# Visualizing Thurston's geometries

Tiago Novello
Vinícius da Silva
Luiz Velho

impa

33º Colóquio
Brasileiro de
Matemática

# Visualizing Thurston's geometries

# *Contents*

# *Preface*

In 2018, NVidia introduced the RTX series of GPUs enabling the implementation of real-time ray-tracing algorithms in Euclidean spaces, which allows visualization applications with a high degree of photorealism. In the same year, Luiz Velho and Vinícius da Silva started the Ray-VR project, at IMPA's Visgraf Laboratory, to integrate Virtual Reality (VR) and ray tracing. In 2019, Tiago Novello joined the project, which started using the developed framework to visualize non-Euclidean spaces in an immersive and interactive way.

Examples of non-Euclidean spaces date back to Thurston's geometrization conjecture, which states that any three-dimensional compact manifold one decomposes into geometrically modeled pieces by just eight geometries. Ray-VR gave rise to a system for immersive and interactive visualization (in VR) of spaces modeled by the classic Thurston geometries (Euclidean, spherical, and hyperbolic), later, the results were extended to manifolds modeled by the "twisted" geometries (Nil, Sol, and SL2): the least trivial Thurston's geometries.

This book presents a compilation of Ray-VR results in the intrinsic visualization of Thurston's geometries. This is an active research topic in mathematical visualization that combines the areas of geometry and topology, with concepts of computer graphics. The content of this book serves both experts and students. Although this is a short book, it is self-contained since it considers all the ideas, motivations, references, and intuitive explanations of the required fundamental concepts.

It is important to highlight that several conditions made this a special moment for such a topic. On one hand, the development of mathematical research and

graphics algorithms has provided the theoretical framework. On the other hand, the evolution of media technologies allows us to be immersed in three-dimensional spaces using VR.

The target reader of this book would be interested in geometry, topology, mathematical education, and also interested in new visualization techniques to explore abstract spaces. For the public interested in media, this work offers the possibility to explore new mathematical scenarios. These visualizations have the potential to be applied in entertainment, arts, education, cinema, and games.

We thank the Organizing Committee of the 33rd Brazilian Colloquium of Mathematics for the opportunity to present our results in mathematical visualization.

# 1 Background on Manifolds and Orbifolds

## 1.1 History

This chapter sets the stage with an historical account of the quest to investigate 2D and 3D spaces, as well as the context related to the Poincaré Conjecture inspiring the classification / geometrization of compact two and three dimensional manifolds.

### 1.1.1 Henri Poincaré

In 1895, Henri Poincaré published his *Analysis situs* (Poincaré 1895), in which he presented the foundations of topology by proposing to study spaces under continuous deformations; position is not important. The main tools for topology are introduced in this paper: manifolds, homeomorphisms, homology, and the fundamental group. He also discussed about how three-dimensional geometry was real and interesting. However, there was a confusion in his paper: Poincaré treated homology and homotopy as equivalent concepts.

In 1904, Poincaré wrote the fifth supplement 1904 to Analysis situs, where he approached three-dimensional manifolds. This paper clarified that homology was not equivalent to homotopy in dimension three. He presented the *Poincaré dodec-*

*ahedron* as an example of a 3-manifold with trivial homology but with nontrivial homotopy. In Section 1.4.1, we present an inside view of such space. Poincaré proposed the conjecture: Is the 3-sphere the unique compact connected 3-manifold with trivial homotopy?

Poincaré stimulated a lot of mathematical works asking whether some manifold exists. Works on this question were awarded three Fields medals. In 1960, Stephen Smale proved 2007 the conjecture for $n$-manifolds with $n > 4$. In 1980, Michael Freedman proved 1982 Poincaré conjecture for 4-manifolds. The problem in dimension three was open until 2003 when Grigori Perelman proved (Perelman 2002, 2003a,b) Thurston's geometrization conjecture, and consequently the Poincaré conjecture as a corollary.

Poincaré also worked on an important problem in dimension two, the *uniformization theorem*. This states that every simply connected *Riemann surface* (one-dimensional complex manifolds) is *conformally equivalent* to the unit disc, the complex plane, or the Riemann sphere. This was conjectured by Poincaré in 1882 and Klein in 1883, and proved by Poincaré and Koebe in 1907. The history details can be found in the recent book by Ghys (2017). A big step in the history of the geometry was the generalization of this result for dimension three, *Thurston's geometrization theorem*.

## 1.1.2   William P. Thurston

Thurston's works in 3-manifolds have a geometric taste with roots in topology. He tried to generalize the uniformization theorem of compact surfaces to dimension three. Five more geometries arise; the hyperbolic still playing the central role.

In 1982, Thurston stated the *geometrization conjecture* (Thurston 1982) with solid justifications. It is a three-dimensional version of the uniformization theorem, where hyperbolic geometry is the most abundant because it models all surfaces with genus greater than one. In dimension three, Thurston (ibid.) proved that the conjecture holds for a huge class of 3-manifolds, the *Haken manifolds*, implying that hyperbolic plays, again, the central role. The result is known as the *hyperbolization theorem*. Thurston received in 1982 a Fields medal for his contributions to 3-manifolds. The *Elliptization conjecture*, the part which deals with spherical manifolds, was open at that time.

### 1.1.3 Grigori Perelman

In 2000, the Clay Institute selected seven problems in mathematics to guide mathematicians in their research, the *seven Millennium Prize Problems* (Jaffe 2006). Poincaré conjecture was one of them. They did not know that the Poincaré conjecture was about to be solved by Grigori Perelman as a corollary of the proof of the geometrization conjecture.

In 2003, Perelman published three papers (Perelman 2002, 2003a,b), in arXiv solving the Geometrization conjecture. He used tools from geometry and analysis. Specifically, he used the *Ricci flow*, a technique introduced by Richard Hamilton to prove the Poincaré conjecture. Hamilton proved the conjecture for a special case when the 3-manifold has positive *Ricci curvature*. The idea is to use Ricci flow to simplify the geometry along time. However, this procedure may create *singularities* since this flow expands regions with negative Ricci curvature and contracts regions of positive Ricci curvature. Hamilton suggested the use of *surgery* before the manifold collapse. The procedure gives rise to a simpler manifold, and we can evolve the flow again. Perelman, proved that this algorithm stops and each connected component of the resulting manifold admits one of the Thurston geometries. In other words, Perelman proved the geometrization conjecture, and consequently the Poincaré conjecture.

## 1.2   2-Manifolds

We present some results involving topology and geometry of surfaces. We assume all surfaces been compact, connected, and oriented. Starting with the *classification theorem* in terms of the *connected sum*, one can represent a surface through a polygon with an appropriate edge gluing. This polygon can be embedded in one of the three two-dimensional geometry models (Euclidean, spherical, and hyperbolic). The resulting surface has the geometry modeled by one of these geometries.

### 1.2.1   Classification of compact surfaces

The classical way to state the classification theorem is by the *connected sum*. Removing disks $D_1$ and $D_2$ from surfaces $S_1$ and $S_2$, one obtains their connect sum $S_1 \# S_2$ by identifying the boundaries $\partial D_1$ and $\partial D_2$ through a homeomorphism. The theorem says that any compact surface is homeomorphic to a sphere or a connected sum of tori. The theorem proof uses a computational representation of a compact surface $S$ through an appropriate pairwise gluing of edges in a polygon:

- Take a triangulation $T$ of $S$; it is a well-known result;

- Cutting along edges in $T$ we obtain a list of triangles embedded in the plane without intersection; the edge pairing must be remembered;

- We label each triangle edge with a letter according to its gluing orientation;

- Gluing the triangles through their pairwise edge identifications without leaving the plane produces a polygon $P$. The boundary $\partial P$ is an oriented sequence of letters;

- Let $a$ and $b$ be a couple of edges in $\partial P$. If the identification of $a$ and $b$ reverses the orientation of $\partial P$ we denote $b$ by $a^{-1}$, and simply $a$ otherwise;

- A technical result states that by cutting and gluing $P$ leads us to an equivalent polygon $Q$ with its boundary having one of following configurations:

    – $aa^{-1}$, which is a sphere;
    – $\sum aba^{-1}b^{-1}$, a connected sum of tori $aba^{-1}b^{-1}$.

To model the geometry of those surfaces, we embed, in a special way, the polygon in one of the two-dimensional model geometries.

### 1.2.2   Geometrization of compact surfaces

We remind the well-known *geometrization* theorem of compact surfaces which states that any topological surface can be modeled using only three geometries.

**Theorem 1** (Geometrization of surfaces). *Any compact surface admits a geometric structure modeled by the Euclidean, the hyperbolic, or the spherical space.*

The Euclidean space $\mathbb{E}^2$ models the geometry of the 2-torus through the quotient of $\mathbb{E}^2$ by the group of translations. The sphere is modeled by the spherical geometry.

For a hyperbolic surface, consider the *bitorus*, which topologically is the connect sum of two tori. The bitorus is presented as a regular polygon $P$ with 8 sides $aba^{-1}b^{-1}cdc^{-1}d^{-1}$ as discussed above. All vertices in $P$ are identified into a unique vertex $v$. Then, the 8 corners of $P$ are glued together producing a topological disk. Considering $P$ with the Euclidean geometry, the angular sum around $v$ equals to $6\pi$. To avoid such a problem, let $P$ be a regular polygon centered in the hyperbolic plane, with an appropriate scale, its angles sum $\pi/4$. The

edge pairing of $P$ induces a group action $\Gamma$ in the hyperbolic plane $\mathbb{H}^2$ such that $\mathbb{H}^2/\Gamma$ is the bitorus. In terms of *tessellation*, $\Gamma$ tessellates $\mathbb{H}^2$ with regular 8-gons. Analogously, all surfaces represented as polygons with more than four sides are hyperbolic. Implying that hyperbolic is the most abundant geometry.

The above discussion handled all orientable surfaces. The well-known Gauss–Bonnet theorem implies that these geometric structures must be unique.

## 1.3  3-Manifolds

It took time to formulate the modern idea of a manifold in a higher dimension. For example, a version of Theorem 1 for 3-manifold seemed not possible until 1982, when Thurston proposed the geometrization conjecture 1982. There are exactly eight geometries in dimension 3, which are presented in more detail in Section 1.4. Scott (1983) is a great text on this subject.

### 1.3.1  Classification of compact 3-manifolds

As for surfaces, there is a combinatorial procedure to build three-dimensional manifolds from identifications of polyhedral faces. To do so, endow a finite number of polyhedra with an appropriate pairwise identification of its faces. Each couple of faces has the same number of edges and it is mapped homeomorphically to each other. Such gluing gives a *polyhedral complex $K$*, which is a 3-manifold iff its Euler characteristic is equal to zero (Theorem 4.3 in (Fomenko and Matveev 2013)).

We now take the opposite approach. Let $M$ be a compact 3-manifold, we represent $M$ as a polytope $P$ endowed with a pairwise identification of its faces. The following algorithm mimics the surface case presented in Section 1.2.1.

- Let $T$ be a triangulation of $M$; endorsed by the triangulation theorem;

- Detaching every face identification in $T$ gives rise to a collection of tetrahedra which can be embedded in $\mathbb{E}^3$. Remember the pairwise face gluing;

- Gluing in a topological way each possible coupled tetrahedra without leaving $\mathbb{E}^3$ produces a polytope $P$. The faces in $\partial P$ are pairwise identified.

The combinatorial problem of reducing $P$ to a standard form, as in the surface case, remains open (see page 145 in Lee (2010)). Although there is not (yet) a classification of compact 3-manifold in the sense presented for compact surfaces,

it is still possible to decompose the given manifold into simpler pieces. Thurston conjectured that these pieces can be modeled by eight geometries.

The decomposition used in the geometrization theorem (to be presented in Section 1.3.2) has two stages: the prime and the tori decomposition. The first is similar to the inverse of the connected sum. It consists of cutting the 3-manifold along a 2-sphere such that the resulting two disconnected 3-manifolds are not balls. After attaching balls to the boundary of these parts, one obtains two simpler 3-manifolds. A *prime* 3-manifold does not admit such sphere decomposition. Kneser proved that, after a finite number of steps, a manifold *factorizes* into prime manifolds, and Milnor proved that the decomposition is unique (Milnor 1962) up to homeomorphism.

*Tori decomposition* (Jaco and Shalen 1979; Johannson 1979) consists of cutting a prime 3-manifold along "certain incompressible" tori embedded. The result is a 3-manifold bounded by tori that are left as boundaries because there is no canonical way to close such holes.

Decomposing a 3-manifold through the above procedure produces a list of simpler manifolds, which resembles an evolutionary tree (McMullen 2011) (see Figure 1.1). Thurston conjectured that it is possible to choose the separating tori so that the simplest pieces can be modeled by one of the eight geometries. This is the 3-dimensional case of Theorem 1: the *Thurston-Perelman geometrization theorem*.

### 1.3.2   Geometrization of compact 3-manifolds

The geometrization of surfaces is controlled by the Euler characteristic. 3-manifolds are more complicated. Thurston (1982) proposed that the simpler manifolds given by the prime and tori decomposition can be modeled by eight geometries. These geometries include Euclidean, hyperbolic, and spherical spaces.

**Theorem 2** (Geometrization). *Any compact, orientable, prime 3-manifold can be constructed using just 8 geometry models.*

The other five geometries are the product spaces $\mathbb{S}^2 \times \mathbb{R}$ and $\mathbb{H}^2 \times \mathbb{R}$, endowed with the product metric, and the "twisted" product geometries *Nil*, *Sol*, and $\widetilde{SL_2(\mathbb{R})}$. All the eight geometries are homogeneous, that is, for every pair of points, there is a local isometry sending one to another. Only Euclidean, hyperbolic, and spherical spaces have *isotropic* geometries, that is, isometries on the tangent space at every point can be realized as isometries of the underlying manifold. We give more details of Thurston geometries in Section 1.4.

Let $M$ be an oriented, compact 3-manifold and $M = M_1 \# M_2 \# \cdots \# M_n$ be its prime decomposition. A more formal statement of Theorem 2 is the following. There is a finite collection of disjoint, embedded, incompressible tori in each prime 3-manifold $M_i$, so that each component of the complement admits a geometric structure modeled on one of the eight Thurston's geometries. Remember that a 3-manifold is geometrically modeled by one of Thurston geometries if it is the quotient of one of these spaces by a discrete group. Therefore, the geometrization theorem factorizes the manifold $M$ into pieces (the leaves in Figure 1.1) modeled by the eight geometries. The inverse of the above procedure explains the word *construct* in Theorem 2.

In the surface case, hyperbolic geometry played a central role. The same happens in dimension three, most of the eight geometries are required to describe particular manifolds. Thurston (ibid.) said that hyperbolic geometry is by far the most interesting, the most complex, and the most useful among the eight geometries. In Section 1.4, we present some ideas explaining the abundance of manifolds modeled by hyperbolic geometry.

The geometrization theorem implies the Poincaré conjecture. A compact simply connected 3-manifold is prime and does not contain a torus non-trivially embedded (its fundamental group is trivial). The theorem implies that the manifold is modeled by one of the eight geometries. As the fundamental group is finite, the manifold must be the quotient of the sphere by a discrete group (*Elliptization theorem*), which should be trivial since it is isomorphic to the fundamental group.

At this point, we should clarify two hard questions. Why are there exactly eight geometries? How can Theorem 2 be proved? We present some informal intuitions and ideas of the proofs. The first question is approached in Section 1.4. The technique using Gauss–Bonnet theorem does not work in this case.

Perelman's proof of the geometrization theorem involves geometry and analysis tools that are beyond the scope of this paper. Informally, Perelman's argument consists of starting from a 3-manifold endowed with a Riemannian metric $g_0$. Then running Hamilton's Ricci flow $\frac{\partial g_t}{\partial t} = -2Ric(g_t)$, where $g_t$ is the metric which evolves along time controlled by the *Ricci curvature*. This smooths the metric giving a more "uniform" shape to the manifold (similar to the heat equation). This procedure may produce singularities since (in some sense) the differential equation may create critical elements. Perelman overcomes this by cutting the manifold into certain pieces (prime and tori decomposition) just before the collapse appears. Then he repeats the method on each of the individual pieces. He proved that this algorithm decomposes the manifold in a "tree" with each leaf been a manifold with geometry modeled by one of the Thurston geometries, see

Figure 1.1.
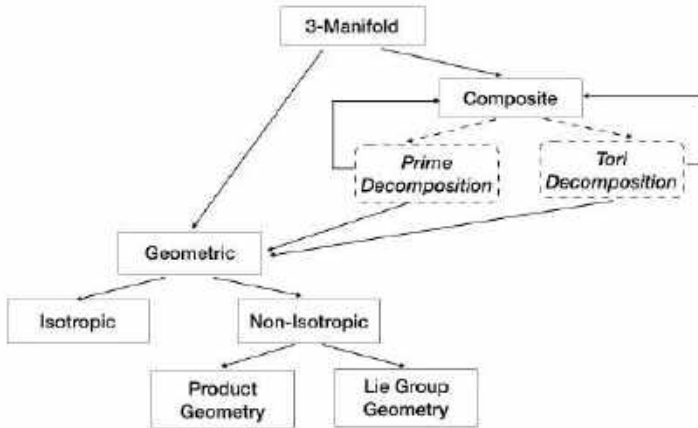


Figure 1.1: Evolutionary tree of a compact orientable 3-manifold. It operates like an algorithm. The first two layers indicate the prime and tori decomposition of the 3-manifold. The last two is the geometrization theorem.

## 1.4   8 Thurston Geometries

We provide the definitions and some features of the geometries that appear in the geometrization theorem. We also justify why the hyperbolic geometry is the richest, presenting all the manifolds modeled by the Thurston geometries. Other great presentations of the eight geometries include (Martelli 2016; Scott 1983; Thurston 1982; Weeks 2020b).

The classification mentioned above uses the *Seifert manifolds*: closed manifolds admitting a decomposition in terms of disjoint circles. Martelli (2016) describes two results. The first states that a closed orientable 3-manifold can be modeled by one of the following six geometries: $\mathbb{R}^3$, $\mathbb{S}^3$, $\mathbb{S}^2 \times \mathbb{R}$, $\mathbb{H}^2 \times \mathbb{R}$, $Nil$, $\widetilde{SL_2(\mathbb{R})}$) iff it belongs to a special class of Seifert manifolds. It has a Sol geometric structure iff it admits a particular torus bundle, called *torus (semi-)bundle of Anosov type*.

