Table of Contents

Thick Line Integration with Filtered Sampling

Dóra Varnyú and László Szirmay-Kalos

Budapest University of Technology and Economics, Student Research Group of Balatonfüred and Dept. of Control Eng. and Inf. Tech.
szirmay@iit.bme.hu

Abstract. In particle tracing, motion blur calculation, participating media rendering, or tomography reconstruction, we often need to evaluate the integral of a scalar function in a thick line, i.e. a pipe like domain. Monte Carlo methods would sample the surfaces at the two ends of the pipe and evaluate a line integral between the two sample points, which is approximated by discrete point samples. Although this method produces unbiased estimates, an accurate estimation would require a high number of samples. A more efficient approach would sample the domain not with points or lines but with 3D volumetric structures. In this paper, we examine and compare such sampling schemes, point out their similarity to filtered sampling and the 3D generalization of anti-aliased line drawing algorithms. The proposed methods are used in forward projection of tomography reconstruction and also in model based motion compensation.

1 Introduction

In iterative positron emission tomography (PET) forward and back projections alternate [8]. Forward projection models the physics of the system by computing the expected number of simultaneous γ-photon hits in detector pairs (a.k.a. Line Of Responses or LORs), $\tilde{y}_L$, from the current estimation of the radiotracer density $x(\boldsymbol{v})$, while back projection corrects the current estimation based on the ratio of the measured y_L and computed LOR-hits $\tilde{y}_L$. The *tracer density* function $x(\boldsymbol{v})$ is approximated assuming that the tracer density is x_V in voxel V. The expected hits for a given LOR L is:

$$\tilde{y}_L = \int_{\mathcal{V}} x(\boldsymbol{v})\mathcal{T}(\boldsymbol{v} \to L)\mathrm{d}v \tag{1}$$

where $\mathcal{V}$ is the volume of interest and $\mathcal{T}(\boldsymbol{v} \to L)$ is the *system sensitivity* denoting the probability that a positron born in $\boldsymbol{v}$ causes a γ-photon pair hit in LOR L.

Considering only the geometry, a LOR can be affected only if its detectors are seen at directions $\boldsymbol{\omega}$ and $-\boldsymbol{\omega}$ from emission point $\boldsymbol{v}$. It also means that emission point $\boldsymbol{v}$ and direction $\boldsymbol{\omega}$ unambiguously identify detector hit points $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$, or alternatively, from detector hit points $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$, we can determine those emission points $\boldsymbol{v}$ and direction $\boldsymbol{\omega}$, which can contribute.

To establish a LOR-driven approach, we modify our view point from the emission points and directions to detector points, and using the correspondence between them, the detector response is expressed as an integral over the detector surfaces. The Jacobian of the change of integration variables is:

$$\mathrm{d}\omega \mathrm{d}v = \frac{\cos\theta_{\boldsymbol{z}_1}\cos\theta_{\boldsymbol{z}_2}}{|\boldsymbol{z}_1-\boldsymbol{z}_2|^2}\mathrm{d}l\mathrm{d}z_1\mathrm{d}z_2$$

where $\theta_{\boldsymbol{z}_1}$ and $\theta_{\boldsymbol{z}_2}$ are the angles between the surface normals and the line connecting points $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ on the two detectors, respectively. With this, the LOR integral can be expressed as a triple integral over the two detector surfaces D_1 and D_2 of the given LOR and over the line connecting two points $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ belonging to the two detectors:

$$\tilde{y}_L = \int_{D_1}\int_{D_2} \frac{\cos\theta_{\boldsymbol{z}_1}\cos\theta_{\boldsymbol{z}_2}}{2\pi|\boldsymbol{z}_1-\boldsymbol{z}_2|^2}\left(\int_{\boldsymbol{z}_1}^{\boldsymbol{z}_2} x(\boldsymbol{l})\mathrm{d}l\right)\mathrm{d}z_2\mathrm{d}z_1. \tag{2}$$

