

# 1 Ray Tracing in Poincaré's ball model of hyperbolic space

Boris Ajdin, Jelena Novičić, Radmila Stamenčić, Predrag Janičić

Faculty of Mathematics, University of Belgrade

**Summary.** Ray Tracing is one of the key algorithms in computer graphics. It is used for creating realistic 3D scene images, taking into consideration viewing parameters, geometrical description of objects, lighting, material parameters — anything that affects the final appearance of the scene. Hyperbolic space is a geometrical structure that often goes against our intuition. In order to create a tool for the hyperbolic space visualization, we have developed a hyperbolic version of the Ray Tracing algorithm which, to the best of our knowledge, did not exist before. The obtained images, adapted at certain stage to Euclidian plane for displaying purposes, are visualizations of hyperbolic objects. The developed algorithm, methods for solving problems during development as well as the final results will be shown in this text. This algorithm and the corresponding library (HMathLib) can be applied in teaching and popularization of mathematics, exploring hyperbolic geometry and in digital art.

## 1.1 Introduction

Before the computers powerful enough emerged the visualization possibilities were modest and thus our understanding of geometry, as well as many other related sciences, was limited. That fact is best seen in non-euclidian geometries, whose modern development is intensified with the assistance of computer visualization. The obtained results are often unexpected, intriguing and they differ from our intuition, which is euclidian in its nature. Even though many scientists are exploring non-euclidian geometry visualization, complete results were achieved only in two-dimensions (by exploring the non-euclidian planimetry). Visualization of non-euclidian three-dimensional space is still an open research field waiting for appropriate techniques.

In the late 1980s Charles Gunn and Dell Maxwell from the *Geometry Center* at the University of Minnesota created pioneer hyperbolic space visualization systems. Their work was followed by mathematical foundations for real-time visualization of projective hyperbolic space models ([4, 9]). The first wide known result was the *Not Knot* animation created at the *Geometry Center* which demonstrates hyperbolic space tessellation. Similar results were achieved by Jeffrey Weeks's program *Curved Spaces* ([8]), used for creating *The Shape Of Space* animation. ([2]) describes virtual reality system *ALICE* which allows tours inside hyperbolic space. Interesting results were accomplished by using the technique known as *relativistic Ray Tracing*, which uses Einstein's relativistic theory and the changing of the objects appearance

when the observer travels at speed close to the speed of light ([1]). However, these systems are mainly used for mathematical exploration of hyperbolic space without any intent to provide realistic images.

In this text we shall present the Ray Tracing visualization algorithm adapted to Poincaré's ball model of hyperbolic space. Therefore, from now on we shall name the modified algorithm *hyperbolic Ray Tracing*. Although there are many programs that use Ray Tracing for viewing hyperbolic space, past implementations were based on „outside view”, that is using Euclidian view on hyperbolic space model, with all calculations performed in Euclidian geometry. Our algorithm is, to the best of our knowledge, the first one to implement full hyperbolic Ray Tracing without any geometrical approximations related to the used model of hyperbolic space.

The basic motivation for creating the hyperbolic Ray Tracing algorithm was the desire to define a new tool for hyperbolic space exploration. In order to achieve that we have developed a programming library (*HMathLib*) that offers functionality necessary for creating Poincaré's ball model on the computer. The ultimate aim is to provide realistic view of objects and transformations in hyperbolic space for the purpose of their better understanding. The main condition set before HMathLib library is to perform all calculations in Poincaré's ball model, without any geometrical approximations. Therefore our library supports creation of Poincaré's ball model objects and interaction between the objects, followed by the hyperbolic Ray Tracing algorithm. The final goal was to enable scene description in Poincaré's model and rendering using hyperbolic Ray Tracing, which as a result produces a printable image. Two key problems can be identified: how to define the hyperbolic Ray Tracing algorithm and how to define the transfer between Poincaré's ball model and Euclidian space, at least locally (this transfer has to be performed because computer display devices are Euclidian). These topics will be discussed in more detail later in this text. We assume that the reader is familiar with basic concepts of hyperbolic geometry, Poincaré's ball model and the Ray Tracing algorithm, which will not be discussed. For a detailed introduction to the Euclidian Ray Tracing algorithm see ([3]).

