

Lens Library to Facilitate Composing of Virtual Scene for Optic Experiments

Victor Debelov¹ and Nikita Dolgov²

¹ *Institute of Computational Mathematics and Mathematical Geophysics of Siberian Branch of Russian Academy of Sciences, 6, Ac. Lavrentieva ave., Novosibirsk, 630090, Russia*

² *Novosibirsk State University, 1, Pirogova str., Novosibirsk, 630090, Russia*

Abstract

While the mathematical modeling of optical phenomena, a computer calculation is often performed, confirming the conclusions made. To do this, a virtual computer model of the optical installation is created in the form of a 3D scene. Also, virtual scenes are often used in training when creating presentations.

This paper describes the SphL library, which provides a convenient assignment of spherical lenses and the calculation of the interaction of linear polarized light rays with them. It is focused on applications that use ray tracing. It is known that light of any polarization can be represented on the basis of the mentioned one. The reflected and all rays passing through the lens that arise due to internal reflections are calculated from the ray incident on the scene object. The number of internal reflections is set by the parameter. All output rays are calculated based on the application of Fresnel's equations and are characterized by intensity values and polarization parameters.

In this version of SphL, the main objects at the end-user level are spherical lenses, since they are most often used in optic installations. They are constructed on the basis of the application of the set-theoretic intersection of geometric primitives: a half-space, a sphere, a cone, a cylinder and their complements to the scene space. An advanced user can build their own objects by analogy, for example, cylindrical lenses.

Keywords

Optical experiment, virtual scene, spherical lenses, optically isotropic objects, linear polarized light, ray tracing

1. Introduction

In the mathematical modeling of natural phenomena, computer experiments are often performed with the similar computer model to confirm its reliability. The construction and analysis of a mathematical model often leads to the creation of a 3D scene, which is a virtual analogue of a real installation. Another area of creating virtual 3D scenes is education, especially when it is necessary to demonstrate a phenomenon physically correctly. In optics, when obtaining images, lenses are used in one way or another, for example, a lens, an eyepiece, an eye.

The idea to develop a library of lenses arose after conducting experiments on interference modeling, we often had to change the shape and parameters, and even the type of lens. We came to the conclusion that such objects should be isolated so that the corresponding software modules are responsible for their interaction with linear polarized light rays. Why linear polarized light?

1. Historical background. In connection with the development of algorithms for photorealistic visualization of scenes involving optically anisotropic transparent crystals, one considered only polarized, moreover, linear polarized rays, since a light ray with any polarization incident on these

GraphiCon 2021: 31st International Conference on Computer Graphics and Vision, September 27-30, 2021, Nizhny Novgorod, Russia

EMAIL: debelov@oapmg.sccc.ru (V. Debelov); nikitabrn1999@gmail.com (N. Dolgov)

ORCID: 0000-0002-7577-4700 (V. Debelov); 0000-0001-5622-3586 (N. Dolgov)



© 2021 Copyright for this paper by its authors.

Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

crystals generates up to four linear polarized rays (up to two reflected and two refracted) at the boundary [1].

2. This does not limit the freedom of choice. Light of any polarization (unpolarized, partially polarized, polarized) can be represented as a combination of a certain number of linear polarized rays with a lower intensity, see [1].

3. Note one very important property for ray tracing: when a ray falls on an optically isotropic object, regardless of the state of polarization, the trajectory of the ray is the same. The polarization of the incident ray affects the state of polarization and the intensity of the reflected and refracted rays obtained when interacting with the surface. This is especially evident when performing calculations using Fresnel's formulas [1, 2]. This property does not hold for anisotropic objects.

In [2], the prerequisites for creating a library of lenses, new features compared to existing software were identified. Thus, this article is a natural continuation of the work [2] and contains technical information about the contents of the library and how to use it. First of all, we paid attention to spherical lenses, as the most common. Indeed, in the vast majority of cases, in order to visualize the required optical effects, it is necessary to introduce a lens (camera lens, etc.) into a virtual scene, as well as into a real experimental stage.

The second section describes the contents of the library and the suggested principles of lens design. The third section is devoted to the conclusions.

2. Library of spherical lenses

The lens library should make it easier for the user to design spherical lenses, add them to a 3D scene, and process the interaction of rays of linear polarized light with them. This section describes the requirements that the software package SphL and its implementation must meet. The book [3] describes in detail all types of thin spherical lenses. Since this is the most common type of lenses, we decided to provide users of the software package with a simple design of all six types of spherical lenses: biconvex, plano-convex, concave-convex, biconcave, plano-concave, convex-concave (see Figure 1). Each lens can have a cylindrical or conical rim. The rim is the surface of the lens attachment; it can be made of both transparent (working surface) and opaque material. We decided not to limit ourselves to only thin lenses, but to allow the use of similar geometric shapes of any size. Moreover, since we perform physically correct ray tracing, there are no problems with accuracy as in the cases of using the approximations of thin [3] or thick lenses [4].



