The Real-Time Rendering of Gradient Media

Sergei Riabov [1], Andrey Zhdanov [1], Dmitry Zhdanov [1] and Ildar Valiev [2]

[1] *ITMO University, Kronverksky Pr. 49, bldg., St. Petersburg, 197101, Russia*
[2] *Keldysh Institute of Applied Mathematics, Miusskaya sq. 4, Moscow, 125047, Russia*

Abstract
In the real world, there are optical media that are difficult or impossible to simulate with classical real-time rendering methods. One of such media types is a gradient medium, in which light rays propagate along a curve. However, with advances in the capabilities of the real-time computer graphics rendering hardware, more sophisticated rendering algorithms, e.g., rendering based on the ray tracing technologies which uses a physically correct model of the light propagation and transformation, are becoming more widely used to achieve more realistic images. In the scope of the current article, the authors researched the capabilities of the current accelerated ray tracing-based realistic rendering technologies to perform the physically correct rendering of scenes containing the gradient medium. The presented method can operate in real-time which was proven by presenting the test images and aminations acquired from the implementation of the designed realistic rendering method of the gradient media with NVIDIA OptiX.

Keywords
Rendering, ray tracing, gradient media, NVIDIA OptiX

1. Introduction

With advances in the capabilities of real-time computer graphics rendering hardware, sophisticated rendering algorithms are becoming more widely used to achieve more realistic images. Nowadays, the approach to real-time rendering based on raytracing technologies that use a physical model of light propagation and transformation is gaining popularity.

In the real world, there are optical media that are difficult or impossible to simulate with classical real-time rendering methods. One of the types of such media is a gradient medium, in which light rays propagate along a curve, which complicates their modeling. Gradient media are characterized by an inhomogeneous distribution of the refractive index value. Examples of these environments are:

- the air heated by a hot body (e.g., the air above a fire, heated asphalt, or sand in the desert);
- the areas of mixing the water currents of different temperature and/or salinity;
- the water around the sugar cube dissolving in it;
- the atmosphere of the earth, as the air pressure and density, are changing with increasing altitude.

Physically accurate rendering of such optical effects can be applied in various fields. At first, it can be used to generate realistic images in computer games and movies. Another application of a physically accurate rendering model is data generation for machine learning algorithms. Often, machine learning algorithms use data that is not obtained from the real world but instead is artificially generated. The implementation of models of complex optical phenomena allows the creation of specific data that helps systems using machine learning work correctly in special situations.

GraphiCon 2021: 31st International Conference on Computer Graphics and Vision, September 27-30, 2021, Nizhny Novgorod, Russia
EMAIL: v4lonforth@gmail.com (S. Riabov); andrew.gtx@gmail.com (A. Zhdanov); ddzhdanov@mail.ru (D. Zhdanov); piv@gin.keldysh.ru (I. Valiev).
ORCID: 0000-0001-7559-0015 (S. Riabov); 0000-0002-2569-1982 (A. Zhdanov); 0000-0001-7346-8155 (D. Zhdanov); 0000-0003-2937-8480 (I. Valiev)

Also, such models allow you to create test environments for testing in the early stages of systems, which testing is difficult in real conditions. An example of such systems is the camera lens systems designed to be used on satellites since the earth's atmosphere is a gradient medium.

When solving the problem of light propagation in aт inhomogeneous environment, it is necessary to focus on the target audience. If physically correct modeling is required, then, depending on the problem, one can use either the wave optics solution [1] or the eikonal equation solved by the well-known ray methods [2, 3]. These solutions are used in computational optics, but they have a very limited application area, e.g., gradient lenses, which usually have only one surface which can be intersected by the beam of rays. These methods allow finding a physically correct solution, however, they are quite ineffective if applied to scenes with thousands of geometric primitives that constrain the gradient environment.

Other solutions are usually artificial by their nature, or the non-rectilinearity of the ray trace path is caused by completely different effects. In [4], a non-rectilinear ray tracing method was proposed. The environment is defined by an equation of motion in the form of the ordinary differential equation (ODE). This equation is set by the user and can be solved by any explicit numerical integration scheme. For simplicity, the first-order Euler integration is considered, which, in our opinion, is not entirely correct for a physically correct model of a gradient medium. The authors consider a number of models to describe a medium, but they have nothing in common with a model of a real medium and the eikonal equation. In addition, the iterative approach to the calculation of the possible point of intersection with the boundary surface is not considered.