LOR driven methods are also called *ray based* since they identify voxels that may contribute to a LOR by casting one or more rays between two points on the LOR detectors. Equation (2) can be estimated by taking N_{detline} uniformly distributed point pairs, $(\boldsymbol{z}_1^{(i)}, \boldsymbol{z}_2^{(i)})$ on the two detectors, and selecting N_{step} points $\boldsymbol{l}_{ij}$ of distance Δl_{ij} along each line segment $(\boldsymbol{z}_1^{(i)}, \boldsymbol{z}_2^{(i)})$:

$$\tilde{y}_L \approx \frac{D_1 D_2}{2\pi N_{\text{detline}}}\sum_{i=1}^{N_{\text{detline}}}\left(\frac{\cos\theta_{\boldsymbol{z}_1^{(i)}}\cos\theta_{\boldsymbol{z}_2^{(i)}}}{|\boldsymbol{z}_1^{(i)}-\boldsymbol{z}_2^{(i)}|^2}\sum_{j=1}^{N_{\text{step}}} x(\boldsymbol{l}_{ij})\Delta l_{ij}\right).$$

This formula is the Monte Carlo estimator of the expected LOR hits taking discrete point samples $\boldsymbol{l}_{ij}$ in the voxel domain. In order to get a high accuracy estimate, the domain of each basis function that is relevant for this LOR should be sufficiently densely sampled, which would lead to very high sample numbers.

For the approximation of the line integral along a ray, we may use ray marching or Siddon's algorithm [7]. Siddon's algorithm gives exact integral values in case homogeneous voxels, but the number steps N_{detline} varies for LORs, which is a significant disadvantage in GPU implementation, since it makes parallel threads responsible for different LORs incoherent. Ray marching always take the same number of steps, thus the threads are coherent, but due to the point sampling, it can have high variance estimates. For example, if the line crosses a high activity voxel in a very short line segment, this LOR may have a large value with low probability. Line drawing algorithms of computer graphics, like the Bresenham's algorithm [1], take the same steps if the maximal coordinate change is the same, and select those pixels in 2D where the line-pixel intersection is long enough.

Joseph's method [3] extended the Bresenham's method with filtering operation, by computing the weighted average of two pixels in between the line is passing through. Distance driven techniques [5] mapped both the detector and

voxel onto the same plane, and used the intersections of the projected areas for weighting. Unlike other methods, this technique takes into account where the detector boundaries are, but works only for fan-beam or cone-beam architecture.

Executing filtering before projections [4, 6] is very efficient since then the simplest line integration produces already filtered results, but such approaches also ignore the locations of LOR boundaries.

In this paper, we revisit filtered line integration for PET forward projection. We provide a technique to guarantee unbiasedness and prove that such filtering always reduces the variance of the applied Monte Carlo integration.

2 Filtered sampling

Filtered sampling replaces the integrand by another function that has a similar integral but smaller variation, then its integral can be estimated more precisely from discrete samples. Reducing the variation means the filtering of high frequency fluctuations by a low-pass filter. This filter should eliminate frequencies beyond the limit corresponding to the density of the sample points. On the other hand, it should only minimally modify the integral.

In order to examine the properties of filtered sampling, let us consider the integral I_f of function $f(x)$ in $[a, b]$ by taking M s-independent uniformly distributed random samples $x_1, \ldots, x_M$. The Monte Carlo estimator $\hat{I}_f$ is

$$I_f = \int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{M} \sum_{i=1}^{M} f(x_i) = \hat{I}_f.$$

The expected value of this random estimator $\hat{I}_f$ is equal to the original integrand, i.e. it is unbiased, and its variance is

$$\mathbf{V}\left[\hat{I}_f\right] = \frac{b-a}{M} \left(\int_0^1 f^2(x)\mathrm{d}x - I_f^2 \right).$$