Figure 1: Six types of spherical lenses: biconvex, plano-convex, concave-convex, biconcave, plano-concave, convex-concave. The image is taken from [3]

Our experience has shown that the set of these lenses significantly facilitates the creation of scenes and the execution of virtual optical experiments. Note that even complex optical devices, for example, polarizing microscopes (see Figure 2), are constructed mainly with the help of these lenses: each transparent object in the design of the microscope is a simple spherical lens or is composed of several spherical lenses [5].

Since it is necessary to take into account the polarization of light to calculate interference, the software package must be able to process the interaction of rays of linear polarized monochromatic light with lenses. When tracing a ray through a lens, according to [2], three possible events are taken into account:

- **Event 1.** Total internal reflection. In this case, the user can receive two calculated linear polarized rays, and further ray tracing stops.
- **Event 2.** In this case, the user receives the ray that hits the rim and the coordinates of the hit point, as well as further ray tracing stops.

- Event 3.** The next output ray of linear polarized light has been successfully calculated, or the specified tracing depth inside the lens has been reached. Tracing depth is determined by a separate parameter. For example, if the user has set the tracing depth k , then $k + 1$ output rays should be calculated (see Figure 3).

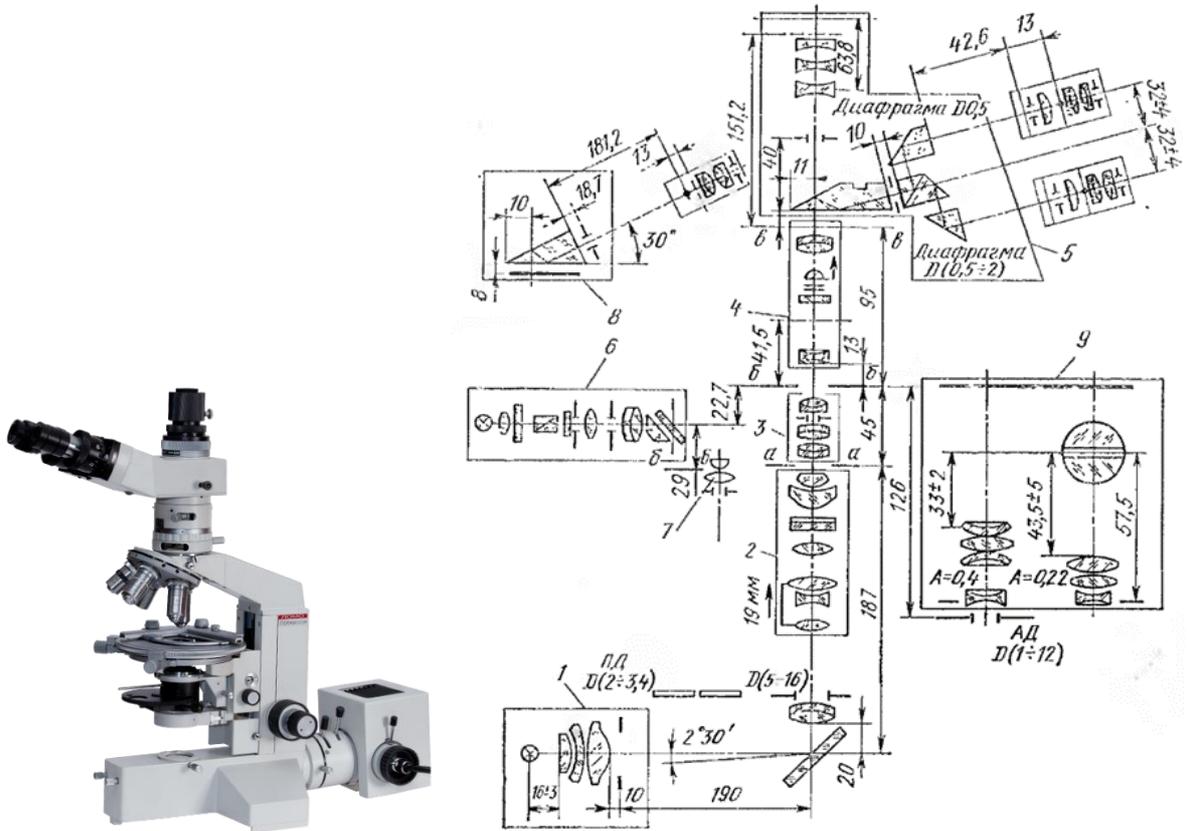


Figure 2: On the left is a polarizing microscope (photo from [6]), on the right are the components of the device of a polarizing microscope (from the book [5])

