POLLACK PERIODICA
An International Journal for Engineering and Information Sciences
DOI: 10.1556/606.2020.15.2.1
Vol. 15, No. 2, pp. 3–12 (2020)
www.akademiai.com

EFFECTIVE VOLUME RENDERING ON MOBILE AND STANDALONE VR HEADSETS BY MEANS OF A HYBRID METHOD

[1,2] Balázs TUKORA*

[1] Department of Information Technology, Faculty of Engineering and Information Technology
University of Pécs, Rókus u. 2, H-7624 Pécs, Hungary, e-mail: balazs.tukora@mik.pte.hu
[2] zMed Zrt., Boszorkány u. 2, H-7624 Pécs, Hungary

Received 17 May 2019; accepted 5 December 2019

Abstract: Numerous volume rendering techniques are available to display 3D datasets on desktop computers and virtual reality devices. Recently the spreading of mobile and standalone virtual reality headsets has brought the need for volume visualization on these platforms too. However, the volume rendering techniques that show good performance in desktop environment underachieve on these devices, due to the special hardware conditions and visualization requirements. To speed up the volumetric rendering to an accessible level a hybrid technique is introduced, a mix of the ray casting and 3D texture mapping methods. This technique increases 2-4 times the frame rate of displaying volumetric data on mobile and standalone virtual reality headsets as compared to the original methods. The new technique was created primarily to display medical images but it is not limited only to this type of volumetric data.

Keywords: Direct volume rendering, ray casting, 3D texture mapping, virtual reality, mobile and standalone virtual reality headsets

1. Introduction

In the last three decades several techniques have been developed for displaying volumetric datasets. A rather old but still relevant list of the most popular methods can be found in the study of Meißner et al. [1], while Zhang and his colleagues give us a comprehensive summary [2] of the topic, mostly focusing on medical applications. The study of Sutherland et al. [3] provides insight into the latest Augmented Reality (AR) and Virtual Reality (VR) technologies to display medical images.

* Corresponding Author

At first volume rendering algorithms were executed by the Central Processing Unit (CPU), and after the appearance of generally programmable Graphics Processing Units GPU's they became run as shader programs on the graphics hardver [4], [5]. The winners of this change have been the ray casting [6] and 3D texture mapping [7], [8] methods, the most popular ones to date, while some other previously dominant techniques, like the splatting [9] and shear-warp algorithm [10], though attempts were made to port them onto GPU's, have actually become extinct by today. A good example of this evolution is the recent version of Visualization Toolkit, in which GPU ray casting is applied as the default volume rendering algorithm, and if it is not supported by the hardware, 3D texture mapping is chosen for interactive and CPU ray casting for still rendering [11].

Ray casting, similarly to the other above-mentioned techniques, is a Direct Volume Rendering method (DVR) as it displays the 3D dataset without computing any intermediate geometry representations [12]. The basic idea behind it is casting a ray into the volume from each pixel of the screen, sampling the volume along the ray in a front-to-back or back-to-front order and finally compositing the optical values at the sampling points, thus obtaining the final color of the pixel. The secret of popularity of the ray casting technique is, beyond its simplicity and easy programmability on both CPU's and GPU's, that it can be optimized in several ways. Some improvements exploit the high coherency between the pixels in image space [13], [14] or between the voxels in object space [15]. Early ray termination can be used when rendering in front-to-back order: When the opacity of the composited pixel color has reached a threshold value that can be considered as fully opaque, the sampling process can be terminated as the further samples wouldn't make significant contribution to the final color. Empty space skipping or space-leaping is another widely-used optimization technique: Skipping the empty spaces within the volume when sampling does not affect the image quality but can speed up the calculations significantly. Numerous solutions (summed up in [1] and [2]) were born in this subject, being distinct in the way of representing the empty and non-empty parts of the volume.