Now let us consider a filtered version $F(x)$ of the integrand with *smoothing filter kernel* $s(x)$:

$$F(x) = \int_{-\infty}^{\infty} s(y) f(x-y)\mathrm{d}y.$$

where the filter kernel is non-negative, normalized and its support is $[-\Delta x, \Delta x]$. We also suppose $s(y)$ to be symmetric, i.e. $s(y) = s(-y)$. Non-negative, normalized filter kernels can also be considered as probability densities, and the filtering operation as the computation of the expected value:

$$F(x) = \mathbf{E}_{s(y)}\left[f(x-y)\right]$$

The integral of the filtered integrand is

$$I_F = \int_{x=a}^{b} F(x)\mathrm{d}x = \int_{x=a}^{b} \mathbf{E}_{s(y)}\left[f(x-y)\right]\mathrm{d}x = \mathbf{E}_{s(y)}\left[\int_{x=a}^{b} f(x-y)\mathrm{d}x\right]$$
$$= \mathbf{E}_{s(y)}\left[\int_{x'=a-y}^{b-y} f(x')\mathrm{d}x'\right] = I_f + \mathbf{E}_{s(y)}\left[\int_{x=a-y}^{a} f(x)\mathrm{d}x - \int_{x=b-y}^{b} f(x)\mathrm{d}x\right].$$

The integral of F is generally not equal to that of f because filtering steps over the two boundaries of the domain, thus evaluating a Monte Carlo quadrature for F results in a biased estimate for the integral of f. The bias caused by the boundary is

$$B = \mathbf{E}_{s(y)}\left[\int_{x=a-y}^{a} f(x)\mathrm{d}x - \int_{x=b-y}^{b} f(x)\mathrm{d}x\right]. \tag{3}$$

This bias can be eliminated by defining integrand f appropriately outside of the original integration domain $[a,b]$. Let $f(x) = f(2a-x)$ when $a-\Delta x < x < a$ and $f(x) = f(2b-x)$ when $b < x < b+\Delta x$, i.e. let us mirror the integrand onto the two boundaries in the domain of the filter kernel. Let us reconsider the integral of F for this case

$$I_F = \mathbf{E}_{s(y)}\left[\int_{x=a-y}^{b-y} f(x)\mathrm{d}x\right].$$

Variable y is a translation for the domain of the integral of f. When y is positive, the domain is translated to the left, modifying the integrand by adding a new area in $[a-y,a]$ and subtracting an area in $[b-y,b]$. As integrand f is symmetric inside $[a-\Delta x, a+\Delta x]$ and also in $[b-\Delta x, b+\Delta x]$, the total modified area is the same when y is multiplied by -1, just addition becomes subtraction and vice versa. As the filter kernel, i.e. the probability density $s(y)$ is also symmetric, these two cases cancel each other. As any y has a corresponding $-y$, the total modification caused by filtering over the boundary is zero.

Let us now compare the variances of $\hat{I}_F$ and $\hat{I}_f$. As their expectations are similar and we take the same number of samples, $\mathbf{V}[\hat{I}_F]$ is smaller than $\mathbf{V}[\hat{I}_f]$ if

$$\int_a^b F^2(x)\mathrm{d}x \leq \int_a^b f^2(x)\mathrm{d}x.$$

Let us use the definition of $F(x)$:

$$\int_a^b F^2(x)\mathrm{d}x = \int_{x=a}^{b} \left(\mathbf{E}_{s(y)}\left[f(x-y)\right]\right)^2\mathrm{d}x. \tag{4}$$

The integrand is the square of expected value $\mathbf{E}_{b(y)}\left[f(x-y)\right]$. As squaring is a convex function, using Jensen's inequality, we can write

$$\left(\mathbf{E}_{s(y)}\left[f(x-y)\right]\right)^2 \leq \mathbf{E}_{s(y)}\left[f^2(x-y)\right].$$