Works [5, 6] are mainly demonstrative by their nature and are aimed at allowing the audience to study cosmic phenomena that affect the non-rectilinear propagation of light and how this effect can be visualized. The older work [7], which introduces the concept of non-rectilinearity of light propagation due to the action of some external forces, such as gravity, can be attributed to a similar demonstrative paper kind. The author does not limit his solution to the problems of modeling light phenomena; in his opinion, this solution can also be applied to modeling dynamic systems. The same is with work [8], which shows that not only Euler's method but also the Runge-Kutta method can be used to simulate the ray path in a medium defined by the equation of motion in the form of the ODE system. However, neither of these works considers the physical nature of the non-rectilinear propagation of light, determined by the eikonal equation in the geometrical optics.

The scattering of light in the medium can be also attributed to the problems of non-rectilinear light propagation. From the software implementation point of view, [9] is rather interesting and detailed work, however, this work does not consider continuous ray paths, the models are reduced to simplified models of light scattering on particles, optimal sampling methods, and the construction of beautiful visual effects that may be far from physical reality. Methods for determining areas of rectilinear light propagation can be correctly considered only for a scattering medium, but these approaches are in noe applicable for a continuous gradient medium. The same remark can be applied to the works [10, 11], which consider the solution of the rendering equation in an inhomogeneous medium by the Monte Carlo method. The methods for determining the optimal "rectilinear" distance are physically correct only for media with volume scattering.

The current work aims to design and implement a computer model of light propagation in a gradient medium based on physically correct solution and real-time ray tracing methods. In addition, the tasks of the work include researching a mathematical model of light propagation, as well as calculating illumination in a gradient medium.

2. Gradient media model

To design a computer model of the light propagation in a gradient medium, it is necessary to solve two problems. At first, it is the calculation of the ray path in the gradient medium and, secondly, the calculation of the luminance values in the scene containing the gradient media.

In the vector form, the geometrical optics approximates the light propagation in the gradient medium with eikonal equation [1, 2] that is expressed as:

$$\frac{d}{ds}\left[n\big(\vec{r}(s)\big)\frac{d\vec{r}(s)}{ds}\right] = \nabla n\big(\vec{r}(s)\big) \tag{1}$$

where $n(\vec{r}(s))$ is the medium refraction index at the $\vec{r}(s)$ point, $\vec{s}(\vec{r}(s)) = \frac{d\vec{r}(s)}{ds}$ is the unit direction vector of the light propagation in the gradient medium, s is the curved ray path length. In a homogeneous medium, the refraction index does not depend on the position and as result, the ray path becomes a straight line.

The simplest solution would be the discretization of the gradient media to areas with a constant refraction index. As result, the ray path would become a set of straight segments with transformation on the boundaries of the media layers. Despite the attractive simplicity of this approach, it might lead either to a significant error in the resulting ray path or to slowing down the ray tracing because of an increasing number of segments. In addition, in case if the gradient medium has complex properties it might result in an additional complex task to find the internal boundaries between layers of the gradient medium. This approach is shown in Figure 1.

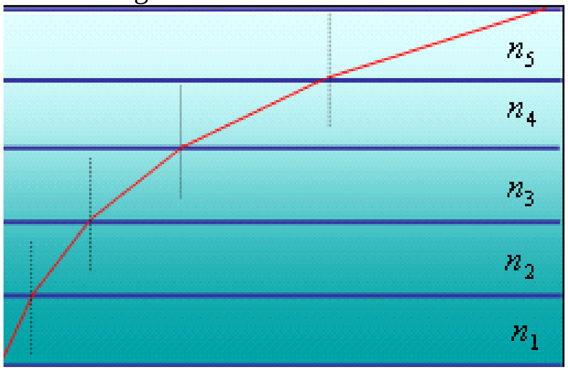

Figure 1: Light propagation in the gradient medium

Another approach to solving the eikonal equation is by using the Runge-Kutta method. Using technology proposed in [2] the following additional variables are introduced:

