equally spaced, parallel planes. For an array of $N_x \times N_y \times N_z$ voxels, the sets of equally spaced, parallel planes can be written as

$$\begin{aligned} x &= i, \ \ 0 \le i \le N_x \\ y &= j, \ \ 0 \le j \le N_y \\ z &= k, \ \ 0 \le k \le N_z \end{aligned} \tag{3}$$

where, for simplicity, the spacing between adjacent parallel planes is assumed to be unity. A convention is made hereafter such that the voxel described by

$$\{(x,y,z) \mid i \le x < i+1, j \le y < j+1, k \le z < k+1\}$$

is assigned with the index (i,j,k).

In Siddon's method, a ray from $P_1(x_1,y_1,z_1)$ to $P_2(x_2,y_2,z_2)$ is represented parametrically as

$$\begin{aligned} x(\alpha) &= x_1 + \alpha\,(x_2 - x_1) \\ y(\alpha) &= y_1 + \alpha\,(y_2 - y_1) \\ z(\alpha) &= z_1 + \alpha\,(z_2 - z_1) \end{aligned} \tag{4}$$

where the parameter α is 0 at P_1 and 1 at P_2. Siddon's algorithm computes the values of the parameter α at each intersection of the ray with each set of parallel planes and sorts them to form an ascending set $\{\alpha(0), \alpha(1), ..., \alpha(n)\}$. After that, the intersecting length l and the voxel index (i,j,k) are computed by

$$l(m) = [\alpha(m) - \alpha(m-1)]\ L, 0 < m \le n \tag{5}$$

and

$$\begin{aligned} i(m) &= \lfloor x_1 + \alpha_{\text{mid}}(x_2 - x_1) \rfloor \\ j(m) &= \lfloor y_1 + \alpha_{\text{mid}}(y_2 - y_1) \rfloor \\ k(m) &= \lfloor z_1 + \alpha_{\text{mid}}(z_2 - z_1) \rfloor, \ 0 < m \le n \end{aligned} \tag{6}$$

[1]This work was supported by NIH Grant #HL54166 of the National Heart, Lung, and Blood Institute; Grant #NS33853 of the National Institute of Neurological Disorder and Stroke; Grant #CA79180 of the National Cancer Institute; and Established Investigatorship of the American Heart Association.

0-7803-5696-9/00/$10.00 (c) 2000 IEEE

where L is the total length of the ray from P_1 to P_2,

$$L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

and α_{mid} is given by

$$\alpha_{\text{mid}} = \frac{1}{2}\left[\alpha\left(m-1\right) + \alpha\left(m\right)\right].$$

In (6), $\lfloor x \rfloor$ is the floor function, *i.e.*, the maximum integer less than or equal to x. Another notation used below is the ceiling function $\lceil x \rceil$, *i.e.*, the minimum integer greater than or equal to x.

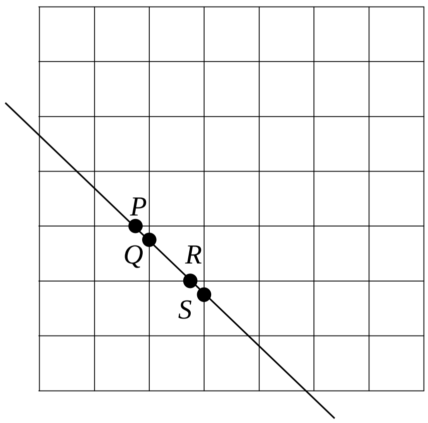

Figure 1: Recursive computation of the voxel indices by incrementing or decrementing one of the indices of the previous voxel in the ray. For example, suppose PQ intersects with voxel (i, j), then QR intersects with voxel $(i', j') = (i + 1, j)$; and RS intersects with voxel $(i'', j'') = (i', j' - 1)$.

It is noted that the indices of neighboring voxels in a ray differ, in general cases, by either $(0, 0, 1)$, or $(0, 1, 0)$, or $(1, 0, 0)$, as illustrated in Figure 1 for two-dimensional situation. Our new algorithm takes advantage of this fact, and sets the index of the first voxel at the beginning and then computes the indices of the following voxels recursively. This novel idea replaces those three multiplications in (6) by a simple increment or decrement operation.

Another fact to be noted is that instead of using parameter α, an alternative parameter λ, which is the distance from P_1 or the starting point of the ray, can be utilized to parametrize the ray, *i.e.*,

$$\begin{aligned} x(\lambda) &= x_1 + \frac{\lambda}{L}(x_2 - x_1) \\ y(\lambda) &= y_1 + \frac{\lambda}{L}(y_2 - y_1) \\ z(\lambda) &= z_1 + \frac{\lambda}{L}(z_2 - z_1). \end{aligned} \quad (7)$$

Then the intersecting length can be computed without any multiplication of (5)

$$l(m) = \lambda\left(m\right) - \lambda\left(m-1\right). \quad (8)$$

In summarizing the two observations above, our new algorithm can be represented as follows.