## 1.2 Hyperbolic Ray Tracing

### 1.2.1 Basic Elements

The elements of the Ray Tracing algorithm that have to be modified according to the laws of hyperbolic geometry are:

1. algorithm description using points, lines and planes;
2. reflections and refractions of rays;
3. light attenuation dependency on distance function;

4. pixel mesh in the view-plane is equidistant rectangular.<sup>1</sup>

For some elements it is easy to define the necessary algorithm modification from Euclidian to hyperbolic space since there exist geometrical equivalents in different types of geometries, whilst for others this can be hard or even impossible. The used modifications are:

1. Instead of using points, lines and planes to describe the Ray Tracing algorithm we shall use *h*-points, *h*-lines and *h*-planes.
2. Since vectors cannot be introduced in hyperbolic geometry ray reflection and refraction have to be defined by using the conformance property of Poincaré’s ball model and the properties of sphere inversion.
3. Imagine a brief flash from a point light-source in intergalactic Euclidian space. The light forms a spherical *wave front*, which expands uniformly out into space. By the time it has traveled a distance *r*, the wave front surface area is  $4\pi r^2$ , while the amount of energy in the wave front remains constant. Therefore the energy per unit area drops as  $1/(4\pi r^2)$ . Ignoring the constants, this says that the energy per unit area drops as  $1/r^2$ . Energy per unit area is what we ultimately perceive as brightness. In hyperbolic space the principle is the same, but the formula for wave front surface area differs. If light travels a distance *r*, then the wave front area is  $4\pi \sinh^2 r$  ([5]). Thus the light energy drops as

$$\frac{1}{4\pi \sinh^2 r}.$$

Combining this formula with a distance function in Poincaré’s ball model produces light attenuation formula in hyperbolic space.

4. Since rectangles cannot be defined in hyperbolic geometry we have to use new approach for creating pixel mesh in the view-plane. Furthermore, we have to define a transfer from hyperbolic plane to Euclidian plane by using some form of projection. The next section gives more detailed description of the used pixel mesh types, projection methods and the achieved results.

### 1.2.2 Projection methods

Projection methods are strongly related to pixel mesh types. In all methods except the hyperbolic orisphere projection the view-point is fixed in the center of the absolute.

**Euclidian central projection** — the pixel mesh used is an equidistant rectangular mesh, that is an Euclidian mesh „inserted” in Poincaré’s

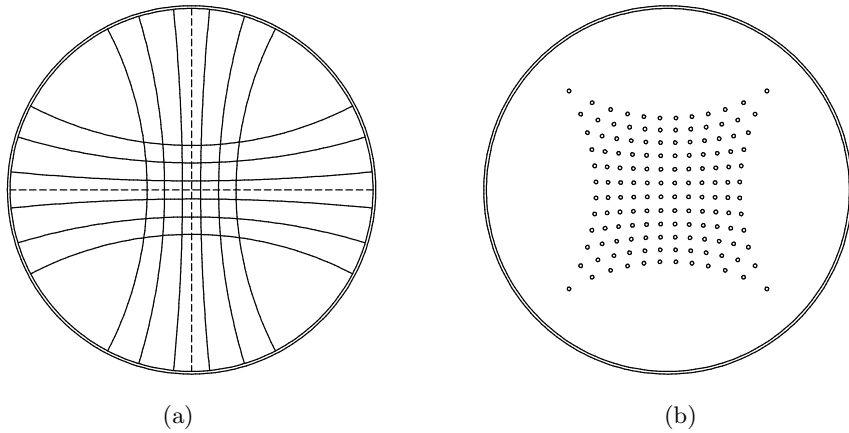
---

<sup>1</sup> The pixel mesh can be divided into identical rectangles with given pixels as vertices and there are no pixels inside these rectangles.

model. By positioning the view-point in the center of the absolute we in fact have central projection in Euclidian space.

Using this projection method all  $h$ -lines will appear as Euclidian lines. However, since we use  $h$ -lines as light rays the appearance of the scene will differ from the appearance of the same scene obtained by the Euclidian Ray Tracing algorithm.

**Hyperbolic central projection** — the pixel mesh is created on the  $h$ -plane that passes through the center of the absolute and then it is  $h$ -reflected to the view  $h$ -plane. Once the hyperbolic Ray Tracing is complete the resulting pixel mesh is bijectively mapped on equidistant rectangular mesh displayable on the computer. Figure 1.1 illustrates the creation of this pixel mesh.



**Fig. 1.1.** Creation of the curved pixel mesh: (a)  $h$ -lines orthogonal to  $x$  and  $y$  axis; (b) the resulting pixel mesh that is  $h$ -reflected to the view  $h$ -plane.