$$\begin{cases} dt = \dfrac{ds}{n} \\ \vec{T}(\vec{r}(s)) = \dfrac{d\vec{r}(s)}{dt} = n(\vec{r}(s))\vec{s}(\vec{r}(s)) = n(\vec{r}(s))\dfrac{d\vec{r}(s)}{ds} \\ \vec{D}(\vec{r}(s)) = n(\vec{r}(s))\nabla n(\vec{r}(s)) \end{cases} \tag{2}$$

where t is the parameter of the ray path segment length, $\vec{T}(\vec{r}(s))$ is the optical ray path vector, and $\vec{D}(\vec{r}(s))$ is the vector of refractive index variation.

After substituting variables (3) into an equation (2) we would get the following equation:

$$\frac{dT(\vec{r}(s))}{dt} = D(\vec{r}(s)) \tag{3}$$

which is the first-order differential equation that could be solved numerically using the Runge-Kutta method [3]. We suppose that ray in the gradient medium is propagated with portions corresponding to some Δt_i step, where *i* is a step-index. For this need additional variables are introduced:

$$\begin{cases} \vec{A} = \Delta t \vec{D}(\vec{r}_i(s)) \\ \vec{B} = \Delta t \vec{D}\left(\vec{r}_i(s) + \dfrac{\Delta t}{2}\vec{T}(\vec{r}_i(s)) + \dfrac{\Delta t}{8}\vec{A}\right) \\ \vec{C} = \Delta t \vec{D}\left(\vec{r}_i(s) + \Delta t \vec{T}(\vec{r}_i(s)) + \dfrac{\Delta t}{2}\vec{B}\right) \\ \vec{T}(\vec{r}_{i+1}(s)) = \vec{T}(\vec{r}_i(s)) + \dfrac{1}{6}(\vec{A} + 4\vec{B} + \vec{C}) \end{cases} \tag{4}$$

Then with help of these variables, the new parameters of the ray (origin, direction, and eikonal) for the next ray path point are approximated:

$$\begin{cases} \vec{r}_{i+1}(s) = \vec{r}_i(s) + \Delta t\left[\vec{T}(\vec{r}_i(s)) + \dfrac{1}{6}(\vec{A} + 2\vec{B})\right] \\ \vec{s}_{i+1}(s) = \dfrac{\vec{T}(\vec{r}_{i+1}(s))}{n(\vec{r}_{i+1}(s))} \end{cases} \tag{5}$$

These steps are repeated until the next ray point ($\vec{r}_{i+1}(s)$) appears outside the gradient medium geometry. Then the ray returns to the previous position $\vec{r}_i(s)$ and reiterations between $\vec{r}_{i+1}(s)$ and $\vec{r}_i(s)$ continue until the desired approximation tolerance between $\vec{r}_{i'}(s)$ point and scene geometry is achieved.

2.1. Ray path in the gradient medium

The calculation of the ray path is performed using numerical methods. In this case, the calculation is performed by splitting the entire ray path into smaller segments according to the method described above. So, the algorithm for calculating the ray path consists of the following steps:

1. obtain the initial parameters of the ray origin and direction after hitting the gradient medium;
2. choose a certain parameter *t*, which is responsible for the length of the ray path segment;
3. calculate the new ray origin and direction at the end of the segment;
4. check the intersection of the calculated segment with the scene objects: in the case if the intersection was found (that is if a rectilinear ray segment intersects the scene geometry) then proceed to step 5; otherwise, the ray stays in the gradient medium, so go back to step 1 with next values of the ray origin and direction;
5. find the intersection of the calculated segment with the boundaries of the gradient medium. In case if the intersection was found use the bisection method to find the coordinates of point where ray leaves the gradient medium;
6. leave the gradient medium.

Steps two to five are repeated until the ray leaves the gradient medium. After leaving the gradient medium the ray is traced further in a standard mode. The described above steps are schematically shown in Figure 2.