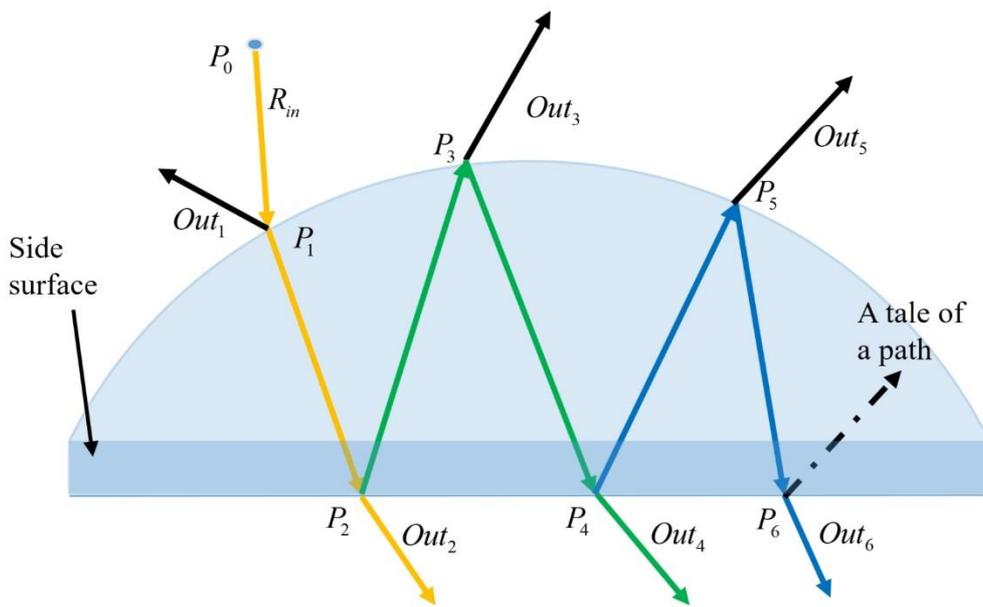


Figure 3: Subtree of rays of depth 5 in the lens. Notation: R_{in} is incident ray, Out_x are rays coming from the lens, P_x are treetops (nodes), E_x are internal reflected rays (image from [2])

To construct lenses, a set-theoretic intersection operation is used over several geometric primitives. The set of primitives includes a half-space, a part of the space inside or outside a sphere, a cylinder, a cone. We consider only objects with a simply connected boundary.

2.1. Half-space

The half-space of the scene can be set by specifying a plane in three-dimensional space using a point and a normal. The normal indicates the selected half of the scene space (Figure 4). Here and further, the wire cube denotes a 3-dimensional space.

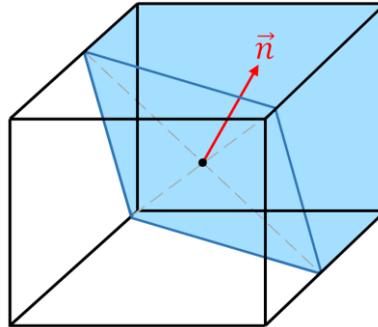


Figure 4: The plane with the normal \vec{n} separates the half-space

2.2. Space inside or outside sphere

The sphere is defined by the center $C(x_c, y_c, z_c)$, the radius R and the Boolean sign INSIDE: *true* means that the inner region is taken, if *false*, then the complement to the space is taken, see Figure 5.

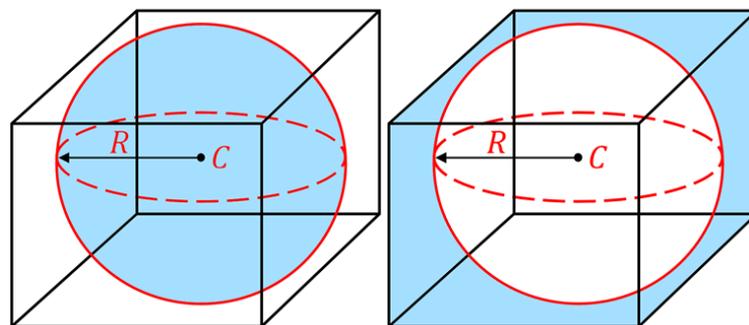


Figure 5: Part of the space inside or outside the sphere. C is the center of the sphere, R is the radius of the sphere

2.3. Space inside or outside cylinder

The infinite cylinder can be set using the coordinates of a point on the axis of rotation, the direction of the axis of rotation, and the radius of the circle. The point is needed to link the vector of the axis of rotation to a certain place in space. Similarly, there are two possible options: a part of the space inside or outside the cylinder is taken (see Figure 6).