All  $h$ -lines used for creating curved pixel mesh belong to  $h$ -planes that pass through the center of the absolute (they belong to Euclidian planes). Therefore, even though these  $h$ -lines are parts of Euclidian circles in Poincaré's ball model, viewed from the center of the absolute they will appear to be Euclidian. Since these  $h$ -lines are bijectively mapped on Euclidian lines it follows that the  $h$ -lines will appear as Euclidian lines in the resulting image.

**Hyperbolic  $h$ -reflection projection** — this projection method uses curved pixel mesh described in the previous paragraph. After Ray Tracing is completed the resulting pixels are transferred using  $h$ -reflection to Euclidian plane and interpolated to equidistant rectangular pixel mesh. Since the obtained projection represents Poincaré's disk model, the projections of  $h$ -lines in Poincaré's ball model will be  $h$ -lines in Poincaré's

disk model. This means that the projected  $h$ -lines will be part of Euclidian circles, and the resulting image will appear to be „curved”.

**Hyperbolic orisphere projection** — since there exists a model of Euclidian planimetry on the orisphere, this projection method enables transfer from hyperbolic to Euclidian space without the loss of information about the curvature of space. The pixel mesh used is the same as in the previous two projection methods, and the view-point is located on the absolute. The projections of  $h$ -lines in Poincaré's ball model will be  $e$ -lines on the orisphere, thus all the projections will appear to be Euclidian lines.

In all projection methods used, as a result of using central projection in Ray Tracing algorithm projections of  $h$ -spheres whose hyperbolic center is not directly in front of the observer will be ellipses.

Sample images can be found in appendix A.

### 1.3 HMathLib library description

The HMathLib library was developed using C++ programming language in the Microsoft Visual Studio 2005 environment. In addition, this project includes *DirectDraw* graphics, MFC (Microsoft Foundation Classes) for graphical user interface and *Flex* and *Bison* for designing the format of the input files.

#### 1.3.1 Class hierarchy

HMathLib includes classes for creating and manipulating objects in Poincaré's ball model and classes needed for the Ray Tracing algorithm. Objects in Poincaré's ball model are created by using the following classes:

- HPoint** — defines  $h$ -points and  $h$ -point transformations  $h$ -reflection,  $h$ -translation and  $h$ -rotation.
- HLine** — defines  $h$ -lines and appropriate transformations. Each HLine object is defined with two distinct  $h$ -points. Transformations of HLine objects are based on  $h$ -point transformations. Methods for determining  $h$ -planes for  $h$ -translation and  $h$ -rotation are also defined in this class.
- HPlane** — defines  $h$ -planes and  $h$ -plane transformations. Each HPlane object is defined with three non- $h$ -colinear  $h$ -points. Like in the HLine class, transformations of the HPlane class are based on  $h$ -point transformations. Methods for intersection with HLine object and HLine reflection and refraction are also defined.
- HPolygon** — defines simple convex  $h$ -polygons. All  $h$ -polygons are defined with an array of its vertices represented by HPoint objects. Both polygonal surfaces and wire-frame polygons are supported. This class inherits methods for  $h$ -line intersection, reflection and refraction from HPlane class and tests whether the intersecting  $h$ -point is inside the  $h$ -polygon.

**HSphere** — defines  $h$ -sphere objects. Each **HSphere** object is uniquely defined by its center and hyperbolic radius, or equivalently by its center and one  $h$ -point on the  $h$ -sphere. Intersection with  $h$ -line and  $h$ -line reflection and refraction methods are also defined.

**HObject** — defines an object in Poincaré's ball model, which can be either a polyhedron or a sphere. A polyhedron is defined as an array of  $h$ -polygons representing its sides.

Classes needed for the Ray Tracing algorithm are:

**Color** — defines color. The model used is 24-bit RGB, where each color component uses 8 bits. Each component is tested for overflow and underflow.

**RCoeffs** — defines material reflection coefficients. Ambient, diffuse and specular reflection components are defined for each color component separately, as well as transparency, reflectiveness, shininess and light refraction coefficients of the material.

**HLight** — defines light in the scene. Two types of lights are supported: point-source and light in infinity (where light source is placed on the absolute). Also, we implemented methods for changing the light position.

**HScene** — defines 3D scene in Poincaré's ball model and the hyperbolic Ray Tracing algorithm. **HScene** class keeps track of all the objects and lights in the scene, creates the pixel mesh and performs the Ray Tracing routine.

