Computers & Graphics 93 (2020) 61–70

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Special Section on SIBGRAPI 2020

Global illumination of non-Euclidean spaces

Tiago Novello[a,*], Vincius da Silva[b], Luiz Velho[a]

[a] IMPA - VISGRAF Laboratory, Rio de Janeiro, Brazil
[b] Pontifical Catholic University of Rio de Janeiro - PUC-Rio, Department of Informatics, Rio de Janeiro, Brazil

ARTICLE INFO

Article history:
Received 24 May 2020
Revised 26 September 2020
Accepted 29 September 2020
Available online 9 October 2020

Keywords:
Global illumination
Path tracing
Non-Euclidean geometries

ABSTRACT

This paper presents a novel path tracer algorithm for immersive visualization of Riemannian manifolds. To do this, we introduce Riemannian illumination, a generalization of classical Computer Graphics illumination models. In this context, global light transport is expressed by extending the rendering equation to Riemannian manifolds. Using Monte Carlo integration to solve this equation results in the novel path tracer for Non-Euclidean spaces. We discuss its basic principles, as well as the general CPU algorithm. Additionally, we discuss in detail how to implement a GPU version, using the RTX pipeline. Finally, we apply the algorithm to render "photorealistic" inside views of the flat torus, Poincaré sphere, and the hyperbolic mirrored dodecahedron. These are examples of Euclidean, spherical, and hyperbolic spaces: the Thurston classical geometries.

1. Introduction

Global illumination is a collection of algorithms in computer graphics that are employed to mimic realistic lighting in 3D scenes. The complexity of these algorithms depends on the *indirect illumination*, which does not directly come from the light sources (*direct illumination*), but from reflections on other surfaces at the scene. Computing *reflections*, *refractions*, and *shadows* are important tasks for computing the global illumination of a given scene.

In *photorealistic rendering*, a set of techniques are used to create an image from a 3D scene that approximates a photograph [1], thus enabling the visualization of the global illumination of the underlying 3D scene. *Path tracing* is an algorithm contained in this set. Informally, it consists of assigning a color for each point (*eye*) and direction (*pixel*) by launching paths of rays through the scene based on reflections, refractions, and shadows. To write a path tracer it is common to simulate *cameras*, *ray–surface intersections*, *light sources*, *visibility*, *surface material*, *indirect light transport*, and *ray propagation*.

The above concepts used in illumination of Euclidean spaces can be extended to Non-Euclidean spaces. These spaces were developed negating Euclid's fifth postulate, so they do not exist in the real world. No real camera can take a picture in such spaces, which makes exploring photorealistic rendering of them very challenging. Attempts to visualize Non-Euclidean geometries produced interesting images with no counterpart in the real world; much of this is due to the nontrivial topology/geometry of such spaces. As we will discuss in Section 2, some previous works on ray tracing proposed to visualize these landscapes, but only simple scenes composed with Lambertian surfaces have been used; no indirect light transport considered (Fig. 1).

We take the challenge of computing the global illumination in Non-Euclidean spaces using a path tracer algorithm. In such spaces, the concept of photorealistic rendering is not well defined because there is no real camera there. However, the possibility of defining a *global illumination equation* in Non-Euclidean geometry opens the door for a path tracing generalization. Then, our challenge is to sample ray paths in Non-Euclidean spaces, where the path segments are nonlinear and the metric is more general than in rendering techniques developed in Euclidean spaces.

The **contributions** of our work are:

- Introduction of the Riemannian global illumination concept in Riemannian manifolds through a generalization of the inner product term of the rendering equation by the underlying Riemannian metric;
- A path tracer algorithm to approximate the solution of the Riemannian global illumination in spaces modeled by Thurston geometries;
- A pseudo-code and implementation of the algorithm on GPU using RTX. The implementation considered only spaces modeled by the Thurston classical geometries;
- We apply the Non-Euclidean path tracer to render "Photorealistic" inside views of the 3D flat torus, Poincaré sphere, and the hyperbolic mirrored dodecahedron. These are examples of

* Corresponding author.
E-mail address: tiago.novello@impa.br (T. Novello).

https://doi.org/10.1016/j.cag.2020.09.014
0097-8493/

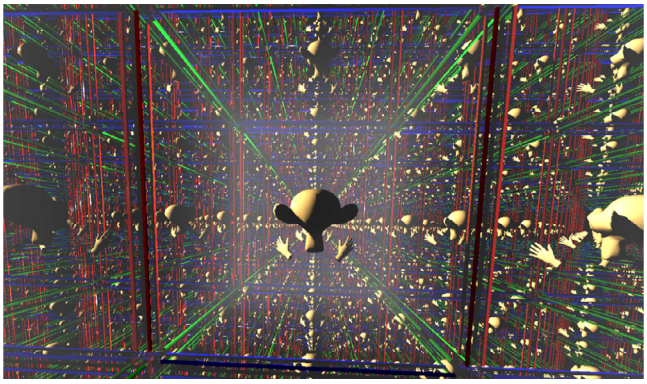

Fig. 1. Inside view of the 3D flat torus using ray tracing [2].

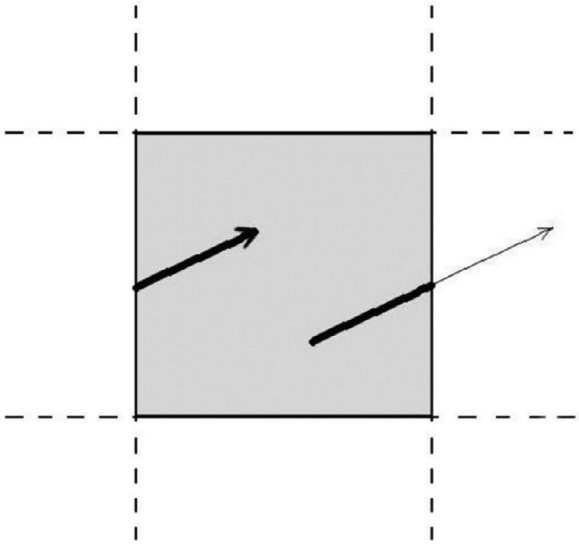

Fig. 2. 2D torus. Opposite edges of the square (fundamental domain) are identified together. Because of that, a geodesic (ray) intersecting an edge of the square returns through the opposite edge. The resulting copies of the square tessellate $\mathbb{R}^2$.

the Euclidean, spherical, and hyperbolic spaces: the Thurston classical geometries.

To visualize these examples of Non-Euclidean spaces, it is important to understand *geodesics* and *fundamental domains*. Geodesics are curves locally minimizing lengths, they are a consequence of the space geometry. The fundamental domain is a "polyhedron" with a special group of transformations associated to its faces. This structure describes the space topology. Fig. 2 gives a schematic example of the 2D torus to illustrate these concepts in dimension 2. Geodesics and fundamental domains are described in Section 3 for the Euclidean, spherical, and hyperbolic geometries.

The fundamental domain of the Euclidean and hyperbolic spaces that we are visualizing in this paper can be modeled as flat polyhedra because we are using Klein's model for hyperbolic geometry. In such a model, lines are straight and planes are flat. This is very helpful for tracing rays using the RTX platform, which supports straight rays only. However, the transformations must be hyperbolic. We use the hyperbolic reflection and an appropriate scale of the fundamental domain to implement the hyperbolic mirrored dodecahedron.

Conversely, spherical geometry cannot be represented by a flat model. The dodecahedron, which is the fundamental domain of the Poincaré sphere, must be spherical. This is a problem for RTX, which does not support curved rays per se. We propose to solve this problem using a parameterization of the dodecahedron. Then, we cast rays in the flat domain first, using RTX. If no scene intersection is found, their positions/directions are updated using the spherical transformations associated with the dodecahedron faces. Therefore, a ray interacts with the fundamental domain using its spherical form until the last interaction. Then, RTX detects the intersection with the scene using a straight ray. This is a good approximation because the fundamental domain of the Poincaré sphere is very small, as we will see in Section 6.2.