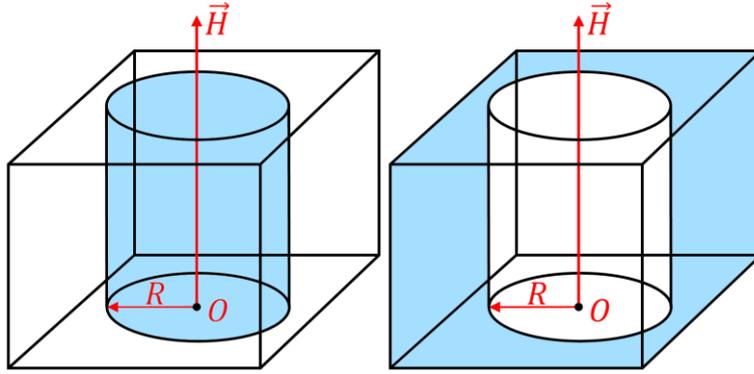


Figure 6: Part of the space inside or outside the cylinder. O is arbitrary point of the cylinder axis, H is the axis vector, R is the radius of the circle

2.4. Space inside or outside cone

The infinite cone is a second-order surface generated by the motion of a straight line (generatrix) passing through a fixed point. In this case, the cone can be set by the coordinate of the vertex of the cone, the vector of the axis of the cone and the value of the angle at the vertex of the cone, i.e., the doubled angle between the axis and the generatrix. As in the case of a sphere, in order to define the primitive completely, it is specified which part of the space is taken: inside or outside the cone (see Figure 7).

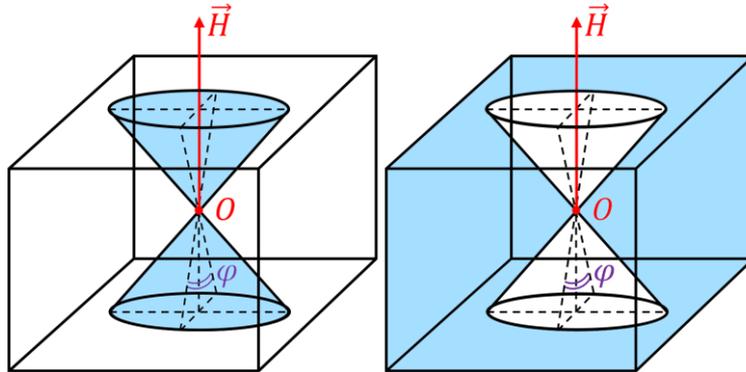


Figure 7: Part of the space inside or outside the cone. O is the vertex of the cone, φ is the angle at the vertex, H is the axis vector

2.5. An example of constructing a lens from primitives

Let us consider the application of the set-theoretic intersection operation on a set of primitives by constructing a certain lens. Let us take for example a biconvex lens. We will set in the scene space the coordinates of the lens center point C , the unit vector of the main optical axis of the lens **axis** and the width of the lens $width$, see Figure 8. A biconvex lens has two spherical surfaces, let us denote their radii R_{front} and R_{back} . Obviously, the following conditions must be met:

$$width > 0 \ \& \ R_{front} \geq \frac{width}{2} \ \& \ R_{back} \geq \frac{width}{2}.$$

Let us find the centers of the spheres. If the radius of the sphere R_{front} is the radius of the curvature of the surface in the positive direction of the **axis** vector, then the center of this sphere is at the point

$$O_1 = C - \mathbf{axis} \cdot \left(R_{front} - \frac{width}{2} \right).$$

Similarly, the center of a sphere with radius R_{back} is at the point

$$O_2 = C + \mathbf{axis} \cdot \left(R_{back} - \frac{width}{2} \right).$$

If the above conditions are met, such spheres will have a non-zero intersection of the width $width$ (see Figure 8). The intersection of the spheres gives a biconvex lens without a rim.