### 1.3.2 Principal methods

Methods for objects transformation defined in all the classes are:

- **hReflect** —  $h$ -reflection with respect to the given  $h$ -plane;
- **hTranslate** —  $h$ -translation for the given  $h$ -segment;
- **hRotate** —  $h$ -rotation around the given  $h$ -line for the given angle.

The following methods are required for the Ray Tracing algorithm:

- **HPoint::operator-** — implements hyperbolic distance function;
- **HPlane::hLineIntersection** — intersection of the given  $h$ -line with **HPlane** object;
- **HPlane::hRayReflection** —  $h$ -line reflection of the **HPlane** object;
- **HPlane::hRayRefraction** —  $h$ -line refraction through the **HPlane** object;
- **HSphere::hLineIntersection** — intersection of the given  $h$ -line with **HSphere** object;
- **HSphere::hRayReflection** —  $h$ -line reflection of the  $h$ -sphere;
- **HSphere::hRayRefraction** —  $h$ -line refraction through the  $h$ -sphere;
- **HPolygon::isInside** — testing whether the intersection  $h$ -point is inside the  $h$ -polygon;
- **HObject::getCosineDiff** — cosine value for the diffuse light reflection component;

- `HObject::getCosineSpec` — cosine value for the specular light reflection component;
- `HScene::chooseNet` — creates the pixel mesh for the selected projection type;
- `HScene::rayTracer` — implements hyperbolic Ray Tracing algorithm.

Documentation for the `HMathLib` library was created using *DOxygen* documentation generator in HTML format. The library itself, along with corresponding program modules and documentation will be publicly available.

## 1.4 Achieved results and future work

The main goal of the project was achieved — we have created a new tool for visualizing Poincaré's ball model using Ray Tracing, with precise geometrical interpretation of the resulting images. The complete functionality of the Poincaré's ball model is implemented, including the creation of objects inside the model, their transformations and interactions. On top of that we have implemented Ray Tracing algorithm and integrated the `HMathLib` library, together with the module for loading input files, into Windows application.

The `HMathLib` library was designed and developed as a modular library, enabling easy modification and functionality upgrading. Unfortunately, our hyperbolic Ray Tracing implementation is not efficient in practice. The principal causes are the complex structure of the Poincaré's ball model and of the Ray Tracing algorithm. Also, optimization techniques for the Ray Tracing algorithm are hard to implement in hyperbolic space. Therefore this implementation should be considered as a starting point for the future improved implementations of Ray Tracing in hyperbolic space.

Although the current stage of the Ray Tracing algorithm development provides the possibility to create realistic images of hyperbolic space, there are still some elements of Ray Tracing that are yet to be implemented:

- adaptive anti-aliasing (eliminating rough polygon edges). Current version supports fixed level anti-aliasing;
- support for textures;
- support for animations;
- library of predefined objects and materials.

Future work includes implementation of the Ray Tracing algorithm in some projective model of hyperbolic geometry (for example, in Klein's ball model). Such version of the Ray Tracing algorithm could be comparable in terms of speed with the Euclidian Ray Tracing implementations. Also, parallel implementation of the Ray Tracing algorithm can be used to increase rendering efficiency ([7]). In the recent years many researchers are working on the problem of real-time Ray Tracing using stream processors like modern

graphic cards ([6]). The same idea could be applied to the hyperbolic Ray Tracing algorithm.

We believe that the HMathLib library, together with the corresponding Windows application and program documentation, can be used for popularization of mathematics and geometric visualization problems. In addition, the library can be used as an education tool for helping students understand hyperbolic geometry better.

**Acknowledgements:** We are honored to thank the members of the “Group for geometry, education and visualizations with applications” from the Faculty of Mathematics for all their support during development. Also, we thank professor Srđan Vukmirović for advice related to Poincaré’s ball model and hyperbolic distance functions and Jeffrey Weeks for help regarding the behavior of light in hyperbolic space.

Professor Konrad Polthier directed us towards the possibility of hyperbolic orisphere projection during the conference *Workshop on Multimedia Technology for Mathematics and Computer Science Education* held in Belgrade in November 2005 and provided us with invaluable support for our presentation at DAAD Spring School 2006 in Berlin. We are thankful for all the assistance and are looking forward to future collaboration.