In 3D texture mapping the volume is uploaded to the GPU as a single 3D texture. A set of polygons perpendicular to the viewing, the so-called proxy geometry, is placed within the volume and the 3D texture is mapped onto them with nearest neighbor or tri-linear sampling. Finally the textured polygons are alpha-blended in a back-to front order to produce the final image on the screen. Rendering with 3D texture mapping doesn't require programmable shaders, GPU's with fixed-function pipeline can also fulfill this task.

The time complexity of the 3D texture mapping and ray casting algorithms is the same, especially as 3D textures are used to upload the volume data to the GPU and the same sampling technique is applied in both cases. The 3D texture mapping method, however, cannot be optimized in the free manner as it can be done so with the programmable shaders in ray casting. Even so effective and robust optimization methods like the early ray termination and empty space skipping cannot be used.

Despite the fact that the ray casting method is effective and easy to optimize, it simply fails on mobile or standalone VR headsets. Far from interactive visualization no more than 4-5 Frames Per Second (FPS), doubled with empty space skipping, could be reached in the case of a common medical volume visualization task (detailed in later

sections). Recent studies [16], with no VR but mobile applications in scope, also confirm these results. In their paper of this year [3] Sutherland and his colleagues claim that rendering of most medical datasets on mobile VR platforms is 'likely to be too challenging'. The reasons can be found in the special characteristics of these devices.

A mobile VR device is a VR platform powered by a smartphone: A headset into which a smartphone is placed that provides the screen and acts also as the computing unit. Standalone VR headsets have their own in-built display and computing unit. They are called standalone as there is no need of a high-performance personal computers to be attached to create and run VR contents on them. The mobile and standalone VR headsets are similar in terms of the hardware components and the typical software tasks to run: A multi-core CPU and integrated graphics unit with shared memory on a low-power SoC (System on a Chip); displaying stereoscopic polygon-based graphics on a quite high-resolution (FHD+, WQHD+) screen at high frame rates.

Though the GPU's on the SoC's are compliant with the latest shader models, their performance is fairly low as compared to the GPU's of desktop computers. The thumb rule of shader programming on these chipsets is keeping the code (especially the fragment shader code) as short as possible. Using program loops that cannot be unrolled by the compiler (reshaped as a repeated sequence in order to eliminate loop-controlling instructions) should also be avoided. A typical ray casting fragment shader with sampling loops executed several hundred times simply cannot fit these requirements.

The 3D texture mapping technique performs slightly better thanks to the simple shader code, but the desirable frame rates cannot be reached as the method does not let further optimization. Here it must be cleared what a desirable frame rate means in the current situation. Displaying static volumetric data, like still medical 3D images, requires an interactive frame rate that ensures an acceptable graphical response time after sending a request from the input peripherals to the program. Miller identified this as 10 frames per second [17], which has been accepted and applied to date by consensus. A frame rate is called real-time, when it gives the impression of smooth, continuous moving of the displayed objects. It is usually 24-30 FPS in 2D and monoscopic 3D environments. VR devices, however, use much higher, not rarely 90 FPS to avoid the virtual reality sickness and to handle the intensive movements and changings in the scene. It was found that for displaying a static medical volumetric image on mobile and standalone VR headsets, where the point of view does not change dynamically, about 30 FPS is acceptable, similarly to the plain 3D visualization. During the measurements described in later sections, the 3D texture mapping technique provided 14 FPS, which is under the desired speed in our case.

To overcome the performance issues the ray casting and 3D texture mapping techniques has been merged. The basic idea is the following: If the 3D texture mapping technique is used with a reduced number of slices and the color on the proxy polygons are calculated by means of hidden ray casting layers, the number of executed program loops decreases and the opportunity to the optimization also remains. To fit the two methods together, the blending functions of 3D texture mapping must be applied during the ray casting composition. To keep the fragment shader code simple a special empty space skipping technique has been chosen. A novel fashion of proxy geometry has been initiated to avoid some resource-consuming calculations. The test results justified our

efforts: Under unchanged conditions the FPS increased to a decent frame rate when a typical volumetric dataset were displayed by the new hybrid technique.

