X-ray synthesis based on triangular mesh models using GPU-accelerated ray tracing for multi-modal breast image registration

J. Maul[1], S. Said[1], N. Ruiter[1], and T. Hopp[1]

Karlsruhe Institute of Technology, Institute for Data Processing and Electronics, Karlsruhe, Germany

Abstract. For image registration of breast MRI and X-ray mammography we apply detailed biomechanical models. Synthesizing X-ray mammograms from these models is an important processing step for optimizing registration parameters and deriving images for multi-modal diagnosis. A fast computation time for creating synthetic images is essential to enable a clinically relevant application. In this paper we present a method to create synthetic X-ray attenuation images with an hardware-optimized ray tracing algorithm on recent graphics processing units' (GPU) ray tracing (RT) cores. The ray tracing algorithm is able to calculate the attenuation of the X-rays by tracing through a triangular polygon-mesh. We use the Vulkan API, which enables access to RT cores. One frame for a triangle mesh with over 5 million triangles in the mesh and a detector resolution of 1080×1080 can be calculated and transferred to and from the GPU in about 0.76 seconds on NVidia RTX 2070 Super GPU. Calculation duration of an interactive application without the transfer overhead allows real time application with more than 30 frames per second (fps) even for very large polygon models. The presented method is able to calculate synthetic X-ray images in a short time and has the potential for real-time applications. Also it is the very first implementation using RT cores for this purpose. The toolbox will be available as an open source.

Keywords: X-ray Simulation, Ray Tracing, GPU, Triangular Mesh, Multi-modal Image Registration, Bio-mechanical Model

1 Background

Image registration is a crucial step for image analysis because valuable information from different images can be combined. The goal of image registration is to find corresponding locations in two or more images and combine the information of these images [9]. Therefore, the accurate integration of the information from two or more images is important for the quality of the result [7].

For early breast cancer diagnosis often multiple modalities are applied. To combine those modalities an image registration is important due to the considerably different patient positioning, dimensionality and deformation state of the breast. To tackle these huge non-linear deformations we develop image registration methods based on biomechanical models of the breast, e.g. for combining the diagnostic values of Magnetic Resonance Images (MRI) and X-ray mammograms. For this purpose a synthetic X-ray image has to be generated from the biomechanical model, which is represented by a polygoal mesh, in order to compare it to the real mammogram for iterative optimization of registration parameters and final combined diagnosis [4][5][10]. For future application in which e.g. biomechanical model generation may be computed on GPUs, the computation time is essential to enable a clinically relevant application.

Beside our own application, the demand for synthetic X-ray images is high because it not just enables image registration but also a wide variety of medical imaging applications such as evaluation of CT reconstruction algorithms, and can furthermore be used e.g. in the field of non-destructive testing [3].

The geometry of (biomechanical) anatomical models are often described with polygon meshes. Typically the polygons are triangles, rectangles or hexagons. In our image registration method, biomechanical models are automatically created from segmented MRI volumes using the iso2mesh [1] toolbox which is based on tetgen [11] and Computational Geometry Algorithms Library (CGAL) [8]. It produces tetrahedral meshes which are subsequently applied for a deformation simulation with FEM. These polygon meshes allow the use of ray tracing algorithms for calculation of the interaction of X-rays with the tissue in order to create synthetic X-ray images.

The available toolboxes for the calculation of synthetic X-ray images are mostly based on volume images using ray casting, thereby requiring an additional processing step to convert the mesh into a voxel volume in case of image registration application [13].

Few implementations exist with respect to X-ray simulation based on polygon meshes. Due to the similarity to ray tracing applications in computer vision, these methods are typically well suited for parallel processing on GPUs. An example for such a ray tracing based algorithm on GPU and with L-Buffers is presented in [12]. The computation time demonstrated is approx. 10 ms for a detector resolution of 1024×768 pixels and a mesh with 11,102 triangles.

An important motivating factor for this work was the development of ray tracing (RT) cores in recent graphics card generations. Because ray tracing is often considered as too computationally intensive, RT cores accelerate the ray tracing principle with hardware-optimized intersection calculation. To our best knowledge, there is no toolbox available using the RT cores of recent GPU generations for X-ray simulations in medical application. This work therefore aims at incorporating the recent advances in GPU computing for fast generation of synthetic X-ray images from potentially very large polygon meshes. In order to combine the advantages of GPU accelerated computing and prototyping in scripting languages, such as MATLAB, we made use of the Vulkan API [14] for implementation of the ray tracing functionality and furthermore developed a MATLAB wrapper to evaluate the results and enable further processing.

