Efficient X-ray projection of triangular meshes based on ray tracing and rasterization

P. Paramonov[a,b], J. Renders[a,b], T. Elberfeld[a,b], J. De Beenhouwer[a,b], and J. Sijbers[a,b]

[a]imec-Vision Lab, Department of Physics, University of Antwerp, Antwerp, Belgium
[b]DynXlab: Center for 4D Quantitative X-ray Imaging and Analysis, Antwerp, Belgium

ABSTRACT

X-ray based inspection often relies on triangular meshes, for example to inspect objects that were manufactured from CAD models. In this work, we present three complementary implementations of X-ray mesh projectors, obtained by adapting state-of-the-art rendering techniques to the simulation of X-ray imaging. The first technique is *rasterization*, where the interaction of each triangle with the X-ray beam is simulated in parallel using the NVIDIA CUDA toolkit. The second approach is *ray tracing*, where the interaction of each ray with the mesh is simulated in parallel using the NVIDIA OptiX framework. Both recursive and non-recursive versions of ray tracing are described. The simulated XCT setup is described in terms of a cone beam projection geometry that is compatible with the corresponding geometry in the ASTRA toolbox. All three projectors were benchmarked on a series of tests with varying resolution of both the mesh and the detector. The rasterizer exhibited the best computation time in most benchmark scenarios, coupled with the best scalability w.r.t. both the mesh size and the detector size. However, the recursive ray tracing approach offers more capabilities towards implementing additional optical effects such as refraction.

Keywords: X-ray imaging, rasterization, ray tracing, mesh model, GPU computing

1. INTRODUCTION

X-ray computed tomography (XCT) is widely applied in industry as a 3D imaging technique for non-destructive testing (NDT). A typical approach is based on a full 3D XCT imaging of the inspected sample, followed by a conversion of the reconstructed voxel representation to a surface mesh for comparison with a reference Computer Aided Design (CAD) model.[1,2] An alternative approach for X-ray based inspection compares X-ray projections of the sample with those simulated from either a reference voxel representation[3] or a CAD model.[4] Both options require efficient tools for accurate and effective simulation of forward X-ray projections of 3D meshes.

A well-known approach to simulate X-ray projections is based on Monte Carlo simulations.[5,6] Although being the most physically accurate, it is also extremely computationally expensive due to the need of casting a large number of photons as to keep the signal-to-noise ratio of the simulated projections at an acceptable level. It becomes particularly cumbersome for full XCT simulations, when radiographs have to be simulated from a large number of viewing angles. Hence, simpler techniques based on either ray tracing[7,8] or rasterization[9,10] have been proposed for both radiograph and XCT simulation. Both approaches assume that X-rays propagate strictly along straight lines. In the ray tracing approach, the attenuation detected by the corresponding pixel is approximated by a single line integral computed along the source-pixel path. The X-ray beam is then modelled by a set of rays that are cast towards the centers of the detector pixels. Another way of computing ray attenuation is by arranging the simulation in a "triangle-wise" order instead of "pixel-wise". This can be accomplished by rasterization, which is a rendering technique based on looping over the mesh triangles and finding pixels they cover.[11]

In this paper, we describe and analyze both ray tracing and rasterization algorithms for X-ray projection simulation. In our implementation, the representation of the projection geometry is compatible with the volumetric

Further author information: (Send correspondence to P.Paramonov)
P.Paramonov: E-mail: pavel.paramonov@uantwerpen.be

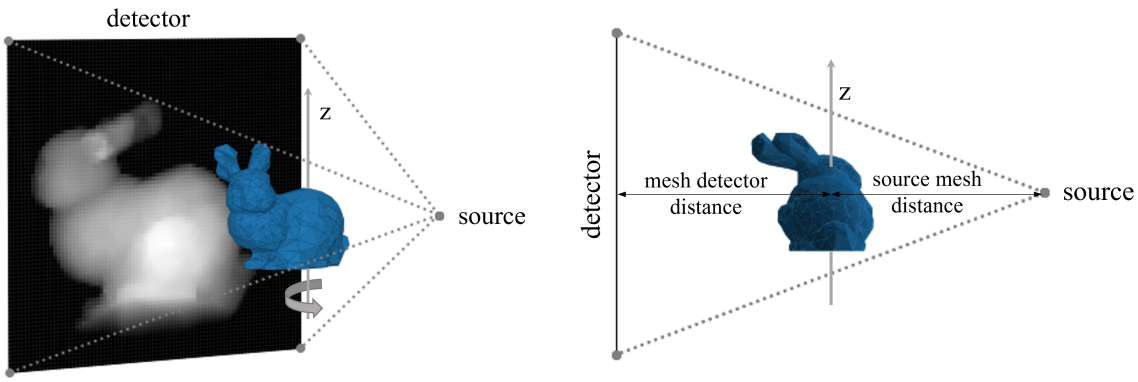


Figure 1: An example of the "Standford Bunny" mesh (10^3 faces) projected with the circular cone beam projection geometry.