The second result states that if a 3-manifold is geometrically modeled by one of Thurston geometries, it is specified by the manifold fundamental group:

| Fundamental group | Model geometry | | |
|---|---|---|---|
| Finite | $\mathbb{S}^3$ | | |
| Virtually cyclic | $\mathbb{S}^2 \times \mathbb{R}$ | | |
| Virtually abelian | $\mathbb{R}^3$ | | |
| Virtually nilpotent | $Nil$ | | |
| Virtually solvable | $Sol$ | | |
| Contains a normal cyclic group | Quotient lifts a finite-index subgroup | $\mathbb{H}^2 \times \mathbb{R}$ | |
| | Otherwise | $\widetilde{SL_2(\mathbb{R})}$ | |
| Otherwise | $\mathbb{H}^3$ | | |

The seven classes of fundamental groups aforementioned represent a restricted portion of the set of all possible fundamental groups, which implies that the hyperbolic manifolds are more abundant. We skip these group definitions because they deviate from the scope of this paper.

Thurston geometries can be divided in three classes. The *classical* geometries which consist of isotropic spaces: Euclidean, spherical, and hyperbolic spaces. The remaining are the non-isotropic geometries and are composed of the *product* geometries $\mathbb{S}^2 \times \mathbb{R}$ and $\mathbb{H}^2 \times \mathbb{R}$ and the *"twisted" product* geometries $Nil$, $Sol$, and $\widetilde{SL_2(\mathbb{R})}$. All these spaces are homogeneous, every pair of points admits similar neighborhoods. The classical geometries admit constant *sectional curvature* since they are isotropic (Carmo 1992).

## 1.4.1   Classical geometries

For dimension $n \geqslant 2$ exists a unique complete, simply connected Riemannian manifold having constant sectional curvature 1, 0, or $-1$. These are the isotropic geometries: the sphere, the Euclidean space, and the hyperbolic space. Conversely, if a complete manifold has constant sectional curvature 1, 0, or $-1$, it must be the quotient of such models geometries by a discrete group (Proposition 4.3 in (ibid.)). We present these geometries, examples of manifolds modeled by them, and the behavior of rays in such spaces.

### Euclidean space

In dimension two, every orientation preserving isometry in Euclidean space is a translation. Then, if $\mathbb{E}^2/\Gamma$ is a compact orientable surface, it must be the torus

(see Sec 6.2 of (Martelli 2016)). In dimension three the list is increased by five more orientable manifolds since we can compose rotations with translations.

The *Euclidean space* $\mathbb{E}^3$ is $\mathbb{R}^3$ endowed with the Euclidean inner product $\langle u, v \rangle_{\mathbb{E}} = u_x \cdot v_x + u_y \cdot v_y + u_z \cdot v_z$, 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 $\gamma(t) = p + t \cdot v$ describes a *ray* leaving a point $p$ in a direction $v$. Analogously, for any $n > 0$ the Euclidean space $\mathbb{E}^n$ is constructed.

The *flat torus* $\mathbb{T}^3$ is a 3-manifold modeled by $\mathbb{E}^3$, it is obtained by gluing opposite faces of the unit cube in $\mathbb{E}^3$. $\mathbb{T}^3$ is also the quotient of $\mathbb{E}^3$ by its group of translation spanned by $(x, y, z) \rightarrow (x \pm 1, y, z)$, $(x, y, z) \rightarrow (x, y \pm 1, z)$, and $(x, y, z) \rightarrow (x, y, z \pm 1)$. The unit cube is the fundamental domain.

A ray leaving a point $p \in \mathbb{T}^3$ in a direction $v$ is parameterized as $r(t) = p + t \cdot v$. For each intersection between $r$ and a face $F$ of the unit cube, we update $p$ by $p - n$ in the opposite face; $n$ is normal to $F$. The direction $v$ does not need to be updated.

Then, we have the ingredients for an inside view of $\mathbb{T}^3$. The fundamental domain receives the scene and the rays in $\mathbb{T}^3$ can return to it, resulting in many copies of the scene. The immersive perception is $\mathbb{E}^3$ tessellated by scene copies. Section 4.1.1 will present inside views of the flat torus.

Beyond the torus, there are exactly five more compact oriented 3-manifold with geometry modeled by the Euclidean geometry, see Figure 1.2.

### Hyperbolic space

Here we present the *hyperboloid and Klein models* of the hyperbolic geometry. There are plenty of hyperbolic manifolds, making this concept a central actor in the topology of 3-manifolds (ibid.).

The *Lorentzian space* is $\mathbb{R}^4$ with the product $\langle u, v \rangle_{\mathbb{H}} = u_x v_x + u_y v_y + u_z v_z - u_w v_w$, where $v, u \in \mathbb{R}^4$. The *hyperbolic geometry* $\mathbb{H}^3$ is represented by the hyperboloid $\{p \in \mathbb{R}^4 | \langle p, p \rangle_{\mathbb{H}} = -1\}$ endowed with the metric

$$d_{\mathbb{H}}(p, q) = \cosh^{-1}(-\langle p, q \rangle_{\mathbb{H}}),$$

where $p$ and $q$ are points in $\mathbb{H}^3$. Due to its similarity to the sphere definition, $\mathbb{H}^3$ is also known as *pseudosphere*.

A tangent vector $v$ to a point $p$ in $\mathbb{H}^3$ satisfies $\langle p, v \rangle_{\mathbb{H}} = 0$. Moreover, the *tangent space* $T_p\mathbb{H}^3$ coincides with the set $\{v \in \mathbb{R}^4 | \langle p, v \rangle_{\mathbb{H}} = 0\}$. The Lorentzian inner product is positive on each tangent space.
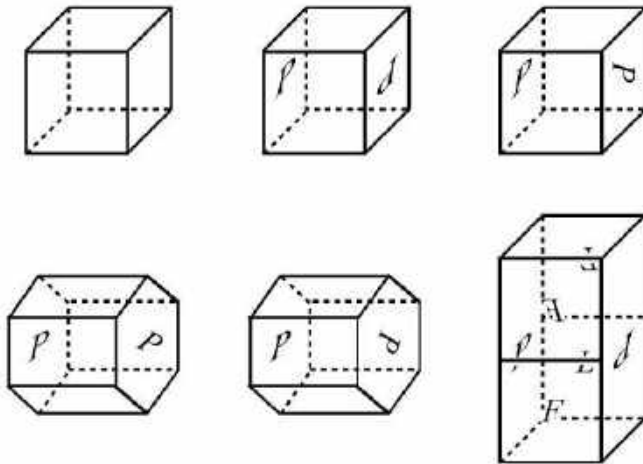
Figure 1.2: The six compact oriented flat manifolds. These are built through pairwise gluing: faces are identified isometrically according to their labels, otherwise, it is glued to its opposite in an obvious way. From (Martelli 2016).

*Rays* in $\mathbb{H}^3$ arise from intersections between $\mathbb{H}^3$ and planes in $\mathbb{R}^4$ containing the origin. A ray leaving a point $p \in \mathbb{H}^3$ in a tangent direction $v$ is the intersection between $\mathbb{H}^3$ and the plane spanned by the vectors $v$ and $p$. Its parameterization is $r(t) = \cosh(t)p + \sinh(t)v$. Thus, rays in $\mathbb{H}^3$ can not be straight lines.

It is possible to model $\mathbb{H}^3$ in the unit open ball in $\mathbb{R}^3$ — known as the *Klein model* $\mathbb{K}^3$— such that in this model the rays are straight lines. More precisely, each point $p \in \mathbb{H}^3$ is projected in the space $\{(x, y, z, w) \in \mathbb{R}^4 | \ w = 1\}$ by considering $p/p_w$, the space $\mathbb{K}^3$ is obtained by forgetting the coordinate $w$.

The hyperbolic space is a model of a *Non-Euclidean* geometry, since it does not satisfy the Parallel Postulate: given a ray $r$ and a point $p \notin r$, there is a unique ray parallel to $r$. In hyperbolic space, within a two-dimensional hyperbolic plane, for a ray $r$ and a point $p \notin r$ there is an infinite number of rays going through $p$ which do not intersect $r$.

For a compact 3-manifold geometrically modeled by hyperbolic geometry considers the *Seifert–Weber dodecahedral space*. It is the dodecahedron with an identification of its opposite faces with a clockwise rotation of $3\pi/10$. The face pairing groups edges into six groups of five, making it impossible to use Euclidean geometry. The regular Euclidean dodecahedron has a dihedral angle of $\sim 116$ degrees. The desired dodecahedron should have a dihedral angle of 72 degrees, which is

possible in hyperbolic space considering an appropriate dodecahedron diameter.

Then, we ray trace Seifert–Weber dodecahedron. A ray leaving a point $p \in M$ in a tangent direction $v$ is given by $\gamma(t) = p + tv$ since we are using Klein's model. For each intersection between $\gamma$ and a dodecahedron face, we update $p$ and $v$ through the hyperbolic isometry that produces the face pairing above. This isometry is quite distinct from Euclidean isometries (Gunn 1993). The immersive perception of $M$ using this approach is a tessellation of $\mathbb{H}^3$ by dodecahedra with a dihedral angle of 72 degrees. Section 4.3.1 will illustrate inside views of the Seifert–Weber dodecahedron.

### Spherical space

The 3-*sphere* $\mathbb{S}^3$ is $\{p \in \mathbb{E}^4 | \langle p, p \rangle_{\mathbb{E}} = 1\}$ with the metric

$$d_{\mathbb{S}}(p, q) = \cos^{-1} \langle p, q \rangle_{\mathbb{E}}.$$

As in the hyperbolic case, a tangent vector $v$ to a point in $\mathbb{S}^3$ satisfies $\langle p, v \rangle_{\mathbb{E}} = 0$. The *tangent space* $T_p\mathbb{S}^3$ corresponds to the set of vectors $\{v \in \mathbb{S}^3 | \langle p, v \rangle_{\mathbb{E}} = 0\}$. 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 tangent direction $v$ is the arc produced by intersecting $\mathbb{S}^3$ with the plane spanned by $v$, $p$, and the origin of $\mathbb{E}^4$. Such ray is parameterized as $r(t) = cos(t)p + sin(t)v$.

$\mathbb{S}^3$ is a Non-Euclidean geometry because it fails the Parallel Postulate: given a ray $r$ and a point $p \notin r$, there is a unique ray parallel to $r$. In $\mathbb{S}^3$, within a 2-sphere $\mathbb{S}^2$, each pair of distinct rays always intersects at exactly two points.

Gluing the opposite faces of the dodecahedron with a clockwise rotation of $\pi/5$ we get the *Poincaré dodecahedral space* (or *Poincaré homological sphere*); its first homological group is trivial. The face pairing groups edges into ten groups of three edges. Then, we need a dodecahedron with dihedral angle of 120. In this case, we use spherical geometry by finding a dodecahedron in the 3-sphere with an appropriate diameter. The immersive visualization of Poincaré dodecahedral space is a tessellation of $\mathbb{S}^3$ by 120 dodecahedra. This is a 4-dimensional regular polytope: the 120-*cell*. Section 4.2.1 will give inside views of the Poincaré sphere.

### 1.4.2   Product geometries

The eight three dimensional geometries include products of lower-dimensional geometries, which are $\mathbb{S}^2 \times \mathbb{R}$ and $\mathbb{H}^2 \times \mathbb{R}$ endowed with the product metric. We do not focus on visualizing these spaces because they model few manifolds (Martelli 2016). Visualizations of these geometries are given by (Weeks 2006).

## $\mathbb{S}^2 \times \mathbb{R}$ space

The geometry $\mathbb{S}^2 \times \mathbb{R}$ models very few manifolds. The sectional curvature is 1 along with horizontal directions and 0 along with verticals. Recall that sectional curvature of a plane is the Gauss curvature associated with the surface generated by such a plane.

The manifold $\mathbb{S}^2 \times \mathbb{S}$ endowed with the product metric can be modeled by $\mathbb{S}^2 \times \mathbb{R}$. The geometry of $\mathbb{S}^2 \times \mathbb{S}$ can not be modeled by classical geometries, since $\mathbb{S}^2 \times \mathbb{S}$ has $\mathbb{S}^2 \times \mathbb{R}$ as its universal covering and it is not isotropic.

## $\mathbb{H}^2 \times \mathbb{R}$ space

The geometry $\mathbb{H}^2 \times \mathbb{R}$ is given by the product metric. Analogous to the $\mathbb{S}^2 \times \mathbb{R}$, horizontal and vertical planes have sectional curvature $-1$ and 0.

### 1.4.3 "Twisted" product geometries

The remaining three non-isotropic geometries to analyze are not products, but they admit a kind of "bundle structure". The first attempt to visualize these geometries in real-time (using VR) appeared in 2019 (Novello, V. da Silva, and Velho 2020e). We give a brief introduction of these geometries. In Chapter 5, we present inside views of these geometries. See (Coulon et al. 2020a,b,c; Kopczyński and Celińska-Kopczyńska 2020; Rogue 2020; Skrodzki 2020) for other great works.

#### Nil space

We follow the definitions of (Martelli 2016). *Nil space* is a Lie group consisting of all $3 \times 3$ real matrices

$$\begin{bmatrix} 1 & x & z \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix}$$

with the multiplication operation. We identify $\mathbb{R}^3$ and *Nil* using

$$(x, y, z) \in \mathbb{R}^3 \rightarrow \begin{bmatrix} 1 & x & z \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix} \in Nil$$

as a parameterization (for ray tracing, we set up our scene in $\mathbb{R}^3$). Then, the multiplication of $(x, y, z)$ and $(x', y', z')$ is given by:

$$(x, y, z) \cdot (x', y', z') = \begin{bmatrix} 1 & x & z \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & x' & z' \\ 0 & 1 & y' \\ 0 & 0 & 1 \end{bmatrix}$$

$$= (x + x', y + y', z + z' + xy').$$

In other words, the multiplication of elements in $Nil$ is the sum of its coordinates, with an additional term in the last one. This term makes all the difference, since, to put geometry in $Nil$, we consider the left multiplication $(x, y, z) \rightarrow p \cdot (x, y, z)$, for each $p \in Nil$, being an isometry.

We construct a metric in Nil. The origin of $Nil$ is $e = (0, 0, 0)$, and $T_e Nil = \mathbb{R}^3$ is the tangent space at $e$ with the Euclidean product $\langle u, v \rangle_e = u_x v_x + u_y v_y + u_z v_z$. Let $p$ be a point in $Nil$, we define a scalar product $\langle \cdot, \cdot \rangle_p$ in $T_p Nil$. Since $p^{-1} = (-p_x, -p_y, -p_z + p_x p_y)$ we have the isometry that translates $p$ to the origin $e$:

$$\varphi(x, y, z) := L_{p^{-1}}(x, y, z) = (x - p_x, y - p_y, z - p_z + p_x(p_y - y)).$$

The differential of the map $\varphi$ at a point $p$ in the tangent direction $v$ is given by $d\varphi_p(v) = v_x \cdot d\varphi_p(e_1) + v_y \cdot d\varphi_p(e_2) + v_z \cdot d\varphi_p(e_3)$, where $\{e_1, e_2, e_3\}$ is standard base of $\mathbb{R}^3$. The partial derivatives are $d\varphi_p(e_1) = (1, 0, 0)$, $d\varphi_p(e_2) = (0, 1, p_x)$, and $d\varphi_p(e_3) = (0, 0, 1)$, hence $d\varphi_p(v) = (v_x, v_y, v_y p_x + v_z)$. The following equation defines the product between two tangent vectors $u$ and $v$ at $p$:

$$\langle u, v \rangle_p = \langle d\varphi_p u, d\varphi_p v \rangle_e = u^T \begin{bmatrix} 1 & 0 & 0 \\ 0 & p_x^2 + 1 & -p_x \\ 0 & -p_x & 1 \end{bmatrix} v.$$

The $3 \times 3$ matrix above defines a metric $g_{ij}$ in the tangent space at $p$. Varying $p$ we obtain a Riemannian metric $\langle \cdot, \cdot \rangle$, since each matrix entry is differentiable. The vectors $(1, 0, 0)$, $(0, 1, x)$, and $(0, 0, 1)$ form an orthogonal basis at $p = (x, y, z)$. Also, the volume form of Nil space coincides with the standard one from the Euclidean space, since $\det[g_{ij}] = 1$.

To compute the Christoffel symbols of $Nil$ at $(x, y, z)$, we use its well-known formula from Riemannian geometry (Carmo 1992). Their are all zero except

for (Martelli 2016):

$$\Gamma_{22}^1 = -x, \ \Gamma_{23}^1 = \Gamma_{32}^1 = \frac{1}{2},$$

$$\Gamma_{12}^2 = \Gamma_{21}^2 = \frac{x}{2}, \ \Gamma_{13}^2 = \Gamma_{31}^1 = -\frac{1}{2},$$

$$\Gamma_{12}^3 = \Gamma_{21}^3 = \frac{(x^2 - 1)}{2}, \ \Gamma_{13}^3 = \Gamma_{31}^3 = -\frac{x}{2}.$$

The computation (Szilágyi and Virosztek 2003) can be done using the software *Maple*. Replacing the Christoffel symbols of $Nil$ in the formula of the *geodesic flow* (Carmo 1992), we obtain the geodesic flow of $Nil$. This admits a solution (Szilágyi and Virosztek 2003). A ray $\gamma(t) = (x(t), y(t), z(t))$ starting at $(0, 0, 0)$ in the unit tangent direction $v = (c\cos(\alpha), c\sin(\alpha), w)$ has the following form:

$$x(t) = \frac{c}{w}(\sin(wt + \alpha) - \sin(\alpha))$$

$$y(t) = -\frac{c}{w}(\cos(wt + \alpha) - \cos(\alpha))$$

$$z(t) = t(w + \frac{c^2}{2w}) - \frac{c^2}{4w^2}(\sin(2wt + 2\alpha) - \sin(2\alpha))$$

$$+ \frac{c^2}{2w^2}(\sin(wt + 2\alpha) - \sin(2\alpha) - \sin(tw)).$$

To compute a geodesic $\beta(t)$ starting at $p$ in the direction $v$, we translate $\beta$ to the origin using the left multiplication $\varphi(x, y, z) = (-p_x, -p_y, -p_z + p_x p_y) \cdot (x, y, z)$. Observe that, $\varphi(0) = e$ and $d\varphi_p(v) = v - (0, 0, p_x v_y)$. Therefore, the curve $\varphi \circ \beta(t) = (x(t), y(t), z(t))$ is a ray starting at $e$ in the direction $v - (0, 0, p_x v_y)$, and it can be computed using the above closed formula. To compute $\beta(t)$ we simply apply $L_p$ to $\varphi \circ \beta(t)$ to obtain the desired formula

$$\beta(t) = p \cdot (x(t), y(t), z(t)) = (p_x + x(t), p_y + y(t), p_z + z(t) + p_x y(t)).$$

## $\widetilde{SL_2(\mathbb{R})}$ space

We follow the definitions of (Gilmore 2008). The *special linear group* $SL_2(\mathbb{R})$ consisting of all $2 \times 2$ matrices with unit determinant is a Lie group. Indeed, the product of two matrices with unit determinant has unit determinant, the same for the inverse matrix. To understand the richness of $SL_2(\mathbb{R})$, we present two interpretations: one geometrical and another more topological.