Substituting into Eq. 4, we get

$$\begin{aligned}\int_a^b F^2(x)\mathrm{d}x &\leq \int_{x=a}^{b} \mathbf{E}_{s(y)}\left[f^2(x-y)\right]\mathrm{d}x = \mathbf{E}_{s(y)}\left[\int_{x=a}^{b} f^2(x-y)\mathrm{d}x\right] \\ &= \mathbf{E}_{s(y)}\left[\int_{x=a}^{b} f^2(x)\mathrm{d}x\right] = \int_a^b f^2(x)\mathrm{d}x. \end{aligned} \tag{5}$$

3 Anti-aliased line drawing in 3D

3.1 Ray marching

Ray marching divides the line into N_{march} segments of equal length, i.e. the sample points are located at a predetermined distance of Δl from each other. Each sampled voxel contributes to the number of detector hits in the LOR with the same weight.

3.2 Filtered ray marching

With the filtered ray marching algorithm the voxel array is pre-filtered before sampling. We used the symmetric 3D filter kernel that gives weight $w = 0.9$ to the center voxel and $(1-w)/6$ to its face neighbors.

3.3 Siddon

The Siddon algorithm is based on the observation that the intersection points of an arbitrary straight line and a parallel set of the voxel bounding planes are always located at the same distance.

The coordinates of the current voxel are stored in the X, Y and Z integer variables. The t_x, t_y and t_z ray parameters keep track of the distance until the next intersection in their direction. The smallest of the three variables mark the direction where the line will intersect a voxel boundary next. In every step we move on to the next voxel along the axis belonging to the smallest parametric variable (i.e. increment X, Y or Z) and increase the affected parametric variable by a predetermined step value. The contribution of every visited voxel is weighted proportionally to the length of the line segment that falls in the voxel, i.e. the difference of the smallest parametric variables when the line stepped out of and stepped into the voxel.

3.4 Bresenham

The Bresenham algorithm iterates through the direction in which the line changes the most rapidly in unit steps. In every iteration the algorithm chooses whether the two coordinates belonging to the two other directions should be incremented or not based on which is the closest to the line. This is decided with the help of an error variable, which keeps track of the slope error of line sampling caused by the previous decisions.

3.5 Antialiased Bresenham

The Antialiased Bresenham complements the Bresenham algorithm with on-the-fly box filtering. For this purpose, the intersection of the one-voxel wide line segment and the voxel concerned has to be calculated. Considering first only the 2D case, we can conclude that a maximum of three voxels may intersect the line segment in each column if the slant is between 0 and 45 degrees (Fig. 1). Let the vertical distance of the three closest voxels to the center of the line be r, s and t respectively, and suppose $s < t \leq r$.

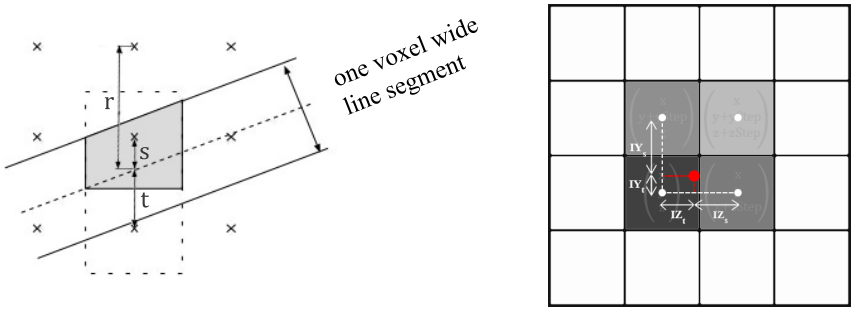


Fig. 1. Left: Box filtering of a line segment. Right: Linear interpolation in the Antialiased Bresenham algorithm reduces the number of memory accesses from 4 to 1 per iteration.

