Rendering the Real-Time Caustics with DirectX Raytracing

Egor Komarov [1], Dmitry Zhdanov [1] and Andrey Zhdanov [1]

[1] *ITMO University, 49 Kronverksky Pr., St. Petersburg, 197101, Russia*

Abstract

Caustic illumination frequently appears in the real life, however, this type of illumination is especially hard to be rendered in real-time. Currently, some solutions allow to render the caustic illumination caused by the water surface, but these methods could not be applied to the arbitrary geometry. In the scope of the current article, we present a method of real-time caustics rendering that uses the DirectX Raytracing API and is integrated into the rendering pipeline. The method is based on using additional forward caustic visibility maps and backward caustics visibility maps that are created for light sources and the virtual camera correspondingly. The article presents the algorithm of the developed real-time caustics rendering method and the results of testing its implementation on test scenes. The analysis of the dependence of the rendering speed on the depth of specular ray tracing, the number of light sources, and the number of rays per pixel is carried out. Testing shows promising results which can be used in the modern game industry to increase the realism of the virtual world visualization.

Keywords

Ray tracing, backward photon mapping, caustics, DirectX Raytracing

1. Introduction

The caustic illumination [1] frequently appears in real-life experience. The simplest example of this effect would be the illumination of a table by a light passing through a glass of water as is shown in Figure 1.

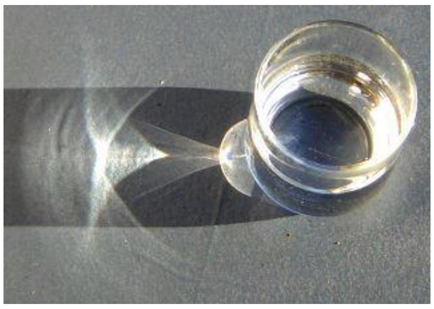

Figure 1: An example of caustics caused by the glass of water

Even though this type of illumination is quite common in real life it is not that feasible to be simulated by the methods of computer graphics, especially if real-time image rendering is required, e.g., in computer games. Rendering methods used in computer games mainly use rasterization techniques that is a method for projecting the scene objects to the virtual camera [2], however recently the DirectX Raytracing (DXR) [3] was presented, that allows using the raytracing API and combine different rendering techniques in the real-time applications. At the same time, NVIDIA presented the Turing technology for their GPUs allowing them to speed up the ray hit search up to ten times comparing to the previous generations [4]. So, the adaptation of the existing rendering algorithms, realistic rendering methods, and various realistic optical effects to be effectively used with the real-time rendering with

GraphiCon 2021: 31st International Conference on Computer Graphics and Vision, September 27-30, 2021, Nizhny Novgorod, Russia
EMAIL: komarov.99@yandex.ru (E. Komarov); ddzhdanov@mail.ru (D. Zhdanov); andrew.gtx@gmail.com (A. Zhdanov)
ORCID: 0000-0003-3719-995X (E. Komarov); 0000-0001-7346-8155 (D. Zhdanov); 0000-0002-2569-1982 (A. Zhdanov)

the DirectX Raytracing is an urgent challenge. In the scope of the current paper, we present an approach for effective accounting for the caustic type of illumination when designing the realistic rendering algorithms to be used with the DirectX Raytracing API.

2. Related works

For the correct caustic illumination calculated the rendering method should be based on physically correct laws of the light propagation and transformation. This type of rendering method is called realistic rendering and is based on calculating the visible luminance $L(\vec{p},\vec{v}_r)$ for each pixel $\vec{p}$ in the observation direction $\vec{v}_r$ of the virtual camera by solving the James Kajiya rendering equation [5]:

$$L(\vec{p},\vec{v}_r) = \tau(\vec{p})\frac{{n_r}^2}{{n_0}^2}\left(L_0(\vec{p},\vec{v}_r) + \int_{\omega} L_i(\vec{p},\vec{v}_i) f(\vec{p},\vec{v}_i,\vec{v}_r)\cos(\vec{n},\vec{v}_i)\,d\omega\right), \quad (1)$$

where $L_0(\vec{p},\vec{v}_r)$ is the self-luminance of a surface at the point $\vec{p}$; $\tau(\vec{p})$ is the transmission coefficient of the medium on the path from the virtual camera to the observed surface; n_r is the refractive index of the virtual camera medium; n_0 is the refractive index of the medium on the surface in the direction of observation; $L_i(\vec{p},\vec{v}_i)$ is the luminance, incident on the surface in direction $\vec{v}_i$; $f(\vec{p},\vec{v}_i,\vec{v}_r)$ is the Bidirectional Scattering Distribution Function (BSDF) value for the direction of the incident light $\vec{v}_i$ and the direction of observation $\vec{v}_r$; $\vec{n}$ is the surface normal at the point of observation.

Since the rendering equation is an integral equation with an infinite recursion it is traditionally solved with the Monte-Carlo method. So the formula (1) is transformed the following formula:

$$L(\vec{p},\vec{v}_r) \approx \tau(\vec{p})\frac{{n_r}^2}{{n_0}^2}\left(L_0(\vec{p},\vec{v}_r) + \frac{1}{\pi\cdot N}\sum_{i=1}^{N} L_i(\vec{p},\vec{v}_r) f(\vec{p},\vec{v}_i,\vec{v}_r)\cos(\vec{n},\vec{v}_i)\right), \quad (2)$$

where N is the number of samples.

This equation is traditionally solved with methods of backward ray tracing [6], forward ray tracing [7], bidirectional ray tracing [8] and its modifications [9], forward [10], and backward [11] photon mappings. The most effective methods for caustics visualization are based on the photon mapping [12][13] or backward photon mapping [14]. When talking about real-time caustics visualization in gaming industry the most common type is the visualization of water that is performed by calculating the animated texture of water illumination and normals in advance and then projecting them to the underwater surfaces [15]. Another real-time caustics visualization method is using caustic maps [16], however, this solution is limited by a single event of specular reflection or refraction. To speed up caustic maps creation they could also be built using the deferred shading [17].