**Geometrical interpretation:**   We can interpret $SL_2(\mathbb{R})$ as a 3-manifold in $\mathbb{R}^4$ using the expression $\{(a, b, c, d) \in \mathbb{R}^4 |\ ad - bc = 1\}$. Moreover, rephrasing $ad - bc = 1$ we obtain:

$$\left(\frac{a + d}{2}\right)^2 - \left(\frac{a - d}{2}\right)^2 + \left(\frac{b - c}{2}\right)^2 - \left(\frac{b + c}{2}\right)^2 = 1,$$

which describes the equation of a 3-hyperboloid in $\mathbb{R}^4$.

**Topological interpretation:**   We can also identify $SL_2(\mathbb{R})$ with the product of the (hyperbolic) *upper half plane* $\mathbb{H} = \{z \in \mathbb{C}|\ Im(z) > 0\}$ and the unit circle $\mathbb{S}^1$. Before, we provide some additional definitions. An element $g = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \in SL_2(\mathbb{R})$ induces an *action* in $\mathbb{H}$:

$$gz = \begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} z \\ 1 \end{bmatrix} = \begin{bmatrix} az + b \\ cz + d \end{bmatrix} = \frac{az + b}{cz + d}. \tag{1.1}$$

We used homogeneous coordinates in Equation (1.1).

The action in Equation (1.1) is well defined, that is $cz + d \neq 0$ and $Im(gz) > 0$. As $ad - bc = 1$, either $c$ or $d$ are non null, so is $cz + d$. To verify that the imaginary part of $gz$ is greater than zero we multiply the numerator and denominator of $gz$ by $c\bar{z} + d$, where $\bar{z}$ is the conjugate of $z$. After some calculations we get $Im(gz) = Im(z)/|cz + d|^2$, which is greater than zero.

We now provide a decomposition of $g$ in two components $AN$ and $K$, known as *Iwasawa decomposition*. Where $K$ will be responsible by rotations around the point $i \in \mathbb{H}^2$ and $AN$ will be translations of $i$. This procedure provides the decomposition of $SL_2(\mathbb{R})$ into the product $\mathbb{S}^1 \times \mathbb{H}^2$. Specifically, define

$$AN = \begin{bmatrix} \dfrac{1}{\sqrt{c^2 + d^2}} & \dfrac{ac + bd}{\sqrt{c^2 + d^2}} \\ 0 & \sqrt{c^2 + d^2} \end{bmatrix}, \quad K = \begin{bmatrix} d & -c \\ c & d \end{bmatrix} \cdot \dfrac{1}{\sqrt{c^2 + d^2}}.$$

It can be verified that the product between $AN$ and $K$ gives rise to the matrix $g$. Observe that $K$ can be written as $\begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}$, for some angle $0 \leqslant \phi \leqslant 2\pi$. Thus, the matrices $K$ can be parameterized by $\mathbb{S}^1$. We now verify that $AN$ parameterize $\mathbb{H}^2$.

Let $z = x + iy$ be a point in $\mathbb{H}^2$, then

$$z = \begin{bmatrix} \sqrt{y} & \frac{x}{\sqrt{y}} \\ 0 & \frac{1}{\sqrt{y}} \end{bmatrix} \begin{bmatrix} i \\ 1 \end{bmatrix} = x + iy. \tag{1.2}$$

Every point in $\mathbb{H}^2$ can be written as the action of some matrix of the linear special group $SL_2(\mathbb{R})$ on $i$. Taking $y = c^2 + d^2$ and $x = ac + bd$ the correspondence between $AN$ and $\mathbb{H}^2$ follows.

The identification of $SL_2(\mathbb{R})$ with $\mathbb{H}^2 \times \mathbb{S}^1$ provides a topological interpretation of $SL_2(\mathbb{R})$: its fundamental group is $\mathbb{S}^1$. Then $SL_2(\mathbb{R})$ is not simply connected, hence it is not a model geometry. However, its *universal cover* $\widetilde{SL_2(\mathbb{R})}$ is a model geometry(Thurston 1979). We focus on $SL_2(\mathbb{R})$ since they have identical geometry and we are dealing with visualization.

We could use the above parameterization of $SL_2(\mathbb{R})$ by $\mathbb{H}^2 \times \mathbb{S}^1$, however, we take an easier coordinate system. We parameterize a neighborhood of the identity of $SL_2(\mathbb{R})$ by a neighborhood of the origin of $\mathbb{R}^3$ using the map (Gilmore 2008):

$$X(x, y, z) = \begin{bmatrix} 1 + x & y \\ z & \frac{1 + yz}{1 + x} \end{bmatrix}. \tag{1.3}$$

Observe that the element $X(0, 0, 0)$ is the identity of $SL_2(\mathbb{R})$, and that in the plane $x = 1$ the map is not well defined. We use the map $X$ to push-back the metric of $SL_2(\mathbb{R})$ to $\mathbb{R}^3$. In other words, we construct a metric in the $SL_2(\mathbb{R})$ and use the map defined in Equation (1.3) to push it to $\mathbb{R}^3$.

The identity matrix is the identity $e$ of $SL_2(\mathbb{R})$. Let $T_e SL_2(\mathbb{R})$ be the tangent space at $e$ with the well known (ibid.) matrix scalar product $\langle u, v \rangle_e = Tr(u \cdot v)$ between two $2 \times 2$ matrices $u$ and $v$ in $T_e SL_2(\mathbb{R})$. $Tr$ is the matrix trace. We define a scalar product $\langle \cdot, \cdot \rangle_p$ in $T_p SL_2(\mathbb{R})$ for a point $p$ using the left multiplication $\varphi$, where

$$p^{-1} = \begin{bmatrix} \frac{1 + p_y p_z}{1 + p_x} & -p_y \\ -p_z & 1 + p_x \end{bmatrix}.$$

Therefore we obtain the parameterization $\varphi(x, y, z) = p^{-1} \cdot X(x, y, z)$. Its differential at $p$, applied to the tangent vector $v$, can be expressed by the form $d\varphi_p(v) =$

$v_x \cdot d\varphi_p(e_1) + v_y \cdot d\varphi_p(e_2) + v_z \cdot d\varphi_p(e_3)$. The partial derivatives are:

$$d\varphi_p(e1) = \begin{bmatrix} \dfrac{1 + p_y p_z}{1 + p_x} & \dfrac{p_y + p_y^2 p_z}{(1 + p_x)^2} \\[3mm] -p_z & -\dfrac{1 + p_y p_z}{1 + p_x} \end{bmatrix},$$

$$d\varphi_p(e2) = \begin{bmatrix} 0 & \dfrac{1}{1 + p_x} \\[3mm] 0 & 0 \end{bmatrix}, \; d\varphi_p(e3) = \begin{bmatrix} -p_y & -\dfrac{p_y^2}{1 + p_x} \\[3mm] -1 + p_x & p_y \end{bmatrix}.$$

After some computations using the scalar product at $T_e SL_2(\mathbb{R})$ we obtain the metric tensor at $p$:

$$\begin{bmatrix} 2\dfrac{(1 + p_y p_z)}{(1 + p_x)^2} & -\dfrac{p_z}{1 + p_x} & -\dfrac{p_y}{1 + p_x} \\[3mm] -\dfrac{p_z}{1 + p_x} & 0 & 1 \\[3mm] -\dfrac{p_y}{1 + p_x} & 1 & 0 \end{bmatrix}. \tag{1.4}$$

To compute a ray in $SL_2(\mathbb{R})$ we need the Christoffel symbols at each point $\varphi(p)$, which are all zero except for

$$\Gamma_{11}^1 = -\frac{1 + p_y p_z}{1 + p_x}, \quad \Gamma_{12}^1 = \frac{p_z}{2}, \quad \Gamma_{13}^1 = \frac{p_y}{2}, \quad \Gamma_{23}^1 = -\frac{1 + p_x}{2}$$

$$\Gamma_{11}^2 = -\frac{p_y + p_y^2 p_z}{(1 + p_x)^2}, \quad \Gamma_{12}^2 = \frac{p_y p_z}{2 + 2 p_x}, \quad \Gamma_{13}^2 = \frac{p_y^2}{2 + 2 p_x}, \quad \Gamma_{23}^2 = -\frac{p_y}{2}$$

$$\Gamma_{11}^3 = -\frac{p_z + p_z^2 p_y}{(1 + p_x)^2}, \quad \Gamma_{12}^3 = \frac{p_z^2}{2 + 2 p_x}, \quad \Gamma_{13}^3 = \frac{p_y p_z}{2 + 2 p_x}, \quad \Gamma_{23}^3 = -\frac{p_z}{2}.$$

Remember that $\Gamma_{ij}^k = \Gamma_{ji}^k$. We obtain the geodesic flow by replacing the Christoffel symbols in the general geodesic flow formula (Carmo 1992).

$$\begin{cases} x_k' = y_k, \; k = 1, 2, 3. \\[2mm] y_1' = \dfrac{(1 + p_y p_z)y_1^2}{1 + p_x} - p_z y_1 y_2 - p_y y_1 y_3 + (1 + p_x)y_2 y_3 \\[3mm] y_2' = \dfrac{(1 + p_y p_z)p_y y_1^2}{(1 + p_x)^2} - \dfrac{p_z p_y}{1 + p_x}y_1 y_2 - \dfrac{p_y^2}{1 + p_x}y_1 y_3 + p_y y_2 y_3 \\[3mm] y_3' = \dfrac{(1 + p_y p_z)p_z y_1^2}{(1 + p_x)^2} - \dfrac{p_z^2}{1 + p_x}y_1 y_2 - \dfrac{p_y p_z}{1 + p_x}y_1 y_3 + p_z y_2 y_3 \end{cases}$$

We use Euler's numerical method to integrate this equation.

### Sol space

Sol is the least symmetric one among Thurston geometries. As in the Nil space, we follow the definitions of Martelli (2016). The *Sol space* is a Lie group consisting of all matrices

$$\begin{bmatrix} e^z & 0 & x \\ 0 & e^{-z} & y \\ 0 & 0 & 1 \end{bmatrix}$$

with the multiplication operation. *Sol* identifies with $\mathbb{R}^3$, since

$$(x, y, z) \in \mathbb{R}^3 \rightarrow \begin{bmatrix} e^z & 0 & x \\ 0 & e^{-z} & y \\ 0 & 0 & 1 \end{bmatrix} \in Sol$$

defines a parameterization. We push-forward the differentiable structure of $\mathbb{R}^3$ to *Sol* and use $\mathbb{R}^3$ to set our scene for ray tracing.

Let $(x, y, z)$ and $(x', y', z')$ be elements in Sol space, their multiplication is given by the formula:

$$\begin{aligned} (x, y, z) \cdot (x', y', z') &= \begin{bmatrix} e^z & 0 & x \\ 0 & e^{-z} & y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} e^{z'} & 0 & x' \\ 0 & e^{-z'} & y' \\ 0 & 0 & 1 \end{bmatrix} \\ &= (x'e^z + x, y'e^{-z} + y, z + z'). \end{aligned}$$

Specifically, the multiplication in *Sol* is the sum of the coordinates of the elements with an additional exponential term. We now define a geometry in *Sol*.

The element $e = (0, 0, 0)$ is the identity of *Sol*. We consider the tangent space $T_e Sol$ at $e$ to be endowed with the Euclidean scalar product. Let $p$ be a point in *Sol*, we define a scalar product $\langle \cdot, \cdot \rangle_p$ in $T_p Sol$ as in Nil geometry case. Then, we use the left multiplication $\varphi := L_{p^{-1}}$, and its differential $d\varphi_p$, to transpose vectors from $T_p Sol$ to $T_e Sol$. As $p^{-1} = (-p_x e^{p_z}, -p_y e^{-p_z}, -p_z)$, we have $\varphi(x, y, z) = (xe^{-p_z} - p_x e^{p_z}, ye^{p_z} - p_y e^{-p_z}, z - p_z)$.

The differential of the map $\varphi$ applied to a tangent vector $v$ has the form $d\varphi_p(v) = v_x \cdot d\varphi_p(e_1) + v_y \cdot d\varphi_p(e_2) + v_z \cdot d\varphi_p(e_3)$. Computing the partial derivatives, $d\varphi_p(e1) = (e^{-p_z}, 0, 0)$, $d\varphi_p(e2) = (0, e^{p_z}, 0)$, and $d\varphi_p(e3) = (0, 0, 1)$, we obtain $d\varphi_p(v) = (v_x e^{-p_z}, v_y e^{p_z}, v_z)$.

We derive the scalar product between two tangent vectors $u$ and $v$ at $p$ as

$$\langle u, v \rangle_p = \langle d\varphi_p(u), d\varphi_p(v) \rangle_e = u^T \begin{bmatrix} e^{2p_z} & 0 & 0 \\ 0 & e^{-2p_z} & 0 \\ 0 & 0 & 1 \end{bmatrix} v.$$

The $3 \times 3$ matrix above defines a metric at $p$. Varying $p$ we obtain a Riemannian metric $\langle \cdot, \cdot \rangle$, since each matrix entry is differentiable. The volume form of *Sol* coincides with the standard one from $\mathbb{R}^3$, since the determinant of the above matrix is one. To compute a ray in *Sol* we need the Christoffel symbols at each point $(x, y, z)$, which are all zero except ((Martelli 2016)):

$$\Gamma_{13}^1 = \Gamma_{31}^1 = 1, \ \ \Gamma_{23}^2 = \Gamma_{32}^2 = -1,$$
$$\Gamma_{11}^3 = -e^{2z}, \ \ \Gamma_{22}^3 = \Gamma_{31}^3 = e^{-2z}.$$

We use the Christoffel symbols of *Sol* to obtain its *geodesic flow*:

$$\begin{cases} x_k' & = y_k, \ k = 1, 2, 3. \\ y_1' & = -2y_1 y_3 \\ y_2' & = 2y_2 y_3 \\ y_3' & = e^{2p_z} y_1^2 - e^{-2p_z} y_2^2 \end{cases} \tag{1.5}$$

Let $p \in Sol$ and $v \in T_p Sol$ be an initial condition for Equation (1.5). Then, there is no solution for this problem in terms of elementary functions (Bölcskei and Szilágyi 2007). To overcome this we use Euler's numerical method to integration as described in Section 3.3.3.

We believe that the development of Computer Graphics techniques in Riemannian geometry could be an ally in the mathematical research in low dimension. In particular, several works have been using Virtual Reality to interactively visualize the eight Thurston geometries (Novello, V. da Silva, and Velho 2020d,e; Weeks 2020a,c). In the next chapters we introduce the new ray tracing techniques which allow (real-time) visualizations of Thurston geometries.

# 2

# *Immersive Visualization in Virtual Reality*

## 2.1   3D Visualization

A 3D visualization algorithm renders an image of a three-dimensional scene according to a view specification. The input of the algorithm is a scene description composed of ambient three-dimensional space, 3D shapes placed in this ambient space, and a viewpoint, among other parameters. The output is a two-dimensional view. In that sense, the rendering process transforms geometric three-dimensional information into visual two-dimensional information.

### 2.1.1   The Viewing Transformation Pipeline

In order to understand this process, let us recall the *viewing transformation pipeline*, which relates the different spaces and coordinate systems involved in the computation of a rendered image. To introduce the reader to the visualization of non-Euclidean spaces using computer graphics techniques, we consider the ambient space to be a Riemannian 3-manifold $M$.

We describe a scene $S$ inside $M$ — the *world space*. Each object $o \in S$ has a position $p \in M$ and an orthogonal transformation $T$ in the tangent space $T_p M$. Thus $T_p M$ is a natural candidate to be the *object space*, and the exponential map

$\exp_p : T_p M \rightarrow M$ provides the *object coordinate system*. The object can be described parametrically or implicitly in $T_p M$. All the objects are placed in $M$ by a modeling transformation. Specifically, we embed each object $o \in S$ in the world space using the map $\exp_p \circ T$: composition of the transformation $T$ encoding the object orientation with the exponential map $\exp_p$ associated with the object space.

Let $q$ be the *camera position* in the 3-manifold $M$. The view is specified in $T_q M$ (the *camera space*) using the exponential map $\exp_q$ which defines the *camera coordinate system* relative to the world. The objects that are visible from the camera are mapped to the *image coordinate system* (which implements the viewing window). More precisely, we model a *view frustum V* directly in the tangent space $T_q M$. The exponential map is used to release the rays, which in this case, will be geodesics. Thus objects of the scene inside the projected view frustum $\exp_q(V)$ are mapped to the image space. This pipeline is shown in Figure 2.1.
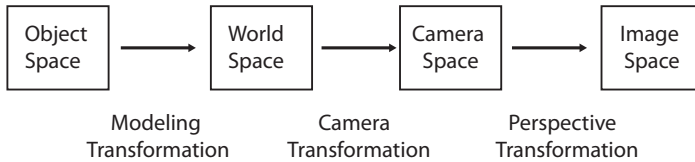


Figure 2.1:  Viewing Transformation Pipeline .

Since Molnár (1997) presented a projective interpretation of the Thurston geometries, the transformations of the viewing pipeline can be defined in such space by real projective mappings using homogeneous coordinates in $\mathbb{P}^3$. This scheme has been adopted as a standard in most graphics systems (in Euclidean space) because it unifies all the transformations involved using $4 \times 4$ projective matrices. In that way, the 3D objects are immersed in projective space, transformed and then, projected to a 2D image space.