The areas of intersection, I_s, I_t and I_r depend not only on r, s and t, but also on the slant of the line segment. This dependence, however, can be rendered unimportant by using the following approximation:

$$I_s \approx (1 - s),\ I_t \approx (1 - t) = s,\ I_r \approx 0.$$

These formulae can be evaluated incrementally beside the Bresenham's incremental coordinate calculation. In every column, the contributions of the two sampled voxels are weighted with the area of their intersection with the line segment, i.e. I_s and I_t.

The 3D generalization of the Antialiased Bresenham algorithm samples 4 voxels in every iteration, 2-2 both horizontally and vertically. Each voxel gets two independent weights (one horizontal and one vertical weight), which are computed by two simultaneous 2D Antialiased Bresenham algorithms. These weights are then multiplied to form the final weight of the voxel's contribution.

On GPU, the built-in linear interpolation of the texture memory can be utilized in order to reduce the number of memory accesses from 4 to 1 per iteration. Fig. 1 shows how the horizontal and vertical weights are used in the interpolation of the four voxels.

3.6 Gupta–Sproull

Instead of box filtering, the Gupta-Sproull algorithm [2, 9] executes cone filtering. For this purpose, the volume of the intersection between the one-voxel wide line segment and the one-voxel radius cone centered around the voxel concerned has to be calculated. The height of the cone must be $3/\pi$ to guarantee that the volume of the cone is 1. Considering first only the 2D case, we can conclude that a maximum of three voxels may have intersection with a base circle of the cone in each column if the slant is between 0 and 45 degrees (Fig. 2).

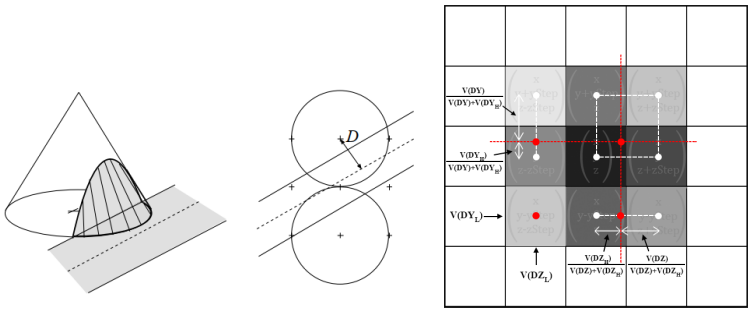


Fig. 2. Left: Cone filtering of a line segment. Right: Linear interpolation in the Gupta-Sproull algorithm reduces the number of memory accesses from 9 to 4 per iteration.

Let the distance between the voxel center and the center of the line be D. For possible intersection, D must be in the range of $[-1.5; +1.5]$. For each voxel the convolution integral - that is the volume of the cone segment above the voxel - depends only on the value of D, thus it can be computed for discrete D values and stored in a lookup table $V(D)$ during the design of the algorithm.

The 3D generalization of the Gupta-Sproull algorithm samples 9 voxels in every iteration, 3-3 both horizontally and vertically. Each voxel gets two independent weights (one horizontal and one vertical weight), which - like in the 3D Antialiased Bresenham - are computed by two simultaneous 2D algorithms.

These weights are then multiplied to form the final weight of the voxel's contribution, with the addition of a normalization factor to ensure that the sum of weights in the nine voxels equals to 1.

On GPU, the built-in linear interpolation of the texture memory can be utilized in order to reduce the number of memory accesses from 9 to 4 per iteration. Fig. 2 shows how the horizontal and vertical weights are used in the interpolation of the nine voxels.

3.7 Cylindrical Gupta–Sproull

Another variation of the Gupta–Sproull algorithm uses cylindrical filtering where the volume of the intersection between the one-voxel wide line segment and the one-voxel radius cylinder centered around the voxel concerned has to be calculated. This variation differs only in the values of the $V(D)$ lookup table from the original Gupta-Sproull algorithm.