1.1. Structure of the paper

The paper is structured as follows: Section 2 reviews previous and related works; Section 3 introduces concepts from Non-Euclidean geometry; Section 4 presents the possibilities of photorealistic immersive visualization of 3-manifolds and introduces Riemannian illumination; Section 5 introduces the path tracing algorithm to compute the illumination of a given space; Section 6 presents photorealistic inside views of the flat torus, the Poincaré sphere, and the hyperbolic space using the path tracing; finally, Section 7 provides concluding remarks.

2. Related works

Ray tracing in curved spaces is an ancient topic in computer graphics. Bar [3] used a nonlinear approach to ray trace smooth surfaces efficiently. The surfaces are considered to be bent flat planes, which are generated by a deformation of the space. Such deformation corresponds to a coordinate system, where the intersection problem considers the surface being flat, and the rays being curved.

The main effort for visualization of Non-Euclidean spaces took place at the Geometry Center from 1994 to 1998. This initiative, under the leadership of William Thurston, resulted in a scientific program to study and disseminate modern geometry using interactive visualization. A platform called *Geomview* [4] was developed based on OpenGL for interactive viewing in Euclidean, spherical and hyperbolic spaces. Weeks [5] proposed real-time visualization of these classical geometries, and in [6], he extended it to the product geometries.

Researchers at the Geometry Center used Virtual Reality for providing insights of curved spaces. They created simple VR installations to allow the user not only to have a glimpse at the visual landscape inside a 3-manifold, but also to experience the sensation of being immersed in such an environment. Two of their projects are Mathenautics [7] and Alice [8]. Recently, Hart et al. [9] presented a more sophisticated immersive exploration of such curved spaces using VR, and Weeks [10,11] improved it by presenting a framework that allows game development.

The early work on the visualization of Non-Euclidean spaces, reported above, was based on the traditional OpenGL rasterization pipeline [5]. Therefore, the rendering algorithms employed a scene-based architecture. There are two problems with such an approach. First, the scene must be replicated to "unroll" spaces with complicated topology. Second, to rasterize a scene it is necessary to project objects in the scene based on camera position. This is a hard problem in manifolds with nontrivial geometry, where classic perspective projection cannot be applied. The ray tracing allows us to overcome such difficulties operating intrinsically in the manifold geometry and topology.

The first work to propose an *Image-Based* architecture for visualization of Non-Euclidean spaces was from Berger et al. [12]. Their rendering algorithm exploited programmable compute shaders and CUDA to implement ray tracing on the GPU. This was restricted to Euclidean/hyperbolic geometries. Rays in such spaces are modeled by straight lines which allow the use of classical algorithms from computational geometry.

Recently, a framework [13–16], implemented on top of Nvidia RTX GPUs, was introduced for real-time immersive and interactive visualization of Non-Euclidean spaces: Euclidean, Hyperbolic, Spherical, Nil, Sol, and $\widetilde{SL_2(\mathbb{R})}$ geometries; the six more interesting Thurston geometries. They introduced a novel ray tracing model for Riemannian manifolds focusing primarily on the topology of the underlying manifolds rather than the geometry. In [17], they designed and visualized Riemannian metrics in the space $\mathbb{R}^3$. Other attempts to visualize and explore Non-Euclidean geometries are [18–20].

All the aforementioned approaches adopt only a local illumination model for shading. As a consequence, this prevents the use of global ray tracing effects such as reflection and refraction. This work introduces the Riemannian global illumination and presents a novel path tracing to synthesize "photorealistic" inside views of Non-Euclidean spaces. Unfortunately, the computational performance is not interactive yet, which prevents the use of these effects in VR.

Other related works can be found in *visualization of physics*. Stam and Languénou [21] presented a ray tracing algorithm to render mirages and other optical effects such as "Fata Morgana", caused by perturbations in the participating media, thus integrating this phenomenon in the basic equations from geometrical optics that govern the propagation of rays in a medium varying its index of refraction continuously. The curvature of the rays depends on the air's index of refraction. Nonlinear ray tracing [22] was also considered in visualizing relativistic effects, the geometric behavior of nonlinear dynamical systems, and the movement of charged particles in a force field (e.g., electron movement).

3. Non-Euclidean spaces

In dealing with path tracing algorithms we need the following requirements: (1) The space must be locally similar to a Euclidean space — a *manifold*. This allows us to model the camera and the scene objects; (2) For each point p we need *tangent* vectors pointing in all directions. The *inner product* between two tangent vectors is required to simulate light effects; (3) For a point p and a vector v tangent at p, we need a *ray*, and its intersections with the scene objects.

Geometric manifolds satisfies the above properties. Such objects are locally-geometrically similar to special spaces called *model geometries*. In dimension two, for example, there are exactly three models: Euclidean, hyperbolic, and spherical spaces. In dimension three, there are five more model geometries, however, in this work, we focus on the three classical spaces. We describe these topics in detail at the end of this section. Great texts on this subject are Thurston [23], Martelli [24], and Scott [25].

A *3-manifold* M is a topological space locally identical to the Euclidean space $\mathbb{E}^3$. Specifically, there is a neighborhood of every point in M mapped homeomorphically to $\mathbb{E}^3$. These maps are called *charts*, and provide *coordinate systems* for the point neighborhoods. The change of charts between two neighborhoods in M must be differentiable. Examples of 3-manifolds include the Euclidean, hyperbolic, and spherical spaces.

Inner product and *rays* are important concepts when working with path tracing, since light travels along with rays and spreads in the surfaces through the inner product. *Riemannian metric* and *geodesics* are extensions of these concepts to manifolds.

A *Riemannian metric* is a smooth map g assigning to each point p of a 3-manifold M an inner product $g_p(\cdot, \cdot)$ on the tangent space T_pM. Let $p \in M$, and u, v be tangent vectors of T_pM. Expressing u and v in terms of the basis associated to a chart $\mathbf{x}(x_1, x_2, x_3)$, that is, $u = \sum u_i \frac{\partial}{\partial x_i}(p)$ and $v = \sum v_i \frac{\partial}{\partial x_i}(p)$, we obtain the inner product $\langle u, v\rangle_p = \sum u_i v_j g_p(\frac{\partial}{\partial x_i}, \frac{\partial}{\partial x_j})$. The metric g on M is determined by the symmetric positive definite matrix $[g_{ij}] := [g_p(\frac{\partial}{\partial x_i}, \frac{\partial}{\partial x_j})]$. A *Riemannian manifold* is a pair (M, g), where M is a manifold and g is a Riemannian metric on M.

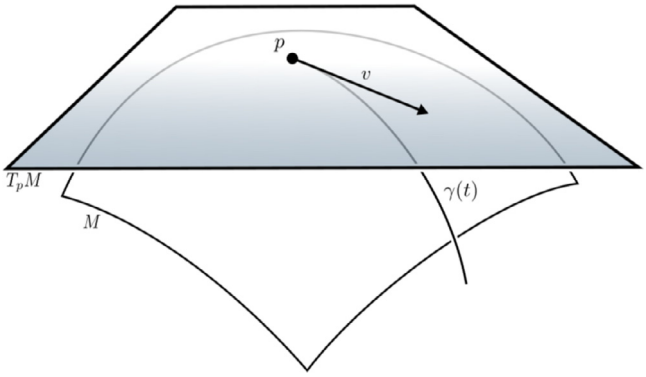

Fig. 3. A manifold M and its tangent space T_pM at the point p. The geodesic $\gamma(t)$ leaves p towards the direction v.

The metric g provides the length of *vectors* and consequently the distances between points in M. A *geodesic* (ray) $\gamma(t)$ leaving a point p in M and tangent to the direction v is a curve locally minimizing length (Fig. 3).

As this work focuses on Euclidean, spherical, and hyperbolic geometries, we only detail their geodesic formulas (definitions below). For the precise definition of geodesics in Riemannian manifolds see Carmo [26], and Novello et al. [16] for a presentation of this concept in the ray tracing context.

We remember an algebraic (very computational) way to construct manifolds from simple ones. Let M be a connected manifold and Γ be a discrete group of isometries acting on M, the *quotient* M/Γ is the set $\{\Gamma \cdot p |\ p \in M\}$ where $\Gamma \cdot p = \{g(p) |\ g \in \Gamma\}$ is the *orbit* of p. For example, the quotient of $\mathbb{E}^3$ by the group of translations gives rise to the *flat torus* $\mathbb{T}^3$. A ray r leaving a point $p \in \mathbb{T}^2$ in a direction v is described by considering the fractional part of the coordinates of $r(t) = p + t \cdot v$.