When designing the real-time realistic rendering based on the DirectX Raytracing it should follow the DXR raytracing pipeline and use the corresponding API [18]. The most promising CPU rendering method is the progressive backward photon mapping [19] as it uses fewer resources with the same rendering quality, however, due to different computation architecture and resource limits, it is not possible to use the CPU rendering method directly, so new algorithmic approaches should be developed.

3. The rendering method

In the scope of the current research, we distinguish two types of caustics. The first type is caustics which is observed directly, and the second type is observed through other specular objects. The two types of caustics are shown in Figure 2.

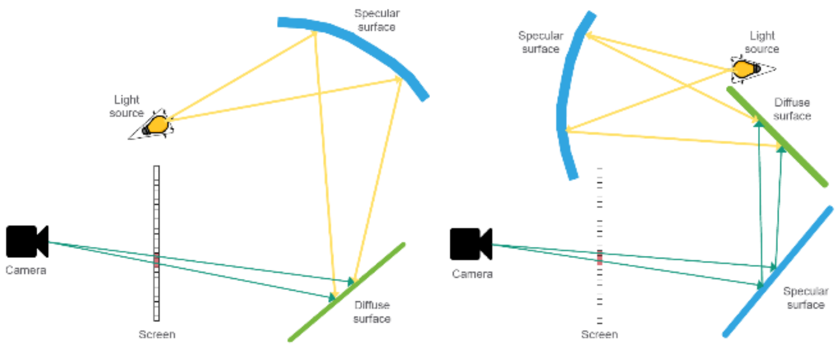

Figure 2: The caustics of the first type (to the left) and the second type (to the right)

Our research of different rendering methods showed that the most realistic caustic illumination with the same rendering speed is achieved when using the backward photon mapping method. However, in its pure form, the result of its implementation on GPU looks ineffective, so the final algorithm combines these ideas with forward ray tracing and caustic maps. As a result, the developed rendering method can be divided into four main phases integrated into the single DirectX rendering pipeline:

1. the global illumination calculation,
2. calculation of the caustics of the first type,
3. calculation of the caustics of the second type,
4. image filtering.

3.1. The global illumination

To calculate the global illumination, we use the method of Deferred Shading. Its main idea is to separate the calculation of positions and properties of surfaces from the final illumination. In the general case, this is achieved by a single rendering of all visible objects of the scene into several render targets (RT), which are called geometric buffers (G-Buffers), in which the position of the displayed object is recorded for each point of the image (usually in the form of the distance from the observer), normal to the surface and any other parameters of the material that are involved in the luminance calculation.

Thus, the calculation of the global illumination in the simplest case consists of three steps:

1. creation of G-Buffers,
2. calculation of shadowing, transparencies, and reflections,
3. the final luminance calculation.

These algorithms are well described in works [20] and [21] and are outside of the scope of the current article.

3.2. The caustics of the first type

To calculate the caustics of the first type, it is required that the global illumination is calculated in advance and depth map G-Buffer is present.

At first, the creation of the G-Buffers for depth maps, surface normals, and surface optical properties, for each of the scene light sources should be performed. To speed up the further calculations the light sources that do not form any caustic illumination are omitted in the following steps. It should be noted that this step can be performed at the same time as calculating the scene shadowing. Figure 3 illustrates the creation of these G-Buffers. In this example the 3rd light source does not form the caustic illumination, so no visibility map is created.

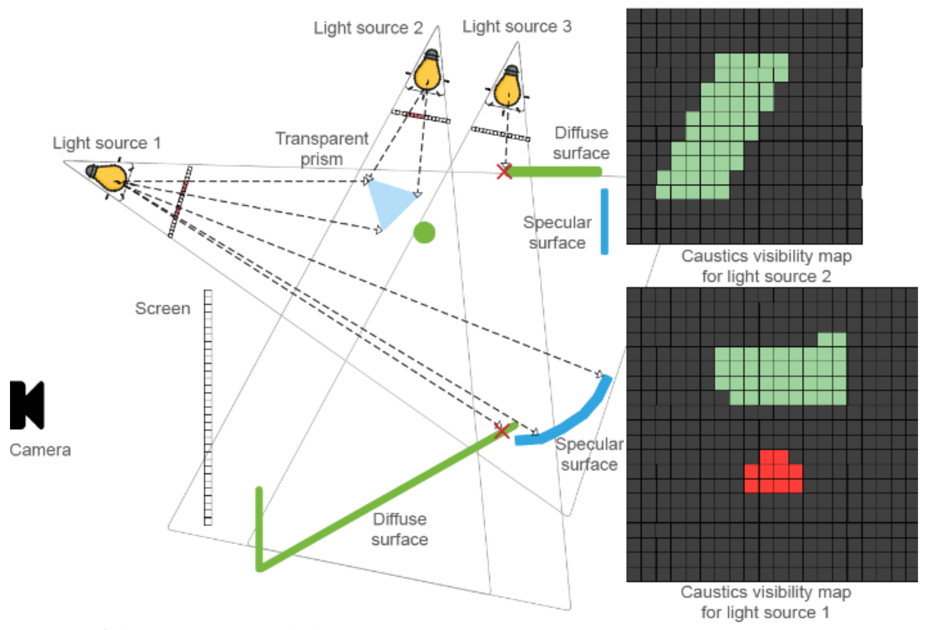

Figure 3: Creation of the caustic visibility maps

The creation of the caustic visibility maps replaces the first step of ray tracing. The use of rasterization in this case significantly reduces the computation time and allows determining areas within which it makes sense to trace further rays. This also allows us to adjust the rays emitting PDF.

The primary generation of rays is carried out on a grid with a dimension equal to the visibility map resolution. So, it makes sense to combine light sources with the same grid resolutions into texture arrays, adding a third dimension to the generation grid with a value equal to the size of the array. Before generation, the values of the reflectance and transparency coefficients of the surface are read from the G-Buffer at the point corresponding to the index of the ray in the grid. If both values are 0, then no ray emission is required at that point. Otherwise, depending on the coefficient values, a reflected and/or refracted ray is emitted. The ray origin is taken from the depth map, the direction and flux are determined using importance sampling with a probability distribution corresponding to the surface BSDF, for this, the value of the normal map and, optionally, other surface properties are read from the corresponding G-Buffers.