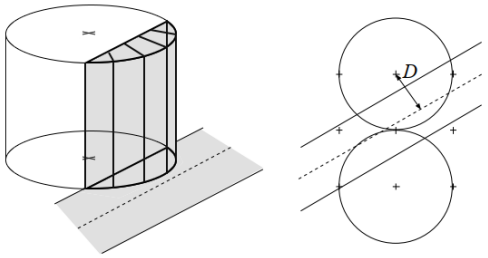


Fig. 3. Cylindrical filtering of a line segment.

4 Results

In our measurements, we compare the efficiency of the previously discussed line drawing algorithms to approximate the line integral of a given LOR during forward projection. The evaluation is performed by two programs written in C++. The first one runs on the CPU and was later transposed to the CUDA platform, which thus runs on the GPU and utilizes its massively parallel architecture and built-in services to accelerate the computations.

The measured volume is represented by a 3D voxel array. During the simulation, the number of hits of one single LOR is calculated separately with the different algorithms. This simplified model consists only of the two opposite detector crystals and the voxel array between them. The LOR is sampled by selecting 128 random $(\boldsymbol{z}_1, \boldsymbol{z}_2)$ point pairs following uniform distribution on the detector surfaces.

In our model, detector crystals have a 8×8 voxel wide surface. The distance of the two crystals were 128, 256 and 512 voxels respectively. In the voxel array a predetermined number of random voxels following uniform distribution were selected to have positive activity, while all others had zero activity. The examined parameter in this regard was the proportion of active voxels in the whole array (12.5%, 25%, 50% or 75%).

Our research included the measurement of the accuracy and the execution time of the implemented line drawing algorithms, which were then combined to form an efficiency index.

4.1 Accuracy

In Fig. 4 the relative L_1 error of the examined algorithms are plotted as a function of the number of samples on the detector surfaces. The reference value was computed with high precision (10^7 samples) using ray marching. To minimize the sampling error, each method is executed 20 000 times and an average of their result is calculated. On GPU, this is done by launching 20 000 threads (*gathering type* computation). Since relative error is a logarithmic function of the sample number, axes of the graphs are logarithmically scaled to make easier the differentiation of the error curves.

Table 1 summarizes the average error of each algorithm from the five measurements and sets up a ranking among them. It can be concluded that while the execution time on the CPU increases linearly with the size of the voxel array, the GPU implementation shows much better scaling.

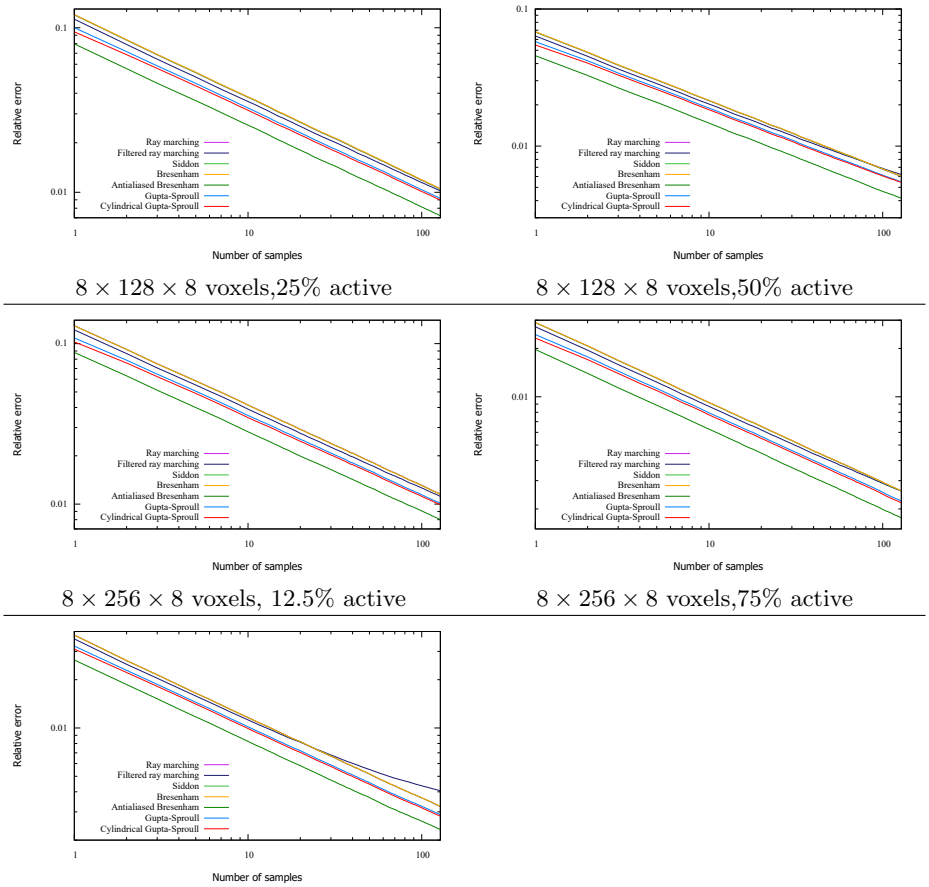
Rank	**Algorithm**	**Average error**
1	Antialiased Bresenham	0.004695
2	Cylindrical Gupta-Sproull	0.005867
3	Gupta-Sproull	0.005982
4	Siddon	0.006760
5	Ray marching	0.006765
6	Bresenham	0.006784
7	Filtered ray marching	0.006845