This particular way in which the viewing pipeline is defined opens up the possibility to render views of ambient spaces different than the Euclidean space. For example, Weeks (2002b) describes the above pipeline in detail for the spherical and hyperbolic geometries.

### 2.1.2   Inside Views in Non-Euclidean Spaces

With a few exceptions, most 3D visualization software support only rendering of scenes in the Euclidean space $\mathbb{E}^3$. However, exploiting the fact that the visu-

alization process resorts to projective transformations in order to create images of three-dimensional spaces, it is possible to render different model geometries without structural changes to the visualization algorithm, by using the appropriate transformations in the viewing pipeline.

Thus, besides the Euclidean space $\mathbb{E}^3$, other model geometries, such as spherical and hyperbolic, can be rendered by the above visualization algorithms without difficulty. The specialization required in the pipeline amounts to taking care of the correct transformations that respect the intrinsic metric of the model geometry, as discussed previously.

The isometries for a given model geometry define the set of model transformations in the viewing pipeline. They are combined with the camera and perspective transformations, which are defined by the view specification. Perspective transformations are examples of projective transformation which are presented in (Molnár 1997) for the eight Thurston geometries.

These transformations are subsumed as elements of $PGL(\mathbb{R}, 4)$, the projective general linear group, and represented as $4 \times 4$ transformations matrices in the visualization algorithm (see (Gunn 1993; Molnár 1997; Phillips and Gunn 1992)).

To render inside views of three-dimensional spaces, we generate images that would be seen by an observer (e.g. a camera) placed in that space. Light paths follow geodesics and reveal the geometry/topology of the space. Chapter 3 will discuss the visualization of 3-manifolds in more details.

### 2.1.3 Types of Algorithms

There are two main types of three-dimensional visualization algorithms. They can be classified into:

- Object space;

- Image space.

Object space algorithms apply the direct viewing transformation to points of the objects, while image space algorithms apply the inverse viewing transformation to rays originating from the camera and corresponding to image pixels.

The structure of these two types of algorithms is described by the pseudocodes:

---

**Algorithm 1:** Object-Space Visualization

---

    **for** each $o \in S$ **do**
       Map $o$ from scene to camera space
       **if** $o$ is visible **then**
          Project $o$ to image space
       **end if**
    **end for**

---

The object-space algorithm is widely adopted in Computer Graphics and is the one used in the OpenGL standard.

---

**Algorithm 2:** Image-Space Visualization

---

    **for** each pixel $p \in I$ **do**
       Generate a ray $r$ in camera space
       Transform $r$ to scene space
       Find the intersection $i(r)$ with visible object $o \in S$
       **if** $i(r) \neq \emptyset$ **then**
          Paint pixel
       **end if**
    **end for**

---

The image-space algorithm is the basis of *ray tracing* rendering methods. In these algorithms, rays are launched toward the pixels, and when they hit an object's surface the shading is computed. This is a very convenient technique to render inside views of Riemannian 3-manifolds, we just have to set the view frustum in the tangent space of the observer and use the exponential map to release the rays. The visualizations presented in this book are outputs of a generalization of the ray tracing algorithm to Riemannian geometry, see Chapter 3 for the details.

Note that object-space algorithms work at geometric precision and, in principle, must perform full evaluation when transforming objects in the scene, while image-space algorithms work at image resolution and may perform a lazy evaluation of transformations required by each ray.

Furthermore, the opposite nature of these two algorithms has a determinant impact on the complexity of the visualization process and on the strategies used to make them more efficient. This depends on various factors, such as scene and depth complexity, among others.

### 2.1.4   Rendering Acceleration

Rendering acceleration methods exploit some kind of coherence in different classes of scenes embedded in three-dimensional spaces.

In the case of object space rendering for immersive views of 3-manifolds (modeled by the classical geometries), transformed copies of the objects to be drawn need to be computed first because these algorithms operate in the universal covering of the manifold. Thus we have to tessellate the universal covering space, which is the considered model geometry, by copies of the scene using the underlying discrete group. As a result, object-space algorithms generate many occluded objects which may not be needed, because they are not visible. Therefore, object-space acceleration techniques, such as occlusion culling do not help much, because of the burden of generating and transforming the objects in the scene.

For this reason, attempts to adapt traditional object-space acceleration methods, or even develop new ones, will not be very effective. In most cases, the limited efficiency gains will not justify the complexity of implementing such methods. Perhaps, this is the reason that most existing object-space algorithms rely mainly on the GPU acceleration of OpenGL.

On the other hand, image-space algorithms for inside views of 3-manifolds take full advantage of the classical ray tracing acceleration methods based on space subdivision (see (Arvo and Kirk 1989)). This is because to render inside views of a scene embedded in a 3-manifold (modeled by one of Thurston geometries), we can define the scene inside the fundamental domain of the manifold and launch the rays from the observer. Each time a ray intersects a face of the fundamental domain, we update it using the discrete group. Thus, these methods are naturally incorporated into the algorithm and do not require additional memory. Furthermore, they can take full advantage of modern programmable GPU's.

We will return to this point further in Chapter 3, regarding the immersive visualizations of Riemannian 3-manifolds using an image-space algorithm.

## 2.2   GPU Ray Tracing using RTX / Falcor 3.2.1

Computer Graphics history has several examples of important hardware milestones. They changed the way real time algorithms could be designed and implemented and created vast opportunities for advances in research.

One aspect of graphics cards that has been advancing consistently in the last decades is shader flexibility. In the beginning we had graphics libraries using a fixed rendering pipeline, which could only receive data and instructions from

the CPU. No GPU side programming could be done at that time. This aspect was changed later, with the advent of programmable shaders. Vertex and Pixel Shaders were introduced, creating a revolution in the possibilities for real time graphics. Later on, those capabilities were increased with the exposition of more programmable rendering stages (Fernando R. 2004). Applications could implement Tesselation and Geometry Shaders to have access to customizable geometry resolution and primitive connectivity.

Recently, the same approach was used to create a solution for Real time Global Illumination. The so-called RTX platform can produce faithful images using Ray Tracing (RT), which is historically known to have prohibitive performance for real time applications. This landmark creates interesting opportunities for new visualization applications. In particular, content makers for Virtual Reality (VR) can greatly benefit from the added realism to create immersive, meaningful experiences.

Thus, the demand for a VR/RT integrated solution is clear. However, realistic VR needs stereo images for parallax sensation. The obvious consequence is a duplication of the performance hit caused by ray tracing. A good algorithm should balance performance and image quality, something that can be done using RTX Ray Tracing and a proper trace policy. The recent announcement of ray tracing support for older architectures (Burnes 2019) emphasizes even more the necessity of a flexible algorithm for such task. Another point that must be taken into consideration is stereo camera registration. Depending on how the ray directions are calculated based on camera parameters, the stereo images may diverge when seen in a head mounted display (HMD).

### 2.2.1   RTX Ray Tracing

Historically, GPUs process data in a predefined rendering pipeline, which has several programmable and fixed processing stages. The main idea is to start with a group of stages that process the vertices, feeding a fixed Rasterizer, which in its turn generates data for pixel processing in another stage group. Finally, the result image is output by the final fixed stage.

Currently, programmable shaders are very flexible in essence. The Vertex Shader works on the input vertices, using transformation matrices to map them to other spaces. The Hull, Tesselator and Domain Shaders subdivide geometry and add detail inside graphics memory, optimizing performance. The Geometry Shader processes primitives and mesh connectivity, possibly creating new primitives in the process. The Fragment Shader works on the pixels coming from the

Rasterizer and the Output stage outputs the resulting image. Figure 2.2 shows the rendering pipeline in detail.



Figure 2.2: Rendering pipeline in depth. The gray boxes are programmable shaders and the black boxes are fixed stages. Adapted from (Wyman and Marrs 2019).

NVidia RTX is a hardware and 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, the memory to transfer data between them and the Acceleration Structures for fast geometry culling when intersecting rays and 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. On one hand, the bottom level Acceleration Structure contains the rendering primitives (triangles for example). On the other hand, the top level Acceleration Structure is a hierarchical grouping of bottom level ones. Finally, the GPU role is to run instances of the ray tracing shaders in parallel. This is similar to the well-established rasterization rendering pipeline.

The ray tracing GPU device code runs in a pipeline to the rasterization scheme discussed previously. The differences are the stages taken. The goal of the first stages is to generate the rays. Afterwards, a fixed intersection stage calculates the intersection of the rays with the scene geometry. Then, the intersection points are reported to the group of shading stages. Notice that more rays can be created at this point, resulting in a recursion in the pipeline. The final fixed stage outputs the generated image.

The details of the pipeline are as follows. A Ray Generation Shader is responsible for creating the rays, which are defined by their origins, directions and payloads (custom user-defined data). A call to TraceRay() launches a ray. The next stage is a fixed traversal of the Acceleration Structure, which is defined by the CPU host code beforehand. The Acceleration Traversal uses an Intersection Shader to calculate the intersections. All hits found pass by tests to verify if they are the closest hit or if they must be ignored because of transparent material. In

case a transparent material is detected, the Any-Hit Shader is called for all hits so shading can be accumulated, for example. After no additional hits are found, the Closest-Hit Shader is called for the closest intersection point. In case no hits are found, the Miss Shader is called as a fallback case. It is important to note that additional rays can be launched in the Closest-Hit and Miss shaders. Figure 2.3 shows an in-depth pipeline scheme. More detailed information about RTX Ray Tracing can be seen in (Wyman and Marrs 2019) and applications can be found in (Haines and Akenine-Möller 2019b).
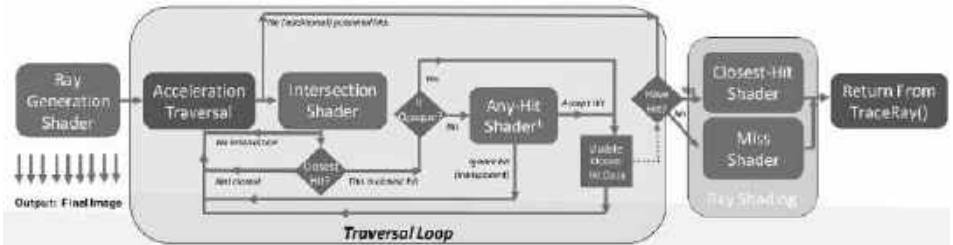


Figure 2.3: Ray tracing pipeline in depth. Fixed stages are in light gray and programmable shaders are in dark grey. Modified from (Wyman and Marrs 2019).

## 2.2.2 Falcor 3.2.1

RTX Ray Tracing can be accessed in four ways. On one hand there are the low level APIs Vulkan (Bailey 2018) (Sellers and Kessenich 2016), DirectX 12 (Luna 2016) (Wyman and Marrs 2019) and OptiX (Parker et al. 2010). They provide more flexibility but less productivity. On the other hand there is Falcor (Benty et al. 2018b), an open-source real-time rendering framework designed specifically for rapid prototyping. It has support for ray tracing shaders and is the recommended way to use RTX Ray Tracing in a scientific environment.

Falcor code and installation instructions can be found at Github [1] (ibid.). It is important to use version 3.2.1 because version 4.0 onward do not support VR anymore. All mentions to Falcor from now on refer to this version. The bundle comes with a Visual Studio solution, structured in two main components: a library project called Falcor with high level components and Sample projects which use those components to perform computations, effects or to provide tools for other supportive purposes.

---

[1]https://github.com/NVIDIAGameWorks/Falcor

Each Sample project consists at least of a main class inheriting from Renderer and a Data folder. The Renderer class defines several relevant callbacks which can be overridden as necessary. Examples include onLoad(), onFrameRender(), onGuiRender(), onMouseEvent() and so forth. The Data folder is where non-C++ files necessary for the Sample (usually HLSL Shaders) should be placed. Falcor automatically copies them at compilation time to the binary's folder so programs have no access problems.

## 2.3 Ray Tracing and Stereo Rendering

Our goal is to build a stereo-and-ray-tracing-capable renderer for VR. For this purpose, we will exploit the functionalities of Falcor that provide support for Stereo Rendering and Simple Ray Tracing.

Falcor is designed to abstract scene and Acceleration Structure setup so our focus will be on describing Shader code and the CPU host code to set it up. The next subsections explain the logic for two Falcor Samples with the objective of using their components later on as building blocks for our new renderer. We refer to code in the Falcor Samples, so it is advisable to access it in conjunction with this section for a better understanding.

### 2.3.1 Simple Ray Tracer

HelloDXR is a simple ray tracer with support for mirror reflections and shadows only. As would be expected, the Sample specifies two ray types: primary and shadow.

The ray generation shader rayGen() is responsible of converting pixel coordinates to ray directions. This is done by a transformation to normalized device coordinates followed by another transformation using the inverse view matrix and the camera field of view. The function TraceRay() is used to launch the rays. The ray type index and the payload are provided as parameters.

The Closest-Hit Shader primaryClosestHit () computes the pixel final color. It has two components: an indirect reflection color and a direct color. The reflection color is calculated by getReflectionColor (), which reflects the ray direction using the surface normal and shoots an additional ray in that direction. The payload has a ray depth value used to limit recursion. The direct color is the sum of the contributions of each light source at the pixel, conditioned by the shadow check checkLightHit (). If the light source is not occluded, the contribution is calculated by evalMaterial (), a Falcor built-in function to shade pixels based on materials.

### 2.3.2   Stereo Rendering

The StereoRendering Sample is an application to render stereo image pairs using the rasterization algorithm. The CPU host code ensures connection with the HMD (initVR()), issues the Shaders to generate the images and transfers them to the device (submitStereo()). Specifically, it maintains a struct containing the camera matrices and properties for both eyes. The geometry is drawn once, but it is duplicated inside the GPU by the Shaders. A frame buffer array with two elements is maintained for that purpose (mVrFbo). When a frame finishes, each array slice has the view of one eye.

The GPU code consists of a Vertex Shader, a Geometry Shader and a Pixel Shader. The Vertex Shader (StereoRendering.vs.hlsl) just passes ahead the vertex positions in world coordinates and additional rendering info such as normals, colors, bitangent, texture and light map coordinates, if available.

The projection is left to the Geometry Shader (StereoRendering.gs.hlsl), which is also responsible for duplicating the geometry. It receives as input three vertices of a triangle and outputs six vertices. Each input vertex is projected twice, once for each of the view-projection matrices available at the camera struct. The geometry for each eye is output into the related array slice by setting a render target index at struct GeometryOut.

Finally, the Pixel Shader (StereoRendering.ps.hlsl) samples material data using the built-in function prepareShadingData() and accumulates the contributions of each light source using the built-in function evalMaterial().

## 2.4   Integrating Ray Tracing and VR

Now we have the tools to develop a new renderer that combines the capabilities described in the previous section. It should be capable of stereo rendering and ray tracing in real time. This section describes the process and the possible choices and alternatives to address the problems encountered.

### 2.4.1   Stereo Convergence

One key problem of integrating VR and RT is the stereo image registration. Depending on how this process is done, the images may diverge and it can be impossible for the human vision to focus correctly on the scene objects. This phenomenon may result in viewer discomfort or sickness.

To understand the ray generation process it is good to think about perspective projection and the several related spaces it involves. (Pharr, Jakob, and Humphreys 2016b) contains an exceptional explanation of this topic, which will be summarized here.

Conceptually, the process consists of a chain of transformations starting at the world space, passing through the camera space and the normalized device coordinate space and ending at the raster space. The camera space is the world space with a translated origin to the camera position. The normalized device coordinate space is the camera space with the near and far planes transformed. The near plane is at the square with top-left corner at (0,0,0) and bottom-right corner at (1,1,0) and the far plane is at the square with top-left corner at (0,0,1) and bottom-right corner at (1,1,1). Finally, the raster space is the normalized device coordinate space scaled by the image resolution. Figure 2.4 shows how the spaces relate to each other.
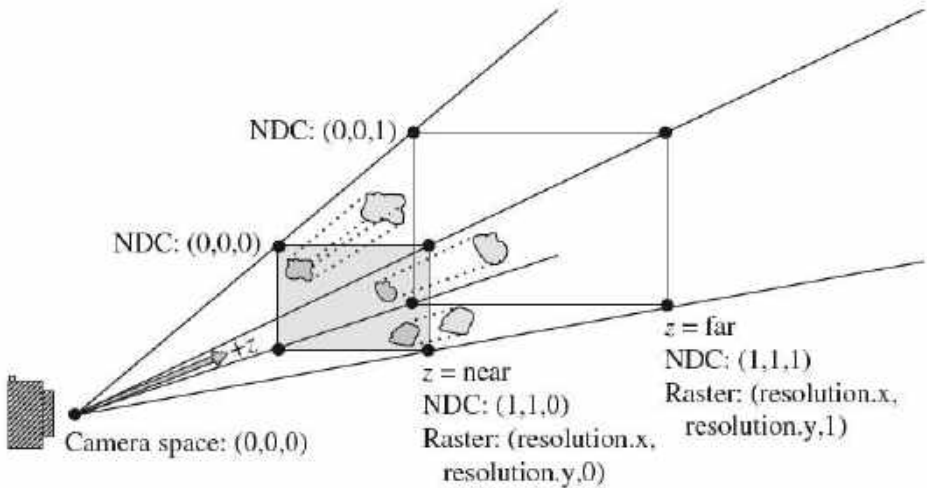


Figure 2.4: Several camera-related coordinate spaces: camera, normalized device coordinates and raster. As in (Pharr, Jakob, and Humphreys 2016b).

The transformation for a projection camera can be constructed in two steps. First, building a canonical perspective matrix with distance to the near plane $n$ and distance to the far plane $f$. The projected coordinates $x'$ and $y'$ are equal to the original ones divided by the $z$ coordinate. $z'$ is remapped so the values in the near plane have $z' = 0$ and the values in the far plane have $z' = 1$:

$$x' = \frac{x}{z}, \quad y' = \frac{y}{z}, \quad z' = \frac{f(z-n)}{z(f-n)}.$$

This operation can be encoded as a $4 \times 4$ matrix using homogeneous coordinates:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & \frac{-f \cdot n}{(f-n)} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

As a side note, the original position of the projection planes would be important for rasterization because the map of $z$ to $z'$ is not linear, what could result in numerical issues at depth test, for example. However, we are only interested in the projection directions for ray tracing, thus those distances can be totally arbitrary.