The quotient M/Γ of M by a discrete group Γ could be a non-manifold. In this case, M/Γ is called an *orbifold*. For an example, consider the group spanned by the reflections of a cube inside the Euclidean space.

Interesting cases arise when M is a *geometry* model. In this paper, we consider the classical examples of geometry models, the Euclidean, hyperbolic, and spherical. The space M/Γ inherits the *geometric structure* of M. For example, $\mathbb{T}^3$ has a geometric structure modeled by $\mathbb{E}^3$. The *fundamental domain* Δ plays an important role in the above construction being the region of M containing exactly one point for each orbit. The unit cube is the fundamental domain of $\mathbb{T}^3$.

In this paper, we will give examples of the global illumination of the classical Thurston geometries. For this reason, we will give the details of the Euclidean, spherical, and hyperbolic spaces. Section 6 will present examples and visualizations.

Definition 3.1 (Euclidean space)**.** The *Euclidean space* $\mathbb{E}^3$ is the three-dimensional vector space $\mathbb{R}^3$ endowed with the classical *inner product* $\langle u, v\rangle_{\mathbb{E}} = u_x v_x + u_y v_y + u_z v_z$ where u and v are vectors in $\mathbb{R}^3$. The *distance* between two points p and q is $d_{\mathbb{E}}(p, q) = \sqrt{\langle p - q, p - q\rangle_{\mathbb{E}}}$. The curve $\gamma(t) = p + tv$ describes a *ray* leaving a point p in a unit direction v. Analogously, for any $n > 0$ the Euclidean space $\mathbb{E}^n$ is constructed.

Definition 3.2 (Elliptic Space)**.** The *3-sphere* $\mathbb{S}^3$ is the set $\{p \in \mathbb{E}^4 |\ \langle p, p\rangle_{\mathbb{E}} = 1\}$ endowed with the metric $d_{\mathbb{S}}(p, q) = \cos^{-1}\langle p, q\rangle_{\mathbb{E}}$. A tangent vector v to a point in $\mathbb{S}^3$ satisfies $\langle p, v\rangle_{\mathbb{E}} = 0$, then the *tangent space* $T_p\mathbb{S}^3 = \{v \in \mathbb{E}^4 | \langle p, v\rangle_{\mathbb{E}} = 0\}$. The space $T_p\mathbb{S}^3$ inherits

the Euclidean inner product of $\mathbb{E}^4$. A *ray* in $\mathbb{S}^3$ passing through a point p in a unit direction v is the intersection between $\mathbb{S}^3$ and the plane spanned by v, p, and the origin. Such ray can be parameterized as $r(t) = \cos(t)p + \sin(t)v$.

Definition 3.3 (Hyperbolic space)**.** The *Lorentzian space* is $\mathbb{R}^4$ endowed with the *Lorentzian product* $\langle u, v\rangle_{\mathbb{H}} = u_x v_x + u_y v_y + u_z v_z - u_w v_w$, where u, v are vectors in $\mathbb{R}^4$. The *hyperbolic space* $\mathbb{H}^3$ is the hyperboloid $\{p \in \mathbb{R}^4 |\ \langle p, p\rangle_{\mathbb{H}} = -1\}$ endowed with the special metric $d_{\mathbb{H}}(p, q) = \cosh^{-1}(-\langle p, q\rangle_{\mathbb{H}})$.

A tangent vector v to a point p in $\mathbb{H}^3$ satisfies $\langle p, v\rangle_{\mathbb{H}} = 0$, then the *tangent space* $T_p\mathbb{H}^3 = \{v \in \mathbb{R}^4 |\ \langle p, v\rangle_{\mathbb{H}} = 0\}$. *Rays* in $\mathbb{H}^3$ are the intersections between $\mathbb{H}^3$ and the planes in $\mathbb{R}^4$ containing the origin. For instance, the ray leaving a point $p \in \mathbb{H}^3$ in a unit tangent direction v is the intersection between $\mathbb{H}^3$ and the plane spanned by the vectors v and p in $\mathbb{E}^4$. Such ray can be parameterized as $r(t) = \cosh(t)p + \sinh(t)v$.

$\mathbb{H}^3$ contains no straight line, thus its rays cannot be straight. However, we can model $\mathbb{H}^3$ in the unit ball of $\mathbb{R}^3$ — known as *Klein model* $\mathbb{K}^3$— such that the rays are straight. Specifically, each point $p \in \mathbb{H}^3$ is projected in $\{(x, y, z, w) \in \mathbb{R}^4 |\ w = 1\}$ by considering p/p_w. We use this model to visualize the hyperbolic mirrored dodecahedron in Section 6.3.

In this work, we use the Nvidia RTX platform to visualize spaces modeled by the above geometries, however, RTX only supports ray tracing using straight lines. This is well suitable to cast rays in Euclidean and hyperbolic geometries since we have Klein's model for the hyperbolic case. However, there is no flat model to represent the spherical geometry. To render inside views of spaces modeled with the spherical geometry, we use an alternative approach. This consists of casting a ray using RTX first. If no intersection is found, we update the ray position and direction using the precise spherical setting. Then we repeat the procedure. To set the scene and alternate between the spherical and Euclidean settings, we use a parameterization of the spherical fundamental domain.

4. Illumination of non-Euclidean spaces

This section focuses on extending the global illumination equations to a Riemannian manifold (M, g), the next section presents a path tracer algorithm for the class of geometric manifolds. We consider that light propagates along with rays in (M, g), and that it is constant in a vacuum. We use the *Riemannian ray tracing* model developed in [16] to define the global illumination of Riemannian manifolds.

Inside views of a scene in M can be rendered by tracing rays. Let p be a point (the eye) in M, and V_p be the set of direction within the observer's field of view (the view frustum). For each pixel in the image, we trace a ray towards the associated direction. Specifically, p is the origin of the tangent space T_pM, we attribute a RGB color to each direction $v \in V_p$. This color is computed using *Riemannian illumination* through the *direct* and *indirect* light contributions. The direct illumination considers the rays coming directly from light sources and the indirect gathers the light rays bounced by other surfaces. Fig. 4 gives a schematic view of this procedure in dimension 2.

Illumination is the process of simulating the light leaving a point in a given direction. Techniques developed in Euclidean spaces to perform such task are not suited for Riemannian manifolds, so we propose a more general definition.

Let $\mathcal{C}$ be the space of RGB radiant energy. The *Riemannian illumination* of a 3-manifold M, with a scene embedded on it, is a function $L : M \times \mathbb{S}^2 \to \mathcal{C}$, where for each point $p \in M$ and unit tangent direction $v \in T_pM$ we have a color $L(p, v)$ representing the light leaving this direction. Because we are considering the light

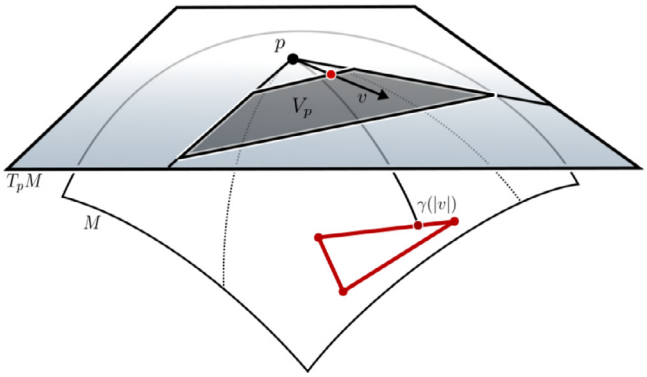

Fig. 4. Tracing rays in a Riemannian manifold M. Let p be the observer and V_p be the view frustum (gray region) in the tangent space T_pM. We launch a ray γ towards each vector $v \in V_p$. If γ hits a visible object (red triangle) in $\gamma(|v|)$ we define a RGB color for the corresponding point in the near plane of V_p by considering the direct and indirect illumination. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

constant along geodesics, the function L only needs to be computed on the scene surface objects.