In the following sections the main components of the introduced technique are detailed: The creation of 3D texture mapping proxy geometry; the hybrid compositing algorithm; the solution of empty space skipping. These sections are followed by the test results to show the correctness of the conception, as compared to the solutions of others and the original techniques. At last the conclusions are drawn from the experiences.

2. Materials and methods

2.1. Proxy geometry

In 3D texture mapping the intersections of the volume bounding box and the series of slicing planes perpendicular to the direction of view form the polygons of proxy geometry. After calculating the six or less intersection points of the bounding box edges and a plane, these points are used as the vertices of the polygon on that plane (see *Fig. 1* left). These calculations are performed by the CPU and the newly created polygons are uploaded onto the GPU at every rendering cycle whenever the point of view has changed.

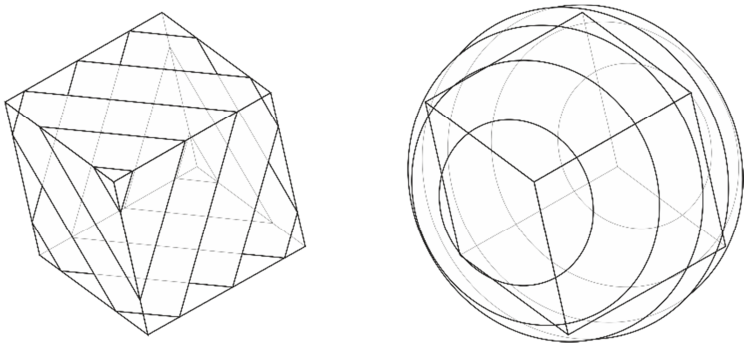

Fig. 1. Dynamic (left) and static (right) proxy geometry

When rendering VR contents, the viewer's position changes continuously. This means that the creation and uploading of the proxy geometry has to be repeated at every single frame. The limited resources of mobile and standalone VR headsets do not allow us this dynamic way of proxy geometry creation. A static set of polygons are used instead that are created at the program setup time and placed into the screen space to be anchored to the point of view. The geometry consists of the slices of the bounding sphere around the volume, as seen in *Fig. 1* right. In practice the circles are approximated by regular polygons. It has been found that the use of 10-gons results in good approximation without the need of dealing with too many rectangles by the GPU. The unnecessary samplings outside the volume boundaries are discarded in the fragment shader, on the basis of the sampling point coordinates in the object space.

2.2. Hybrid composition

The applied composition method is a mix of alpha blending of overlapping polygons (as in 3D texture mapping) and compositing along rays casted into the volume (as in ray casting). The process is represented in *Fig.* 2. On one hand, the colors on the proxy polygons are blended together by the traditional over operator [18] of alpha compositing in a back-to-front order, thus getting the final pixel colors on the screen. On the other hand, the colors on the proxy polygons are calculated by a limited-length ray casting composition. The casting rays start at the camera center point, go through the screen pixels and intersect the proxy polygons. To get the color of a certain pixel on a polygon, some samplings are done in front of it and behind it along the intersecting ray. The number of samplings determines the number of hidden ray casting layers. The composition calculations are performed by the fragment shader, similarly to the traditional ray casting technique. The composition equations corresponds the equations of over operator of alpha blending:

$$C_d = \frac{C_s\alpha_s + C_d\alpha_d(1-\alpha_s)}{\alpha_s + \alpha_d(1-\alpha_s)}, \tag{1}$$

$$\alpha_d = \alpha_s + \alpha_d(1-\alpha_s). \tag{2}$$

In the equations C_d and α_d stand for the color and opacity of the destination pixel (the pixel to be updated by the composition), while C_s and α_s are the source color and opacity (the values that alter the appearance of the destination pixel).