projectors of the open source tomography toolbox ASTRA.[12] Firstly, we describe their effective implementations that leverage NVIDIA technologies for parallel computing (NVIDIA CUDA toolkit[13] for rasterization, and NVIDIA OptiX[14] for ray tracing). Then, we compare their efficiency in different simulation scenarios and discuss their strong and weak points.

2. ALGORITHMS FOR X-RAY PROJECTION SIMULATION

All of our projectors simulate the propagation of monochromatic X-rays along straight lines. A set of rays is generated from a point source, each ray aiming towards the center of the corresponding detector pixel. The detected attenuation I_i is found for each pixel as a sum of ray path segments l_j inside the mesh with the corresponding attenuation coefficient μ_j:

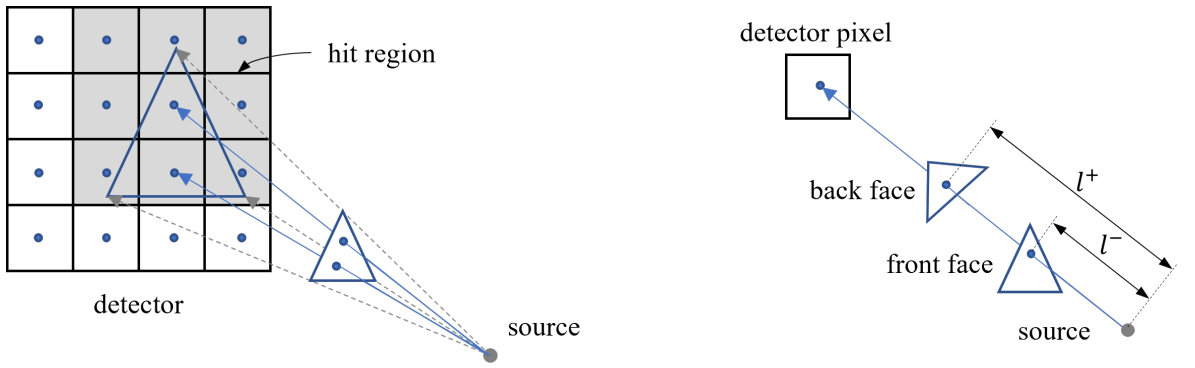
$$I_i = \sum_j \mu_j l_j. \tag{1}$$

To obtain correct projections, all projectors require the mesh to be watertight. Although our implementation does not support functionally graded materials by, e.g., representing them with adaptive tetrahedral meshes,[15] projection of heterogenous objects is still possible by nesting the meshes that correspond to homogeneous material regions.

To describe the XCT imaging setup, our implementation relies on the notation of a projection view and a projection geometry, as it is defined in the ASTRA toolbox. The projection view is a combination of a source position, detector position, size, and orientation. Running the projector on a single projection view results in a single radiograph. The projection geometry combines multiple projection views, necessary for obtaining a sinogram. In this work, a circular cone beam projection geometry with a point source was used in the computational experiments. It is described by the distances from the mesh to the detector and from the source to the detector, the size and the resolution of the detector, and a set of projection angles (see Fig. 1).

2.1 Rasterization

This algorithm projects each triangle to the detector plane, and computes its input to the ray attenuation for the pixels that it covers, as it is depicted in Fig. 2. To that end, each of the three triangle vertices are projected onto the detector, determining the hit region. For all the pixels in the hit region, it is determined if the pixel center is inside the projected triangle. For those pixels that are inside the projected triangle, the distance from the source to the triangle face is computed, multiplied by the attenuation coefficient, and added to the value recorded at that pixel. If the triangle face normal points towards the source (front face), the distance value is assigned a negative sign (l^- in Fig. 2b). If the normal points away (back face), the distance value is positive (l^+ in Fig. 2b).