Let q be a point in an embedded surface $S \subset M$. The *Riemannian illumination* of q comes from direct geodesics connecting q to the light sources and indirect geodesics connecting other Riemannian-illuminated points to q. The *radiant intensity* at q can be modeled using the *Lambertian reflectance*, which depends on the inner product, or more generally from a BRDF. Thus, it is natural to adapt this model to the Riemannian geometry by replacing the standard inner product with the Riemannian metric of the underlying 3-manifold.

4.1. Global illumination in non-Euclidean spaces

Kajiya introduced the "Rendering Equation" [27], the classical *global illumination function* of the Euclidean space $\mathbb{E}^3$. For each point p, it computes the amount of light emitted in a given direction v. This is modeled through the integral equation over the unit hemisphere $\Omega(p) = \{v \in \mathbb{S}^2 |\ \langle v, N(p)\rangle_{\mathbb{E}} \geq 0\}$:

$$L(p, v) = L_e(p, v) + \int_{\Omega(p)} f_r(p, v, w_i) L(p, w_i) \langle w_i, N\rangle_{\mathbb{E}} dw_i. \tag{1}$$

The *bidirectional reflectance distribution function* (BRDF) $f_r(p, v, w_i)$ defines how the light reflects at p, and $L_e(p, v)$ is the light emitted at p in the direction v.

Eq. (1) integrates over all directions entering the hemisphere Ω. Replacing the Euclidean metric $\langle \cdot, \cdot \rangle_{\mathbb{E}}$ with the Riemannian metric $g(\cdot, \cdot)$, we obtain a global illumination function for a Riemannian manifold (M, g),

$$L(p, v) = L_e(p, v) + \int_{\Omega(p)} f_r(p, v, w_i) L(p, w_i) g_p(w_i, N) dw_i. \tag{2}$$

The term g_p in the equation is the Riemannian metric g evaluated at the point p. Specifically, g_p is an inner product at T_pM, that is, a bilinear positive definite form. Such objects can be expressed as 3×3 symmetric positive definite matrices.

Taking into account the importance contribution of the directions w_i leaving the point p, we divide Eq. (2) into the emitted, the direct and indirect components $L(p, v) = L_e(p, v) + L_{dir}(p, v) + L_{ind}(p, v)$, where $L_e(p, v)$ is the light emitted from the surface, $L_{dir}(p, v)$ is the *direct contribution* coming from the light sources, and $L_{ind}(p, v)$ is *indirect contribution* reflected by other surfaces.

For computing the direct illumination, we evaluate the surface BRDF f_r using unit tangent vectors. Then, perform the next event estimation, i.e. find a direction towards a light source with appropriate importance sampling. Finally, compute form factors, i.e.

determine the size of the projected solid angle for a given subset of the manifold.

For the indirect illumination $L_{ind}(p, v)$, we use *Monte Carlo integration* to estimate an approximation of the light reflected from other surfaces. Let $\{w_i^1, \ldots, w_i^n\}$ be n vectors on the hemisphere Ω chosen using a distribution density d_w, the Monte Carlo integration states that

$$L_{ind}(p, v) \approx \frac{1}{n} \sum_{k=1}^{n} \frac{f_r(p, v, w_i^k) L(p, w_i^k) g_p(w_i^k, N)}{d_w(w_i^k)}. \quad (3)$$

Eq. (3) corresponds to an unbiased estimator for indirect lighting and will converge to the correct estimate when increasing the number of samples. The estimator's variance will depend mostly on the sampling strategies (i.e., BSDF sampling, emitter sampling). The function L is evaluated recursively, resulting in a *path tracing* algorithm. It is well-known in computer graphics, since Kajiya [27], that depending on the scene geometry, surface reflectance, and the quality of importance sampling strategies, the number of samples m does not need to be very large for computing realistic images.

5. Path tracing in non-Euclidean spaces

We combine Riemannian ray casting and Riemannian global illumination to synthesize inside views of Non-Euclidean spaces. This generalizes the classical *path tracing*. We focus on a geometric manifold M/Γ since it admits a well-behaved geometry M and a combinatorial description of its topology in terms of Γ.

Our method approximates the Riemannian global illumination of the visible surfaces inside a geometric manifold using the ray tracing capabilities of the RTX platform. We will discuss first the basic principles of path tracing in Non-Euclidean spaces, as well as, the general algorithm in CPU. We also show and detail how to implement the computation in GPU using the RTX pipeline.

5.1. Overview of the method

Path tracing is the most popular technique to produce photo-realistic images of Euclidean scenes. Thus, extending the technique to Non-Euclidean spaces would enable photo-realism inside such abstract spaces. This task requires knowledge about the geometry/topology of the Non-Euclidean space. The first step is to simulate the ray path as it travels inside the space, starting from the point of observation until it intersects with a visible object — the Riemannian ray casting. The second step amounts to the illumination, evaluating the light scattered from the environment in the ray direction — the Riemannian global illumination. Because of the non-trivial topology, the ray path is updated as it exits the fundamental domain.

In a nutshell, the method solves the following two problems:

1. How to compute geodesics in the space, because light or rays follow geodesics;
2. How do the light-material interaction formulas change because of the geometry.

5.2. Algorithm in CPU

We present the basic path tracing algorithm for a geometric manifold M/Γ, and compare it with the traditional path tracing of Euclidean space.

We start with an algorithm to trace rays inside M/Γ. Let p be a point inside the fundamental domain Δ of M/Γ, and v be a direction at T_pM, Algorithm 1 traces a ray γ from p towards the direction v. If γ intersects a scene object in Δ, the point and direction are updated.

Algorithm 1: Trace ray.

```
Data: point p, direction v
Result: bool hit
1  Trace a ray γ from (p, v) inside Δ;
2  repeat
3      Find closest intersection γ(t) with objects O in Δ;
4      if γ(t) ≠ ∅ then
5          Update p = γ(t) and v = γ'(t);
6          return true;
7      else
8          Find the intersection of γ with faces of Δ ;
9          Compute the new origin p' and ray γ';
10         ++i;
11     end
12 until i ≤ maxlevel;
13 return false;
```

Algorithm 2: Path Tracing in Non-Euclidean spaces.

```
1 for each pixel σ ∈ I do
2     Let p := 0 and v be the direction associated to σ;
3     if TraceRay(p, v, d) then
4         Define a depth d and a color c = 0;
5         Shade σ using L_dir(p, v) + L_ind(p, v, d, c);
6     end
7 end
```

As it can be verified in Algorithm 1, the rays are intersected with visible objects (line 3) and if there is a hit (line 4), the ray point and the direction is updated (line 5). The whole computation has the *fundamental domain* as a base, which is modeled by a polyhedron Δ. Therefore, as the ray hits a face F of Δ (line 7), we transport it using the corresponding transformation of the discrete group Γ (line 8–9). This is the most important step because it depends on the geometry and topology of the space. As such, it is specific for each type of space. For practical computational reasons, we cannot continue the ray indefinitely, thus a maximum level is set to stop the path (line 12).

Algorithm 2 describes the path tracer. The rays are generated from the observer's point of view (lines 1–2). Using Algorithm 1 the visible points are computed and if there is a hit (line 3), shading is done using direct and indirect illumination (line 4–5). Note that Algorithm 2 computes the Riemannian ray casting. To compute a good approximation of the image, it is common to accumulate the resulting image of Algorithm 2 and line 2 chooses the direction associated with the pixel using a distribution density function.

To add the shadows we need to cast an additional ray to evaluate the visibility between the sampled point and the current shading point.

Sampling the light sources turns out to be an interesting problem when rendering inside views of geometric spaces. In this work, we adopt a strategy to randomly select an element in the set of light sources contained in the nth neighborhood of the fundamental domain. Specifically, we apply the underlying discrete group on the light sources n times, each iteration adds a new neighborhood. For the spherical case, where the covering space is compact, we only need to repeat this procedure a finite number of times. For infinite spaces, we consider an approximation. This feature can be exploited in importance sampling because we can prioritize, with a probability distribution, the closest light sources. In the examples we are using, we consider the first neighborhood of copies of the

Algorithm 3: Indirect illumination.