The second step is scaling the matrix so points inside the field of view project map to coordinates between $[-1, 1]$ on the view plane. For square images, both $x$ and $y$ lie between the expected interval after projection. Otherwise, the direction in which the image is narrower maps correctly, and the wider direction maps to a proportionally larger range of screen space values. The scaling factor that maps the wider direction to the range $[-1, 1]$ can be computed using the tangent of half of the field of view angle. More precisely, it is equal to $1/\tan(\frac{fov}{2})$, as can be seen in Figure 2.5.

To launch rays from pixels we use the inverse transformation chain. We start at the raster space, passing through the normalized device coordinate and camera spaces and ending at the world space. More specifically, to compute a ray direction we must convert the raster coordinates of its associated pixel to normalized device coordinates, scale by the reciprocal of the factor used to map the field of view to the range $[-1, 1]$ and use the inverse view transformation matrix to map the result to the world space. The conversion from raster coordinates $r \in [d_x, d_y]$ to normalized device coordinates $n \in [-1, 1]$, given the image dimensions $d = (d_x, d_y)$, is expressed by the following equation:

$$n = \frac{2(r + 0.5)}{d} - 1;$$

which is composed of a normalization by the image dimensions and operations to map the resulting image space from the interval $[0, 1]$ to $[-1, 1]$.

The remains of the transformation chain can be done in a optimized way, using a precomputed tangent of half the field of view angle $f$ (in dimension $y$), the
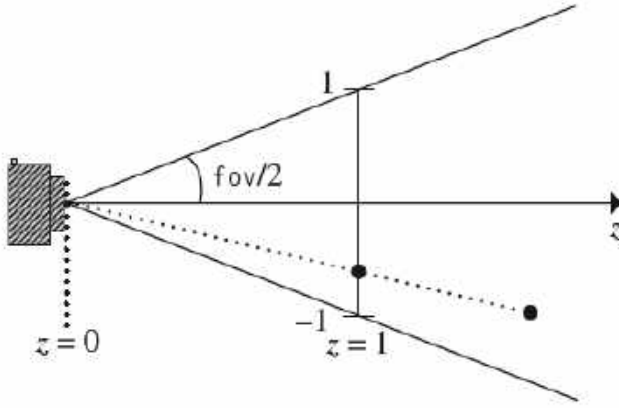
Figure 2.5: Relation between a field of view and normalized device coordinates. As in (Pharr, Jakob, and Humphreys 2016b).

aspect ratio $a$ and the basis vectors of the inverse view matrix $I = [u|v|w]$. The operation is done by the expression:

$$normalize(af(n_x u) - (f(n_y v)) - w),$$

As can be seen, this expression transforms the normalized device coordinates using the parts of the inverse view matrix that would affect each of the coordinates and scales them using the tangent of the field of view for dimension $y$. The scale value is corrected by the aspect ratio for the $x$ dimension.

In our tests we could not generate correctly registered stereo images using this optimized expression, because it does not take into account stereo rendering. For this reason, we used two other approaches: the inverse of the projection matrix, and a rasterization G-Buffer prepass. Both options ensure correct stereo images, with different pros and cons. The first option does not need any additional raster-ization pass or memory for the required texture. However, the G-Buffer provides more flexibility for the algorithm as will be discussed in Section 2.4.2. It is impor-tant to note that the positions in the texture are equivalent to intersection points of rays launched from the camera position. This property comes from the fact that the camera position is equivalent to the projection center and each ray is equivalent to the projection line for the associated pixel.

## 2.4.2    Ray Tracing Overhead

The major drawback of usual stereo rendering is the overhead caused by the additional camera image. This problem is emphasized even more in ray tracing, which demands heavy computation to generate the images. Several techniques have been proposed to address this issue. They usually create the additional image by transforming the contents of the original or by temporal coherence using previous frames. However, artifacts not present when the scene is rendered twice can be introduced in the process.

The RTX platform opens new ways to explore this problem. Additionally, the extension of the ray tracing support for older graphics card architectures (Burnes 2019) encourages new algorithms based on smart ray usage. We benefit from Falcor's design to explore and evaluate the possibilities using a methodology based on fast cycles of research, prototyping, integration and evaluation. The result is a list of several possible approaches, generated by changing component routines of a ray tracing algorithm. In summary, those changes result from the following questions.

1. How the first intersections are calculated?

    (a) Rays shot from camera position.
    (b) G-Buffer prepass.

2. Which effects are applied?

    (a) Direct light and shadows only.
    (b) Perfect-mirror specular reflections and shadows only.

The different algorithms are created by combining the different functionalities described in Section 2.3. We start by integrating Simple Ray Tracing and Stereo Rendering. On one hand, Stereo Rendering includes all logic to connect with the HMD and to control the data flow between the ray tracing shaders and the device. On the other hand, Simple Ray Tracing features a ray tracing shader, which is modified to launch rays based on two view matrices or two position G-Buffers, one for each eye. On the G-Buffer case a rasterization prepass is also performed.

We benefit from Falcor's RenderGraph, which is extremely useful for algorithms with multiple rendering passes. The changes needed are listed next.

1. Adding an additional mirror ray type, equivalent to the primary ray type.

2. Including a function to compute direct light with shadows only. If the G-Buffer is available, the direct contribution comes for free from it.

3. Adding a branch in the Ray Generation Shader to choose between the effects: raster, direct light plus shadows, specular reflections.

An interesting question arises when we analyse the current algorithm. A ray tracing procedure with a G-Buffer prepass is actually a hybrid algorithm based on both rasterization and ray tracing. What if we extrapolate this hybrid paradigm and allow materials to be raster or ray-traced in the scene? This question generated an additional change in the integrated renderer. We introduced a material ID to enable support for per-material effect selection. With this feature, an user can control performance by changing the material IDs of objects in the scene from more complex to simpler effects and vice versa. The final algorithm is very flexible and suited for stereo ray tracing or for older graphics card architectures, environments where performance matters.

<div style="font-size:4em; color:#F5C99B;">3</div>

# *Riemannian Ray Tracing*

## 3.1 Core Concepts

To visualize Riemannian manifolds it is important to understand two related core concepts: geodesics and fundamental domains. They are related to the manifold geometry and topology, respectively. This section gives an informal intuition for both, preparing the reader for details in forthcoming sections.

### 3.1.1 Geodesics and Fundamental Domain

Geodesics are the equivalent to straight lines in the Euclidean space: paths locally minimizing lengths. They are a consequence of the manifold geometry, causing the perception of deformation that a viewer has when exploring the space. Figure 3.1 shows a geodesic $\gamma(t)$ in a Riemannian manifold $M$. It passes through a point $p$ in a tangent direction $v$ lying in the tangent space $T_p M$: the set of all possible directions for geodesics passing through $p$. Note the parametric notation for the geodesic, such as a line in Euclidean space.

The fundamental domain is a "polyhedron" such that a special face identification describes the topology of the manifold. Remember that in dimension 2, for example, a 2D Torus is obtained by identifying the opposing edges of a square,
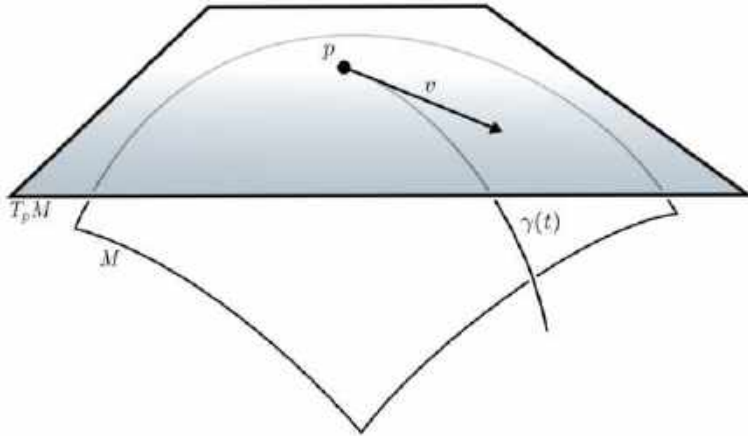
Figure 3.1: We illustrate the manifold $M$ and the tangent space $T_p M$ at the point $p$ as two-dimensional objects. The geodesic $\gamma(t)$ starts at $p$ in the direction $v$.

which is the fundamental domain. Embedding this square in $\mathbb{R}^2$ and "interpreting" the identifications as translations we get a tessellation of $\mathbb{R}^2$ by squares, which is called the covering space of the torus. Figure 3.2 shows the 2D Torus in the covering space.

## 3.1.2 Non-Euclidean Ray tracing

Ray tracing (Whitted 1980) consists of following the path traveled by light in the space to evaluate the contribution of light sources to pixels in an image. The classic algorithm assumes linear paths in Euclidean space.

Novello, V. da Silva, and Velho (2020e) proposed a more general approach by abstracting how light travels through space. Instead of following linear rays, the former now follows geodesics imposed by the latter. In other words, it suffices to know the geodesics of a space to ray trace it. Note that the Euclidean space is a particular case in this model, where the geodesics are lines.

A Riemannian manifold can be visualized using ray tracing because its geodesics propagate on the fundamental domain deforming and tessellating the manifold covering space. Figure 3.2 shows a ray that exits and enters the fundamental domain as it travels the covering space.
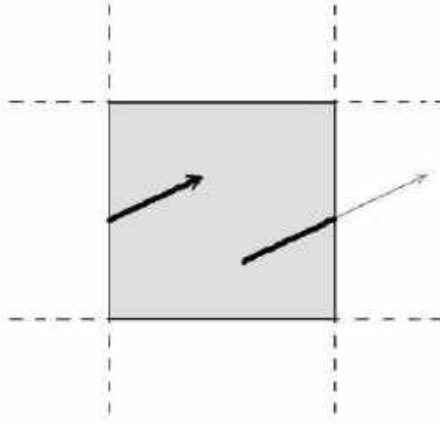
Figure 3.2: 2D torus. Each edge of the square (fundamental domain) is identified with the opposing one. Because of that, everything inside the square is replicated in the entire $\mathbb{R}^2$ (covering space). Image from (Berger, Laier, and Velho 2014).

### 3.1.3   Riemannian Manifolds

To motivate the mathematical definitions used in this work we remember that it deals with an immersive visualization of curved spaces — rays may not be straight lines — using ray tracing. Then the space must satisfy at least three properties:

- Being locally similar to a Euclidean space, a *manifold*. This allows us to model the viewer/scene inside the space;

- To simulate effects produced between the lights and the scene we need the *tangent space* with a *scalar product* at each point, a *Riemannian metric*;

- The *ray* leaving a point in any direction; the *geodesic*. The ray-object intersections are also required.

*Riemannian manifolds* carry the above properties. We are interested in a specific class of them — the model geometries, presented in Section 1.4. These spaces have simple topology (*simply connected* and *complete*) and each pair of points admits neighborhoods with similar geometry (*homogeneous spaces*). In dimension two, there are exactly three models: Euclidean, hyperbolic, and spherical. Chapter 4 will present inside views of three-dimensional spaces modeled by these geometries. In dimension three, there are five more model geometries. We explore three of

them: *Nil*, *Sol*, and $\widetilde{SL_2(\mathbb{R})}$); these were defined in Section 1.4.3. Visualizations of these spaces will be presented in Chapter 5. The remaining geometries are $\mathbb{S}^2 \times \mathbb{R}$ and $\mathbb{H}^2 \times \mathbb{R}$ with the product metric. Since they are simpler and model fewer manifolds, we opt to omit them here.

The model geometries in dimension 3 were described in Section 1.4. Again, these spaces date back to the *geometrization conjecture* which states that every compact three-dimensional manifold decomposes into pieces with the geometry modeled by Thurston geometries. Remember that a manifold $N$ has its geometry modeled by a model geometry $M$ if it can be expressed as a quotient $N = M/\Gamma$, where $\Gamma$ is a discrete group of isometries acting on $M$. The manifold $N$ inherits the metric of $M$ — and it is called a *geometric manifold*. The *fundamental domain* of $N$ is the region in $M$ containing a point for each orbit (the images of a single point under the action of $\Gamma$). Great texts on this subject are (Martelli 2016; Thurston 1979).

## 3.2 Visualization of Riemannian manifolds

There are two obstacles when visualizing a manifold. First, it is necessary to compute *rays*, since light propagates along them. The *geodesic flow* is perfect for that, providing a ray for each point and direction. The second obstacle is the dimension; our eyes only see up to dimension three.

### 3.2.1 Visualization approaches

Two-dimensional manifolds can be visualized using two approaches: *extrinsic* or *intrinsic*. In the extrinsic setting, we consider the manifold $M$ embedded in a higher-dimensional space, typically $\mathbb{R}^3$. In this way, $M$ can be seen from the *outside* using a virtual camera. In the intrinsic setting, we consider the manifold as an abstract independent space. Therefore, $M$ can only be seen from the *inside*. It is also possible to visualize the *universal covering*, i.e., the tessellation produced by the group action through views of the covering space itself. In any case, these visualizations can be based on classical CG algorithms: *rasterization* or *ray tracing*.

The problem of visualizing 3-manifolds is harder. In 1998, Thurston published *How to see 3-manifolds* (Thurston 1998), discussing ways to "see" a 3-manifold using our spatial imagination and computer aid. Many tools in Riemannian 3-manifold are inspired in human mind geometrical instincts. Thus, the human mind

is trained to understand the kinds of geometry that are needed to model certain 3-manifolds.

Since higher-dimensional manifolds "can not" be used to visualize 3-manifolds, we take an immersive approach based on a *ray tracing* algorithm. It follows the ideas from (Berger, Laier, and Velho 2014). Rasterization is not appropriate for this scenario because perspective projection in Non-Euclidean spaces is nontrivial. On the other hand, a scene in a 3-manifold can be ray traced: given a point (eye) and a direction (pixel), we trace a geodesic (ray). When it hits an object we compute its *shading*.

### 3.2.2   Riemannian ray tracing

In computer graphics, *Shading* is the process of assigning a color to a pixel. Classic approaches to perform such tasks are not suited for Riemannian manifolds. Thus, we propose a more general definition for ray tracing Riemannian geometry.

Consider a point $p$ (the eye) in a 3-manifold $M$, and also consider the set of directions within the observer's field of view $V_p$ (the view frustum). We compute a color by tracing rays in the directions associated with image pixels. Specifically, $p$ is the origin of $T_pM$, and for each direction $v \in V_p$ we attribute a color $c$ by launching a ray $\gamma(t)$ from $p$ in the direction $v$. Each time $\gamma$ intersects a visible object at a point $q$ we define an RGB color. Therefore, we have $c : V_p \to \mathcal{C}$, where $\mathcal{C}$ is a color space. We call this procedure *Riemannian shading*. Figure 3.3 presents a schematic view of the above procedure in dimension two.

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. For a given point $p$ in $M$ we compute the Riemannian shading $c : V_p \to \mathcal{C}$ using ray tracing and the Riemannian illumination.

In classic approaches, ray tracing (Whitted 1980) approximates physical illumination. Our ray tracing model for Riemannian 3-manifolds can be also used to compute a Riemannian shading function for local or global illumination.

Depending on what we want to see, we can take a mathematical visualization where we consider pseudocolor based either on properties of the space, such as
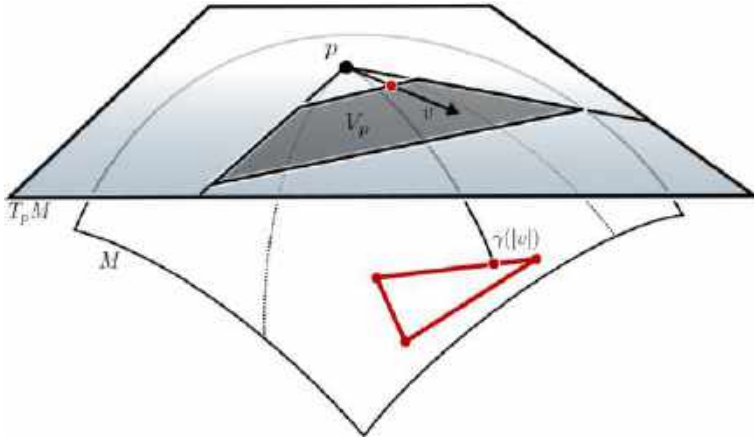
Figure 3.3: Shading in a Riemannian manifold $M$. Let $p$ be the observer and $V_p$ be the view frustum (gray region) in the tangent space $T_p M$. We launch a ray $\gamma$ towards each vector $v \in V_p$. If $\gamma$ hits a visible object (red triangle) in $\gamma(|v|)$ we define a RGB color (red) for the corresponding point in the near plane of $V_p$.

curvature, or attributes of the objects, such as surface normal, to define the Riemannian shading function.

## 3.3   Ray tracing in Riemannian manifolds

We provide an algorithm to compute a Riemannian shading that synthesizes inside views of Non-Euclidean spaces. This generalizes the classical *ray tracing*. In Chapters 4 and 5, we will apply this algorithm to visualize the classical and "twisted" product geometries, respectively.

We focus on a geometric manifold $M/\Gamma$ since the rays in the model geometry $M$ are well behaved and its topology is given by $\Gamma$. Our algorithm computes a Riemannian shading of the visible surfaces. We discuss the basic principles of ray tracing in these manifolds, as well as the general algorithm in CPU. Section 3.3.4 shows the map of the computation to the RTX pipeline.

### 3.3.1   Overview of the Method

Ray tracing is a natural method to visualize 3-manifolds. It is necessary to adapt the traditional ray tracing to consider the geometry/topology of the space. The first

aspect of this task is to simulate the ray path as it travels inside the space. The second aspect amounts to a shading procedure, which computes the illumination and evaluates the light scattered from the environment in the ray direction. Because of the topology, the ray path is updated as it exits the fundamental domain.

### 3.3.2    Algorithm in CPU

Algorithm 3 belongs to the class of image-space algorithms (Algorithm 2) and presents the basic Ray tracing procedure for a geometric manifold $M/\Gamma$. The rays are generated from the observer's point of view (lines 1-4), intersected with visible objects (line 6) and if there is a hit (line 7), shading is done (line 8). Because of practical computational reasons, we cannot continue the ray path indefinitely and set the maximum recursion level (line 13).