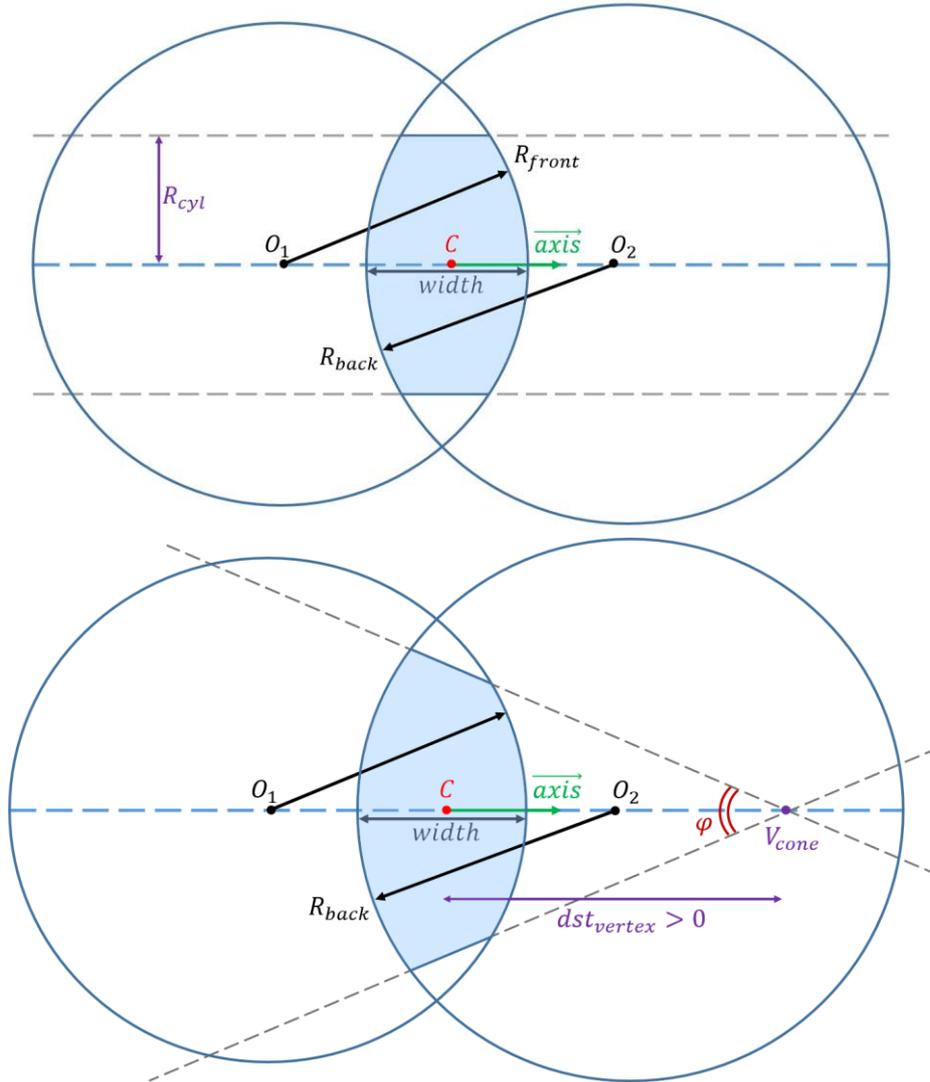


Figure 8: A biconvex lens is formed at the intersection of the spheres. At the top there is a lens with a cylindrical rim, at the bottom there is a lens with a conical rim

Let us add a cylindrical rim to the resulting lens, which is formed when adding a part of the space inside the cylinder to the set of primitives, i.e. the intersection of two spheres and the interior of the infinite cylinder. The cylinder has a rotation axis $\mathbf{H} = \mathbf{axis}$ and a starting point $O = C$. Let h_{max} be the maximum height of the lens, then the radius of the cylinder $R_{cyl} \leq \frac{h_{max}}{2}$. If this condition is met, a lens similar to the upper one shown in Figure 8 will be obtained.

To set a conical rim, you must specify the vertex V_{cone} and the angle φ of the cone, which will cut off the excess part of the lens. The angle φ should be such that the cone has an intersection with the lens. The vertex of the cone is given by the distance dst_{vertex} from the center of the lens along the main optical axis, and then the vertex of the cone can be found by the formula

$$V_{cone} = C + \mathbf{axis} \cdot dst_{vertex}.$$

The condition $|dst_{vertex}| \geq \frac{width}{2}$ must be met, and dst_{vertex} can be either positive or negative (see Figure 8).

Important note: the axis of the rim coincides with the axis of the lens.

The library provides specialized constructors and allows the user to avoid difficulties and errors when designing lenses using set-theoretic intersection. Here is an example of using the lens constructor that eliminates annoying geometric calculations shown above:

```
using namespace Lens;
double cone_angle = 30;
double cone_vertex_distance = 100;
/* Create biconvex lens */
BiconvexLensBuilder builder;
LENS_DATA lens = builder.set_axis(CLENSVector{1, 0, 0})
    .set_center(CLENSVector{0, 0, 0})
    .set_front_radius(200)
    .set_back_radius(400)
    .set_width(50)
    .set_inner_material(GLASS_INDEX)
    .set_outer_material(AIR_INDEX)
    // Specify conical opaque rim
    .set_rim(coneRim(cone_vertex_distance, cone_angle, BORDER))
    .build();
```

Here we show how one can create a biconvex lens with a conical rim using the appropriate constructor. The cylindrical rim is set similarly using the following line

```
.set_rim(cylinderRim(cylinder_radius, BORDER)).
```

In addition to geometric parameters, refraction indices or materials of the lens interior and the external environment are also set.

2.6. Implementation

The SphL software package is designed as a set of libraries in the C++ programming language. For all six types of spherical lenses, constructor classes are implemented. Examples of possible lenses are shown in Figure 9. When designing the lens, the correctness of the parameters set by the user is checked. If it is impossible to construct the correct lens based on the parameters, the user receives an error in the form of an exception.