To implement the ray tracing in the gradient medium the gradient medium program interface should implement methods that would:

- choose the optimal ray travel parameter from the point inside the medium in the specified direction. If the medium does not have the gradient properties this value should be returned as infinite and standard mode ray tracing should proceed;
- calculate the medium refractive index at the specified point inside the medium;
- calculate the medium refractive index gradient and Laplacian at the specified point inside the medium;
- transfer the ray between two points inside the medium and calculate new ray direction at the endpoint;
- calculate the optical and geometric path of the ray between two points inside the medium;
- calculate the absorption of the ray when it travels between two points inside the medium.

2.2. Luminance value estimation in the gradient medium

When ray propagates in the gradient medium and finds the point of intersection with the gradient medium boundary a problem of estimating the luminance value might appear if this point is illuminated through the gradient medium boundary, for example, if a light source is located inside the gradient medium. The possible solutions are considered below, and the limitations of the approach used in the current work are defined.

When using the ray-tracing method for rendering a scene image, the fastest way to calculate the illumination of a scene point is by tracing rays from that point to each of the scene lights. So, when a traced ray hits any of the scene objects, rays are traced from the point of impact to all light sources. If such a ray hits any other object on the path to the light source, that means that the light source is shadowed by this object. However, the presence of a gradient medium environment in the scene creates problems for this method of lighting calculations. If, when tracing a ray from a point to a light source, the ray falls into a gradient medium, then its path will be distorted, due to which the ray may intersect another object, which may cause a shadow at this point. However, there may be a different ray path between the light source and the target point that does not intersect other objects in the scene. As a result, illumination might be calculated incorrectly.