When the rays intersect with the surface, in the first place, similar to the primary generation, the parameters of reflection and refraction are calculated to decide whether new rays should be launched. To limit tracing, you can either use the Russian Roulette method, or set a recursion limit, or set a limit on the number of repeated rays per pixel. The ray-tracing algorithm is shown in Figure 4 (a).

Each hit of the ray on the surface is accompanied by an assessment of the visibility of the intersection point from the camera. For this, its coordinates are transformed from the world space to the camera space using the corresponding matrices. The result is the distance to the camera and the coordinates of the image point from which the intersection is visible. Since the depth map from the camera side has already been calculated in G-Buffer, to determine the visibility, it is enough to compare these data with each other. So, the backward ray tracing is replaced with faster rasterization as is shown in Figure 4 (b).

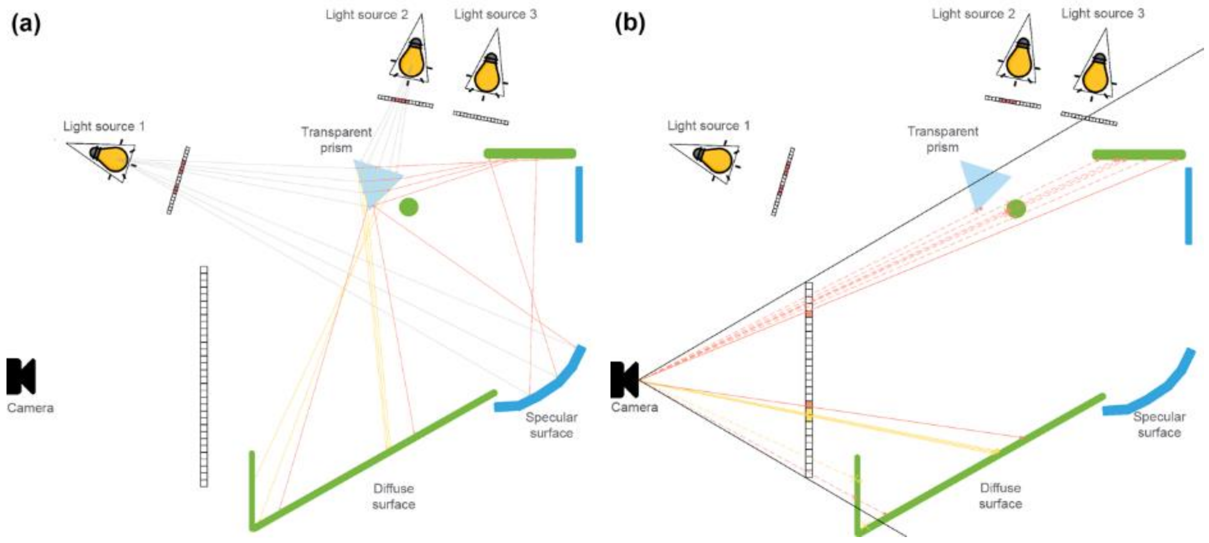

Figure 4: The forward ray tracing (a) and caustics visibility estimation (b)

The resulting caustics luminance carried by each forward ray is calculated according to formula (2) and is atomically added to the previously accumulated luminance at the corresponding point of the image.

3.3. The caustics of the second type

At first, for accounting the caustics of the second type it is required that along with the depth map G-Buffer of the global illumination step, the caustics of the first type are also calculated, and caustics visibility maps are created.

The algorithm for calculating caustics of the second type is using the ideas of backward photon mapping, but to account for the caustic illumination only. At the first step, the camera caustics visibility map is created. The backward rays are emitted and traced through the scene until the event of diffuse scattering occurs. These rays are traced only through pixels that have specular properties stored in corresponding G-Buffers and backward rays are traced from the first hit point which is also stored in the G-Buffers. The backward ray tracing with backward caustics visibility maps creation is shown in Figure 5.

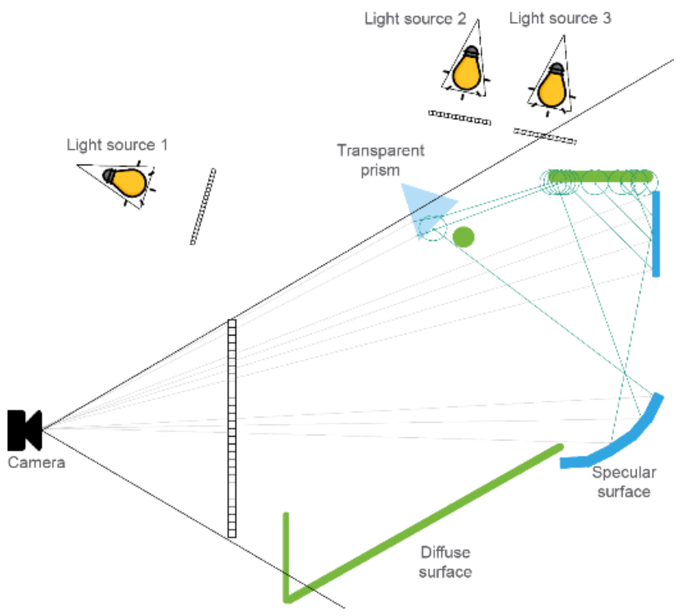

Figure 5: Backward ray tracing and creation of the backward caustics visibility maps

Each of the hit points that were created after the backward ray-tracing step along with other data stores integration sphere radius and the coefficient of transmittance that was accumulated on the ray path. Then the two-level acceleration structure is created for the set of spheres, where bottom-level acceleration structure (BLAS) stores spere and top-level acceleration structure (TLAS) is built over the set of BLAS. Each BLAS is represented with the same spherical object and transformation matrix containing 12 floats. So, this acceleration structure would occupy up to 92Mb of memory for an image with a resolution of 1920x1080.

The last step is checking for the intersection of previously created 1st type caustics with backward caustics visibility maps. The check should be performed only for objects with equal object identifiers and positioned at the same side of the scene surface for reflective objects. Due to ray might hit several visibility spheres, the Any Hit Shader of the ray-tracing pipeline should be used. Figure 6 schematically shows this method.