(a) Detector hit region

(b) Face inputs to the detected attenuation

Figure 2: Steps in the rasterization algorithm: (a) hit region (gray area) is found, (b) face input is added to the pixel attenuation.

Our implementation provides a CUDA kernel that implements this procedure for every triangle of the mesh. The total sum of the distances that equates to the distance the simulated ray travels through the object is found by applying the CUDA atomic sum operation. Because all the ray-triangle intersections are processed in an unordered way, rasterization needs the information about mesh hierarchy to be given explicitly to be able to project nested meshes. The same reason makes it difficult to add refraction and reflection into rasterization.

2.2 Non-recursive ray tracing with NVIDIA OptiX

By "non-recursive" ray tracing we refer to a procedure that runs only once per every ray, and allows to find all the intersections along the ray path. To that end, we implemented two kinds of OptiX programs: *ray generation* and *any-hit*. In the *ray generation* program, the ray origin and initial direction is calculated according to the chosen projection geometry and current projection view. All rays are cast from a point source towards a regular grid of detector pixels. In the *any-hit* program, the same rule as in rasterization algorithm is applied to every ray-triangle intersection, i.e., l^- or l^+ distance is found and added to the pixel attenuation (see Fig. 2b).

The non-recursive OptiX projector is similar to rasterizer is the sense that it performs the same operation but in the different order. Just like the rasterization algorithm, it requires prior information on mesh hierarchy for the correct projection of nested meshes.

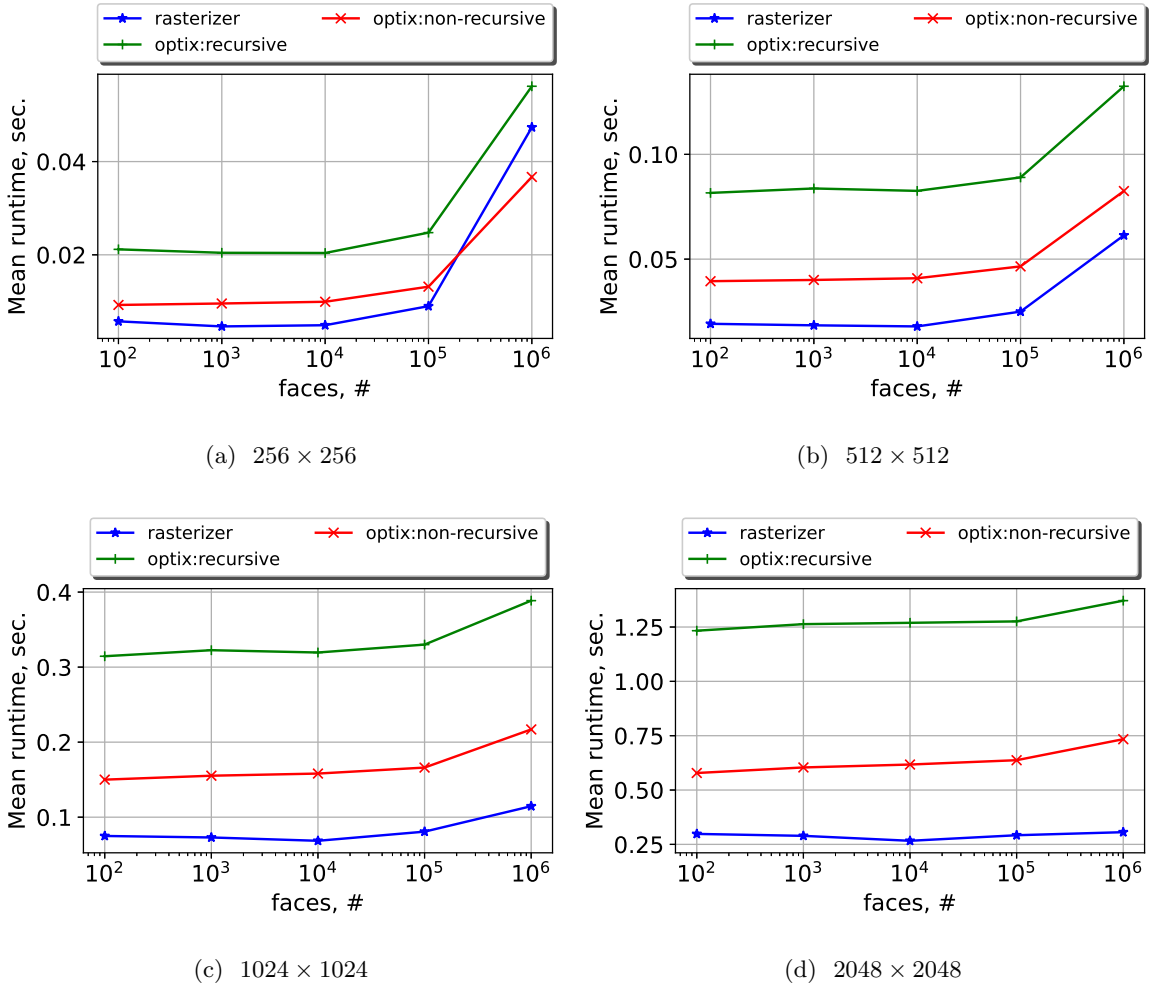
2.3 Recursive ray tracing with NVIDIA OptiX