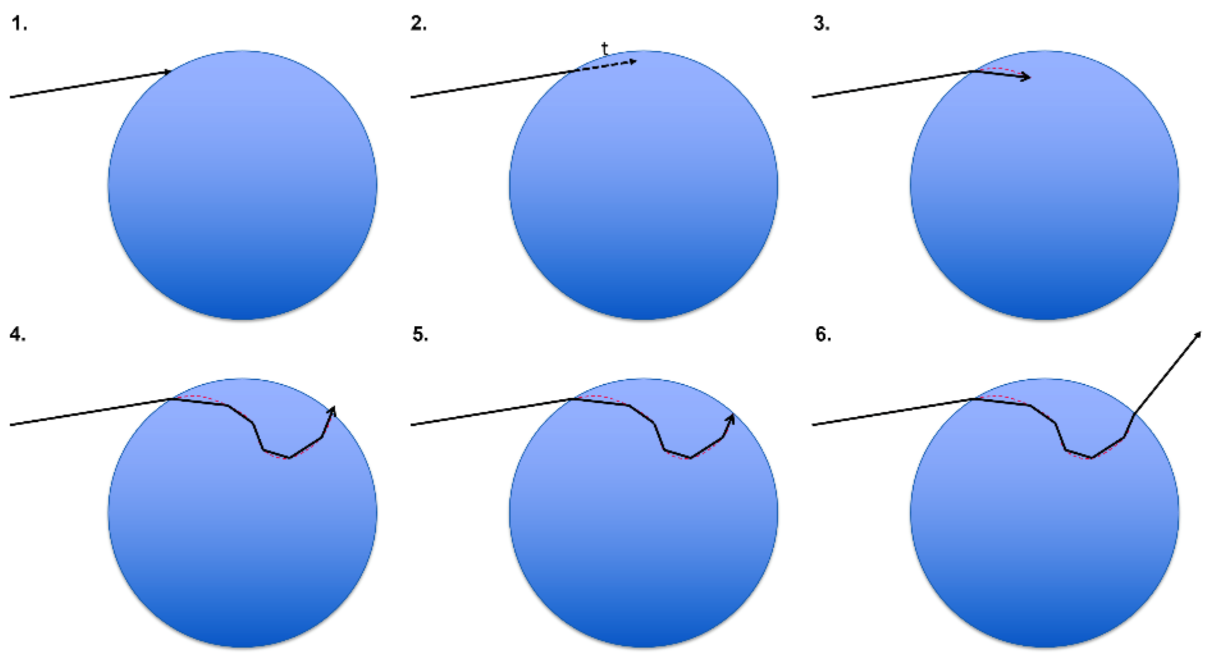

Figure 2: Ray tracing in the gradient medium

So, in the case of a gradient medium, the direct illumination changes to the caustic illumination [12], and the BDF sampling method (the rays are randomly sampled at the gradient medium boundary and the search for an intersection with the scene light sources is performed; if the intersections are found then the luminance values are taken from the lights) should be used to account for it. However, the BDF sampling is not applicable for point of parallel light sources, which are most commonly used in video games, which makes this solution fairly useless. The simplest solution to this problem may be to ignore the gradient environment when calculating the direct illumination. In this case, a standard calculation of illumination occurs – when a ray hits the surface of an object, rays are traced directly to the light sources from the point of impact, and when these rays intersect with the gradient medium, the rays simply continue to propagate in the original direction. Obviously, this method leads to a less realistic image, however, quite often the gradient environment can have a negligible effect on the illumination in the scene, and the result may be acceptable. So, for the designed algorithm we used this simplified method of the direct luminance calculation.

To calculate more realistic lighting, the photon mapping method [13] can be used. The photon mapping method can be used to account for both direct, indirect, and caustic types of illumination by adding variable radius photons for different kinds of illumination [14], however, this method of calculating illumination is much more difficult to implement and it reduces the performance of ray tracing in a gradient environment, but it allows to get a more realistic rendering model, in contrast to the first approach. The main problem of the photon mapping approach is the high computation load caused by it that makes it hardly possible to be implemented in real-time using modern hardware. As result when designing the algorithm, we chose to use a simplified illumination scheme and do not consider any solutions for the indirect illumination.

2.3. Implementation

When implementing algorithms using the modern accelerated ray-tracing technologies there are several most commonly used APIs, these are DirectX Raytracing (DXR) [15], Vulkan [16], and NVIDIA OptiX [17]. The first two are aimed mainly at real-time applications with gaming applications, while NVIDIA OptiX [18] is aimed at the higher quality of the rendered image. So, we used the OptiX API for the implementation of the described above algorithms.

Using the OptiX API we implemented the special spherical object with a gradient medium in which the refraction index can be functionally defined depending on a point position inside the sphere.

The algorithm described above was implemented using the OptiX API, and the parts of the source code related to gradient medium implementation are shown below. Listing 1 contains the implementation of an arbitrary method used to calculate the optical ray, refractive index gradient, and segment travel distance. The following idea is used to calculate the segment travel distance (t): as we know the minimum and maximum values of the refractive index, we know the range that the derivative of the refractive index should lay in. This allows us to use the derivative of the refractive index in the direction of the ray transfers to scale the derivative value into the range between predefined minimum and maximum distance values so that the higher is the gradient value the smaller is the segment length.

Listing 1

Approximation of the optical ray, refractive index variation, and delta values

```
// The constant delta value for gradient approximation
static const float delta = 0.001f;

// Get the medium optical ray
float3 getMediumOpticalRay(float3 position, float3 direction,
                           const Medium& medium)
  {
  return medium.getMediumRefraction(position) * direction;
  }

// Get the medium refraction variation
float3 getMediumRefractionVariation(float3 position,
    const Medium& medium)
  {
  return medium.getMediumRefraction(position) *
         medium.getMediumGradient(position);
  }

// Choosing the delta value for the segment
float getMediumDelta(float3 position, float3 direction,
                     const Medium& medium)
  {
  // Ray deviation from medium gradient
  float dir_der =
    abs(dot(direction, medium.getMediumGradient(position)));
  // Normalization to the medium variation and reverse to 1
  float grad01 = clamp(1.0f - dir_der / medium.amplitude, 0.0f, 1.0f);
  // Interpolate between min and max delta on medium variation
  // in the direction
  float delta = interpolate(medium.min_t, medium.max_t, grad01);

  return delta;
  }
```

Listing 2 contains the implementation of an iteration of calculating the next ray origin and direction.