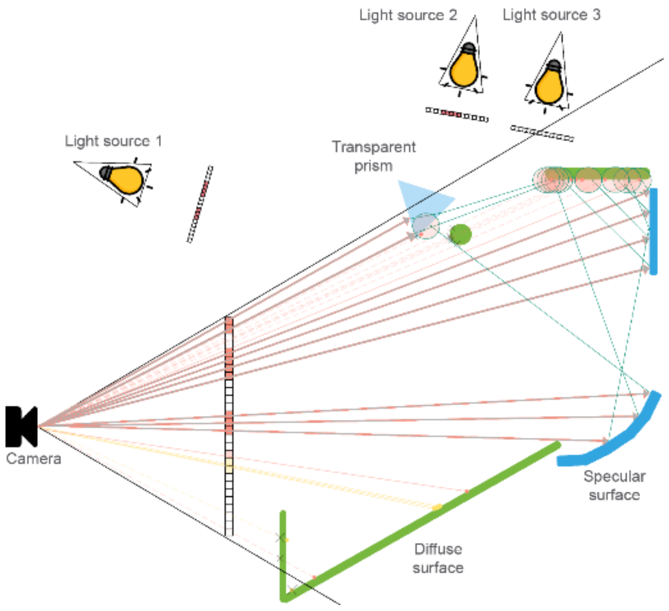

Figure 6: Intersecting the forward caustics with the backward caustics visibility maps

3.4. Image filtering

The use of stochastic rendering methods inevitably leads to a noisy final image. One way to solve this problem is to increase the sampling rate, but since the calculations are done in real-time, the number of traced rays is severely limited. In this case, noise reduction algorithms [22] can be used. The most promising denoiser is NVIDIA Real-Time Denoiser (NRD) [23] that uses a history of 32 frames along with additional image data and requires only about 5ms to filter the single 1920x1080 frame.

Figure 7 shows the complete real-time rendering algorithm scheme with all steps described above.

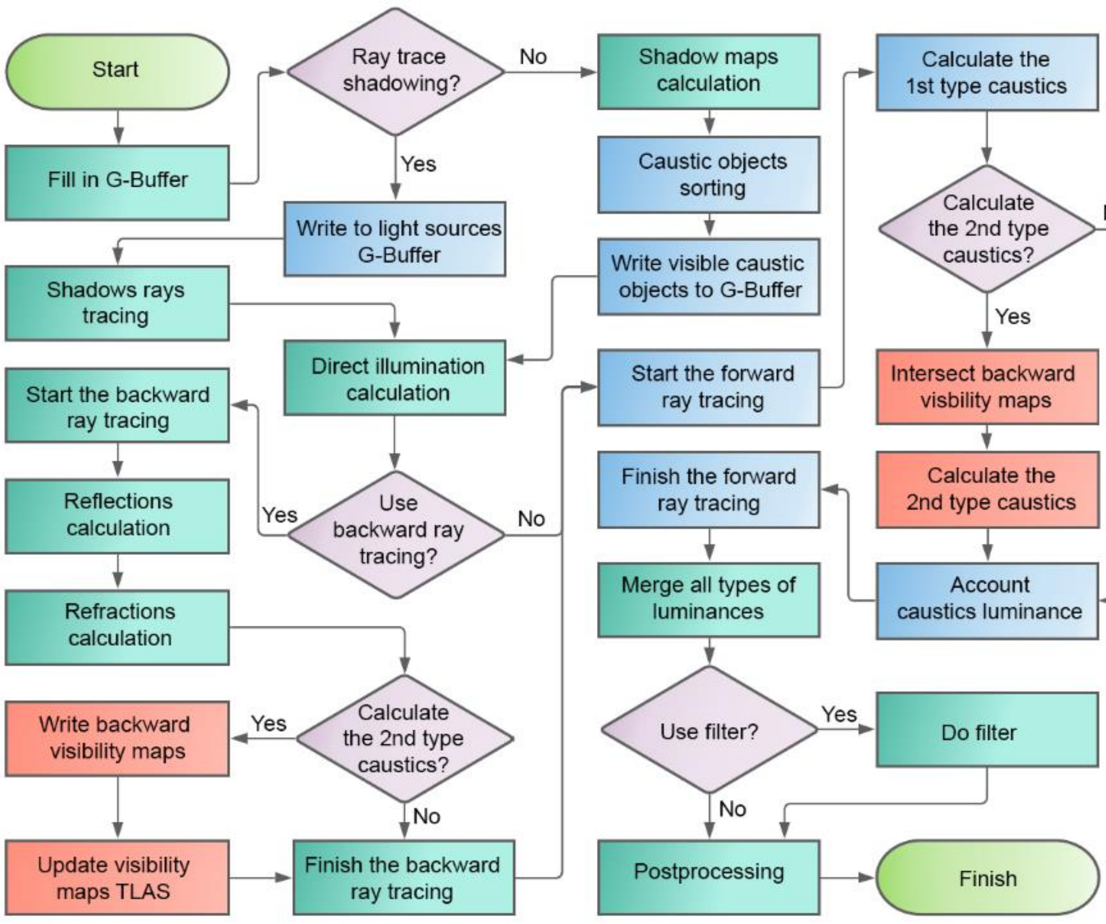
Start
Fill in G-Buffer
Ray trace shadowing?
No
Yes
Write to light sources G-Buffer
Shadows rays tracing
Direct illumination calculation
Start the backward ray tracing
Reflections calculation
Refractions calculation
Use backward ray tracing?
Yes
No
Calculate the 2nd type caustics?
Yes
No
Write backward visibility maps
Update visibility maps TLAS
Finish the backward ray tracing
Shadow maps calculation
Caustic objects sorting
Write visible caustic objects to G-Buffer
Start the forward ray tracing
Finish the forward ray tracing
Merge all types of luminances
Use filter?
Yes
No
Do filter
Postprocessing
Finish
Calculate the 1st type caustics
Calculate the 2nd type caustics?
No
Yes
Intersect backward visbility maps
Calculate the 2nd type caustics
Account caustics luminance

Figure 7: The real-time rendering algorithm principal scheme

In this scheme green and purple blocks represent the general rendering parts and algorithm flow control, blue blocks are for calculating the 1st type of caustic luminance, red blocks are for calculating the 2nd type of caustic luminance.