1. Initialize the recursive ray tracing by computing the indices (v_x, v_y, v_z) of the starting voxel and the initial value of parameter λ.

   (a) Compute the range of values of the λ, *i.e.*, the parameters $\lambda_{\min}$, $\lambda_{\max}$ (similar to Siddon's method), and set the initial λ value to be $\lambda_{\min}$:

   $$\begin{aligned} \lambda_{\min} &= \max\{0, \min\left\{\lambda_x\left(0\right), \lambda_x\left(N_x\right)\right\}, \\ &\quad \min\left\{\lambda_y\left(0\right), \lambda_y\left(N_y\right)\right\}, \\ &\quad \min\left\{\lambda_z\left(0\right), \lambda_z\left(N_z\right)\right\}\} \\ \lambda_{\max} &= \min\{1, \max\left\{\lambda_x\left(0\right), \lambda_x\left(N_x\right)\right\}, \\ &\quad \max\left\{\lambda_y\left(0\right), \lambda_y\left(N_y\right)\right\}, \\ &\quad \max\left\{\lambda_z\left(0\right), \lambda_z\left(N_z\right)\right\}\} \end{aligned} \quad (9)$$

   where

   $$\begin{aligned} \lambda_x\left(i\right) &= \frac{L}{x_2 - x_1}\left(i - x_1\right) \\ \lambda_y\left(j\right) &= \frac{L}{y_2 - y_1}\left(j - y_1\right) \\ \lambda_z\left(k\right) &= \frac{L}{z_2 - z_1}\left(k - z_1\right). \end{aligned}$$

   (b) Compute the index of the first voxel, (v_x, v_y, v_z), on the ray by

   $$\begin{aligned} v_x &= \lfloor x_{\min} \rfloor, \ \text{if } x_{\min} \text{ is not integer or } x_1 > x_2 \\ &= x_{\min} - 1, \text{if } x_{\min} \text{ is integer and } x_1 \le x_2 \end{aligned} \quad (10)$$

   where

   $$x_{\min} = x_1 + \lambda_{\min}\frac{x_2 - x_1}{L}$$

   and v_y, v_z are similarly computed.

   (c) Set the initial values of $(\lambda_x, \lambda_y, \lambda_z)$, the parameters where the ray first intersects in the array with a plane perpendicular to the x-, y-, or z-axis,

   $$\begin{aligned} \lambda_x &= \left(u_x - x_1\right)\frac{L}{x_2 - x_1} \\ \lambda_y &= \left(u_y - y_1\right)\frac{L}{y_2 - y_1} \\ \lambda_z &= \left(u_z - z_1\right)\frac{L}{z_2 - z_1} \end{aligned} \quad (11)$$

   where

   $$\begin{aligned} u_x &= \lfloor x_{\min} \rfloor, \quad \text{if } x_1 \le x_2 \\ &= \lceil x_{\min} \rceil, \quad \text{if } x_1 > x_2 \end{aligned}$$

   and u_y, u_z are similarly computed. If any of the denominators in (11) is zero, then set the corresponding λ value to a number greater than $\lambda_{\max}$.

2. Find the minimum in the triplet $(\lambda_x, \lambda_y, \lambda_z)$, say $\lambda_\xi = \min\{\lambda_x, \lambda_y, \lambda_z\}$, where ξ is on one of the axes x, y, or z. If $\lambda_\xi > \lambda_{\max}$, compute the length of the final voxel

$$l = \lambda_{\max} - \lambda$$

and terminate the ray tracing.

3. Compute the intersecting length in the voxel (v_x, v_y, v_z):

$$l = \lambda_\xi - \lambda. \tag{12}$$

4. Increment λ_ξ by $\left|\frac{L}{\xi_2 - \xi_1}\right|$,

$$\lambda_\xi = \lambda_\xi + \left|\frac{L}{\xi_2 - \xi_1}\right|. \tag{13}$$

Note that this computes *and* sorts the λ parameters after the ray tracing is completed.

5. Compute the indices of the next voxel

$$v_\xi = \begin{cases} v_\xi + 1, \text{ if } \xi_1 < \xi_2 \\ v_\xi - 1, \text{ if } \xi_1 > \xi_2 \end{cases} \tag{14}$$

and then go back to step (2).

It is clearly seen that no multiplication is employed in computing both the indices of the voxels and the intersecting lengths, except for the first voxel on the ray. Note that in the new algorithm, the λ parameters are not precomputed and merged before ray tracing, as in Siddon's algorithm. Instead, the first triplet, $(\lambda_x(0), \lambda_y(0), \lambda_z(0))$, is set before ray tracing, and the following λ parameters are computed recursively and then merged in the tracing process.