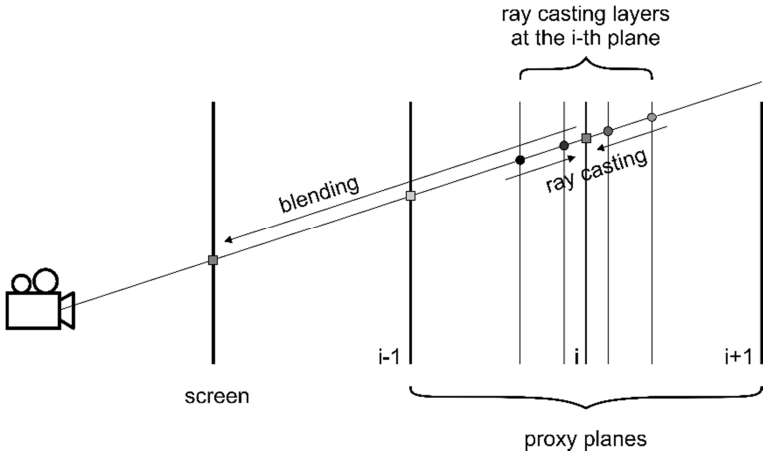

Fig. 2. Hybrid composition

By adjusting the number of 3D texture proxy planes as compared to the hidden ray casting layers, it can be determined which method dominates. By setting the ray casting layers to one per proxy plane the pure 3D texture mapping technique is chosen. By setting the number of proxy planes to one with setting a number of rays casting layers

clearly the ray casting technique works. By performing a few ray casting steps at the proxy planes the unrolled fragment shader loops are kept short with the possibility of applying some limited ray casting optimization techniques.

2.3. Optimization

The most effective branch of optimization techniques in ray casting is the empty space skipping. Before the rendering the volumetric data is completed with information about the empty regions of the volume. During the sampling the ray casting shader reads this information in addition to the voxel properties and calculates if the subsequent sampling points fall in any of the empty regions. If they do so, they become skipped.

The effective empty space skipping requires that the ray casting shader calculates the boundaries of empty regions accurately. However this has a cost. The empty regions are often embedded in a hierarchical structure, e.g. octree [14] that has to be decoded on the fly, or complicated calculations are needed to get the ray-region intersections. Most time this ends up in a complex shader code.

In order to minimize the required calculations on the low-performance GPU's of mobile and standalone VR headsets, the proximity-clouds method [19], [20] has been implemented from the empty space skipping techniques. In this method the distance to the closest occupied voxel is calculated for each voxel in a preprocessing stage. This data read from the voxel at a sampling point directly gives the distance that can be safely skipped along the ray. Though the representation of empty spaces in the proximity-clouds method implies that some unnecessary samplings happen in these regions, the simple shader code makes it very effective on the targeted devices.

3. Results and discussion

To prove the effectiveness of the introduced technique a series of tests was carried out. The device on which the tests run was a Google Daydream View headset powered by a Samsung Galaxy S8 smartphone with a typical hardware of mobile and standalone VR devices: A Samsung Exynos 8895 Octa SoC with an 8-core (4x2.3 GHz Mongoose M2 & 4x1.7 GHz Cortex-A53) processor and Mali-G71 MP20 GPU that share 4GB RAM. The highest available WQHD+ 2960x1440 screen resolution was set with 2x multisampling. The test application had been made with Unity, the de-facto tool for mobile VR development.

The displayed volumetric dataset was a Computer Tomography (CT) scan of a human skull with 256x256x256 voxel resolution and 16 bit depth. The Hounsfield value of each voxel was converted into color and opacity by transfer functions in order to enhance the solid structure of the scull. The size of the displayed skull on the screen was set that it filled the field of vision when looked through the goggles of the headset, which is actually smaller than the available area on the screen, as it can be seen on *Fig. 3*. To save some calculations in the fragment shader, the opacity and the shaded color for each voxel had been pre-calculated by means of the transfer functions and the Blinn-Phong model [21]. These colors were then loaded into a 3D texture and sampled

with tri-linear interpolation during the rendering stage. According to the Blinn-Phong model only the specular component has to be calculated during the rendering as it is view-dependent. Some more speed-up could be reached by omitting this component, as the lack of specular highlights did not reduce particularly the user's experience. For further speed-up the 2x multi-sampling and tri-linear interpolation could have been omitted as well, but it was not done so in order to get an acceptable image quality.