4. Results

The described above algorithms were implemented basing on the Microsoft Mini-Engine renderer [24]. This renderer is based on the DirectX Raytracing and is distributed under MIT license. Along with implementing the caustics illumination algorithm, the following modifications were done:

- the realistic BDF-based surface model [25] was implemented,
- the luminance calculation was modified to be more physically correct,
- the GLTF file format importing was implemented,
- the deferred shading was implemented,
- the PDF-based reflection model was added,
- other minor code corrections.

The testing was performed on the mobile NVIDIA GeForce RTX 2060 GPU. The test scene which is used in the scope of the current article is the modified Crytek Sponza scene. The default rendering parameters are:

- the resolution of the caustic map is 512x512,
- the number of rays per pixel is 1,
- the number of light sources is 3,
- the ray-tracing depth is 4.

The real-time rendering required to have at least 30fps (33.3ms per frame), however, 60fps is preferable (16.6ms per frame) that will be used as a meter of whether the rendering method meets the real-time criteria. The final image filtering was disabled for a better estimation of the rendering quality. Figure 8 shows the comparison of the rendering results with and without additional steps required for the caustics component calculation.

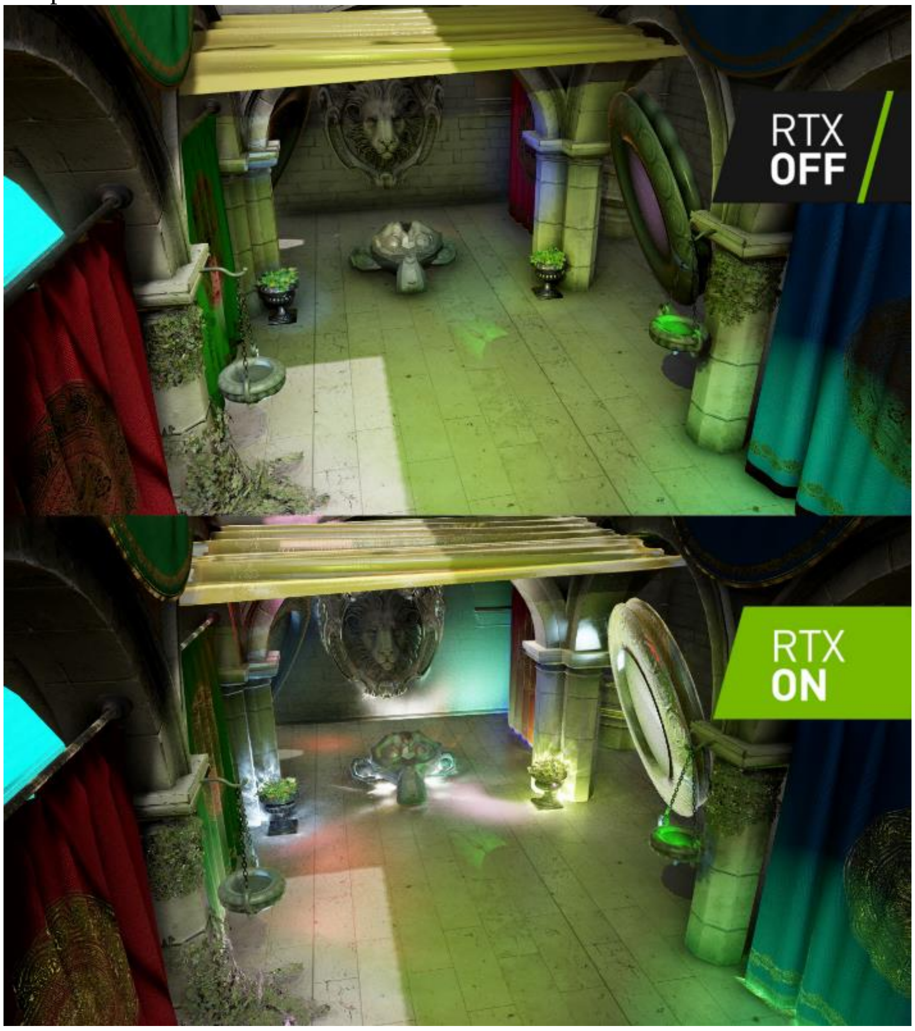

Figure 8: The rendering results without and with additional ray tracing steps

The scalability of the caustic rendering method when increasing the number of light sources was tested. The testing result is shown in Figure 9. Each additional light source required 1.3ms if it is visible from the virtual camera and 1.05ms if it is not.