III. SIMULATION AND DISCUSSION

Monte Carlo simulations were performed to evaluate the performance of the new algorithm and compare it to Siddon's algorithm. In the simulations, voxel arrays of size $N \times N \times N$ were employed and 1 million rays were randomly generated. Simulations were performed for the cases of $N = 21$, 64, 128, 256, 384 and 512, respectively. The time consumed in each computing step for each case were measured. The algorithms of Siddon and ours and the simulations were coded in C on a SGI workstation running IRIX 6.4 operating system.

The resulting voxel indices and intersecting lengths were compared to those computed by Siddon's algorithm. They are found to be identical as expected.

The computation times consumed in the four steps of our new algorithm are shown in Figure 2. From that figure, it is seen that the time spent in computing the voxel indices is greatly reduced, to a negligible percentage of the total computing time (note the scaling of two lines in the figure), while Siddon's algorithm spends a great deal of time on the indices (53% on the voxel indices and intersecting length).

The comparison of the performance of our algorithm with Siddon's algorithm was performed for all the cases. The results are shown in Figure 3. The new algorithm reduced the ray-tracing time dramatically, by approximately $2/3$.

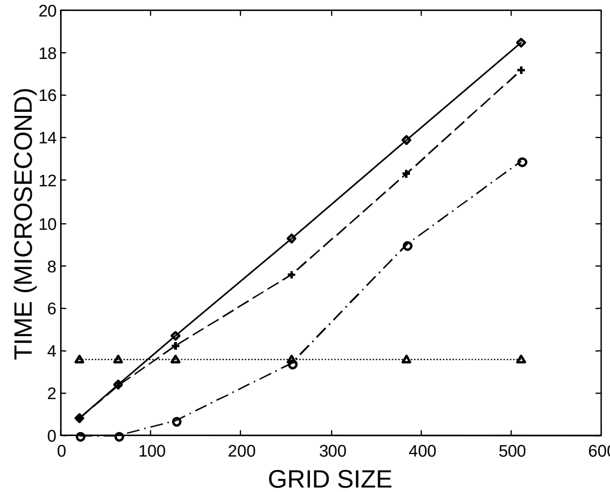

Figure 2: Average time (in microseconds) consumed in each step of the new recursive ray tracing for several grid sizes. Time for initialization is shown by triangles and dotted line; time for computing and sorting the λ parameters, *scaled down by a factor of* 10, is shown by diamonds and solid line; time for computing the voxel indices, *scaled up by a factor of* 10, is shown by circles and dash-dotted line; and time for computing the voxel lengths is shown by $+$ signs and dashed line.

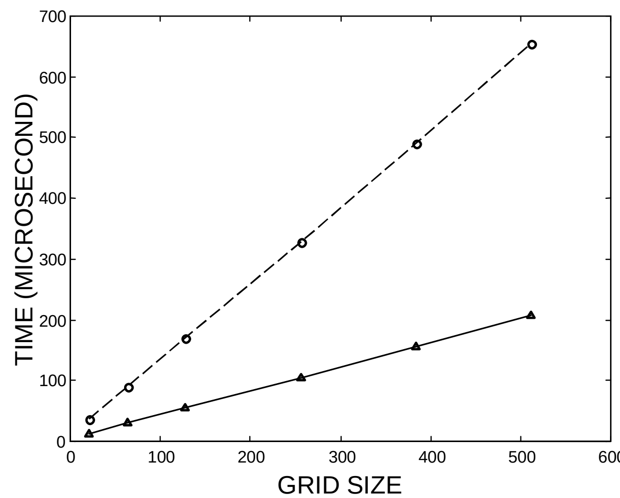

Figure 3: Average time (in microseconds) consumed in tracing one ray using our fast recursive ray-tracing algorithm (triangles and solid line) and using Siddon's algorithm (circles and dashed line) for several grid sizes.

IV. CONCLUSION

A new algorithm is proposed for fast ray tracing aiming to improve Siddon's method. Simulation studies show that this new algorithm reduces ray tracing computation by approximately $2/3$ times as compared with Siddon's method. Due to enormous number of projection rays involved in TCT and ECT studies, a large fraction of computing time is spent on tracing those rays. This new algorithm can be very useful for reduction of the ray-tracing computing time and, therefore, can improve quantitative SPECT imaging [2], as well as radiation treatment planning [3].

V. References

[1] R. L. Siddon, "Fast calculation of the exact radiological path for a three-dimensional CT array," *Med Phys* **12** 252-255, 1986.

[2] Z. Liang, T. G. Turkington, D. R. Gilland, R. J. Jaszczak, and R. E. Coleman, "Simultaneous compensation for attenuation, scatter and detector response for SPECT reconstruction in three dimensions," *Phys Med Biol* **37** 587-603, 1992.

[3] Z. Chen, C. S. Chui, Z. Liang, and L. E. Reinstein, "Comparison of two image reconstruction algorithms for radiotherapy treatment plan optimization," *Med Phys* **22** 1010, 1995.