2 Methods

2.1 Basic principle

X-rays traveling through a material are attenuated depending on the material characteristics. The received X-ray intensity at a detector depends on the initial intensity of the X-ray source, the absorption property of the material and the distance it travels trough the material. This relationship can be expressed by the Beer-Lambert-Law, which is the basic physical principle we use for our application. The Beer-Lambert-Law is defined as

$$I = I_0 \exp\left(- \int \mu(x_n) dx \right) \tag{1}$$

where I_0 represents the initial intensity of the X-ray emitted at the X-ray source, I_n represents the intensity received at a detector position n. Nowadays digital detectors are mainly used for data acquisition, which discretize the field of view into pixels. x_n expresses a position on the ray connecting source and detector. The function $\mu(x_n)$ describes the corresponding attenuation coefficient at x_1. The integral integrates over the traversed part in the material.

Assuming locally isotropic materials, the integral can be discretized as the sum of the product of the partial distances a ray travels through one material and the material's X-ray attenuation coefficient.

To determine the partial distances, which a ray travels through the respective materials in a polygon mesh, intersections of the ray and the polygons need to be calculated, which is a classical ray tracing problem known from computer vision. Once intersection points are known, the partial distance can be calculated by the Euclidean distance between the intersection points. Expressing one ray by a directional vector $\overrightarrow{d}$ from source to detector, all intersection points can be described by $t \cdot \overrightarrow{d}$ where t is a scalar describing the proportional distance from the source. Following this approach the Beer-Lambert law can be re-written as

$$I_n = I_0 \exp\left(- \sum_{k=1}^{(m-1)} \left(\| (t_{k+1} - t_k) \overrightarrow{d} \| \, \mu(x_{m,n}) \right) \right) \tag{2}$$

where m is the number of intersections between ray and polygon mesh and t_k is the relative distance from the source to a specific intersection. An visual example is given in Figure 2.

In our implementation we define a point source for the X-rays. Rays originating from the source are tracked through the polygon mesh using ray tracing until they hit a detector pixel. For this purpose a directional vector from the source to a detector is calculated. Refraction is currently neglected and hence the tracing is performed along straight lines. The expansion of an X-ray is not considered. Detector pixels are modeled as point detectors with an ideal transfer function.

The polygon mesh represents the surfaces of different tissues. Multiple surfaces can be interlaced. In our data structure we encode the material type by assigning region IDs to every polygon, which together with a lookup table are able to identify the attenuation coefficient for the respective material.

2.2 Implementation on special purpose ray tracing units of GPUs

For implementing the algorithms on the special purpose RT cores of GPUs we use the Vulkan API, which is a standardized API for GPU applications provided by the Khronos group [14]. Vulkan executes commands on driver level and allows cross-platform implementation. To achieve compatibility with different GPU brands, it is possible to let Vulkan take over the execution of functions called by the user.

Vulkan provides special data structures for ray tracing problems. The polygon mesh is represented by two entities (nodes and faces). This data structure is also known as face-vertex mesh. These two entities can be forwarded to Vulkan, which creates an acceleration structure (AS). The node entity contains the coordinates of the vertices of a polygon. The faces entity holds the indices of vertices, which are connected to polygons. To use hardware accelerated intersection calculation provided by RT cores, the polygon shapes are restricted to triangles.

The implementation is based on the well known examples provided by Willems [15]. For generating and handling rays, three shaders are used. The “raygen shader” calculates the ray direction $\overrightarrow{d}$ and defines the parameter of the ray. After that the raygen shader starts the ray tracing with an Vulkan API call $traceNV$. The Vulkan library then executes a selected shader whenever an intersection between the ray and the AS is detected. We use the “anyhit shader” to acquire all information about the intersection such as the material type $\mu(x)$ and the distance t, calculated with full precision (32 bit). After all intersections are detected or after a ray does not further intersect with another triangle, the “miss shader” will be called by Vulkan. In this shader we implement the attenuation calculation by sorting the intersections and summing up the product of distances and attenuation coefficients according to the Beer-Lambert law given in equation Equation 2.

Figure 1 illustrates the implementation of shaders for the described functionality. The construction of the AS and the execution of the shaders are performend by the Vulkan library. In order to start the ray tracing shaders, the host calls the $QueueSubmit$ function.