Data: point p, direction v, depth d, color c
Result: color c
1 **if** $d \geq 0$ *and* $TraceRay(p, v)$ **then**
2 Sample a direction $w \in \Omega(p)$ using a distribution d_w;
3 c+=$L_{dir}(p, v) + \dfrac{f_r(p, v, w) L_{ind}(p, w, d-1, c) g_p(w, N)}{d_w(w)}$;
4 **end**
5 **return** c;

fundamental domain, that is $n = 1$. Then, we sample one element in the resulting set of light sources to launch the shadow ray.

For now, we are considering straight lines to trace shadow rays. This works very well in the Euclidean and hyperbolic geometries because in these geometries the rays can be modeled as straight lines. We lose some accuracy in the spherical case where the rays are big circles. In future works, we intend to explore ways to minimize this loss.

The indirect illumination is given in Algorithm 3 . An integer variable *depth* d is considered to control the number of ray light bounces of the path tracer. If d is greater than zero and the ray launched from p towards v intersect the scene (line 1), we sample a hemisphere direction and compute an approximation of the Riemannian global illumination (lines 2–3). Clearly, in line 3, Algorithm 3 is called again. This is a result of Eq. (3), considering only one sample. Algorithm 3 could be replaced by a termination strategy using Russian Roulette, probably resulting in an unbiased estimator. Nonetheless, the validity of this in Non-Euclidean space must be theoretically verified.

5.3. GPU implementation

The implementation of our visualization platform in GPU is built on top of Falcor [28] using DirectX 12 on Windows 10. The Falcor development framework consists of a library with support for DXR at a high level and a built-in scene description format. We use the software Blender to create the scene objects.

The core functionality of our system's architecture consists of a set of shaders implemented for use in the RTX GPU pipeline. We developed generic shaders for each stage of the GPU path tracing pipeline that are independent of the geometric structure of the Non-Euclidean space.

Ray Generation Shader: Creates camera rays using the isometries of the space to transform the ray origin and direction to the camera coordinate system.

Intersection Shader: Computes the ray-object intersection using a parameterization of the ray. Ray and objects are defined based on the model Geometry.

Closest Hit Shader: Performs the shading operation. This includes computing local and global illumination. The local illumination amounts to the direct contribution of light sources that are based on angles between the light direction and the surface normal, as well as, the distance to the light. The global illumination is computed by launching indirect rays based on the surface BRDF.

Miss Shader: Deals with the transport of rays in the covering space, as they leave the fundamental domain of the manifold. For this, the rays are tested for intersection with the boundary of the polyhedron Δ.

5.4. Implementation details: RTX pipeline

Nvidia RTX is a hardware/software platform with support for real-time ray tracing. The ray tracing code of an application using this architecture consists of CPU host code, GPU device code, and the memory to transfer data to the Acceleration Structures for fast geometry culling when intersecting rays with scene objects. Specifically, the CPU host code manages the memory flow between devices, sets up, controls, and spawn GPU shaders and defines the Acceleration Structures.

These shaders correspond to tasks in Algorithms 1, 2, and 3. The *Ray Generation Shader* is responsible for creating the rays (line 1 in Algorithm 2), which are defined by their origins and directions. A call to the RTX intrinsic TraceRay() launches a ray.

The next stage is a fixed traversal of the Acceleration Structure, done by the RTX runtime. This uses an *Intersection Shader* to compute the intersections (line 3 in Algorithm 1). All hits found pass by tests to verify if they are the closest hit. We use the built-in triangle intersection shader. After no additional hits are found, the *Closest-Hit Shader* is called for the closest intersection point (line 3 in Algorithm 1). In case no hits are found, the Miss Shader is called as a fallback case. Note that additional rays can be launched in the Closest-Hit/Miss shaders.

The *scene objects* and the *boundary of the fundamental domain* are treated differently when mapping the algorithm to the RTX pipeline — while the scene objects are tested and shaded in the regular way (lines 5 and 6 in Algorithm 1), the boundary of the fundamental domain is used to transport the rays by the discrete group (lines 8 and 9). This is implemented with a custom designed Miss Shader.

6. Examples and results

When we trace a ray inside a geometric manifold the underlying discrete group makes the ray iterate. If it exits the fundamental domain, it is updated by the corresponding element of the discrete group to reenter the domain. This procedure produces the illusion of a tessellation of the underlying geometry by copies of the fundamental domain. In other words, the scene is replicated following the rules codified in the discrete group.

This section presents output images from our implementation of the algorithm in GPU using RTX. The images are "photorealistic" inside views of some spaces with their geometries modeled by Euclidean, spherical, and hyperbolic space: the classical geometries detailed in Section 3.

We give an empirical discussion of the examples below. Euclidean geometry $\mathbb{E}^3$ models the flat torus, its visualization shows copies of the scene tessellating $\mathbb{E}^3$ by translations. The reflection equation is not affected by the metric. However, in the visualization of the Poincaré sphere, the equation is strongly affected by the spherical metric. In this metric, increasing the distance from the observer, the scene's size first decreases and then increases. This is because triangles in this geometry are fat. On the other hand, the triangles in the hyperbolic geometry are thin. Our hyperbolic example is an orbifold, the mirrored dodecahedron. In the hyperbolic metric, as the discrete group replicates the scene, its size decreases rapidly as the distance from the observer increases. In [29], Weeks present intuitive and empirical schemes on how to understand these geometries.

6.1. Flat torus

Probably the most famous and easiest example of a compact 3-manifold is the *flat torus* $\mathbb{T}^3$. Topologically, it is obtained by gluing opposite faces of the unit cube $[0, 1] \times [0, 1] \times [0, 1] \subset \mathbb{E}^3$. It is easy

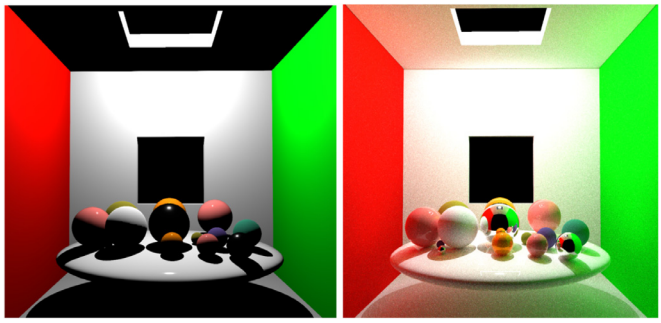

Fig. 5. Cornell box inside the fundamental domain. Rendered considering zero/five bounces.

to check that the neighborhood of each point in $\mathbb{T}^3$ is a 3-ball of the Euclidean space. Thus $\mathbb{T}^3$ is indeed a 3-manifold.

$\mathbb{T}^3$ is modeled by $\mathbb{E}^3$ because it is the quotient of the Euclidean space by the group of translation spanned by $(x, y, z) \to (x \pm 1, y, z)$, $(x, y, z) \to (x, y \pm 1, z)$, and $(x, y, z) \to (x, y, z \pm 1)$. Thus, the unit cube is the fundamental domain of $\mathbb{T}^3$.

Fig. 5 presents the visualization of a scene inside the fundamental domain using the path tracing algorithm. On the left, we do not use the indirect light contribution, on the right we add the indirect light contribution considering five bounces. Note that we are using a *Cornell Box* with an additional window to set up our scene. The spheres inside the cube have specular, diffuse, and composed materials. The only light source is above the box, close to the top window.

We now identify the faces of the torus fundamental domain (the cube). A ray leaving a point $p \in \mathbb{T}^3$ in a direction v is parameterized as $r(t) = p + t \cdot v$ in $\mathbb{E}^3$. For each intersection between r and a face F of the unit cube, we update p by its correspondent point $p - n$ in the opposite face, where n is the unit vector normal to F. The ray direction v does not need to be updated. The rays in $\mathbb{T}^3$ can return to the starting point, providing copies of the scene. The inside perception is $\mathbb{E}^3$ tessellated by cubes: each cube contains one copy of the scene.

Fig. 6 provides an immersive visualization of the 3-dimensional torus $\mathbb{T}^3$ using the path tracer algorithm. On its top, an outside view of the embedded Cornell Box showing how the fundamental domain tessellates the space. On its middle, a closer view on the specular spheres embedded in the scene to show the tessellation being reflected on their surface. On its bottom, the Cornell box is removed.