Table 1. Average errors from the five measurements.

In terms of accuracy, the Antialiased Bresenham algorithm achieves the best results, while the Cylindrical Gupta-Sproull comes out second best. For the filtered ray marching, filtering at the borders introduces significant error as the outside of the voxel array is not defined appropriately.

4.2 Speed

Execution times were measured both on CPU and GPU, and on the latter the impact of the built-in linear interpolation was also examined. Measurements



$8 \times 128 \times 8$ voxels,25% active

$8 \times 128 \times 8$ voxels,50% active

$8 \times 256 \times 8$ voxels, 12.5% active

$8 \times 256 \times 8$ voxels,75% active

$8 \times 512 \times 8$ voxels, 50% active

Fig. 4. Error curves obtained at different number of voxels and ratios of non-zero voxels. The most accurate algorithm is the Antialiased Bresenham in all cases.

lasted until all 20000×128 samples were calculated, and times were rounded to millisecond accuracy. Tables 2 -4 summarize the results on the three different voxel array sizes.

Algorithm	**CPU**	**GPU without interpolation**	**GPU with interpolation**
Ray marching	2831	49	63
Filtered ray marching	2837	49	63
Siddon	2241	113	116
Bresenham	1890	35	29
Antialiased Bresenham	3694	314	30
Gupta-Sproull	10699	791	500
Cylindrical Gupta-Sproull	10667	791	501

Table 2. Runtimes in milliseconds on an array of $8 \times 128 \times 8$ voxels.

Algorithm	**CPU**	**GPU without interpolation**	**GPU with interpolation**
Ray marching	5278	98	125
Filtered ray marching	5280	98	125
Siddon	3760	185	177
Bresenham	3292	64	58
Antialiased Bresenham	6982	602	59
Gupta-Sproull	20911	1628	998
Cylindrical Gupta-Sproull	20944	1628	998

Table 3. Runtimes in milliseconds on an array of $8 \times 256 \times 8$ voxels.

Among the examined algorithms, Bresenham is the fastest. By utilizing the texture memory and the built-in linear interpolation of the GPU, the Antialiased Breenham falls behind by just one millisecond. This represents a 10-fold increase in speed compared to the solution without interpolation and as a result, the Antialiased Bresenham has become the fastest antialiased algorithm.

The two versions of ray marching also achieve high speed. In filtered ray marching, filtering is executed only once at the beginning of the procedure, thus does not slow down the algorithm in the later phases.

Siddon is one of the fastest algorithms on CPU, however on GPU the frequent switching of voxels and the resulting intensive memory use cause a significant slowdown.