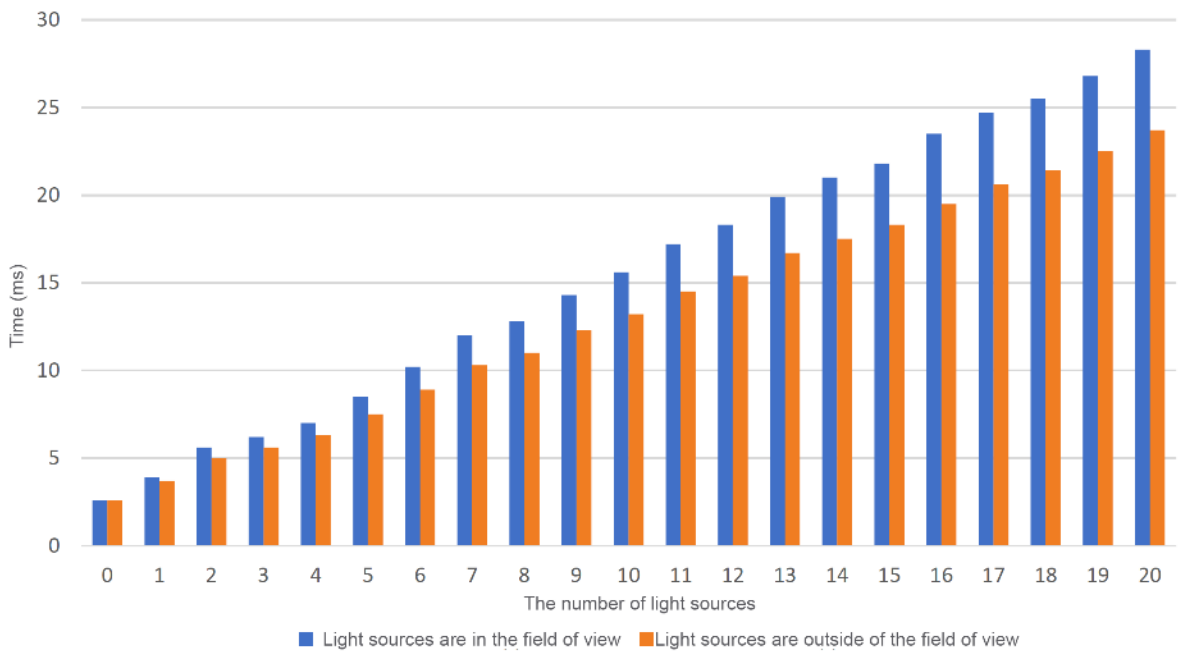

Figure 9: A frame rendering time depending on the number of light sources

In Figure 10 and Figure 11 the test results of rays per pixel variation are shown. As it can be seen the rendering quality of 0.75 rays per pixel is enough for caustics visualization, and the visible image noise can be reduced by using real-time filtering.

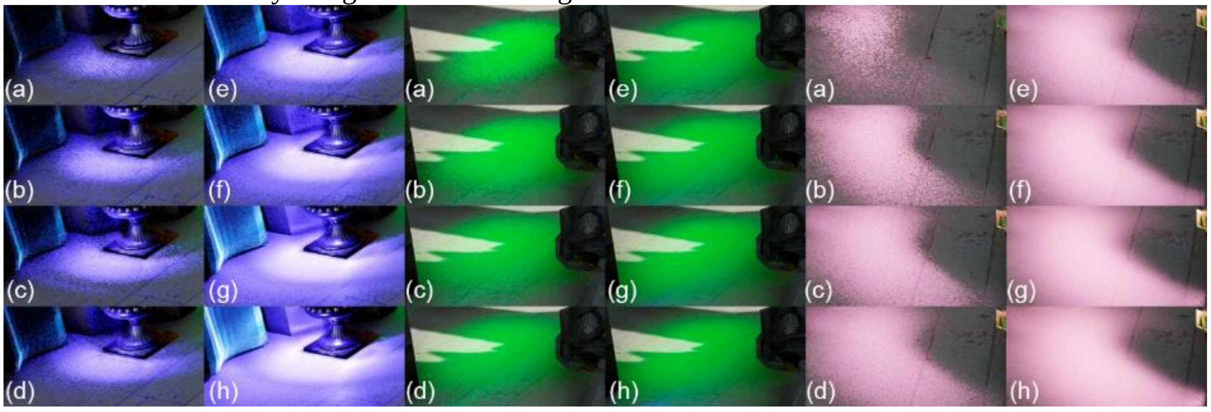

Figure 10: The caustics rendering quality depending on the number of rays per pixel: 0.25 rays per pixel (a), 0.5 rays per pixel (b), 0.75 rays per pixel (c), 1 ray per pixel (d), 2 rays per pixel (e), 4 rays per pixel(f), 8 rays per pixel (g), and 25 rays per pixel (h)

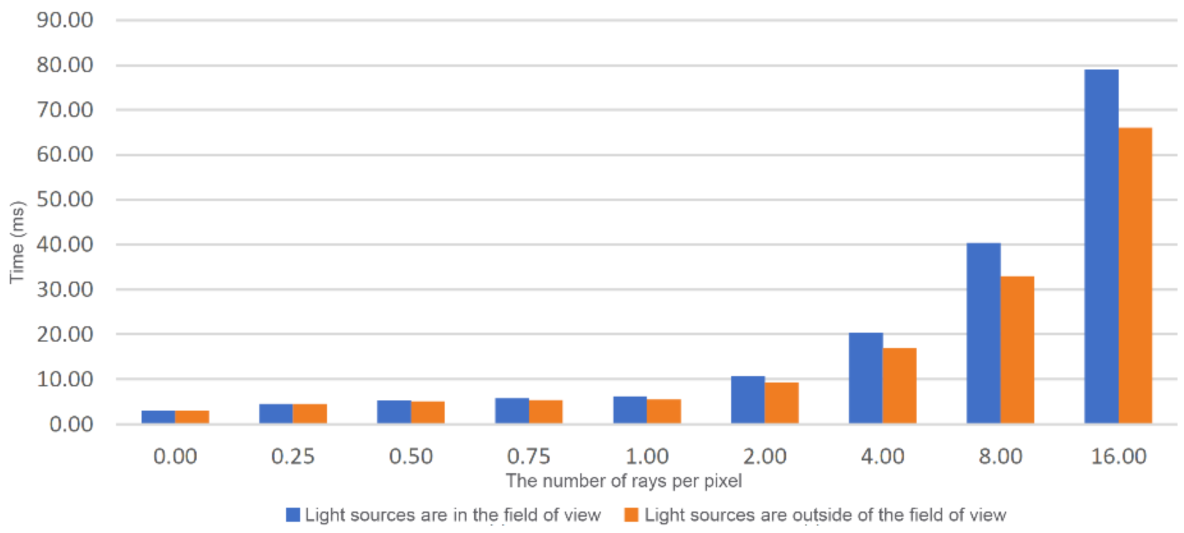

Figure 11: A frame rendering time depending on the number of rays per pixel

In Figure 12 and Figure 13 the testing result of maximum specular ray tracing depth variation is shown. As it can be seen from the presented images, increasing this parameter to more than 4 doesn't give any significant increase in the image quality. Moreover, the non-linear increase of the frame rendering time shows that in most cases the ray tracing stops before reaching the limit.