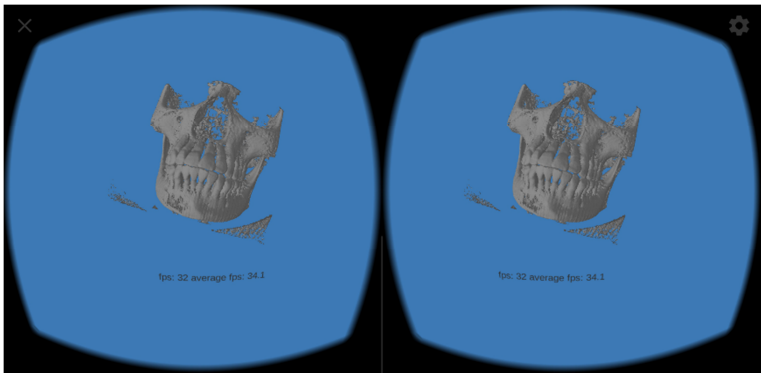

Fig. 3. Stereoscopic volume rendering on VR headset

The tests consisted of a sequence of measurements, when the FPS was registered during displaying the skull object with different proxy plane/ray casting layer ratios. Since the frame rate slightly altered at different points of view, a rotating object was displayed and the average FPS was measured during a whole rotation. (The extent of alternation was not bigger than 1-2 FPS.) The distance of the layers was adjusted to be the same as the voxel size. As the proxy polygons are situated in the bounding sphere around the object, the number of layers must equal √3 times the voxel resolution, which is about 444. In most cases, however, the actual number of layers differed from this number to some degree since it was calculated by the multiplication of two integer numbers. At first the frame rates were measured without optimization and then with applying empty space skipping to exploit the advantage of the hybrid method.

Fig. 4 shows the results of the measurements. The values on the horizontal axis correspond to the total number of layers, given by the number of proxy planes and the number of ray casting layers at each plane.

The leftmost value shows 1 proxy plane and 444 ray casting layers, which indicates that the pure ray casting method was applied here. The reached FPS was fairly low (4.7 without and 8.5 with optimization) due to the long-running fragment shaders. Although no test results were found in the literature about the speed of ray casting visualization on mobile and standalone VR platforms, the measurements of Holub and Winer [16] confirmed the above values. They displayed the structural (static) component of a fMRI dataset with 256x256x128 resolution on an 2016 iPad Pro and got about 18 FPS. It is almost impossible to compare the values of the two tests due to the differences in the test conditions and because of some unknown factors, but it seems correct that the doubled resolution of the volumetric data and the stereoscopic rendering result in a

halved frame rate. To play safe, the Unity ray casting visualization demo, developed by Gilles Ferrand [22] on the basis of the Cg example of NVIDIA [23], was rewritten to fit the conditions of the tests. A fairly view-dependent frame rate was observed with 9 FPS in the worst case, so it can be claimed that our results are typical of the ray casting technique on the targeted platforms.