Similar to the non-recursive case, ray tracing starts with *ray generation* program, but instead of finding all the intersections in one run, only the hit points closest to the source are calculated. These hit events are processed in the OptiX *closest-hit* program, after which tracing starts again from the hit point. This recursive procedure terminates when no more intersections can be found, i.e., when OptiX reports a ray miss event and executes *miss* program.

Unlike the previous two projectors, recursive ray tracing automatically keeps track on the mesh hierarchy. On top of easier support for the nested meshes, the direction of the ray can be updated using Snell's at every intersection, if refraction is considered.

3. SIMULATION RESULTS

To benchmark the implemented projectors, we prepared a series of Stanford bunny-based meshes with a varying number of faces - from 10^2 to 10^6 (cfr. Fig. 1). To measure execution time, we ran every simulation 10 times and took the average recorded time. For the OptiX projectors, we pre-launched the scripts without time measurement to let OptiX run JIT compilation of the PTX code before the benchmark starts. Every run simulated a circular cone beam XCT setup with 250 projection angles and varying detector resolution. In Fig. 3, a comparison between the computation times of XCT simulations with different projectors is shown for various detector resolutions. All



(a) 256×256

(b) 512×512

(c) 1024×1024

(d) 2048×2048

Figure 3: Computation time comparison for the different detector size.

forward projection simulations were run on a workstation with a GeForce RTX 2080 graphics card. The operating system was Ubuntu 20.04, the versions of NVIDIA CUDA and NVIDIA OptiX are 10.2 and 7.4 respectively, the display driver version was 510.73.05.

In all performance tests except one, the rasterizer exhibits lower computational time than both of the OptiX-based projectors. Unlike ray-tracing implementations, it scales well with respect to the detector resolution. The only scenario when rasterization was slower than the non-recursive OptiX projector was with a combination of high-poly mesh with large detector pixels. This causes many faces to overlap from the viewpoint of a single detector pixel, which results in a great number of collision tests in atomic summation and increased waiting time for the CUDA threads. The scalability w.r.t. the mesh size for all of the projectors peaks for the largest detector size, with the rasterizer showing almost no growth in computational time up to 10^6 faces. It also demonstrated the best scalability w.r.t. the detector size: the rasterizer projector slowed down ca. 20 times on a switch from 256×256 to 2048×2048 pixels, while the same switch resulted in about 40 times slower execution for the recursive OptiX projector, and about 44 times longer computation time for the non-recursive OptiX projector.

Although ray tracing does not perform as well as rasterization, the main advantage of the latter technique is the ability to implement more sophisticated light propagation models. Light refraction and reflection can be naturally included into the recursive implementation. On top of that, our implementation of the recursive OptiX

projector supports automatic handling of nested meshes, while both rasterization and non-recursive ray tracing require either the user to explicitly specify the information on mesh hierarchy, or a separate routine that deducts the hierarchy.

4. CONCLUSION

In this paper, three approaches to efficiently simulate X-ray projections from triangular meshes, as well as their GPU implementations, were described and benchmarked. The first approach is based on rasterization and was implement with the NVIDIA CUDA toolkit, while the others are the two versions of the ray tracing technique and were implemented with the NVIDIA OptiX framework. The rasterization-based projector demonstrated the lowest computational time in most simulation scenarios coupled with good scalability with respect to the detector resolution and mesh size. However, it has limited capabilities for further improving the X-ray propagation model (e.g., by including refraction), which limits its application to the domain of conventional XCT. On the other hand, the light propagation model can be relatively easily improved in the recursive OptiX projector, which can make it applicable to other types of XCT, such as edge illumination phase contrast imaging.