These steps are common to any ray tracing algorithm, including the traditional one for the Euclidean space. To trace rays in a geometric manifold $M/\Gamma$, we need extra steps to guide the ray as it iterates in the fundamental domain (lines 9-11).

When $\Gamma$ is the empty set, the ray tracing algorithm coincides with the classical (line 9). In this case, the trace ray function (line 4) will trace curved rays. The most important and critical step is the group action (line 11), which depends on the geometry/topology of the manifold. It is specific for each space type.

---

**Algorithm 3:** Ray tracing a Riemannian manifold $M/\Gamma$

---

1:  **for** each pixel $\sigma \in I$ **do**
2:      Let $p := 0$ and $v$ be the direction associated to $\sigma$;
3:      Let $\Delta$ be the fundamental domain polyhedron;
4:      Trace a ray $\gamma$ from $(p, v)$ inside $\Delta$;
5:      **repeat**
6:          Find closest intersection $\gamma(t)$ with objects $O$ in $\Delta$;
7:          **if** $\gamma(t) \neq \emptyset$ **then**
8:              Shade pixel **break**;
9:          **else if** $\Gamma \neq \emptyset$ **then**
10:             Find intersection of $\gamma$ with faces of $\Delta$ ;
11:             Compute the new ray $(p', v')$, and $++i$ ;
12:         **end if**
13:     **until** $i \leqslant maxlevel$
14: **end for**

---

### 3.3.3 Ray Marching

To launch a ray we need to solve the geodesic flow and subsequently compute the intersection of the given ray with the scene objects. In general, our rays will be *curved rays* that fall into two cases: i) for a few manifolds the flow admits a closed solution resulting in a parametric curve $\gamma(t)$; ii) for most manifolds, however, the geodesic flow does not have a closed-form solution and we have resort to numerical integration methods.

In case i) we generate a polygonal curve by discretizing the parameter $t$, such that $\gamma(t) \approx \{p_i\}$. In case ii) we use Euler's method for integration in the parameterization image, since there we use the Euclidean metric.

The Euler's numerical integration method approximates a ray $\gamma$ starting at $p$ in the direction $v$ by a polygonal $\{p_i\}$, where:

$$\begin{cases} p_{i+1} = p_i + h \cdot \widetilde{\gamma}'(0) \\ v_{i+1} = v_i + h \cdot \widetilde{\gamma}''(0) \end{cases} \tag{3.1}$$

where $h$ is the integration step and $\widetilde{\gamma}(t)$ is the ray satisfying $\widetilde{\gamma}(0) = p_i$ and $\widetilde{\gamma}'(0) = v_i$. We use the geodesic flow to compute $\widetilde{\gamma}''(0)$.

In any case, the polygonal curve $\{p_i\}$ is used to ray trace a scene in $(M, g)$. When $h \to \epsilon$ the scene is rendered with more accuracy. To compute the intersection between a ray and the scene objects we test the intersection of each segment of the polygonal approximation given by the above computation (i.e., ray marching).

The implementations considered in this book are testing the intersection with each segment sequentially. An interesting exercise would be using many segments of the polygonal approximation $\{p_i\}$ to increase the GPU occupancy and optimize the intersection computations.

The Runge–Kutta method could also be used to approximate solutions of the geodesic flow. Instead, we take the Euler method since it uses fewer computations giving GPU performance.

### 3.3.4 RTX Pipeline

NVidia RTX is a hardware and software platform with support for real-time ray tracing. The ray tracing code of an application using this architecture consists of CPU host code and GPU device code.

The ray tracing GPU device code runs under a pipeline scheme composed of a sequence of stages specifically designed for ray tracing operations. The goal of the first stages is to generate the rays. Afterwards, a fixed stage calculates the

intersection of the rays with the scene geometry. Then, the intersection points are reported to the group of shading stages. Notice that more rays can be created at this point, resulting in a recursion in the pipeline. The final fixed stage outputs the generated image.

Each shader can be correlated with the tasks performed by the general CPU procedure described in Algorithm 3. The *Ray Generation Shader* is responsible for creating the rays (line 1), which are defined by their origins, directions and the custom user-defined data, called payloads (line 2). A call to TraceRay() launches a ray (line 3). The next stage is a fixed traversal of the Acceleration Structure which will describe only at high level here. This traversal uses an *Intersection Shader* to calculate the intersections (line 5). All hits found pass by tests to verify if they are the closest hit. After no additional hits are found, the *Closest-Hit Shader* is called for the closest intersection point (line 7). In case no hits are found, the Miss Shader is called as a fallback case. It is important to note that additional rays can be launched in the Closest-Hit and Miss shaders.

Figure 3.4 shows a simplified scheme of the pipeline, where the association of pipeline stages with the steps of the algorithm are indicated by the line numbers. More detailed information about RTX Ray Tracing can be seen in (Wyman and Marrs 2019) and applications can be found in (Haines and Akenine-Möller 2019a).
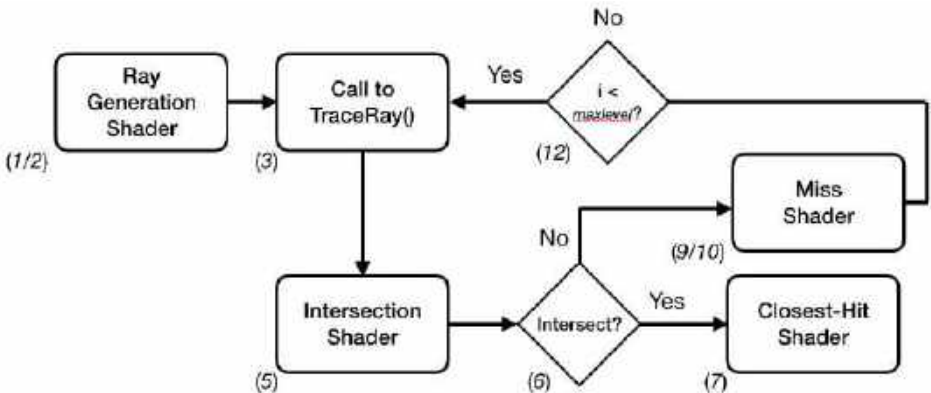


Figure 3.4: Ray Tracing Pipeline - main stages of the RTX GPU computation flow (the numerical labels correspond to line numbers of Algorithm 1).

The above is the general ray tracing GPU pipeline. In the case of ray tracing inside a manifold/orbifold we have two classes of objects: i) the *scene objects* which are embedded in the space; and ii) the *boundary of the fundamental domain* that is

represented by the polyhedron $\Delta$. They 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 7), the boundary of the fundamental domain is used to transport the rays by the group action (lines 9 and 10). This is implemented with a custom designed Miss Shader.

Another important point is related to the Acceleration Structure. The RTX platform defines a hierarchical structure in order to efficiently guide the intersection of rays with scene objects. Bottom-level cells store the actual scene geometry while top-level cells hold pointers of the graph structure. In the diagram of Figure 3.4, this is encapsulated by the block for ray intersection (lines 5 and 6). Note however, that in the algorithm for visualization of manifolds/orbifolds, rays travel through the covering space entering and exiting the fundamental domain multiple times, In that respect, the fundamental domain acts as an special higher-level acceleration structure that defines the topology of the space.

The above description makes clear that the RTX platform potentially opens up new research directions for ray tracing applications, with an impact similar to the introduction of programmable shaders. In particular, for the visualization of non-Euclidean spaces it allows non-trivial advances related to efficient and modular architectures for interactive and immersive exploration of scenes with complex geometry and topology, not possible until now.

### 3.3.5 GPU Implementation

The implementation of the visualization platform in GPU is build on top of Falcor using DirectX 12 on Windows 10. The Falcor development framework consists of a library with support for DXR at high level and a built-in scene description format.

We use the software Blender to create the scene objects and model the fundamental domains, including their boundaries.

The core functionality of our system's architecture consists of a set of shaders that are mapped to the RTX GPU pipeline as described above. In order to make the design of the system extensible and modular, we have adapted the metric neutral approach of (Guimarães, Mello, and Velho 2015) to ray tracing and extended it to 3D geometrical structures. In this context, we have developed generic shaders for each stage of the GPU ray tracing pipeline that are independent of the geometric structure of the manifold/orbifold. They are specialized and instanced based on the metric and topological properties of each individual space. That includes the model of the fundamental domain.

We now describe the tasks performed by the different shader classes, as well

as, the mathematical operations necessary for the visualization of non-Euclidean spaces. Note that these operations are dependent of the model geometry being used in the space.

**Ray Generation Shader**: Creates camera rays. For this purpose it has to use the isometries of the space to transform the ray origin and direction to the camera coordinate system.

**Intersection Shader**: Computes the intersection between the ray and the scene objects. For this purpose it uses the parametric description of the ray. Both the ray and objects are defined according to the model geometry.

**Closest Hit Shader**: Performs the shading operation. This includes computing the local and global illumination. The local illumination amounts to direct contribution of light sources that is based on angles between the light direction and the surface normal, as well as, the distance to the light. All these operations are performed using the model geometry.

**Miss Shader**: Deals with the transport of rays in the covering space, as they leave and enter the fundamental domain. For this, the rays are tested for intersection with the boundary of the polyhedron $\Delta$. Here, both the geometric and topological aspects of the embedding space have to be taken into account.

The distinction of scene objects and the fundamental domain is handled through a feature of Falcor's scene description, i.e., object and material ID's. These two types of entities have different ID's that causes the assignment of the appropriate specific shader classes. In this way, only objects in the scene are processed by the standard ray intersection operations, while the polyhedron representing the fundamental domain is processed only by ray-path propagation mechanism.

In addition, for the development of virtual reality applications, we employ and extend to Non-Euclidean spaces the Ray-VR algorithm (presented in Section 2.3) that implements stereo ray tracing on top of Falcor.

# 4

# *Visualization of Classical Non-Euclidean Spaces*

This chapter presents some expressive output images rendered using the implementation (given in Chapter 3) of the ray tracing algorithm in GPU that is build on top of the RTX platform. We consider examples of well-known 3-manifolds and orbifolds modeled by the Thurston classical geometries: Euclidean, spherical, and hyperbolic. For interesting visualizations using classical rasterization techniques, see Weeks (2002a).

Recall that 3-manifolds are topological spaces such that each point have a neighborhood that is homeomorphic to the Euclidean space. In other words, manifolds are abstract spaces locally similar to the Euclidean space. We present some classical well-known examples — Sections 4.1.1, 4.2.1 and 4.3.1 — of such spaces with their geometry modeled by the classical model geometries.

The rays in such spaces have a particular behavior that can be explained in two ways. Topologically, these space are not simply connected: their fundamental group is nontrivial. Then by *Cartan's theorem* (Carmo 1992), there is a closed ray for each nontrivial element in the fundamental group. Algebraically, these spaces are the quotient of the model geometries by some discrete groups, producing thus a tessellation view inside the model geometry. These arguments explain the multiple

copies of the scene in the examples below.

*Orbifolds* are modeled locally by quotients of a model geometry by discrete groups. Let $M$ be a Euclidean, hyperbolic, or spherical space. The quotient $M/\Gamma$ of $M$ by a discrete group acting on it could be a non-manifold. In this case, $M/\Gamma$ is called an orbifold.

We present two simple orbifold examples: the *mirrored cube*, and *mirrored dodecahedron* — Sections 4.1.2 and 4.3.2.

## 4.1  Euclidean

This section presents and illustrates two classical examples of three-dimensional spaces with their geometries modeled by the Euclidean geometry: the flat torus and the mirrored cube, which are examples of manifold and orbifold, respectively.

### 4.1.1  Flat Torus

Probably the most famous and easiest example of a compact 3-manifold is the *flat torus* $\mathbb{T}^3$. Topologically, this manifold is obtained by identifying, in the obvious way, opposite faces of the unit cube $[0, 1] \times [0, 1] \times [0, 1]$ in $\mathbb{E}^3$. It is trivial to verify that the neighborhood of each point in $\mathbb{T}^3$ is a 3-ball of the Euclidean space therefore homeomorphic to $\mathbb{E}^3$. Thus, the flat torus is indeed a 3-manifold.

The 3-manifold $\mathbb{T}^3$ admits a geometric structure modeled by $\mathbb{E}^3$ since it is also the quotient of the three-dimensional Euclidean space by the group of translation spanned by the isometries $(x, y, z) \rightarrow (x \pm 1, y, z)$, $(x, y, z) \rightarrow (x, y \pm 1, z)$, and $(x, y, z) \rightarrow (x, y, z \pm 1)$. Thus, the face $[0, 1] \times [0, 1] \times 0$ is identified to its opposite face $[0, 1] \times [0, 1] \times 1$ by the translation map $(x, y, z) \rightarrow (x, y, z + 1)$. The remaining two pairs of faces of the unit cube can be identified in an analogous way. The unit cube is clearly the fundamental domain of $\mathbb{T}^3$.

A ray leaving a point $p \in \mathbb{T}^3$ towards a tangent direction $v$ can be parameterized by the formula $r(t) = p + t \cdot v$ in $\mathbb{E}^3$. For each intersection between $r$ and a face $F$ of the unit cube, we update $p$ by its corresponding 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.

Therefore, we have all of the necessary ingredients for a simple immersive visualization of the 3-manifold $\mathbb{T}^3$. The scene objects can be set inside the unit cube because it is the fundamental domain of the manifold. A ray launched inside $\mathbb{T}^3$ can return to its starting point. Such behavior gives rise to many copies of the scene in the rendered image. The inside perception of the flat torus $\mathbb{T}^3$ is the

Euclidean space $\mathbb{E}^3$ tessellated by unit cubes, each cube containing one copy of the scene.

Figure 4.1 provides an immersive visualization of the 3-dimensional torus $\mathbb{T}^3$ using the shader described in Section 3.3.5. Besides the edges (in red, green, and blue) of the unit cube, there is only one monkey's head, the *Suzanne* classical Blender mesh, and a unique pair of hands composing the scene. We attach Suzanne's position and orientation to the camera. The closed rays produce many scene copies of the scene. Algebraically, this image describes the action of the group of translation in the Euclidean space which covers $\mathbb{T}^3$, explaining thus the copy pattern.
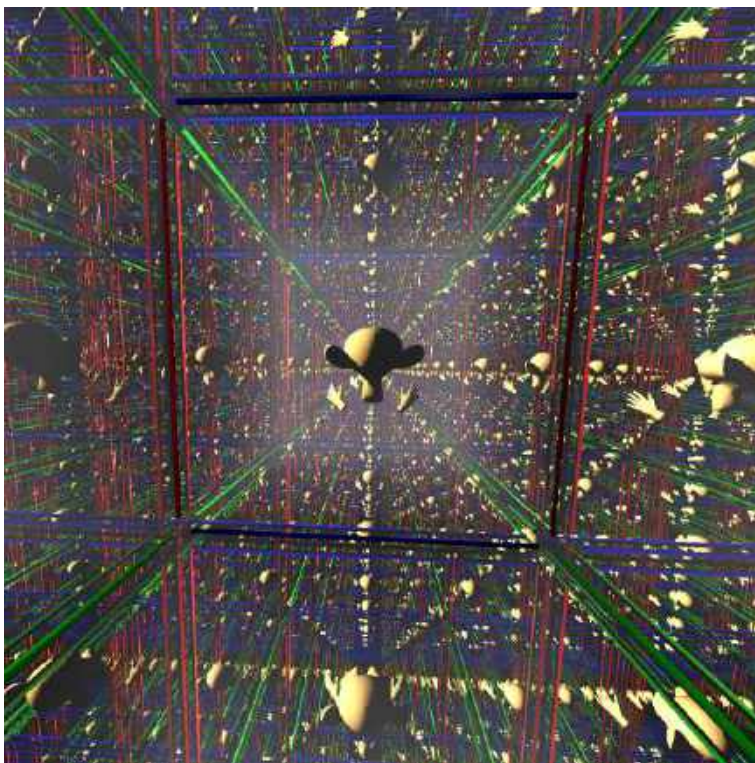


Figure 4.1: Immersive view in the 3-dimensional flat torus. The space is obtained by identifying the opposite faces of a cube (fundamental domain). We use the cube to set up our scene: a unique mesh (Suzanne) endowed with hands, and the cube's edges. The face pairing makes the rays that leave a face return from its opposite face, giving rise to many copies of the scene, tessellating the Euclidean space.

## 4.1.2    Mirrored Cube

The *mirrored cube* $\mathcal{Q}^3$ is an example of an orbifold with the geometric structure modeled by Euclidean geometry $\mathbb{E}^3$ through a special group of reflection $\Gamma$. Such group is generated by the reflections of the planes $x = \pm 1$, $y = \pm 1$, and $z = \pm 1$ in $\mathbb{E}^3$. The unit cube is the fundamental domain of $\mathcal{Q}^3$. Each time a ray $r$ intersects a face of the fundamental domain of $\mathcal{Q}^3$ it is reflected, creating a polygonal curve in $\mathcal{Q}^3$: exactly what happened with the lights in a mirrored room. These polygonal curves suspend to rays in $\mathbb{E}^3$, thus we see a tessellation of $\mathbb{E}^3$ by reflected unit cubes when immersed in $\mathcal{Q}^3$.
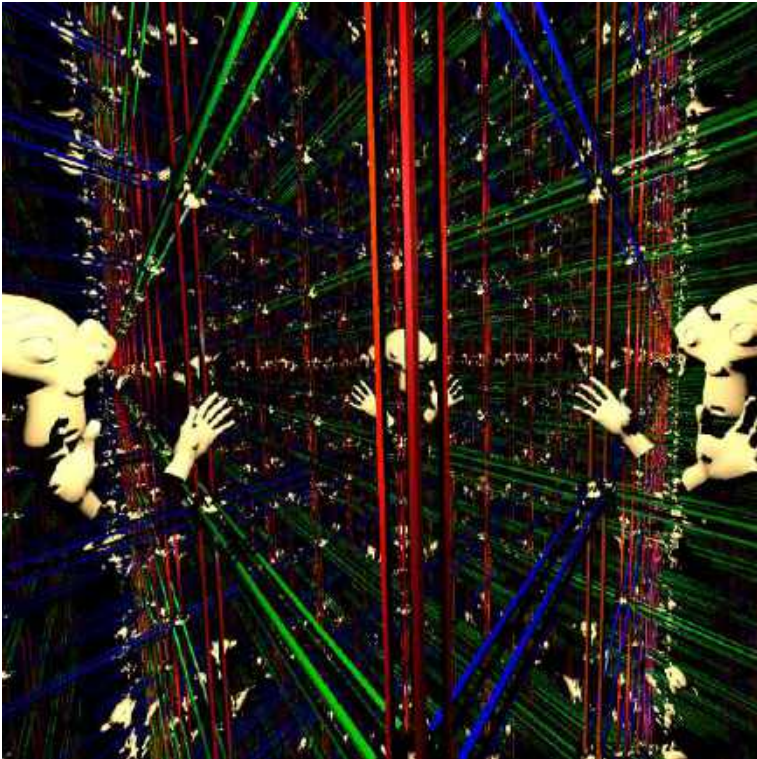


Figure 4.2: Immersive visualization of the mirrored cube, obtained by considering the faces of a regular cube to be perfect mirrors. A unique mesh (Suzanne) and the cube's edges provide the scene. The perfect mirrors make the rays iterate, producing the sensation of being inside a cube tessellation of Euclidean space.