Listing 2

Single iteration of the ray in a gradient medium transformation

```
RayTransform iterateMediumRayTransform(RayTransform previous_transform,
    float delta, const Medium& medium)
  {
  const float3 position = previous_transform.position;
  const float3 direction = previous_transform.direction;

  float3 optical_ray = getMediumOpticalRay(position, direction);
  float3 refraction_variation = getMediumRefractionVariation(position);

  // Calculation of Runge-Kutta coefficients
  float3 a = delta * refraction_variation;
  float3 b = delta * getMediumRefractionVariation(position +
             delta / 2.0f * optical_ray + delta / 8.0f * a);
  float3 c = delta * getMediumRefractionVariation(position +
             delta * optical_ray + delta / 2.0f * b);

  float3 next_optical_ray = optical_ray + (a + 4 * b + c) / 6.0f;
  float3 next_position =
    position + delta * (optical_ray + (a + 2 * b) / 6.0f);
  float3 next_direction =
    next_optical_ray / medium.getMediumRefraction(next_position);

  previous_transform.position = next_position;
  previous_transform.direction = next_direction;

  return previous_transform;
  }
```

Listing 3 contains the implementation of ray hit the gradient boundary or other object inside the gradient medium check.

Listing 3

Check of ray hitting the gradient medium boundary

```
// Start and end ray positions in the world coordinate system
float3 start_position_world =
  localToWorldCoordinates (current_ray_transform.position, sphere);
float3 end_position_world =
  localToWorldCoordinates (next_ray_transform.position, sphere);
// Ray direction from start to end position in the world coordinate
system and pay path
float3 direction_world = end_position_world - start_position_world;
float max_length = length(direction_world);
direction_world = direction_world / max_length;

// Trace ray from the start position to end point on defined length
RadiancePRD new_prd = traceRadianceRay(start_position_world,
  direction_world, 0.0f, max_length, prd.depth, prd.importance,
  FROM_GRADIENT);

// Check if ray hits any object inside the gradient medium
if (new_prd.hit_type == FROM_GRADIENT)
  {
  prd.result = new_prd.result;
  prd.hit_type = new_prd.hit_type;
  break;
  }
```

Listing 4 contains the implementation of the ray exiting the gradient medium point approximation check. The iterations would continue until the desired approximation quality is achieved.

Listing 4
Approximation of the end ray position when leaving the gradient medium check

```
// The "importance" parameter is used to pass the ray path length
float ray_length = new_prd.importance;
float t = ray_length / max_length;

// if the accuracy or maximal number of iterations is achieved
if (iterations < medium.max_iterations &&
    delta_multiplier > medium.boundary_precision)
  {
  delta_multiplier *= t * 0.95f;
  }
else
  {
  float3 interpolated_position =
    interpolate(start_position_world, end_position_world, t);
  float3 interpolated_direction =
    interpolate(current_ray_transform.direction,
                next_ray_transform.direction, t);

  new_prd = traceRadianceRay(interpolated_position,
    interpolated_direction, params.scene_epsilon, 1e16f,
    prd.depth, prd.importance, HIT_OBJECT);

  prd.result = new_prd.result;
  prd.hit_type = new_prd.hit_type;

  break;
  }
```

3. Results

To test the approach described above we used the modified scene from the OptiX package. This scene contains a rectangular surface and two balls above it which are made of glass and steel. The sphere with a gradient medium was placed in front of these two balls partly covering them from the observer. The testing was performed on the mobile NVIDIA GeForce RTX 2060 GPU. The test scene is shown in Figure 3.

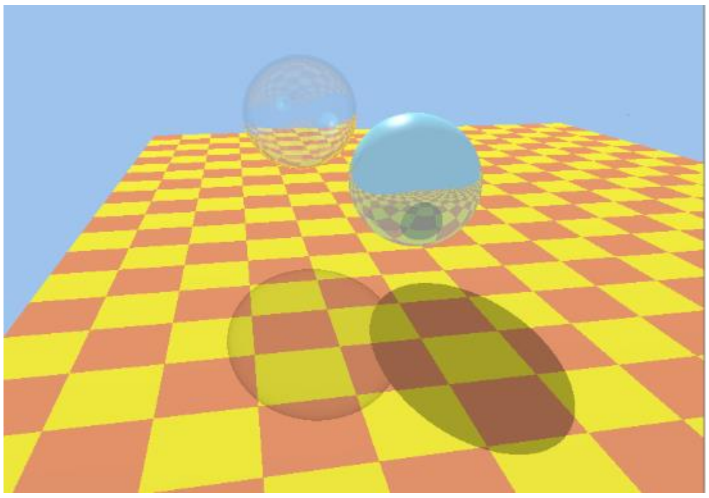
Figure 3: The test scene

In Figure 4 the graph of the refraction index variation depending on the distance from the center of the sphere with the gradient medium is shown. In (a) case it has maximum value in the center of the spere and drops to environment refractive index when approaching the border of the refractive media area. In (b) case it has the opposite gradient direction. The corresponding rendering results are shown in Figure 5. The gradient medium area is highlighted with the white circle. The rendering was performed with 30fps that meets the real-time rendering criterion.