The two variations of the Gupta-Sproull algorithm rank last in terms of execution time as these methods run more than one order of magnitude longer than the other algorithms. This is due to the presence of complex operations (divisions, normalization) and the fact that they sample nine voxels in every

Algorithm	CPU	GPU without interpolation	GPU with interpolation
Ray marching	10214	117	228
Filtered ray marching	10222	177	228
Siddon	6671	304	317
Bresenham	6167	128	114
Antialiased Bresenham	13344	1204	115
Gupta-Sproull	41184	3161	1993
Cylindrical Gupta-Sproull	41334	3257	1994

Table 4. Runtimes in milliseconds on an array of $8 \times 512 \times 8$ voxels.

iteration, which, in addition to the calculation overhead, also involves a high number of memory accesses.

4.3 Efficiency

The results of accuracy and speed measurements were combined to determine the efficiency of the algorithms. For each algorithm in each of the five measurements an efficiency index was calculated as the reciprocal of the product of the squared error and the runtime. Table 5 summarizes the results.

Algorithm	Measurement				
	1	2	3	4	5
Ray marching	145	437	60	1191	417
Filtered ray marching	152	411	64	1197	266
Siddon	79	238	43	847	299
Bresenham	308	931	131	2576	828
Antialiased Bresenham	632	1879	265	5472	1599
Gupta-Sproull	24	66	10	200	60
Cylindrical Gupta-Sproull	25	67	10	211	63

Table 5. Efficiency indices calculated from five measurements.

Based on these results we came to the conclusion that the Antialiased Bresenham is the most effective algorithm among the examined seven algorithms.

5 Conclusion

This paper proposed a filtering method to decrease the variance of the integrand of the high dimensional integrals in the forward projection step of an iterative ML-EM algorithm. We proposed the application of low-pass filtering before the forward projections, while back projection still corrects the original unfiltered voxel array. We have proven that this approach does not compromise the reconstruction and preserves the stability even if high resolution voxel arrays are

reconstructed with a low number of Monte Carlo sampling. All steps are implemented on the GPU where the added computational cost of filtering is negligible with respect to forward and back projection calculations.

Acknowledgements

This work has been supported by OTKA K-124124, EFOP 4.2.1-16-2017-00021, and EFOP-3.6.2-16-2017-00013.

References

1. J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.
2. S. Gupta, R. Sproull, and I. Sutherland. Filtering edges for gray-scale displays. In *Computer Graphics (SIGGRAPH '81 Proceedings)*, pages 1–5, 1981.
3. Peter M. Joseph. An improved algorithm for reprojecting rays through pixel images. *IEEE Transactions on Medical Imaging*, 1(3):192–196, nov. 1982.
4. M. Magdics, L. Szirmay-Kalos, B. Tóth, and T. Umenhoffer. Filtered sampling for pet. In *2012 IEEE Nuclear Science Symposium and Medical Imaging Conference Record (NSS/MIC)*, pages 2509–2514, Oct 2012.
5. Bruno De Man and Samit Basu. Distance-driven projection and backprojection in three dimensions. *Physics in Medicine and Biology*, 49:2463–2475, 2004.
6. László Papp, Gábor Jakab, Balázs Tóth, and László Szirmay-Kalos. Adaptive bilateral filtering for PET. In *IEEE Nuclear science symposium and medical imaging conference*, MIC'14, pages M18–104, 2014.
7. R. L. Siddon. Fast calculation of the exact radiological path for a three-dimensional ct array. *Medical Physics*, 12(2):252–257, 1985.
8. L. Szirmay-Kalos, M. Magdics, and B. Tóth. Multiple importance sampling for PET. *IEEE Trans Med Imaging*, 33(4):970–978, 2014.
9. L. Szirmay-Kalos (editor). *Theory of Three Dimensional Computer Graphics*. Akadémia Kiadó, Budapest, 1995. http://www.iit.bme.hu/~szirmay.