We implemented the ray tracing principle with Vulkan in a headless-mode, i.e. the application does not display the resulting images. Instead we integrated the method into the MATLAB scripting language via a MEX interface.

3 Results

3.1 Validation of attenuation calculation

In order to validate the implementation we calculated the attenuation of an analytical example. For this purpose, triangles belonging to different tissue types are

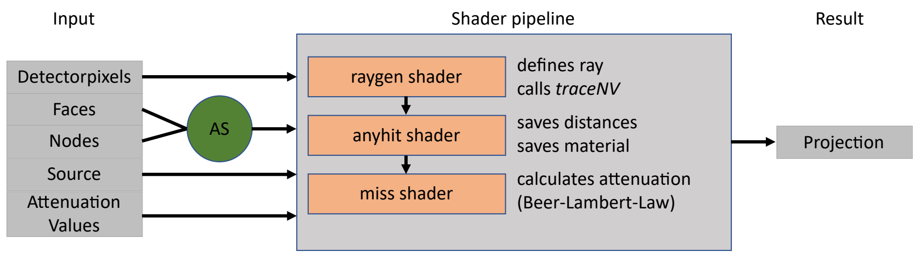

Fig. 1. Simplified X-ray calculation. The parameters are passed via a MEX interface, the AS is built by Vulkan. Then the shaders get executed, which calculates the attenuation with the Beer-Lamber-Law. The result is an synthetic X-ray projection.

placed perpendicular to a ray. We define one detector pixel and source position. Using the analytic positions of source, detector and triangles makes it possible to calculate the analytic attenuation and test the result of the implementation against it (Figure 2).

The deviation is in the range of the data type precision for every tested analytical example. For the example given in Figure 2 the relative error of the computed attenuation was $5.7 \times 10^{-8}\,\%$. The analytic example can be recreated and further tested as a part of the open source toolbox.

In Figure 3 the synthetic X-ray images generated by the presented method for four examples are shown for visual assessment. The breast example shows small artifacts at the bottom edge caused by holes in the underlying mesh and not by the toolbox.

To demonstrate the application for image registration based on biomechanical models, Figure 4 shows a synthetic X-ray image based on the deformed biomechanical model compared to an real X-ray mammogram. We can see, that the shape of the breast and the inner structures look similar, even though the fine structures are not visible in the synthetic X-ray image due to limited resolution of the underlying MRI data.

3.2 Performance evaluation

To evaluate the computational performance we created several examples with a varying number of triangles and a varying resolution of the detector. Both parameters have a direct effect on the calculation time.

Table 1 presents the different properties of the examples and the execution times for one image. Both the computation time with and without data transfer to and from the GPU are illustrated.
Thereby we created two scenarios: (1) Direct application with MATLAB MEX interface for calculating one synthetic X-ray image, (2) interactive application in which only the rendering for the X-ray synthesis on the GPU is considered and data transfer to/from the GPU are neglected. This second scenario may

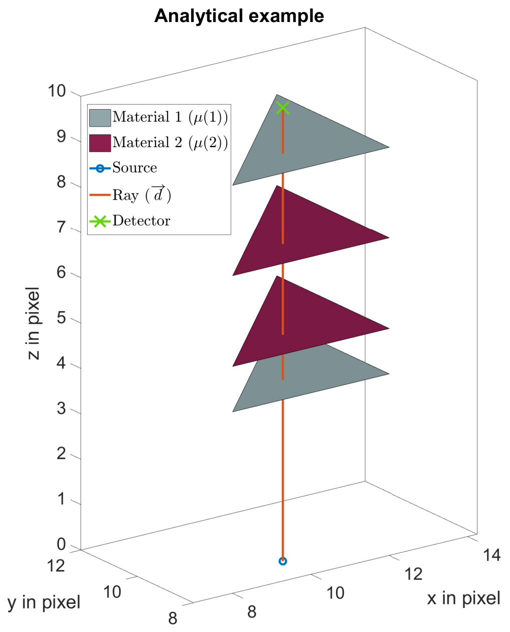

Fig. 2. Structure of the analytical example to determine the accuracy with two layers. The blue circle represents the source. The triangles are at the material surface in a mesh and the different colours represent different material types. The orange line represents the ray and the end is e.g. at the detector plate or at an defined maximum length.

be thought of e.g. a real time renderer with varying source position while the polygon mesh data is not changed.

All calculations are performed on a NVidia RTX 2070 Super GPU on a Windows workstation PC.