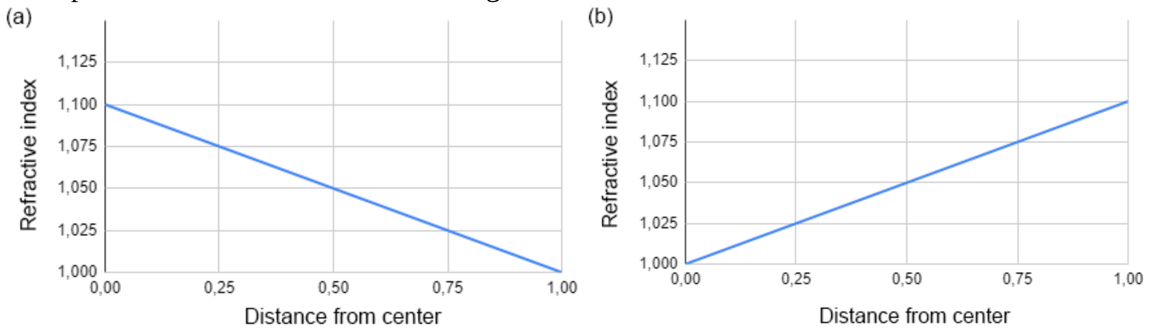

Figure 4: Gradient medium refraction index

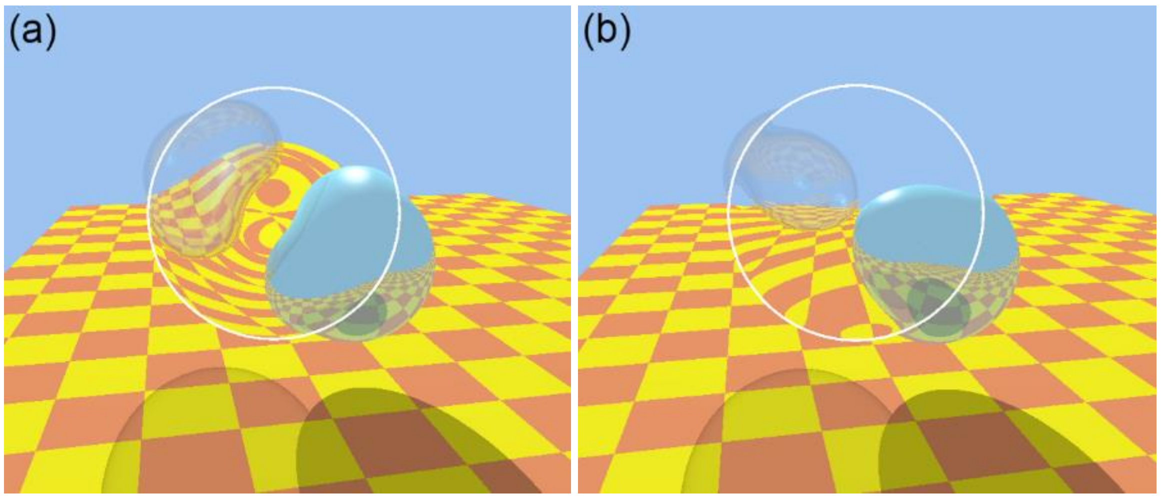

Figure 5: Rendering results of the scene with a gradient medium

The method was also tested with a procedurally created medium gradient defined by the Perlin noise. The rendering results are shown in Figure 6. The rendering performance was 15fps, however, the decrease of speed was caused mainly by the low efficiency of the used implementation of the Perlin noise generator.