Figure 4.2 gives an immersive visualization of the mirrored cube. Again, there is a single Suzanne in the scene attached to the camera. The image is the view of a group of reflection acting on the Euclidean space.

## 4.2 Spherical

### 4.2.1 Poincaré sphere

If the opposite faces of a dodecahedron are identified with a clockwise rotation of $\pi/5$ we obtain *Poincaré sphere*. This is a 3-manifold discovered by Poincaré which is also known as *Poincaré homological sphere* because its first homological group is trivial, but it is not homeomorphic to the 3-sphere.

The face pairing in the Poincaré sphere construction forces many identifications. The edges are grouped into ten groups of three edges. Then, to model the geometry of such space the dihedral angle of the dodecahedron must be 120 degrees. Therefore, it is not possible to model such space using the Euclidean geometry. In this case, we use spherical geometry.

To find the desired dodecahedron we consider it embedded in the 3-sphere. If the dodecahedron is very small its dihedral angle is very close to the flat dodecahedron. Then, with an appropriate scale, the dodecahedron dihedral angle equals to 120 degrees. This dodecahedron can be parameterized by a flat dodecahedron centered at the origin of $\mathbb{E}^3$ using $(x, y, z) \to (x, y, z, 1)/|(x, y, z, 1)|_{\mathbb{E}}$. To move points and directions between the settings, just apply the parameterization to the points and its differential to the directions.

If a ray intersects a face of the dodecahedron we update its position and direction using the face transformation in the spherical setting. Specifically, to compute the intersections between a spherical ray and a dodecahedron's face, we observe such face is contained in a 2-sphere, which can be 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. 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 $\tan t = -\langle p, n \rangle_{\mathbb{E}}/\langle v, n \rangle_{\mathbb{E}}$.

The immersive visualization of the Poincaré sphere is a tessellation of $\mathbb{S}^3$ by 120 dodecahedra. This is one of the 4-dimensional polytopes, known as 120-cell and shown for the first time here.

Figure 4.3 presents an immersive view of Poincaré dodecahedral space. A unique Suzane with hands and the dodecahedron edges compose the scene. For a better understanding of the spherical geometry, we do not attach Suzane to the
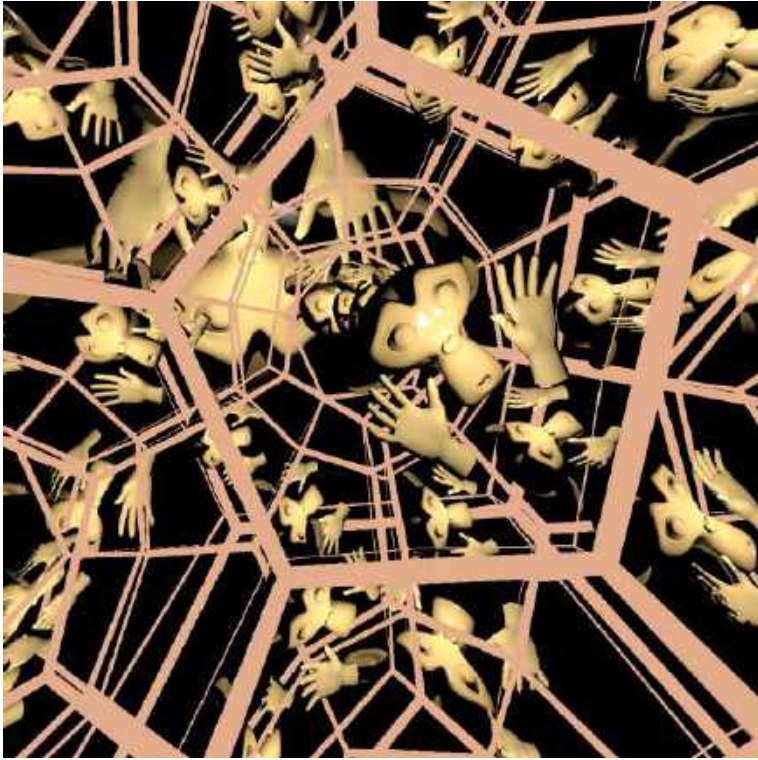
Figure 4.3: Inside view of Poincaré dodecahedron space, which is obtained by identi-
fying, with a rotation of $\pi/5$, the opposite faces of a regular dodecahedron embedded
in 3-sphere. We use a parameterization of the spherical dodecahedron to set our scene:
Suzane with hands and the dodecahedron's edges. The faces pairing make the rays that
leave a face return, with an additional rotation, from its opposite face, giving rise to many
copies of the scene: a tessellation of sphere, the 4-dimensional regular polytope known as
120-*cell*.

camera. Note that as the distance increases, Suzanne's size first decreases and then
begins to increase: there is a large Suzanne upside down at scene background. This
image describes the *icosahedron group* acting on the 3-sphere.

## 4.3   Hyperbolic

This section presents and illustrates two classical examples of three-dimensional spaces with their geometries modeled by the hyperbolic geometry.

### 4.3.1   Seifert–Weber dodecahedral space

We describe a compact 3-manifold with its geometric structure modeled by the hyperbolic geometry. For this, we consider, again, a dodecahedron. Identifying each pair of opposite faces of the dodecahedron with an additional clockwise rotation of $3\pi/10$ gives rise to a 3-manifold which is better know as *Seifert–Weber dodecahedral space $M$*.

The face pairing in the Seifert–Weber space construction produces many identifications, for example, you can verify that edges are grouped into six groups of five. Therefore, it is not possible to fit Euclidean geometry into such a 3-manifold, since the regular Euclidean dodecahedron has a dihedral angle of approximately 116 degrees. The desired dodecahedron should have a dihedral angle of 72 degrees.

We use the hyperbolic geometry to model the geometry of Seifert–Weber space. Let the dodecahedron be centered at the origin of $\mathbb{H}^3$. The dihedral angle of the dodecahedral in the hyperbolic space is smaller than in the Euclidean case. In fact, with an appropriate scale, the dodecahedron admits a dihedral angle of 72 degree as desired.

Using Klein's model of $\mathbb{H}^3$, the rays are straight. So to compute a ray leaving a point $p \in M$ in a direction $v$, we use $r(t) = p + tv$. For each intersection between $r$ and a dodecahedron face, we update $p$ and $v$ through the hyperbolic isometry that produces face pairing above. This isometry is quite distinct from Euclidean isometries (see (Gunn 1993)). For shading purposes, we use the Lorentzian scalar product.

The immersive perception of $M$ using ray tracing is a tessellation of $\mathbb{H}^3$ by dodecahedra with a dihedral angle of 72 degrees.

Figure 4.4 illustrates an immersive view of Seifert–Weber dodecahedral space. Again, there is only one *Suzanne* endowed with hands attached to the camera. The image describes the action of a special discrete group on the hyperbolic space, which provides a dodecahedron tessellation of the hyperbolic space.
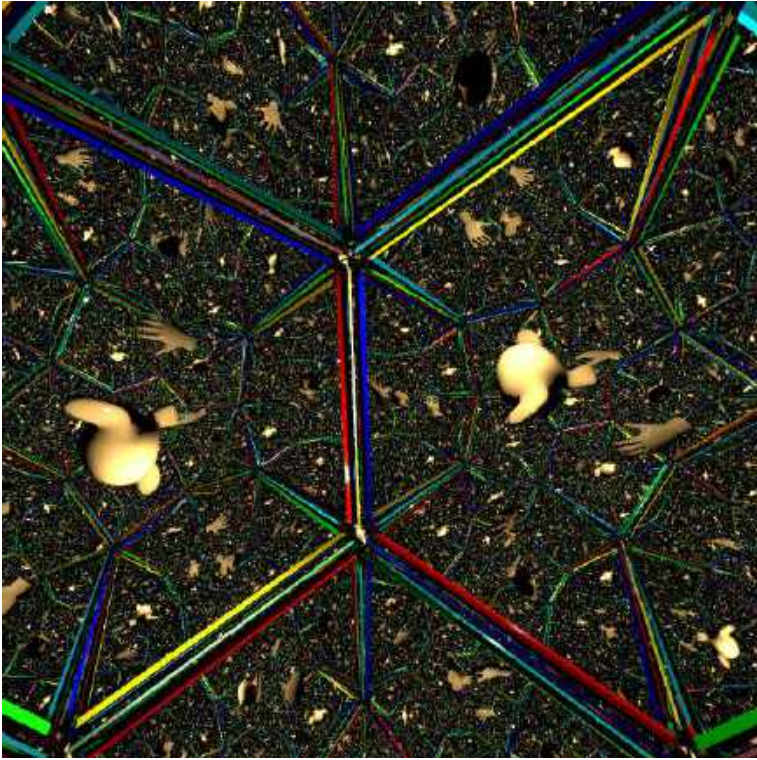
Figure 4.4: Inside view of Seifert–Weber space obtained by gluing, with a rotation of $3\pi/10$, the opposite faces of a special dodecahedron in the hyperbolic space. We use the dodecahedron to set up our scene: a unique Suzane with hands and the dodecahedron's edges. The face pairing make the rays that leave a face return, with an additional rotation, from its opposite face, giving rise to many copies of the scene.

### 4.3.2    Mirrored Dodecahedron

For an example of an orbifold modeled by the hyperbolic space, consider the dodecahedron embedded in $\mathbb{H}^3$. Let $\Gamma$ be the group of reflections generated by the dodecahedral faces. With an appropriate scale, the dihedral angle of the dodecahedron reaches 90 degrees. The quotient $\mathbb{H}^3/\Gamma$ is the *mirrored dodecahedral space*. The group $\Gamma$ tessellates $\mathbb{H}^3$ with right-angle dodecahedra, thus each edge has exactly 4 adjacent cells.

Figure 4.5 provides an immersive visualization of the mirrored dodecahedron using the reflection definition in the hyperbolic space. Again, Suzanne model is

attached to the camera. The image is the view of the group of reflection acting on the Hyperbolic space.
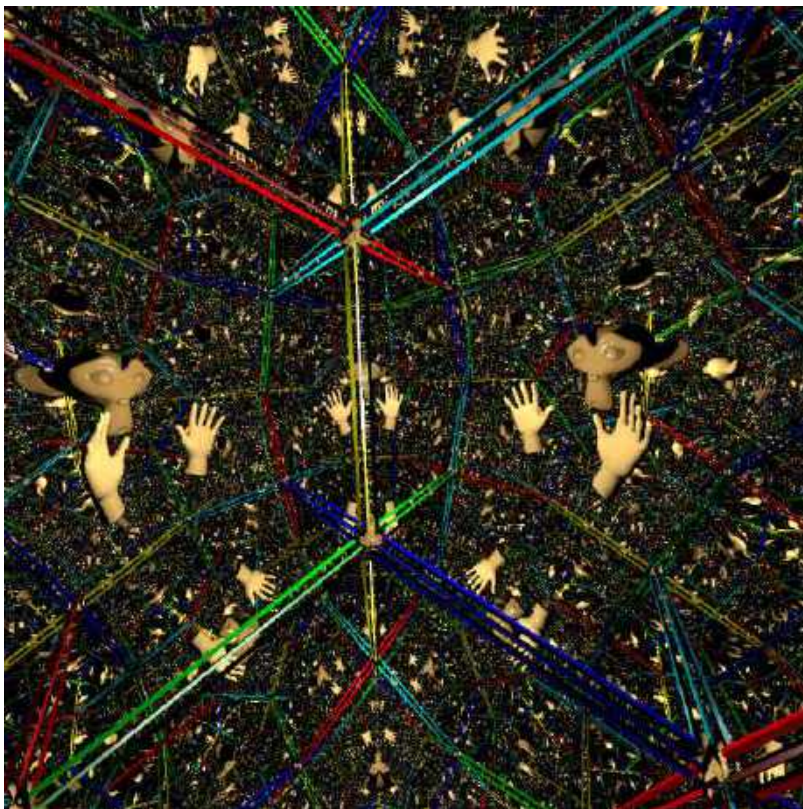


Figure 4.5: Inside view of the mirrored dodecahedron obtained by considering the faces of a hyperbolic dodecahedron to be perfect hyperbolic mirrors. A unique mesh (Suzanne) and the dodecahedron's edges provide the scene. The mirrors make the rays iterate in the scene, producing the sensation of being inside a tessellation of the hyperbolic space.

## 4.4 Analysis

In this section we present a quantitative and qualitative analysis of the results developed using our framework. This includes computational performance, interactivity and space perception.

## 4.4.1　Performance

Here we show the experiments to evaluate our algorithm in respect of performance in current Virtual Reality (VR) devices. The hardware setup consists of a computer with a NVIDIA GeForce 2080 Ti for RTX Ray Tracing support and a HTC Vive for VR visualization. The resolution is set to $1512 \times 1680$ for each eye, resulting in a total resolution of $3024 \times 3360$. A mono version of the algorithm is used as control. Figure 4.6 shows the results.
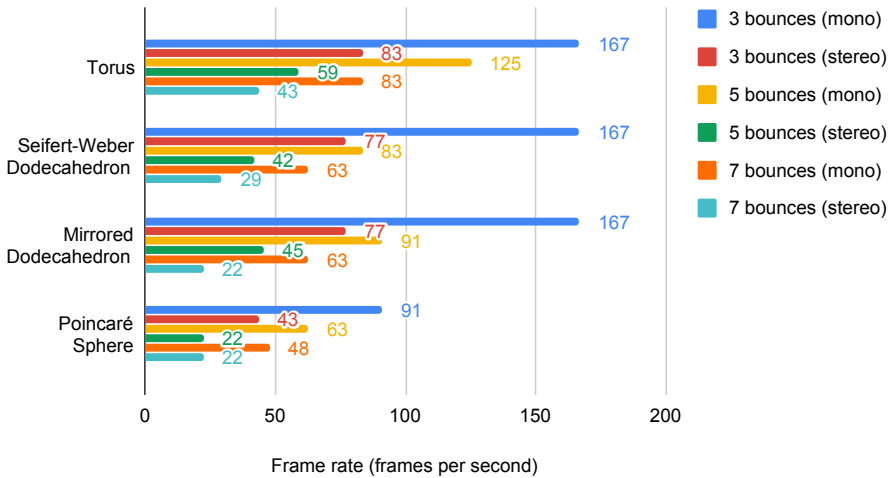
### Performance X Number of ray bounces



Figure 4.6: Performance X Number of ray bounces. The algorithm can generate high resolution stereo images of the spaces, performing up to 80 fps.

Our algorithm achieves performances near 80 fps (frames per second) in high resolution for the frat torus, Seifert–Weber space, and hyperbolic mirrored dodecahedra when using 3 or less bounces. This value is near 90 fps, the peak frame rate recommended for VR experiences in Vive, and ensures a smooth experience for users immersed in those spaces.

### 4.4.2 Interaction

To give the user a better perception of the torus and the mirrored room, we attach, besides Suzanne's head to the camera, models of the left/right hands to the left/right controls of the HTC Vive (see Figure 4.2). Thus interacting in the fundamental domain provides a better sense of being immersed in the quotient spaces. However, we used a very simple approach to map the positions from the physical laboratory to the abstract spaces. For a great discussion about this topic we refer to Weeks 2021.

A good exercise would be to include the motion capture of the user whole body skeleton, using techniques reminiscent from computer vision and artificial intelligence (L. J. S. Silva et al. 2019) (see Figure 4.7). This will allow to include in the scene complete avatars of the users, instead of only the head and hands used in the current implementation.

### 4.4.3 Space Perception

To produce a better understanding of the Non-Euclidean space structures we include the edges of their fundamental domains to the scene. The result images provide a perception of a tessellation of their covering spaces by copies of the fundamental domain polyhedron.

In the examples presented in this chapter, the complete cell structure of their covering spaces is readily apparent since we explicitly marked the boundary edges of the fundamental domain, see Figures 4.1 to 4.5.

More subtle perceptions arise if only some static objects are placed in key landmarks of the domain. Moreover, adding a dynamic behavior may give a transient or pulsating character to the space (i.e., with random or periodic motion, respectively). See Figure 4.8.

As mentioned above, Weeks 2021 has deeply investigated the problem of tracking the user's head and hands from the physical lab. In particular, he discovered that the space *holonomy* would lead to violations of the *coherence* between the head and hands. However, the used visualization framework was based on rasterization, therefore, implementing this using ray tracing would be a great exercise.

In addition, when the viewer is placed inside an opaque cell with a few openings (e.g., a cube with doors and windows), the perception of an infinite space changes to that of a maze.

Another important ingredient in the understanding of the space structure is the scale, which is related to the fundamental domain volume. In Figures 4.1 to 4.3, for example, we are able to see many copies of the fundamental domain,

Figure 4.7: Pose Detection and Motion Capture: currently Head and Hands are captured using HTC Vive Headset and Controllers; in future implementations the user's pose (indicated by the superimposed skeleton) will be estimated and tracked by the AI method described in (L. J. S. Silva et al. 2019).

which produce, again, the view of its covering space. However if we consider a fundamental domain sufficiently larger, the user will be able to visualize mostly the immediate surroundings of the scene restricted to the fundamental domain. This leads us to the philosophical question: *what is the shape of the Universe?* or *could we be living inside a 3D torus?*

Figure 4.8: Inside view of the mirrored dodecahedron. This is the same space depicted in Figure 4.5 without showing the structure of the fundamental domain.