Fig. 7 illustrates the interactive interface of the rendering program with a panel to control the various parameters of the algorithm. Now, the scene is composed of two spheres and a cow model. Although we are using path tracing, we use only one sample when interactively exploring the scene and setting parameters up. The resulting noisy image is improved when the camera is not moving by an accumulation step incorporated in the rendering graph.

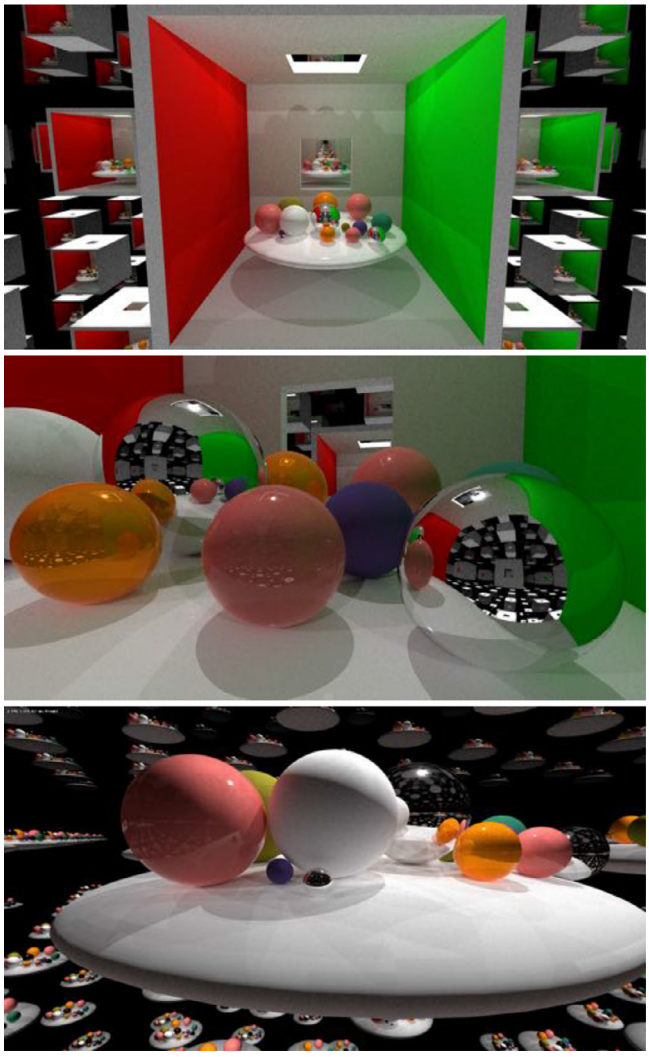

Fig. 6. Inside views of the flat torus. The face pairing makes the rays that leave a face return from its opposite face, giving rise to many copies of the scene.

6.2. Poincaré sphere

If the opposite faces of a dodecahedron are identified by a clockwise rotation of $\pi/5$, we get the *Poincaré dodecahedral space*. This manifold is also known as the *Poincaré homological sphere* since its first homological group is trivial.

The face pairing forces many identifications and the edges are grouped into ten groups of three edges. To model the geometry of such space the dihedral angle of the dodecahedron must be 120. This is not possible to model with Euclidean geometry, but can be done with spherical geometry. To find the required dodecahedron we consider it embedded in the 3-sphere. If the dodecahedron is very small its dihedral angle approximates to the Euclidean dodecahedron. Then, with an appropriate scale, the dodecahedron dihedral angle is equal to 120 degrees.

We adapt Algorithm 1 to trace rays inside Poincaré sphere using RTX. Observe that the RTX is only used in lines 1 and 3. The remaining steps can be performed using the spherical setting. We use a parameterization of the fundamental domain to alternate between the Euclidean and the spherical settings.

The fundamental domain $D_{\mathbb{S}}$ of the Poincaré sphere can be parameterized by a flat dodecahedron D centered at the origin of $\mathbb{E}^3$ using the map $(x, y, z) \to (x, y, z, 1)/|(x, y, z, 1)|_{\mathbb{E}}$. To move points and directions between the settings, just apply the parameterization to the points and its differential to the directions.

To compute the intersections between spherical rays and the fundamental domain faces (used in lines 8–9 of Alg. 1), we observe that faces of $D_{\mathbb{S}}$ are contained in unit 2-spheres. A sphere of this set is written as $\{p \in \mathbb{S}^3 \,|\, \langle p, n\rangle_{\mathbb{E}} = 0\}$, where $n \in \mathbb{S}^3$ is the suspension of the normal vector of a flat dodecahedron face. Then the intersections between a 2-sphere and a ray $\gamma(t) = \cos t \cdot p + \sin t \cdot v$ leaving $p \in \mathbb{S}^3$ towards $v \in T_p\mathbb{S}^3$, are given by the solutions of $\langle \cos t \cdot p + \sin t \cdot v, n\rangle_{\mathbb{E}} = 0$, which is equivalent to $\tan t = -\langle p, n\rangle_{\mathbb{E}} / \langle v, n\rangle_{\mathbb{E}}$.

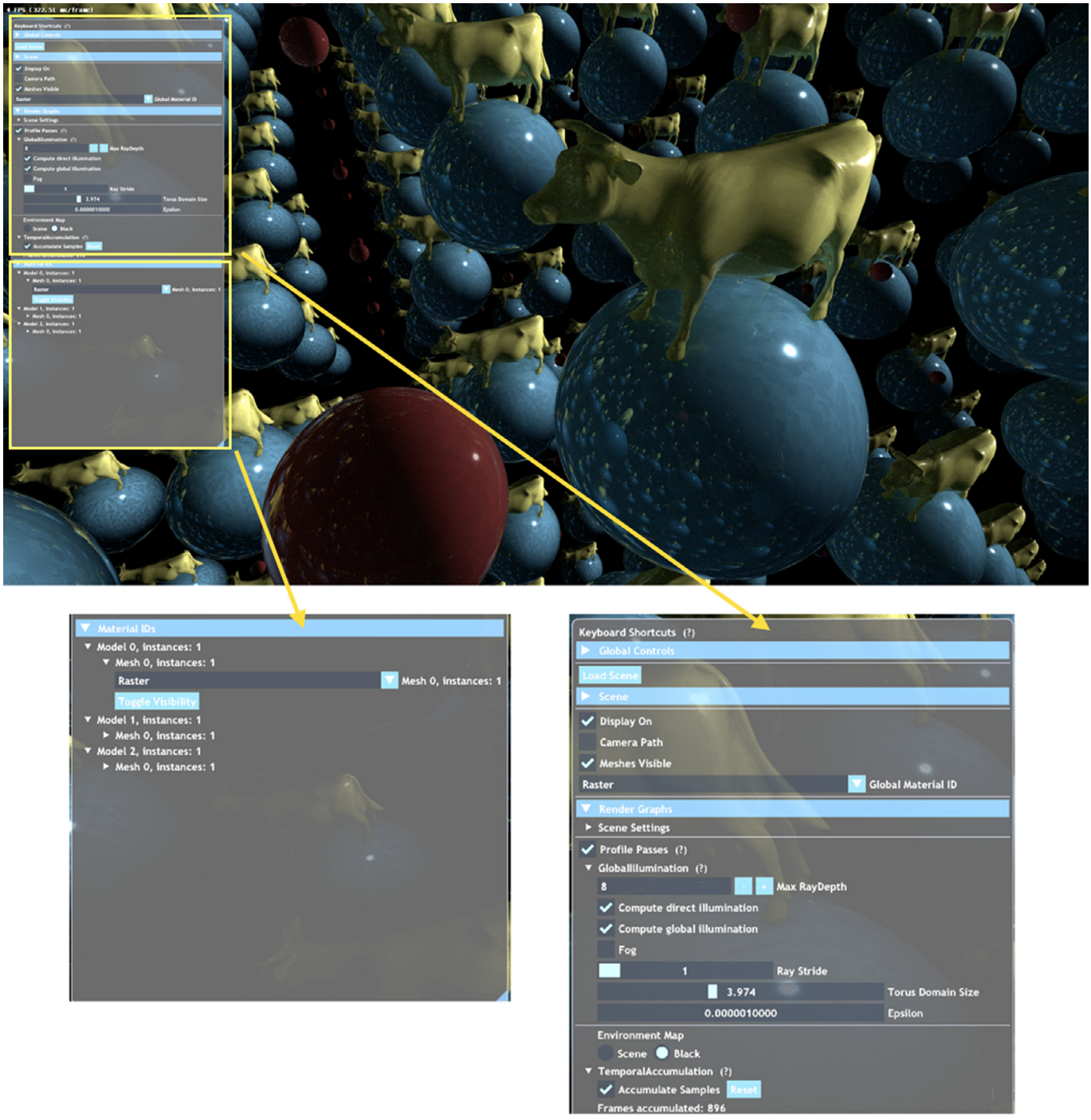

Fig. 7. Interactive program interface with control parameters. For example, from this interface, we can access the scene controls, as the camera and light parameters. The render graph parameters are also available in the interface. For instance, we can control the maximum number of bounces, enable the direct/indirect illuminations, and control the temporal accumulation step.