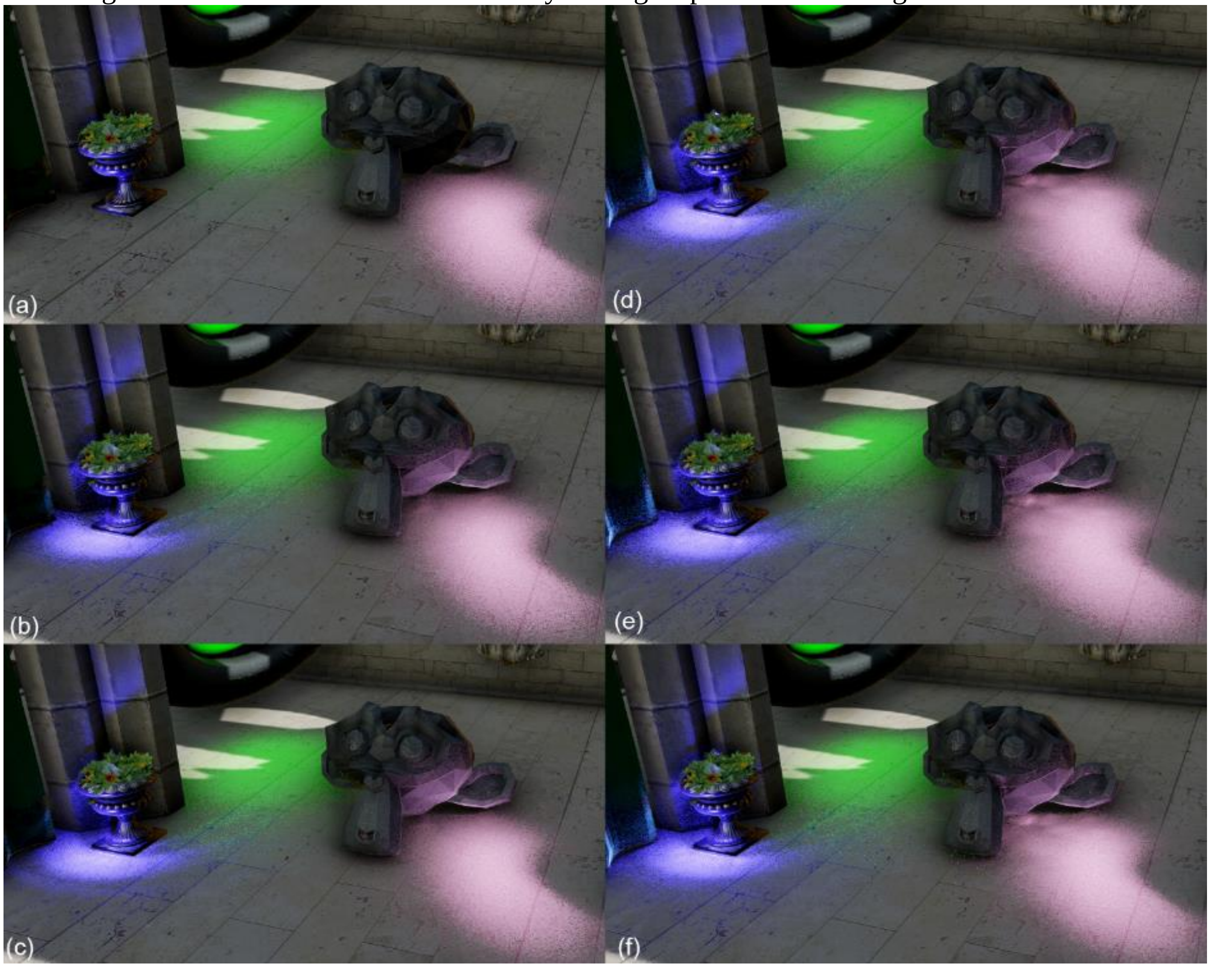

Figure 12: The caustics rendering quality depending on the ray tracing depth: 1 event (a), 2 events (b), 3 events (c), 4 events(d), 8 events(e), and 16 events (f)

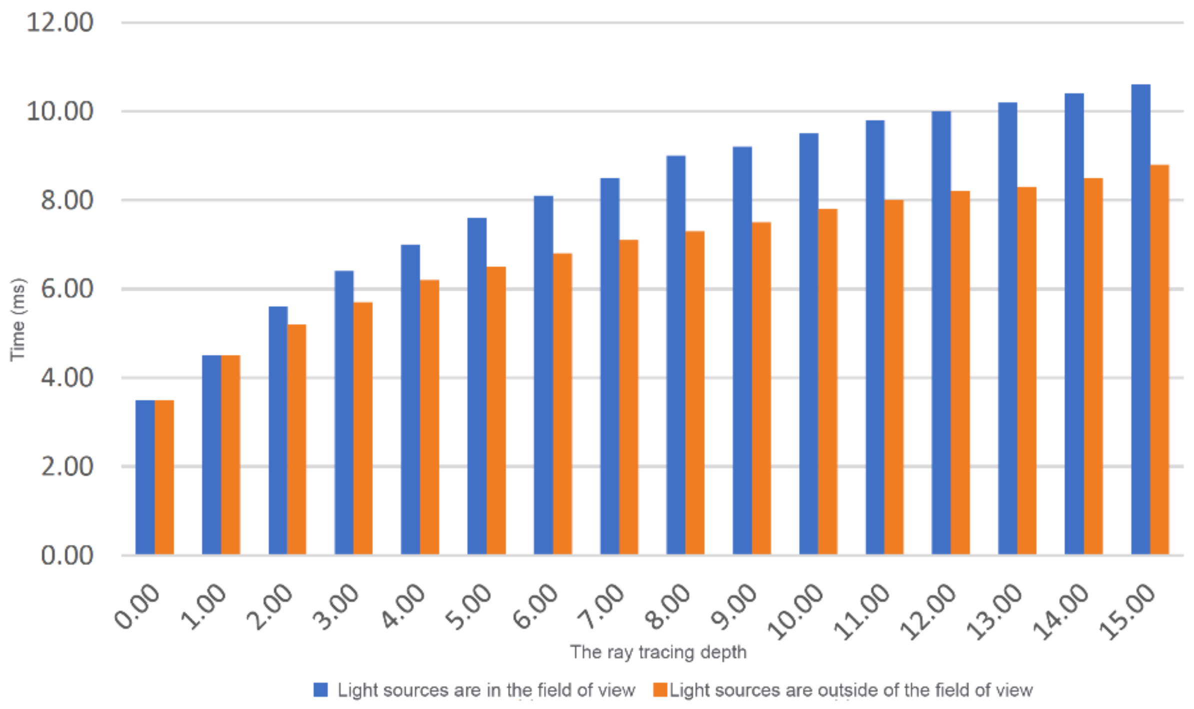

Figure 13: A frame rendering time depending on the ray-tracing depth

Testing showed that the presented rendering method both shows good image quality and meets the real-time criterion on the middle-level NVIDIA RTX GPU. The optimal rendering parameters are ray tracing depth equal to 3 events, and 0.75 rays per pixel. The number of rays per pixel could be lowered up to 0.5 if good filtering method is used.