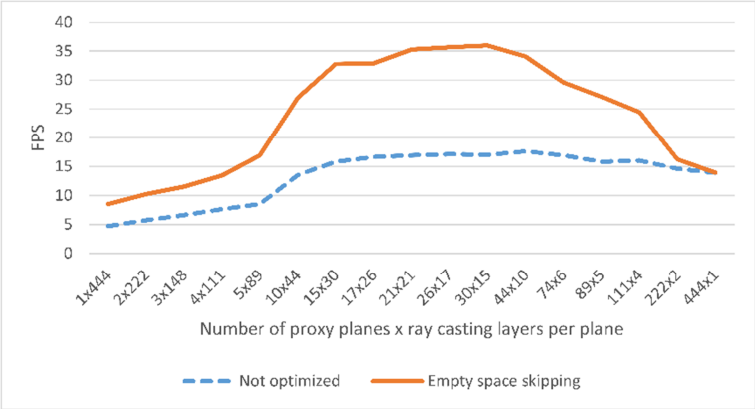

Fig. 4. Test results

On the rightmost side the pure 3D texture mapping method can be found with 444 proxy planes with one sampling point along the rays at each plane. The FPS was 13.9 in this case, which could not be improved by optimization as the empty space skipping had no effect. No previous studies have been found on the efficiency of this technique on mobile or standalone VR devices, and the only found documented test on iPads [24] did not proved comparable. The implementation of the 3D texture mapping algorithm, however, is quite straightforward, so it can be assumed that our method provides the speed that can be expected from this technique.

Going to the left on the diagram, by decreasing the number of proxy planes and increasing the number of ray casting layers, more and more points get involved into the optimization, while the unrolled loops of the fragment shader still fit the limited resources of the GPU. This result in a significant speed-up: 36 FPS could be reached, which is 2.59 times higher than the pure 3D texture mapping technique and 4.24 higher than the optimized ray casting technique can provide. Considering that still medical 3D images are displayed this way, it can be claimed that a decent frame rate, that suits the requirements of VR visualization, has been achieved on the targeted platforms.

Going further to the left on the diagram, the complexity of the compiled fragment shaders exceeds the critical value that causes the sudden drop of the performance.

As it can be seen above, the empty space skipping significantly improves the speed of displaying. The question can be put, if the visualization of datasets with large semi-transparent or opaque areas excessively decreased the performance of the introduced method. The speed, in this case, would not inevitably drop, because of the fact that the empty space skipping is executed *after* applying the transfer functions which results in extended empty areas in common medical visualization tasks.

4. Conclusions

The two most effective thus most popular direct volume rendering techniques in desktop environment underachieve on the mobile and standalone VR headsets. The reasons are different: while the difficult fragment shader code of ray casting lies heavy on the stomach of the limited-performance hardware, the lack of optimization doesn't let the 3D texture mapping to be effective enough. To exploit the advantages of the two methods, the wide range of optimization of the first and the simplicity of the latter, they were mixed together, thus a hybrid method dedicated to the targeted platforms has been born. The created technique provides significantly higher, real-time frame rate on the mobile and standalone VR headsets, opening the door to the qualified volume rendering on these devices.

Acknowledgements

This work was supported by the GINOP-2.2.1-15-2017-00083 grant. The contents of this paper show the achievements of the research work defined in the work packages of the project 'Developing innovative medical solution (zMed), an augmented reality based visualization technology of 3D medical scans and reality that also strengthens the doctor-patient relationship and improves the medical education.'

Open Access statement

References

[1] Meißner M., Wittenbrink C. M., Westermann R., Pfister H. Volume visualization and volume rendering techniques, *Eurographics 2000 Tutorial*, 2000.

[2] Zhang Q., Eagleson R., Peters T. M. Volume visualization: A technical overview with a focus on medical applications, *Journal of Digital Imaging,* Vol. 24, No. 4, 2011, pp. 640–664.

[3] Szűcs Á. I. Improving graphics programming with shader tests, *Pollack Periodica*, Vol. 14, No. 1, 2019, pp. 35–46.

[4] Magoulès F., Ahamed A. K. C., Putanowicz R. Fast iterative solvers for large compressedsparse row linear systems on graphics processing unit, *Pollack Periodica*, Vol. 10, No. 1, 2015, pp. 3–18.

[5] Sutherland J., Belec J., Sheikh A., Chepelev L., Althobaity W., Chow B. J. W., Mitsouras D., Christensen A., Rybicki F. J., La Russa D. J. Applying modern virtual and augmented reality technologies to medical images and models, *Journal of Digital Imaging*, Vol. 32, No. 1, 2018, pp. 38–53.