A basic observation from Table 1 is the considerable overhead of the data transfer to and from the GPU by comparing the calculation times of scenario 1 and scenario 2. With increasing number of triangles the calculation time in scenario 2 increases. Furthermore it can be noticed that e.g. the "veins" example with considerably less triangles than the "breast" example, but more pixels on the detector plane takes longer to compute for scenario 1.

To further investigate the dependencies of the computation time, the geometry used in the "breast" example was taken as a basis to create polygon meshes with different number of triangles. Additionally the detector resolution was varied. Figure 5 shows the results of this analysis in frames per second (fps).

As we can see, the detector resolution has a stronger influence on the computational performance than the number of triangles, because the higher resolution leads to more possible intersections as more rays are generated and more hits

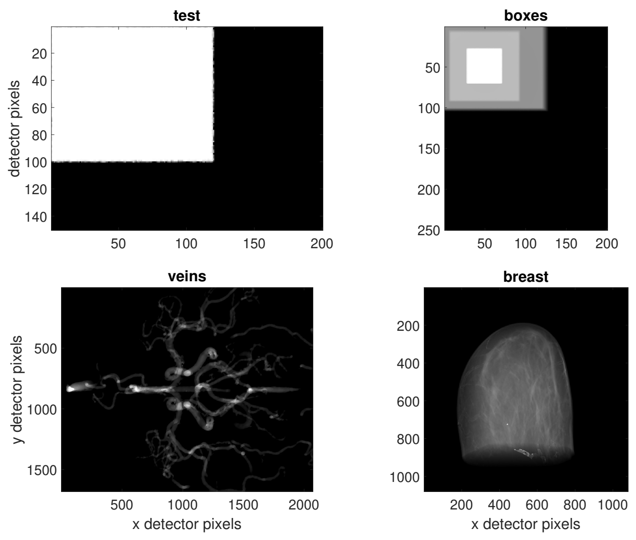

Fig. 3. Exemplary results of the X-ray simulation for the examples from Table 1. The "test" example models a simple plate made from one material. The "boxes" example models three interleaved boxes of different material. The "veins" example was provided by [6] and displays a part of the veins in the human body. The "breast" example models the geometry of a breast based on a segmented MRI scan as typically used in our biomechanical model based image registration.

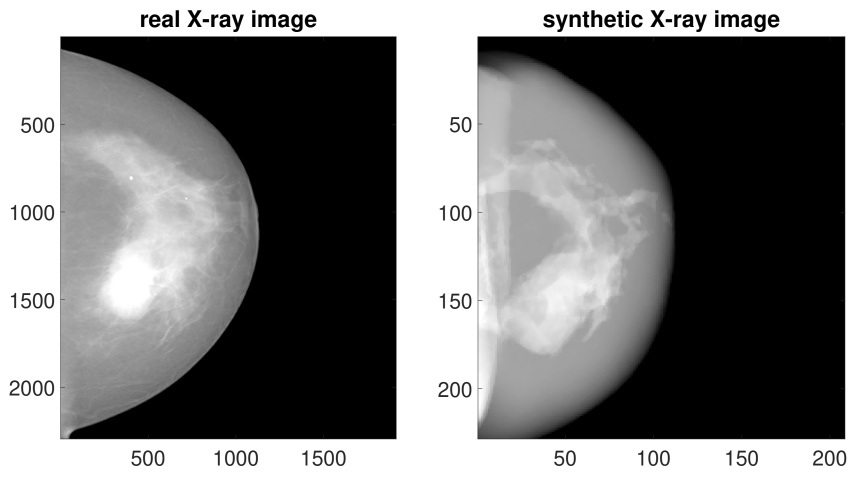

Fig. 4. Comparison between real and synthetic X-ray image.

Example	Triangles	Pixels	Calc. time (scen. 1)	Calc. time (scen. 2)
test	15,626	150×200	0.31 ± 0.0096	0.0009 ± 0.000087
boxes	47,580	250×200	0.31 ± 0.0089	0.0014 ± 0.000088
veins[6]	215,728	1681×2071	0.86 ± 0.0117	0.0041 ± 0.000031
breast	5,185,459	1080×1080	0.76 ± 0.0117	0.025 ± 0.0018

Table 1. Calculation time results with different examples. Properties of the examples (first column), number of triangle in the polygon mesh, number of pixels on the detector. Times are given for one frame in seconds as mean of 100 measurements ± the standard deviation. The first calculation time shows the time with transfer from and to MATLAB (i.e. scenario 1), the second calculation time shows the time for computing 1 frame without data transfer, e.g. real time application (i.e. scenario 2).