Let us represent a linear polarized zero-thickness tracing ray with attributes based on the work [7] in the form

$$R = \{P_0, \mathbf{d}, \mathbf{X}, \mathbf{Y}, I, \lambda, L_{id}, Op, \Sigma\}, \quad (1)$$

where: $\{P_0, \mathbf{d}\}$ is the mathematical ray, P_0 is the origin of the ray, \mathbf{d} is the direction, $\{\mathbf{X}, \mathbf{Y}, \mathbf{d}\}$ is the associated right-hand coordinate system. In other words, the ray is represented as

$$P(t) = P_0 + t \times \mathbf{d}.$$

The oscillations of the electric vector of electromagnetic wave occur along the \mathbf{X} axis, I is the intensity, λ is the wavelength of the light, L_{id} is the identifier of the point light source that generated the ray. Two rays are coherent if their source IDs are non-zero and match. Op is the optical path of a geometric path traveled from the source, which is used to calculate the current phase of the electromagnetic wave [1]. Σ is the phase jump accumulated during reflections from a denser medium, see [2, 7] for details.

The generated rays (reflected and refracted) inherit some of the attributes, or they are recalculated during the contact of the incident ray with the scene surface. Note that the Fresnel's equations apply.

A corresponding data structure was written for each type of primitive. The primitive is defined, firstly, by the parameters that determine its geometry and location in space, and secondly, by the refraction indices of two media: belonging (interior) and not belonging to the primitive (medium). Basic operations on primitives are the following:

- Use a given point to determine whether it belongs to a primitive,
- Using a given geometric ray, if possible, calculate: a) the value of the parameter t for the point of its intersection with the primitive boundary, b) the coordinates of the intersection point, c) the normal vector at this point. There may be several such points.

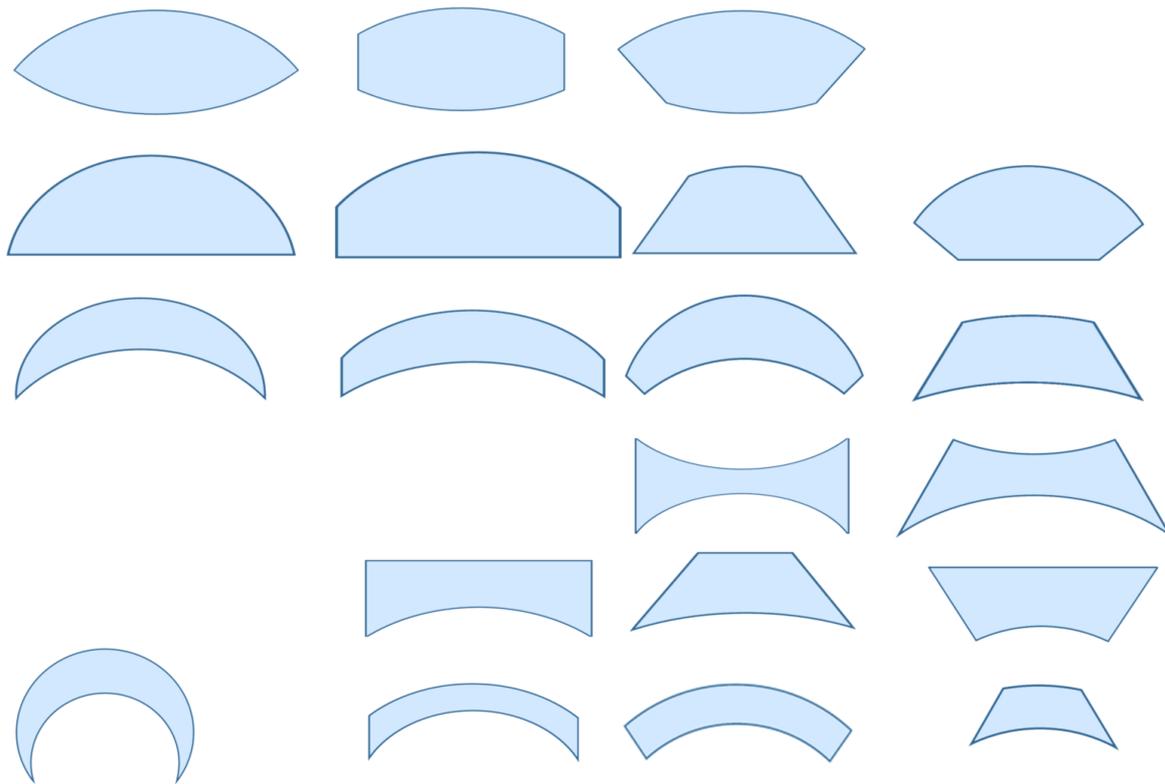


Figure 9: All types of spherical lenses supported by the SphL software package

Let us consider a simple case presented in Figure 10. The lens is represented as a list of primitives. When tracing, it is necessary to determine which primitive the ray falls on, the coordinates of the hit point, the normal at this point, and the refraction indices of the media. Knowing these values, it is possible to calculate the reflected and refracted rays using Fresnel's formulas. To support the intersection operation, it is necessary to check for each point that it belongs to all primitives.