[6] Drebin R. A., Carpenter L., Hanrahan P. Volume rendering, *ACM SIGGRAPH Computer Graphics*, Vol. 22, No. 4, 1988, pp. 65–74.
[7] Cabral B., Cam N., Foran J. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware, *Proceedings of the 1994 Workshop on Volume Visualization*, Tysons Corner, Virginia, USA, 17-18 October 1994, pp. 91–98.
[8] Dachille F., Kreeger K., Chen B., Bitter I., Kaufman A. High-quality volume rendering using texture mapping hardware, *Proc. of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, Lisboa, Portugal, 31 August-1 September 1998, pp. 69–76.
[9] Westover L. Interactive volume rendering, *Proceedings of the 1989 Chapel Hill Workshop on Volume Visualization*, Chapel Hill, North Carolina, USA, 5-8 December 1989, pp. 9–16.
[10] Lacroute P., Levoy M. Fast volume rendering using a shear-warp factorization of the viewing transform, *Proc. of the 21st Annual Conference on Computer Graphics and Interactive Techniques,* Orlando, Florida, 24-29 July 1994, pp. 451–457.
[11] vtkSmartVolumeMapper Class Reference, VTK 7.1.1 Documentation, https://www.vtk.org/doc/release/7.1/html/classvtkSmartVolumeMapper.html#details, (last visited 17 September 2018).
[12] Gordon D., Reynolds R. A. Image space shading of 3-dimensional objects, *Computer Vision, Graphics and Image Processing*, Vol. 29, No. 3, 1985, pp. 361–376.
[13] Bergman L., Fuchs H., Grant E., Spach S. Image rendering by adaptive refinement, *ACM SIGGRAPH Computer Graphics*, Vol. 20, No. 4, 1986, pp. 29–37.
[14] Levoy M. Volume rendering by adaptive refinement, *The Visual Computer*, Vol. 6, No. 1, 1990, pp. 2–7.
[15] van Walsum T., Hin A. J. S., Versloot J., Post F. H. Efficient hybrid rendering of volume data and polygons, in *Advances in Scientific Visualization. Focus on Computer Graphics* (*Tutorials and Perspectives in Computer Graphics*), Post F. H., Hin A. J. S. (Eds.) Springer, 1992.
[16] Holub J., Winer E. Enabling real-time volume rendering of functional magnetic resonance imaging on an iOS device, *Journal of Digital Imaging*, Vol. 30, No. 6, 2017, pp. 738–750.
[17] Miller R. B. Response time in man-computer conversational transactions, *Proc. of Fall Joint Computer Conference,* Part I, San Francisco, California, 9-11 December 1968, pp. 267–277.
[18] Wallace B. Merging and transformation of raster images for cartoon animation, *ACM SIGGRAPH Computer Graphics*, Vol. 15, No. 3, 1981, pp. 253–262.
[19] Cohen D., Shefer Z. Proximity clouds - an acceleration technique for 3D grid traversal, *The Visual Computer: International Journal of Computer Graphics*, Vol. 11, No. 1, 1994, pp. 27–38.
[20] Zuiderveld K. Z., Koning A. H. J., Viergever M. A. Acceleration of ray casting using 3D distance transform, *Proceedings of Visualization in Biomedical Computing*, Chapel Hill, NC, 22 September 1992, Vol. 1808, pp. 324-335
[21] Blinn J. F. Models of light reflection for computer synthesized pictures, *ACM SIGGRAPH Computer Graphics*, Vol. 11, No. 2, 1977, pp. 192–198.
[22] Ferrand G. *Unity-RayTracing*, https://github.com/gillesferrand/Unity-RayTracing, (last visited 16 October 2018).
[23] *NVIDIA OpenGL SDK 10 Code samples*, http://developer.download.nvidia.com/SDK/10/opengl/samples.html, (last visited 16 October 2018).
[24] Noon C., Holub J., Winer E. Real-time volume rendering of digital medical images on an iOS device, *Proc. of SPIE, Multimedia Content and Mobile Devices*, Vol. 8667, Burlingame CA, United States, 7. March 2013, pp. N/A.