ACKNOWLEDGMENTS

The research was supported by the Research Foundation- Flanders (FWO)(G090020N, G0F9117N, G094320N, S003421N, 11D8319N) and EU Interreg Flanders-Netherlands Smart*Light (0386).

REFERENCES

[1] De Chiffre, L., Carmignato, S., Kruth, J.-P., Schmitt, R., and Weckenmann, A., "Industrial applications of computed tomography," *CIRP Annals - Manufacturing Technology* **63**(2), 655 – 677 (2014).

[2] Thompson, A., Maskery, I., and Leach, R., "X-ray computed tomography for additive manufacturing: A review," *Measurement Science and Technology* **27**(7) (2016).

[3] van Dael, M., Verboven, P., Dhaene, J., Van Hoorebeke, L., Sijbers, J., and Nicolai, B., "Multisensor X-ray inspection of internal defects in horticultural products," *Postharvest Biology and Technology* **128**, 33–43 (2017).

[4] Presenti, A., Sijbers, J., and De Beenhouwer, J., "Dynamic few-view X-ray imaging for inspection of CAD-based objects," *Expert Systems with Applications* **180** (2021).

[5] Langer, M., Cen, Z., Rit, S., and Létang, J. M., "Towards Monte Carlo simulation of X-ray phase contrast using GATE," *Optics Express* **28**(10), 14522 – 14535 (2020).

[6] Sanctorum, J., De Beenhouwer, J., and Sijbers, J., "X-ray phase contrast simulation for grating-based interferometry using GATE," *Optics Express* **28**(22), 33390–33412 (2020).

[7] Jacobs, F., Sundermann, E., De Sutter, B., Christiaens, M., and Lemahieu, I., "A fast algorithm to calculate the exact radiological path through a pixel or voxel space," *Journal of Computing and Information Technology* **6**(1), 89 – 94 (1998).

[8] Vidal, F. P., Garnier, M., Freud, N., Létang, J. M., and John, N. W., "Simulation of X-ray attenuation on the GPU," in [*Theory and Practice of Computer Graphics 2009, TPCG 2009 - Eurographics UK Chapter Proceedings*], 25–32 (2009).

[9] Kooa, J., Dahla, A. B., Bærentzena, J. A., Chenb, Q., Balsb, S., and Dahla, V. A., "Shape from projections via differentiable forward projector for computed tomography," *Ultramicroscopy* **224**, 113239 (May 2021).

[10] Renders, J., De Beenhouwer, J., and Sijbers, J., "Mesh-based reconstruction of dynamic foam images using X-ray CT," in [*2021 International Conference on 3D Vision (3DV)*], 1312–1320, IEEE (2021).

[11] Parker, S. G., Friedrich, H., Luebke, D., Morley, K., Bigler, J., Hoberock, J., McAllister, D., Robison, A., Dietrich, A., Humphreys, G., McGuire, M., and Stich, M., "GPU ray tracing," *Communications of the ACM* **56**(5), 93 – 101 (2013).

[12] van Aarle, W., Palenstijn, W. J., Cant, J., Janssens, E., Bleichrodt, F., Dabravolski, A., Beenhouwer, J. D., Batenburg, K. J., and Sijbers, J., "Fast and flexible X-ray tomography using the ASTRA toolbox," *Opt. Express* **24**, 25129–25147 (Oct 2016).

[13] Nickolls, J., Buck, I., Garland, M., and Skadron, K., "Scalable parallel programming with CUDA," *Queue* **6**(2), 40 – 53 (2008).

[14] Parker, S. G., Bigler, J., Dietrich, A., Friedrich, H., Hoberock, J., Luebke, D., McAllister, D., McGuire, M., Morley, K., Robison, A., and Stich, M., "OptiX: A general purpose ray tracing engine," *ACM Transactions on Graphics* **29**(4) (2010).

[15] You, Y., Kou, X., and Tan, S., "Adaptive tetrahedral mesh generation of 3D heterogeneous objects," *Computer-Aided Design and Applications* **12**(5), 580 – 588 (2015).