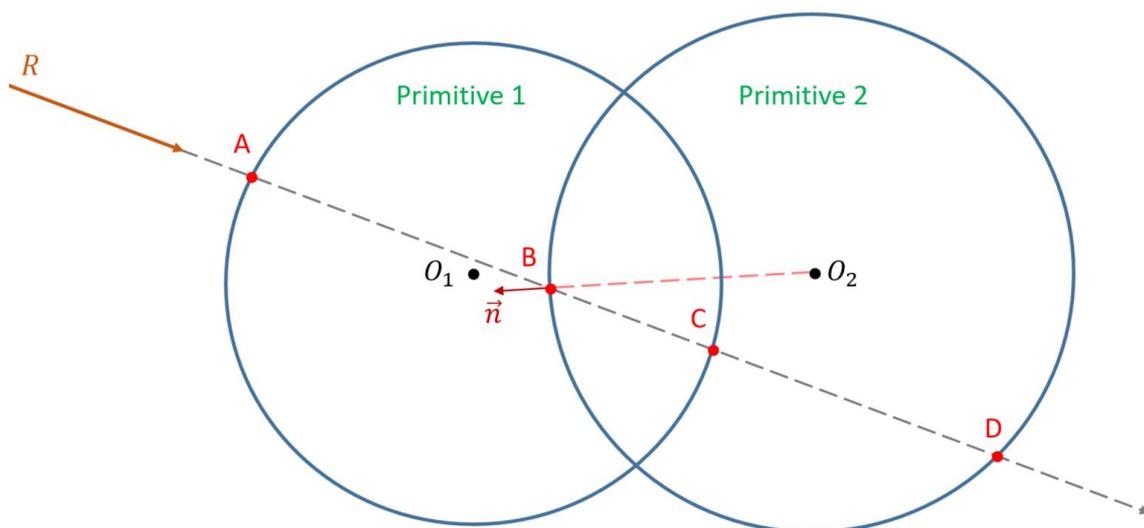


Figure 10: Intersection points of the ray R with primitives

Based on the example, we will consider how the set-theoretic intersection operation is implemented. It is based on the well-known *even-odd* principle when calculating intersections of a ray with the boundaries of primitives. In Figure 10, the ray R falls on the surface of a lens constructed from two primitives: parts of space inside spheres with centers O_1 , O_2 . The ray R crosses the boundary of the

Primitive $Prim1$ at points A and C, the boundary of the Primitive $Prim2$ at points B and D. Thus, we get a set of values of t :

$$t(A, Prim1) - t(C, Prim1) - t(B, Prim2) - t(D, Prim2),$$

sort it in ascending order t :

$$t(A, Prim1) - t(B, Prim2) - t(C, Prim1) - t(D, Prim2),$$

analyze the sequence and find the segment BC belonging to both primitives, i.e. their intersection.

We have considered the simplest case of the mutual arrangement of the ray and the boundaries of primitives. A well-known problem is when the ray hits the intersection point of the primitives themselves. In order to have an accurate idea of how to perform sorting, a different implementation was made for each type of lens.

When creating a lens, the user sets handlers for the three above-mentioned events he is interested in. A user can use the *standard (or default) handlers* already included in the library, or implement them himself. Each of the three possible events has its own handler class interface. Let us look at the work of standard handlers:

- The handler for the next calculated output ray allows the user to get a list of calculated output rays after the tracing is completed.
- The handler for the case of total internal reflection allows the user to check whether a total internal reflection has occurred after the tracing is completed and, if it has occurred, to obtain two calculated linear polarized rays (see [2, 7]).
- The handler for the case of a ray hitting the rim allows the user, after the tracing is completed, to check whether a certain ray has hit the rim and, if it has happened, to get this ray and the coordinates of the hit point.

The handlers are installed after the lens is set. Different lenses may have different handlers. After setting the handlers, the user can trace the ray through the lens. To do this, the user needs to set the input ray, the tracing depth, the energy level of the ray ε , at which further tracing stops, the lens through which the tracing will be performed, and finally calls the tracing function.

After the trace is executed, the user can get the result and apply the appropriate handler. The value returned by the tracing function is the code of the last event that occurred:

- FRESNEL_SUCCESS means that the required number of rays has been successfully calculated,
- FRESNEL_RIM means a ray was found that hit the rim,
- FRESNEL_TIR means the case of total internal reflection,
- FRESNEL_DATA_ERROR means that error in user data is detected,
- FRESNEL_NO_ENERGY means that the incident ray has an energy less than ε ,
- FRESNEL_NO_INTERSECTION means that the ray does not intersect with the lens.