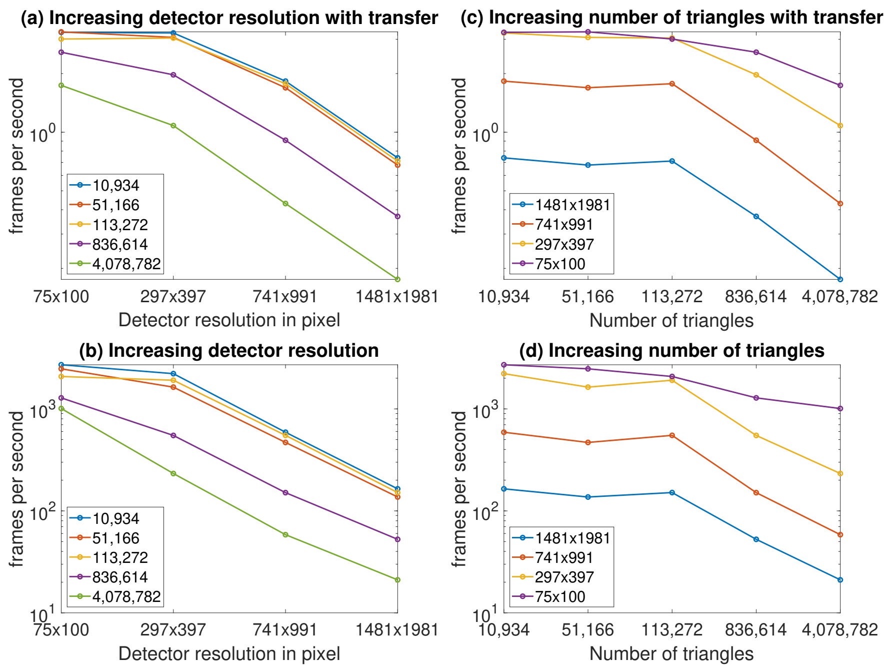

Fig. 5. Rendering times in frames per second for different detector resolutions and different amount of triangles in the polygon mesh. Left column: dependency on detector resolution. (Resolutions: $75 \times 100 = 0.911$mm/px, $297 \times 397 = 0.23$mm/px, $741 \times 991 = 0.091$mm/px, $1481 \times 1981 = 0.046$mm/px) Right column: dependency on number of triangles. Top row: results for scenario 1. Bottom row: results for scenario 2.

are detected. The difference in fps is small up to the 113,000 triangle example for both scenario 1 and 2, while a constant decrease in fps can be observed for increasing detector resolution.

For all examples, the overall fps is considerably higher without data transfer. In scenario 1, for large problems (around 4 million triangles, detector resolution 1481×1981 pixels) the fps is nearly 1, while small problems (around 11,000 triangles, detector resolution 75×100 pixels) reach about 30 fps. Scenario 2 reaches up to 21 fps for the large problem and up to 2700 fps for the small one.

Deeper analysis of the differences between scenario 1 and scenario 2 showed that the data transfer and the initialization of the Vulkan library need more than 95% of the computation time of one frame in scenario 1. As these steps have to be done only once for a real time application, this explains the huge time differences between scenario 1 and scenario 2.

4 Discussion and Conclusion

In this paper we presented a method for synthesizing X-ray attenuation images based on polygon meshes using ray tracing. To our best knowledge, the provided toolbox is the first example using RT cores for such purposes as other toolboxes concentrate on memory-management [12][13] or volume based methods [2]. Because this is the first approach to use RT cores to generate synthesizing X-ray attenuation images, this paper focuses on the implementation. In future, comparison with other proposals will be made in more detail. With the MEX interface an easy access from a high level scripting language, which is often used in scientific applications, is possible.

The main purpose of this method is to accelerate the biomechanical model based image registration of MR images and X-ray mammograms in future. While this has been easily achieved compared to the former applied method using ray casting through a voxel grid [4], the real time potential of the method is even higher and may open up new fields of application. The method has limitations, like a straight ray approximation, negligence of refraction and considering a monochromatic case only. However these limitations may be overcome in future, because e.g. refraction is common in ray tracing problems in computer vision and may as such be easily introduced in the algorithm.

The computational performance reached by the current implementation is very promising. Further optimization may concern the data transfer to and from the GPU as well as potentially reducing the overhead by the Vulkan library. Furthermore in the current implementation the execution time of the miss shader needs between 30% and 50% of the total rendering time for one frame. This may be optimized by replacing the current sorting method of intersections with e.g. the approach presented in [12].