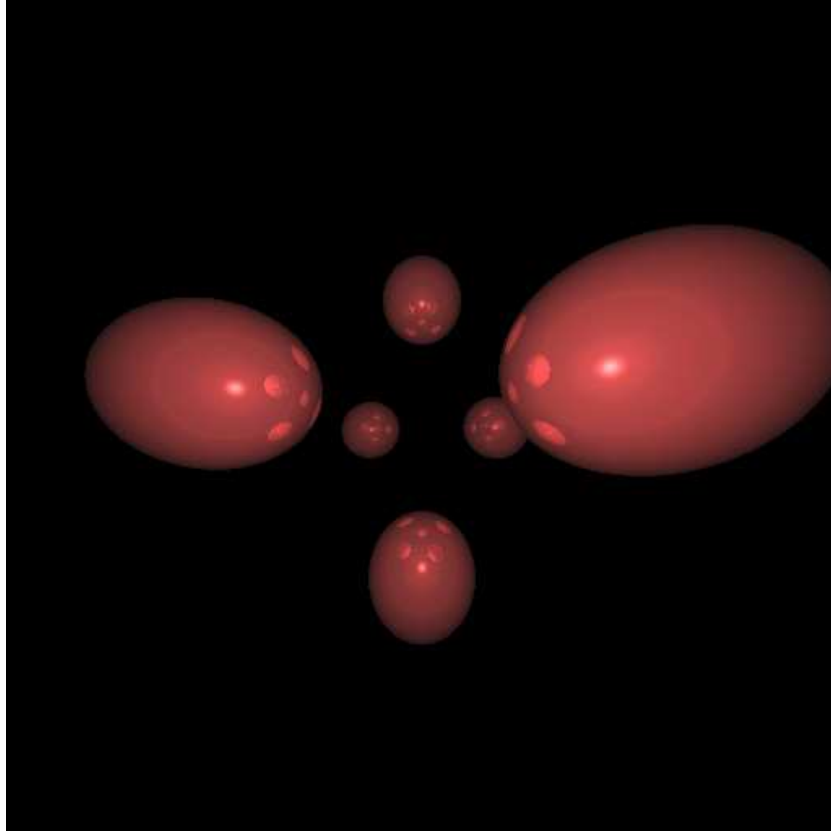
Finally, we thank professor Aleksandar B. Samardžić for answering all our questions related to various aspects of the development. With many advice, remarks and critical comments he motivated us and helped us make a better implementation of the hyperbolic Ray Tracing algorithm.

## References

1. Les Kitchen, Andrew Howard, Sandy Dance. Relativistic ray-tracing: simulating the visual appearance of rapidly moving objects.
2. Henry J. Kaczmarski, Benjamin J. Schaeffer, John M. Sullivan, George K. Francis, Camille M.A. Goudeseune. Alice on the eightfold way: Exploring curved spaces in an enclosed virtual reality theater.
3. Steven K. Feiner, John F. Hughes, James D. Foley, Andries van Dam. *Computer Graphics: Principles And Practice*. Addison-Wesley Publishing Company, 1990.
4. Charles Gunn, Mark Phillips. Visualizing hyperbolic geometry: Unusual uses of 4x4 matrices. *Eurographics*, 1992.
5. Tamara Munzner. H3: Laying out large directed graphs in 3d hyperbolic space. *IEEE Symp. Information Visualization*, 1997.
6. Timothy John Purcell. *Ray Tracing on a Stream Processor*. PhD thesis, Stanford University, 2004.
7. Aleksandar B. Samardžić. Paralelna implementacija ray tracing algoritma. Master’s thesis, Matematički fakultet, Beograd, 1998.
8. Jeffrey Weeks. Curved spaces software.
9. Jeffrey Weeks. Real-time rendering in curved spaces. *IEEE Computer Graphics and Applications*, 22(6), 2002.