2.7. Additional library objects

Spherical lenses represent only the starting set of the library. Already in the course of our experiments, we were faced with the need for such basic geometric shapes as a cube for the instrument glass, a cylinder for the instrument glass, a wedge for experiments with interference. We illustrate these additional objects in Figure 11.

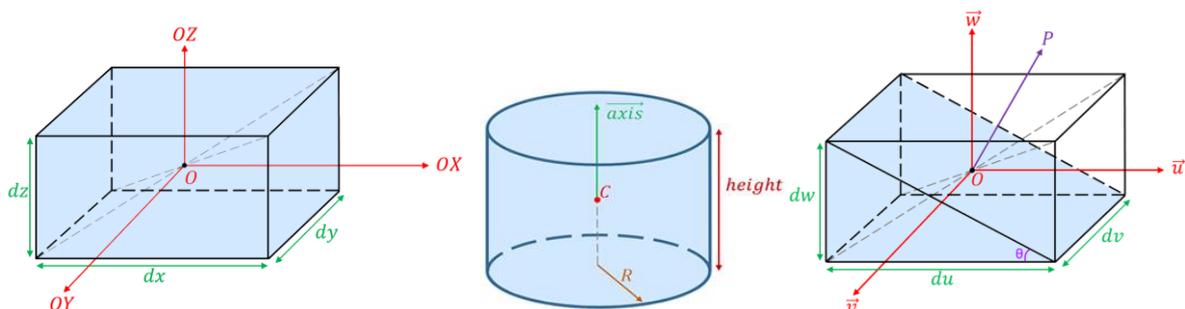


Figure 11: Additional forms for the interference experiment

Advanced users can create their own additional forms in a similar way. They should keep in mind possible collisions at the intersection points of two or more primitives and correctly handle in case of a ray hitting such points.

2.8. Function Fresnel: base level of library

Tracing is carried out by rays of linear polarized light. To calculate their interaction with optically isotropic transparent objects, the Fresnel function is developed, which, based on the application of the Fresnel's equations and the parameters of the ray incident on the boundary between two optically isotropic transparent media, calculates the characteristics of the generated reflected and refracted rays: direction, polarization, and others according to the representation (1). This function also determines the events of total internal reflection and others mentioned above.

3. Conclusions

In this paper, we described the current state of the development of SphL Library. Testing and debugging were carried out on simple scenes, for example, calculating interference patterns related to fringes of equal thickness [1, 3], including the calculation of Newton's rings in transmitted and reflected light.

We are planning to expand it step by step with other types of geometric shapes: cylindrical and aspherical lenses, other optically isotropic transparent objects necessary for particular experiments, etc.

Note that the library allows one to find out all the data about the progress of ray tracing in the lens (see Figure 3), and, therefore, the user can link the movement of the rays to the virtual scene, and prepare presentations and reports on the flow of the experiment.

4. Acknowledgements

This work was carried out under state contract with ICMMG SB RAS (0251-2021-0001).

5. References

- [1] M. Born, E. Wolf. Principles of optics: Electromagnetic theory of propagation, interference and diffraction of light, Cambridge University Press, 1980.
- [2] V. A. Debelov, K. G. Kushner, L. F. Vasilyeva. Lens for a computer model of a polarizing Microscope, *Mathematica Montisnigri*, 41 (2018) 151–165.
- [3] G. S. Landsberg, *Optics*, 6th. ed., Fizmatlit, Moscow, 2003. [in Russian]
- [4] B. A. Barsky, D. R. Horn, S. A. Klein, J. A. Pang, M. Yu. Camera Models and Optical Systems Used in Computer Graphics: Part I, Object-Based Techniques, in: V. Kumar, M. L. Gavrilova, C. J. K. Tan, P. L'Ecuyer (Eds.), *Proceedings of International Conference on Computational Science and Its Applications, ICCSA 2003, Montreal, Canada, May 18–21, 2003*, volume 2669 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin. Heidelberg, 2003, pp. 246–255. doi: 10.1007/3-540-44842-X_27.
- [5] G. I. Fedotov, R. S. Ilin, et al., *Laboratory optical devices. Textbook for optical specialties of Universities*, 2nd. ed., Mashinostroenie, Moscow, 1979. [In Russian]
- [6] Laboratory polarizing microscope of transmitted light, 2021. URL: <http://www.lomo.ru/production/grazhdanskogo-naznacheniya/mikroskopy/mikroskopy-tekhnicheskie/polam-1-213m/>. [In Russian]
- [7] V. A. Debelov, L. F. Vasilieva. Visualization of interference pictures of 3D scenes including optically isotropic transparent objects, *Scientific visualization*, 12.3 (2020) 119–136. doi:10.26583/sv.12.3.11.