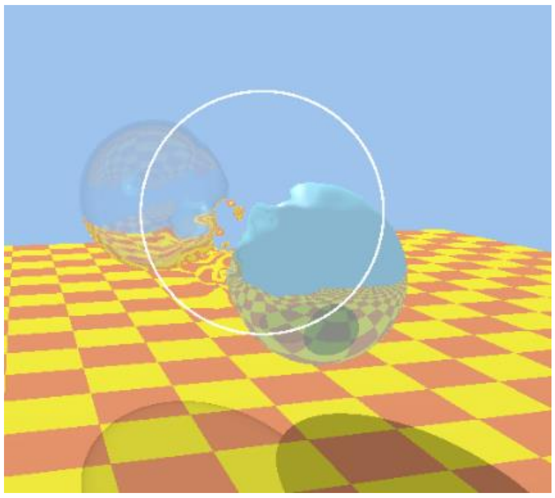
Figure 6: Rendering results of the scene with the Perlin noise gradient medium

4. Conclusion

In the scope of the current research, the gradient medium model was implemented to be rendered in real-time using NVIDIA OptiX API. The further research direction would be methods for more effective choice of the ray travel path distance for faster ray travel in the gradient medium and the realistic illumination of the scenes containing the gradient medium in real-time by methods of forward or backward photon mapping.

5. Acknowledgements

The research was partially supported by state financial support of RFBR grant No. 19-01-00435.

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] A. Zhdanov, D. Zhdanov, V. Sokolov, I.S. Potemin, S. Ershov, V. Galaktionov, Problems of the realistic image synthesis in media with a gradient index of refraction, in: Optical Design and Testing X, International Society for Optics and Photonics, 2020, Vol. 11548, pp. 115480W.

[3] J.C. Butcher, On the implementation of implicit Runge-Kutta methods, BIT Numerical Mathematics 16(3) (1976) 237–40. doi:10.1007/BF01932265

[4] D. Weiskopf, T. Schafhitzel, T. Ertl, GPU-Based Nonlinear Ray Tracing, Computer Graphics Forum 23 (2004) 625-633. doi:10.1111/j.1467-8659.2004.00794.x

[5] O. James, E. von Tunzelmann, P. Franklin, K.S. Thorne, Gravitational lensing by spinning black holes in astrophysics, and in the movie Interstellar, Classical and Quantum Gravity 32(6) (2015) 065001. doi:10.1088/0264-9381/32/6/065001

[6] O. James, E. von Tunzelmann, P. Franklin, K.S. Thorne, Visualizing Interstellar's wormhole, American Journal of Physics 83(6) (2015) 486-499.

[7] E. Gröller, Nonlinear ray tracing: Visualizing strange worlds, The Visual Computer 11 (1995) 263–274. doi:10.1007/BF01901044

[8] D. Weiskopf, T. Schafhitzel, T. Ertl, GPU-Based Nonlinear Ray Tracing, Computer Graphics Forum 23 (2004) 625-633. doi:10.1111/j.1467-8659.2004.00794.x

[9] J. Fong, M. Wrenninge, C. Kulla, R. Habel, Production volume rendering, SIGGRAPH 2017 course, in: ACM SIGGRAPH 2017 Courses (SIGGRAPH '17), 2017, Article 2, pp. 1–79. doi:10.1145/3084873.3084907

[10] S. Marschner, Volumetric Path Tracing, 2015. URL: https://www.cs.cornell.edu/courses/cs6630/2015fa/notes/10volpath.pdf

[11] P. Shirley, Ray Tracing: The Next Week, 2020. URL: https://raytracing.github.io/books/RayTracingTheNextWeek.html#volumes

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

[13] 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

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

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

[16] G. Sellers, J. Kessenich, Vulkan programming guide: The official guide to learning vulkan, Addison-Wesley Professional, 2016.

[17] How to Get Started with OptiX 7, 2021, URL: https://developer.nvidia.com/blog/how-to-get-started-with-optix-7/

[18] S.G. Parker, J. Bigler, A. Dietrich A, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, M. Stich, Optix: a general purpose ray tracing engine, Acm transactions on graphics (TOG) 29(4) (2010) 1–3. doi: 10.1145/1778765.1778803