Even though the resulting toolbox was originally developed for synthesizing X-ray images for our model-based image registration, the performance of the GPUs has the potential, that the simulator can be used as a real time application for other purposes than registration, as we demonstrate in the result section.

The presented methods have been assembled in a toolbox such that it can be used in a wide field of biomedical applications. To achieve this goal, we share this toolbox as open source project including all examples presented in this paper, which can be accessed at the following web page:

`https://git.scc.kit.edu/dach/raytracingx-rayprojectionsimulator`

References

1. Fang, Q., Boas, D.A.: Tetrahedral mesh generation from volumetric binary and grayscale images. In: 2009 IEEE ISBI: From Nano to Macro. pp. 1142–1145 (2009). https://doi.org/10.1109/ISBI.2009.5193259
2. Folkerts, M., Jia, X., Gu, X., Choi, D., Majumdar, A., Jiang, S.: MO-FF-A4-05: Implementation and Evaluation of Various DRR Algorithms on GPU. Medical Physics **37**(6Part6), 3367 (2010). https://doi.org/10.1118/1.3469159
3. Hanke, R., Fuchs, T., Uhlmann, N.: X-ray based methods for non-destructive testing and material characterization. Nucl. Instrum. Methods Phys. Res. **591**(1), 14–18 (2008). https://doi.org/10.1016/j.nima.2008.03.016
4. Hopp, T., Dietzel, M., Baltzer, P.A., Kreisel, P., Kaiser, W.A., Gemmeke, H., Ruiter, N.V.: Automatic multimodal 2D/3D breast image registration using biomechanical FEM models and intensity-based optimization. Medical image analysis **17**(2), 209–218 (2013). https://doi.org/10.1016/j.media.2012.10.003
5. Hopp, T., Baltzer, P., Dietzel, M., Kaiser, W.A., Ruiter, N.V.: 2D/3D image fusion of X-ray mammograms with breast MRI: visualizing dynamic contrast enhancement in mammograms. IJCARS **7**(3), 339–348 (2012). https://doi.org/10.1007/s11548-011-0623-z
6. Lickteig, S.: Segmentierung & Visualisierung von Blutgefäßen
7. Oliveira, F.P.M., Tavares, J.M.R.S.: Medical image registration: a review. CMBBE **17**(2), 73–93 (2014). https://doi.org/10.1080/10255842.2012.670855
8. Rineau, L., Yvinec, M.: 3D Surface Mesh Generation (2008), `https://doc.cgal.org/latest/Surface_mesher/index.html`, 26.07.2021
9. Rueckert, D., Schnabel, J.A.: Medical Image Registration. In: Deserno, T.M. (ed.) Biomedical image processing, pp. 131–154. Biological and Medical Physics, Biomedical Engineering, Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-15816-2_5
10. Said, S., Clauser, P., Ruiter, N.V., Baltzer, P.A.T., Hopp, T.: Image registration between MRI and spot mammograms for X-ray guided stereotactic breast biopsy: preliminary results. In: Linte, C.A., Siewerdsen, J.H. (eds.) Medical Imaging 2021: Image-Guided Procedures, Robotic Interventions, and Modeling. p. 45. SPIE (15022021 - 20022021). https://doi.org/10.1117/12.2581820
11. Si, H., Gärtner, K.: Meshing Piecewise Linear Complexes by Constrained Delaunay Tetrahedralizations. 14th IMR (2005). https://doi.org/10.1007/3-540-29090-7_9
12. Vidal, F.P., Garnier, M., Freud, N., Létang, J.M., John, N.W.: Simulation of X-ray Attenuation on the GPU. In: Wen Tang, John Collomosse (eds.) Theory and Practice of Computer Graphics. The Eurographics Association (2009). https://doi.org/10.2312/LocalChapterEvents/TPCG/TPCG09/025-032
13. Vidal, F.P., Villard, P.F.: Development and validation of real-time simulation of X-ray imaging with respiratory motion. CMIG **49**, 1–15 (2016). https://doi.org/10.1016/j.compmedimag.2015.12.002
14. Vulkan: Home — Vulkan — Cross platform 3D Graphics (2021), `https://www.vulkan.org/`, 20.05.2021
15. Willems, Sascha: SaschaWillems/Vulkan Examples (28052021), `https://github.com/SaschaWillems/Vulkan`, 28.05.2021