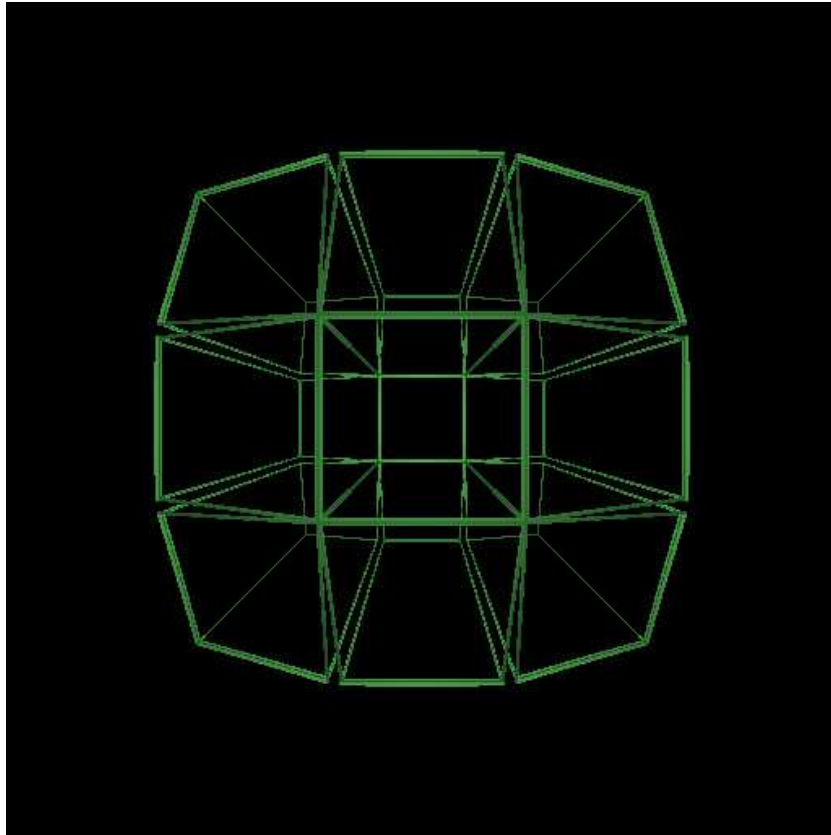
## A Examples

### A.1 Example 1



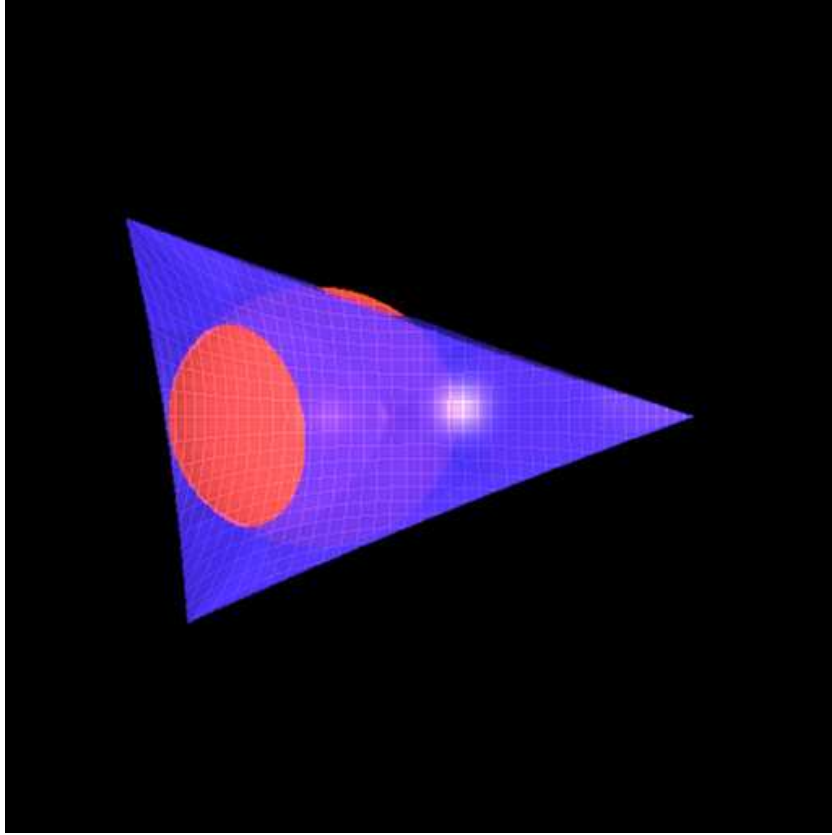
**Fig. 1.2.** Six identical  $h$ -spheres placed with their hyperbolic centers at the vertices of regular octahedron. Euclidian central projection, two light sources, recursion level two.

A.2 Example 2



**Fig. 1.3.** An attempt of space tessellation with identical wire-frame cubes. Hyperbolic central projection, one light source, recursion level zero.

A.3 Example 3



**Fig. 1.4.**  $H$ -sphere inside transparent octahedron. Hyperbolic  $h$ -reflection projection, one light source, recursion level two.