If a ray intersects a face of the dodecahedron we update its position and direction using the face transformation in the spherical setting. The immersive visualization of the Poincaré dodecahedral space is a tessellation of $\mathbb{S}^3$ by 120 dodecahedra. This is one of the 4-dimensional polytopes, known as 120-cell and shown in Fig. 8 using global illumination.

On the top, the scene is composed of a single cow, a pentagon floor, four spheres (specular red, totally specular, and two diffuses: yellow and purple). We also include the edges of the fundamental domain of the Poincaré sphere. These edges are brown and show the space structure. They are computed analytically using the spherical setting without using the RTX. The face identifications produce a tessellation of the 3-sphere. Looking towards a dodecahedron's face we see copies of the scene rotated by $\pi/5$. Note that as the distance increases, the scene's size first decreases and then begins to increase: there is a large cow upside down in the background. In the middle, we increase the specularity of the pentagon floor and replace the scene by a single specular sphere and a single monkey's head, the *Suzanne* classical Blender mesh. Again, as the distance increases, Suzanne's size first decreases and then increases. The dodecahedron's edges are removed. On the bottom, a specular sphere and a diffuse ellipsoid with its

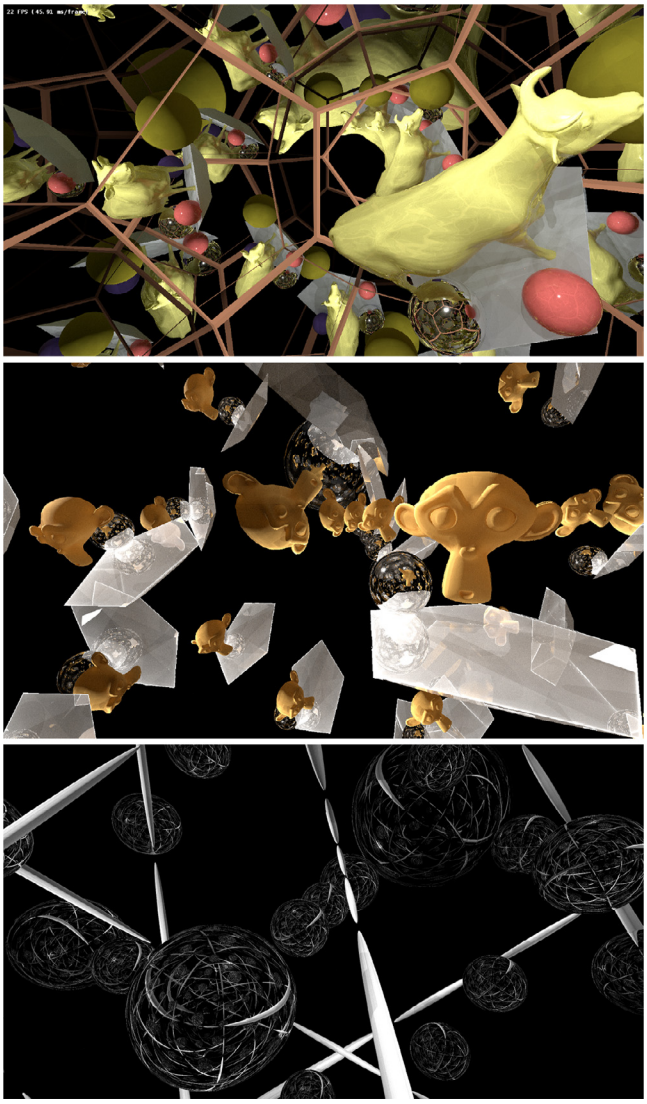

Fig. 8. Inside views in the Poincaré sphere, which is obtained by identifying, with a rotation of $\pi/5$, the opposite faces of a dodecahedron embedded in the 3-sphere. We use a parameterization of this dodecahedron to set our scenes. The face pairing makes the rays that leave a face return, with an additional rotation, from its opposite face, resulting in many copies of the scene: a tessellation of the sphere, the 4-dimensional regular polytope known as *120-cell*.

axis aligned with a geodesic of the underlying manifold. The face gluing makes copies of the ellipsoid which are distributed along the geodesic in the Poincaré sphere. The specular sphere illustrates the perfect reflection of the space.

6.3. Mirrored dodecahedron

For an example of a Non-Euclidean space modeled by the hyperbolic geometry, consider the dodecahedron embedded in the Klein's model $\mathbb{K}^3$ of the hyperbolic space. Let Γ be the group of hyperbolic reflections generated by the dodecahedral faces. With an appropriate scale, the dihedral angle of the dodecahedron reaches 90 degrees. The quotient $\mathbb{K}^3/\Gamma$ is an orbifold, denominated *mirrored dodecahedral space*. The discrete group Γ tessellates $\mathbb{K}^3$ with dodecahedra with right dihedral angle, thus each edge is adjacent to exactly 4 cells.

As the rays are straight in the Klein's model $\mathbb{K}^3$, we can use RTX to launch them without difficulty. Each time a ray intersects a dodecahedron face F at a point p, we update the ray direction v using its hyperbolic reflection $v - 2n\langle v, n\rangle_{\mathbb{H}}/\langle n, n\rangle_{\mathbb{H}}$, where the vector v is presented in homogeneous coordinates, and n is the homogeneous coordinates of the plane containing the dodecahedron face F. For more details on hyperbolic transformations see Phillips and Gunn's paper [30]. Observe that when tracing rays in the hyperbolic mirrored dodecahedron, we only have to replace the Euclidean inner product by the hyperbolic inner product.

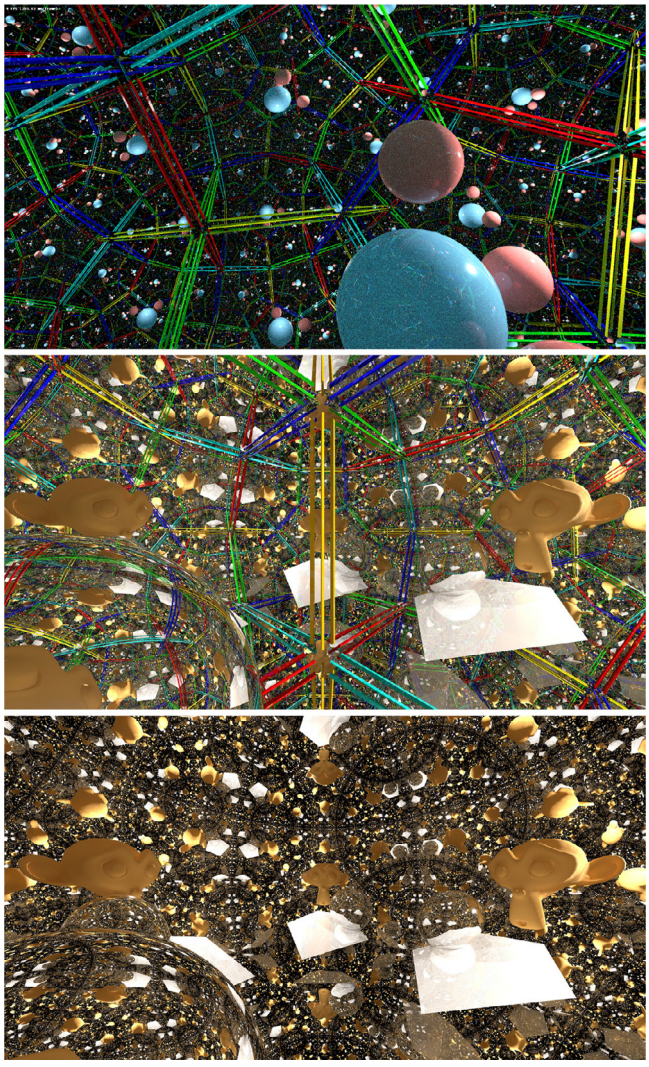

Fig. 9. Immersive visualization of the mirrored dodecahedron. This space is obtained by considering the faces of a hyperbolic regular dodecahedron to be perfect hyperbolic mirrors.