5. Conclusion

In the scope of the current research, authors have developed the realistic real-time caustic visualization method that used DirectX Raytracing API and can be integrated into the DirectX Raytracing visualization pipeline. The caustics visualization speed linearly depends on the number of light sources that can result in caustic illumination. Testing results showed that the designed solution can be executed in real-time with up to 60fps. The achieved results may be used in the development of the modern games and in virtual reality systems that should significantly increase the immersion into the game or virtual world.

6. References

[1] M. Born, E. Wolf, Foundations of geometrical optics, M. Born (Ed.), Principles of optics: electromagnetic theory of propagation, interference and diffraction of light, Cambridge University Press, Cambridge, 1999. pp. 134-135. doi:10.1017/CBO9781139644181

[2] Microsoft, Rasterization Rules (Direct3D 9), 2018. URL: https://docs.microsoft.com/en-us/windows/win32/direct3d9/rasterization-rules.

[3] Microsoft, D3D Team, Announcing Microsoft DirectX Raytracing!, Microsoft Developer Blogs, 2018. URL: https://devblogs.microsoft.com/directx/announcing-microsoft-directx-raytracing/.

[4] NVIDIA Corporation, NVIDIA Turing GPU Architecture, WP-09183-001_v01, 2018. URL: https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf.

[5] J.T. Kajiya, The rendering equation, in: SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques, August 1986. pp. 143-150. doi:10.1145/15886.15902

[6] T.L. Kay, J.T. Kajiya, Ray tracing complex scenes, in: ACM SIGGRAPH computer graphics, 1986, Vol. 20, No. 4, pp. 269-278. doi:10.1145/15886.15916

[7] S. Chattopadhyay, A. Fujimoto, Bi-directional ray tracing, in: Computer Graphics 1987, Springer, Tokyo, 1987, pp. 335-343. doi:10.1007/978-4-431-68057-4_21

[8] E.P. Lafortune, Y. Willems, Bi-directional path tracing, in: Compugraphics' 93, 1993, pp. 145-153.

[9] I. Georgiev, J. Křivánek, T. Davidovic, P. Slusallek, Light transport simulation with vertex connection and merging, in: ACM Trans. Graph, 2012, Vol. 31, No. 6, pp. 192:1-192:10. doi:10.1145/2366145.2366211

[10] H.W. Jensen, P. Christensen, High quality rendering using ray tracing and photon mapping, in: ACM SIGGRAPH 2007 courses, 2007. doi:10.1145/1281500.1281593

[11] A.D Zhdanov, D.D. Zhdanov, Progressive backward photon mapping, Programming and Computer Software 47(3) (2021) 185–193.

[12] M. McGuire, D. Luebke, Hardware-accelerated global illumination by image space photon mapping, in: Proceedings of the Conference on High Performance Graphics 2009, 2009. doi:10.1145/1572769.1572783

[13] D.K. Bogolepov, D.P. Sopin, V.E. Turlapov, Simplified photon mapping for real-time caustics rendering, Programming and Computer Software 37(5) (2011) 229-235. doi:10.1134/S036176881105001X

[14] J. Jendersie, K. Rohmer, F. Brüll, T. Grosch, Pixel cache light tracing, in: Proceedings of the conference on Vision, Modeling and Visualization (VMV '17). Eurographics Association, Goslar, DEU, 2017, pp. 137–144.

[15] K. Iwasaki, Y. Dobashi, T. Nishita, A fast rendering method for refractive and reflective caustics due to water surfaces, in: Eurographics. 2003, pp. 283-291. doi:10.1111/1467-8659.t01-2-00708

[16] M. A. Shah, J. Konttinen, S. Pattanaik, Caustics Mapping: An Image-Space Technique for Real-Time Caustics, in: Transactions on Visualization and Computer Graphics, 2007, Vol. 13, pp. 272-280. doi:10.1109/TVCG.2007.32

[17] C. Wyman, G. Nichols, Adaptive caustic maps using deferred shading, in: Computer Graphics Forum, Vol. 28, No. 2, 2009. doi:10.1111/j.1467-8659.2009.01370.x

[18] Microsoft, DirectX Raytracing (DXR) Functional Spec, 2021. URL: https://microsoft.github.io/DirectX-Specs/d3d/Raytracing.html.

[19] I.S. Potemin, D.D. Zhdanov, A.D. Zhdanov, N.N. Bogdanov, A.G. Voloboy, Hybrid ray tracing method for photorealistic image synthesis in head-up displays, in: Proceedings of SPIE, Optical Design and Engineering VII, International Society for Optics and Photonics, 2018, Vol. 10690, pp. 106900I. doi: 10.1117/12.2312589

[20] T. Akenine-Moller, E. Haines, N. Hoffman, A. Pesce, M. Iwanicki, S. Hillaire, Real-Time Rendering, 4th ed. Boca Raton: A K Peters/CRC Press, 2018. doi:10.1201/b22086

[21] E. Haines, T. Akenine-Moller, Ray Tracing Gems High-Quality and Real-Time Rendering with DXR and Other APIs, Apress (2019).

[22] J. Hasselgren, J. Munkberg, M. Salvi, A. Patney, A. Lefohn, Neural Temporal Adaptive Sampling and Denoising, in: Computer Graphics Forum, No. 39(2), May 2020, pp. 147-155. doi:10.1111/cgf.13919

[23] D. Zhdan, Fast Denoising with Self Stabilizing Recurrent Blurs, GTC, 2020.

[24] Microsoft, DirectX-Graphics-Samples, GitHub, URL: https://github.com/microsoft/DirectX-Graphics-Samples.

[25] M. Pharr, W. Jakob, G. Humphreys, Physically Based Rendering: From Theory to Implementation, San Francisco, Morgan Kaufmann Publishers Inc. (2016).