Fig. 9 illustrates immersive visualizations of the hyperbolic mirrored dodecahedron. On the top, the scene is composed of exactly three spheres, two red and one blue. We add the fundamental domain edges to highlight the hyperbolic space tessellated by dodecahedra. The image is the inside view of the group of reflection Γ acting on the Hyperbolic space. In the middle, we maintain the dodecahedron edges and replace the scene by another a little bit more complex — similar to the one used in Fig. 8 (middle). On the bottom, we remove the dodecahedron edges.

7. Conclusions and future works

We presented photorealistic inside views of the three-dimensional flat torus, the Poincaré sphere, and the mirrored

hyperbolic dodecahedron using a novel path tracing algorithm. Previous works were restricted to simple rendering techniques using Lambertian surfaces.

The algorithm was based on Falcor, NVIDIA's scientific prototyping framework, which uses the power of the new generation of RTX GPUs.

A probability distribution was used to sample the directions in the superior hemisphere of each shaded point. In future works, we intend to explore how to apply importance-sampling strategies with different Riemannian metrics. We also intend to explore other materials and more complex BSDFs.

To improve the path tracing efficiency, it is common to use next-event estimations, for example, the shadow rays. In this work, we have chosen an approach based on two considerations. First, we used a linear approximation of shadow rays, which works well on Euclidean and hyperbolic geometries, but needs to be improved in the Spherical case. Second, we considered the first neighborhood of the replicated scenes to sample the light sources used in the direct illumination. In future works, we intend to improve these tasks which becomes difficult depending on the geometry considered.

Another interesting future work is the application of our path tracing to other geometries. Interesting cases include the other Thurston geometries: $\mathbb{S}^2 \times \mathbb{R}$, $\mathbb{H}^2 \times \mathbb{R}$, Nil, Sol, and $\widetilde{SL_2(\mathbb{R})}$. In particular, the case of Nil geometry is attractive because its rays have a closed expression in terms of elementary functions.

One limitation of our work is that, because of the computational cost of path tracing, we cannot view these spaces in real-time. For now, we are using an accumulation step in the rendering graph, then we can move the camera inside the scene when using a few bounces to find a good position. We believe that the use of neural networks could help us in such a task.

An interesting problem and another path for future work is to adapt the algorithm for VR. Immersive photorealistic VR can be a great tool for visualizing such spaces. However, the camera is never truly static in this context, which renders the accumulation strategy useless.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Tiago Novello: Conceptualization, Investigation, Methodology, Software, Writing - review & editing. **Vincius da Silva:** Conceptualization, Investigation, Methodology, Software, Writing - review & editing. **Luiz Velho:** Conceptualization, Investigation, Methodology, Software, Writing - review & editing.

References

[1] Pharr M, Jakob W, Humphreys G. Physically based rendering: from theory to implementation. Morgan Kaufmann; 2016.
[2] Velho L, Novello T, Silva V, Lucio D. Visualization of non-Euclidean spaces using ray tracing. Tech. Report. Visgraf-Impa; 2019.
[3] Barr AH. Ray tracing deformed surfaces. ACM SIGGRAPH Computer Graph 1986;20(4):287–96.
[4] Amenta N, Levy S, Munzner T, Phillips M. Geomview: a system for geometric visualization. In: Proceedings of the eleventh annual symposium on Computational geometry. ACM; 1995. p. 412–13.
[5] Weeks J. Real-time rendering in curved spaces. IEEE Comput Graph Appl 2002;22(6):90–9.
[6] Weeks J. Real-time animation in hyperbolic, spherical, and product geometries. In: Non-Euclidean geometries. In: Mathematics and Its Applications, 581. US: Springer; 2006. p. 287–305.
[7] Hudson R, Gunn C, Francis GK, Sandin DJ, DeFanti TA. Mathenautics: using VR to visit 3-d manifolds. In: Proceedings of the 1995 symposium on interactive 3D graphics. New York, NY, USA: ACM; 1995. p. 167–70. ISBN 0-89791-736-7.
[8] Francis GK, Goudeseune CMA, Kaczmarski HJ, Schaeffer BJ, Sullivan JM. Alice on the eightfold way: exploring curved spaces in an enclosed virtual reality theatre. In: Visualization and mathematics III; 2003. p. 305–15.
[9] Hart V, Hawksley A, Matsumoto EA, Segerman H. Non-euclidean virtual reality I: explorations of h^3. In: Proceedings of bridges 2017: mathematics, music, art, architecture, culture; 2017.
[10] Weeks J. Non-euclidean billiards in VR. In: Proceedings of bridges 2020: mathematics, music, art, architecture, culture; 2020.
[11] Weeks J. Virtual reality simulations of curved spaces. In: preparation 2020.
[12] Berger P, Laier A, Velho L. An image-space algorithm for immersive views in 3-manifolds and orbifolds. Vis Comput 2014.
[13] da Silva V., Velho L.. Ray-VR: ray tracing virtual reality in falcor. arXiv preprint arXiv:200611348 2020.
[14] Velho L, Silva Vd, Novello T. Immersive visualization of the classical non-Euclidean spaces using real-time ray tracing in VR. In: Proceedings of graphics interface 2020; 2020. p. 423–30. ISBN 978-0-9947868-5-2. doi:10.20380/GI2020.42.
[15] Novello T., da Silva V., Velho L.. How to see the eight Thurston geometries. arXiv preprint arXiv:200512772 2020.
[16] Novello T., Silva V.d., Velho L.. Visualization of nil, sol, and sl2 geometries. Comput Graph2020;91:219–231. doi:10.1016/j.cag.2020.07.016.
[17] Novello T., da Silva V., Velho L.. Design and visualization of Riemannian metrics. arXiv preprint arXiv:200505386 2020.
[18] Coulon R., Matsumoto E.A., Segerman H., Trettel S.. Non-euclidean virtual reality iii: nil. arXiv preprint arXiv:200200513 2020.
[19] Coulon R., Matsumoto E.A., Segerman H., Trettel S.. Non-euclidean virtual reality iv: sol. arXiv preprint arXiv:200200369 2020.
[20] Kopczyński E., Celińska-Kopczyńska D.. Real-time visualization in non-isotropic geometries. arXiv preprint arXiv:200209533 2020.
[21] Stam J, Languénou E. Ray tracing in non-constant media. In: Rendering techniques' 96. Springer; 1996. p. 225–34.
[22] Gröller E. Nonlinear ray tracing: visualizing strange worlds. Vis Comput 1995;11(5):263–74.
[23] Thurston W. The geometry and topology of three-manifolds. Princeton University; 1979.
[24] Martelli B.. An introduction to geometric topology. arXiv preprint arXiv:161002592 2016.
[25] Scott P. The geometries of 3-manifolds. Bull Lond Math Soc 1983;15(5):401–87.
[26] Carmo MPd. Riemannian geometry. Birkhäuser; 1992.
[27] Kajiya JT. The rendering equation. In: ACM SIGGRAPH computer graphics, 20. ACM; 1986. p. 143–50.
[28] Benty N., Yao K.-H., Foley T., Oakes M., Lavelle C., Wyman C. The Falcor rendering framework. 2018. Https://github.com/NVIDIAGameWorks/Falcor.
[29] Weeks J. The shape of space. Marcel Dekker; 2002. ISBN 9780824707095.
[30] Phillips M, Gunn C. Visualizing hyperbolic space: unusual uses of 4x4 matrices. In: Proceedings of the 1992 symposium on interactive 3D graphics. New York, NY, USA: ACM; 1992. p. 209–14. ISBN 0-89791-467-8. doi:10.1145/147156.147206.