# 5

# *Visualization of Nil, $\widetilde{SL_2(\mathbb{R})}$, and Sol*

This chapter explores the Riemannian ray tracing (introduced in Chapter 3) in the non-isotropic geometries to render inside views of the most non-trivial Thurston Geometries: Nil, Sol, and $\widetilde{SL_2(\mathbb{R})}$. These Riemannian manifolds are fundamental in the Geometrization conjecture as we saw in Chapter 1.

Nil, Sol, and $\widetilde{SL_2(\mathbb{R})}$ geometries were described in Section 1.4.3 as examples of *non-isotropic* spaces: their geometries are not the same in all directions because local two-dimensional slides admit different curvatures. As a consequence, their geodesics are not (in general) straight lines in the Euclidean sense, only a few exceptions are, for example, the geodesics in Nil geometry towards the $z$ axis. Another property is that these spaces are Lie groups: a class of group admitting manifold structures. Geodesics in Lie groups can be calculated on the identity of the group and then translated to each element. Such property is very suitable for ray tracing.

For the Nil space, we have the analytical solution of the geodesic flow (Szilágyi and Virosztek 2003), i.e., case i) of Section 3.3.3. In (Divjak et al. 2009; Marenitch 2008), the authors provide the geodesics of $SL_2(\mathbb{R})$, however, they use a different parameterization of the space. An exercise would be to explore those formulas to increase performance in the visualizations. (Troyanov 1998) obtained a formula for geodesics in *Sol*, however, it contains many coefficients that can not be com-

puted in a closed formula. Therefore, for the Sol and $\widetilde{SL_2(\mathbb{R})}$ spaces we only approximate the geodesics by numerical integration of the flow ODE's, i.e., case ii) of Section 3.3.3.

To visualize Nil, Sol, and $\widetilde{SL_2(\mathbb{R})}$ geometries, we consider manifolds geometrically modeled by them. For Nil and Sol spaces, we choose compact manifolds created by taking the quotient of these spaces by "simple" discrete groups: "translations" in the direction of axis $x$, $y$, and $z$. We use the edges of the respective fundamental domains to visualize the structure of Nil and Sol. The result is a tiling produced by the fundamental domain edges. For $\widetilde{SL_2(\mathbb{R})}$ space, we take a different approach, we visualize the well-known *special linear group* of the $2 \times 2$ real matrices (with unit determinant) $SL_2(\mathbb{R})$ which has the geometry modeled by $\widetilde{SL_2(\mathbb{R})}$. To explore the distortions of such a manifold, we consider rendering a grid defined in the domain of a parameterization of $SL_2(\mathbb{R})$ by $(x, y, z)$ coordinates.

However, expressing the structures of those spaces with static images is not trivial. We recommend the use of VR for better understanding them, due to their non-isotropic properties. Being inside allows a person to better explore and control variations in point-of-view and position, and understand how they change visualization. An intermediate solution is to explore them through a video. In the figures, we will illustrate several points of view inside these geometries commenting on how the underlying structures vary.

## 5.1 Visualizing Nil space

Nil geometry was briefly described in Section 1.4.3. To visualize this space, we consider a compact manifold $M = Nil/\Gamma$ modeled by $Nil$. Here $\Gamma$ is a discrete group of isometric actions on $Nil$. Then, setting a camera inside such abstract space we obtain images of Nil space (universal covering space of $M$) tessellated by copies of the fundamental domain of $M$. Each of these copies corresponds to elements of the discrete group $\Gamma$.

Specifically, let $\Gamma$ be the discrete group generated by the "translations" in the direction of axis $x$, $y$, and $z$: $\Phi_1(p) = (x + 1, y, y + z)$, $\Phi_2(p) = (x, y + 1, z)$, and $\Phi_3(p) = (x, y, z + 1)$. We use the word translation in quotes because $\Phi_1$ is a translation in the axis $x$ composed with a shear transformation. The manifold $M = Nil/\Gamma$ inherits the geometry of the Nil space. Moreover, it provides a 2-torus for each fixed $x$, resulting in $M$ admitting a foliation by tori. The unit cube of $\mathbb{R}^3$ is the fundamental domain.

We set the scene inside the unit cube and ray trace it using the geodesic (ray) formula of *Nil* defined in Section 1.4.3. Each time a ray intersects a face of the fundamental domain, we update the ray position and ray direction using the corresponding element of discrete group $\Gamma$. Specifically, to visualize the scene objects we define the shading function $c_{Nil} : V_p \to \mathcal{C}$, where $V_p$ is the observer's field of view in the tangent space at the observer position and $\mathcal{C}$ is the RGB color space. The function $c_{Nil}$ is defined using *Nil* rays and *Nil* metric, and the definitions discussed in Section 3.2.2.

Figure 5.1 illustrates four points of view inside of the 3-manifold $M = Nil/\Gamma$. The scene is composed of thickening in 2D of the boundary of the fundamental domain faces. Opposite faces receive the same shading. Red, green, and blue for the faces parallel to the axis, $x$, $y$, and $z$. In the images, the lines shaded with red and green are extended to the infinite as straight lines. These lines are geodesics of *Nil* and are perpendicular to the plane $yz$; lines with such property foliate the space *Nil*.

In the top left image, the camera points towards $z$-axes, where the gluing of the cube faces are trivial (like in the 3D torus). This gives copies of the cube by left translation along $z$-axes. Looking towards the direction of the green faces would provide a similar result, thus we avoid this perspective. In top right and bottom left, the camera points toward $x$-axes, in both senses. The identification of the cube faces in this direction if nontrivial, producing this tessellation of Nil. Finally, in the bottom right a view towards the diagonal of the cube.

## Ray-plane intersection

Let $\beta(t)$ be a geodesic in the Nil space, such that $\beta(0) = p$ and $\beta'(0) = v$. In Section 1.4.3, we saw that $\beta(t) = (p_x + x(t), p_y + y(t), p_z + z(t) + p_x y(t))$; $(x(t), y(t), z(t))$ is a ray leaving the origin.

We compute the intersection between the geodesic $\beta(t)$ and the planes $x = c$ and $y = c$, where $c$ is a constant. For the plane $x = c$, we have to solve the equation $x(t) + p_x = c$, which is equivalent to $\frac{c}{w}(\sin(wt + \alpha) - \sin(\alpha)) + p_x = c$. After some computations, we obtain $t = \left(\arcsin((c - p_x)\frac{w}{c} + \sin(\alpha)) - \alpha\right)/w$.

In an analogous way we compute the intersection between $\beta(t)$ and the plane $y = c$. The parameter is $t = \left(\arccos((c - p_y)\frac{-w}{c} + \cos(\alpha)) - \alpha\right)/w$.

We could not compute (yet) explicitly the intersection of $\beta(t)$ and the plane $z = c$, which is given by the solution of the equation $p_z + z(t) + p_x y(t) = c$. We avoid this problem by computing the intersections of rays and the fundamental domain faces using a *binary search* algorithm. This is reachable because we have the
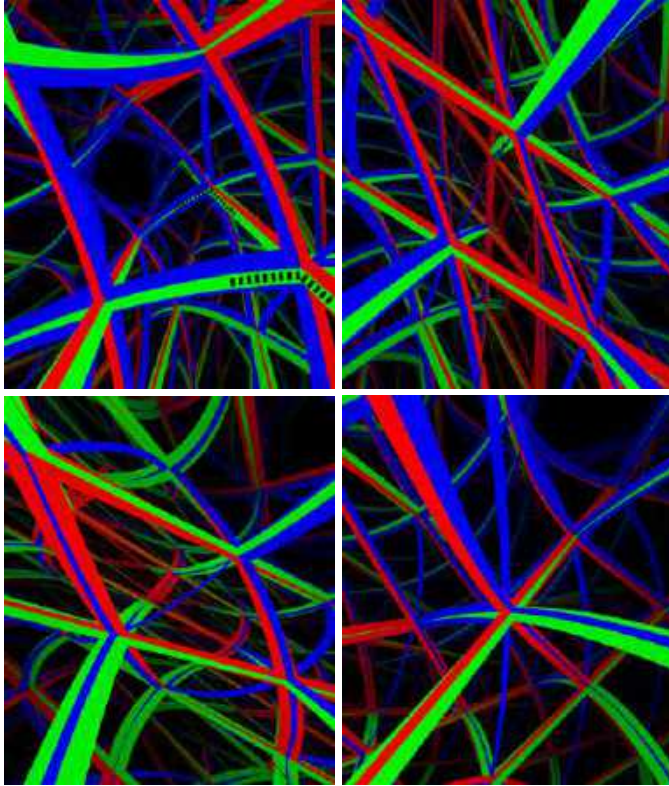
Figure 5.1: Inside views of a Nil manifold. The cube's edges compose the scene. In the top left, the camera point towards $z$-axes, where the group transformation are translations. Thus we see translated copies of the cube distorted by the Nil geometry. To highlight the distortions we mark (black dotted line) the corner of the cube's bottom face (in green): as the fundamental domain is translated the green face receives an anticlockwise torsion. In the top right and bottom left, the observer looks toward $x$-axes, in both senses respectively. The identification of the cube faces in this direction if nontrivial producing this tessellation of Nil. In the bottom right, we set the camera towards the cube diagonal.

analytic expression of the geodesics of Nil space. This algorithm results in better rendering performance in comparison with the other geometries (Sol and $\widetilde{SL_2(\mathbb{R})}$ spaces), which do not have analytic expressions (using elementary functions) for their geodesics.

## 5.2    Visualizing Sol space

To visualize Sol geometry, we consider a compact 3-manifold $M = Sol/\Gamma$ modeled by $Sol$. Here $\Gamma$ is a discrete group of isometries acting on $Sol$. Defining a camera inside such space, we obtain images of Sol space tessellated by the fundamental domain of $M$.

Let $\Gamma$ be the discrete group spanned by the translations along $x$- and $y$-axes, the maps $\Phi_1(p) = (x+1, y, z)$ and $\Phi_2(p) = (x, y+1, z)$, and the map $\Phi_3(p) = (x \cdot e^{-2\ln\phi}, y \cdot e^{2\ln\phi}, z + 2\ln\phi)$; where $p = (x, y, z)$ is a point in $\mathbb{R}^3$ and $\phi$ is the well-known golden ratio number. The 3-manifold $M = Sol/\Gamma$ inherits the geometry of Sol space, and for each fixed $z_0$, it provides a 2D torus which is the quotient of the plane $\mathbb{R}^2 \times \{z_0\}$ by the discrete group generated by the translation maps $\Phi_1$ and $\Phi_2$. Thus the 3-manifold $M$ is foliated by tori. The parallelepiped $D \times [0, 2\ln\phi)$ is the fundamental domain of $M$; where $D$ is the unit square $[0, 1]^2$.

We explain the role of the element $\Phi_3(p) = (x \cdot e^{-2\ln\phi}, y \cdot e^{2\ln\phi}, z + 2\ln\phi)$ in the discrete group $\Gamma$ discussed above. It is an automorphism between the planes $\mathbb{R}^2 \times \{0\}$ and $\mathbb{R}^2 \times \{2\ln\phi\}$. Observe that the numbers $-2\ln\phi$ and $2\ln\phi$ are the eigenvalues of the $2 \times 2$ matrix $A = \left(\begin{smallmatrix} 2 & 1 \\ 1 & 1 \end{smallmatrix}\right)$, which is an automorphism of $\mathbb{R}^2$ preserving the lattice $\mathbb{Z}^2 \subset \mathbb{R}^2$. Thus the matrix $A$ induces a homeomorphism of the 2D torus. Lining up the $(x, y)$-coordinates of Sol space with the eigenvectors of $A$, we get an identification of $A$ with $\Phi_3$. This procedure to create the Sol manifold $M$ seems to be naïve, however, it can be extended for every automorphism of $\mathbb{R}^2$ preserving $\mathbb{Z}^2$. For details see (Thurston 1997) and (Coulon et al. 2020b).

We set the scene inside the fundamental domain (the parallelepiped $D \times [0, 2\ln\phi)$) of $M$ and ray trace it using Sol rays. Each time a ray hits a parallelepiped face it is updated by the discrete group $\Gamma$. As in the Nil section, we define a Riemannian shading for Sol and create the scene by thickening the boundary of the fundamental domain faces and opposite faces receive the same shading. Figure 5.2 provides four points of view of $M = Sol/\Gamma$. We look toward the fundamental domain faces and vertex to explore the non-isotropic nature of Sol which is the most non-symmetric geometry. In the top left and right of the image, the camera points towards $z$-axes, in both senses, respectively. Here the identification of the parallelepiped faces is nontrivial, producing together with its geometry, this tessellation of Sol. The space looks compressed horizontally and vertically, depending on the sense. These regions are highlighted with green dotted lines in the figure. In the bottom left, the camera points in the direction of $x$-axes, the identification of the cube faces in this direction is trivial (like in the 3D torus). This gives the impression of translated copies of the fundamental domain very distorted
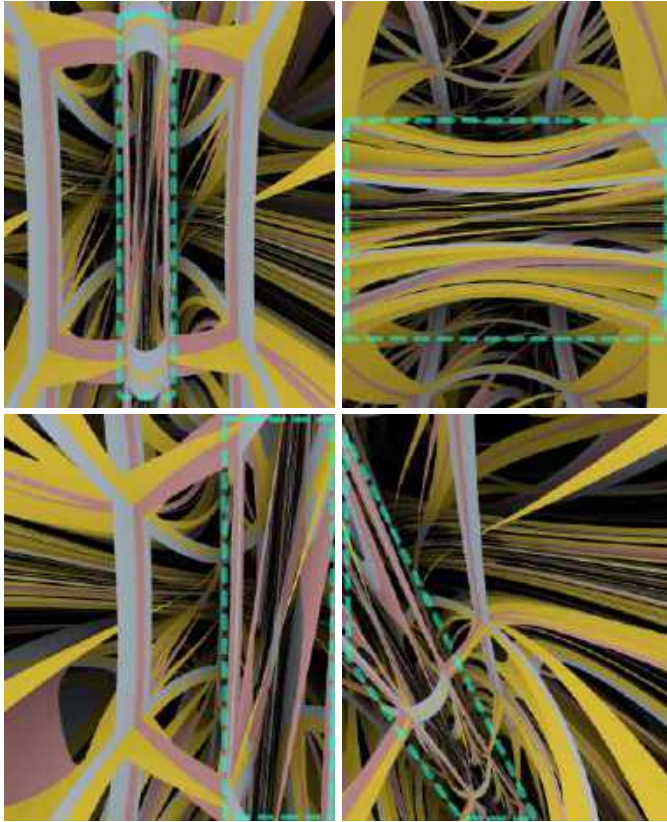
Figure 5.2: Inside views of a Sol manifold. The parallelepiped's edges compose the scene. The borders of each face are shaded with different colors. In the top, the camera points towards $z$-axes, in both senses. In this direction the gluing of the parallelepiped faces is nontrivial, producing together with its geometry, this tessellation of Sol. The space looks compressed horizontally and vertically, depending on the sense. We highlight these regions with dotted lines. In the bottom left, the camera points in the direction of $x$-axes, the group transformation in this direction is trivial. Thus we see copies of the fundamental domain distorted by Sol geometry, they surround the horizontal compressed region. On the bottom right side, a view towards the diagonal of the fundamental domain.

by Sol geometry, surrounding the vertically compressed region. Looking towards $y$-axis is similar, however, the copies surround the horizontal compressed region. Finally, on the bottom right side of the image, a view towards the diagonal of the fundamental domain. The non-symmetry of Sol is remarkable.

## 5.3    Visualizing $\widetilde{SL_2(\mathbb{R})}$ space

Here we are not visualizing a 3-manifold generated by taking quotients of $\widetilde{SL_2(\mathbb{R})}$ by discrete groups. Instead, we focus on the visualization of the special linear group $SL_2(\mathbb{R})$ which is modeled by $\widetilde{SL_2(\mathbb{R})}$; see Section 1.4.3 for the details. This 3-manifold corresponds to the set of $2 \times 2$ matrices with unit determinant. This space is well known due to its variety of properties, therefore its visualization is interesting.

     As in the Nil and Sol spaces, we define a Riemannian shading for $SL_2(\mathbb{R})$ and we illustrate it in Figure 5.3. This image refers to an immersive view of a grid defined in the domain of a parameterization of $SL_2(\mathbb{R})$ by $(x, y, z)$ coordinates. The parameterization distorts the 3D grid following the geometry of $SL_2(\mathbb{R})$. We choose empirically the RGB color to be the $(x, y, z)$ coordinates of the hit points. In future work, we intend to explore the rendering using the sectional curvature of $SL_2(\mathbb{R})$.

     We give four point of views in Figure 5.3. To explain the images, we use the correspondence between the spaces $SL_2(\mathbb{R})$ and $\mathbb{H}^2 \times \mathbb{S}$ with an appropriate metric (Martelli 2016). We consider the half-plane model of the hyperbolic plane $\mathbb{H}^2$. In the top left image, we look towards the direction of the $x$-axes (the red edge). This is a geodesic and coincides with a straight line in the Euclidean sense. It also matches with the line $(0, t, 0)$ in the model $\mathbb{H}^2 \times \mathbb{S}$. We avoid looking at the other sense because the parameterization of $SL_2(\mathbb{R})$ that we are using is not defined for the plane $x = 1$. In the top right image, the camera points towards $y$-axes, which coincides with the green line. In the model $\mathbb{H}^2 \times \mathbb{S}$, it is the line $(t, 1, 0)$. In the bottom left image, the observer is aligned to the $z$-axes (the blue edge). In the model $\mathbb{H}^2 \times \mathbb{S}$, this line is $(t, t^2 + 1, \phi)$, where $\cos(\phi) = 1/\sqrt{t^2 + 1}$. Finally, in the bottom right image, the camera points towards the diagonal direction.

## 5.4    Experiments and comparisons

To evaluate the proposed model, we consider the implementation, discussed in Chapter 3, using NVIDIA's Falcor (Benty et al. 2018a). Falcor provides a platform that supports many features for real-time visualization, including OpenVR and DirectX Raytracing. However, ray tracing and virtual reality do not work originally in an integrated way there. Thus, we relied on an integration extension which was presented in Chapter 2. The test system is a i7-8700 CPU at 3.20GHz with 16GB RAM and a RTX 2080 Ti GPU.
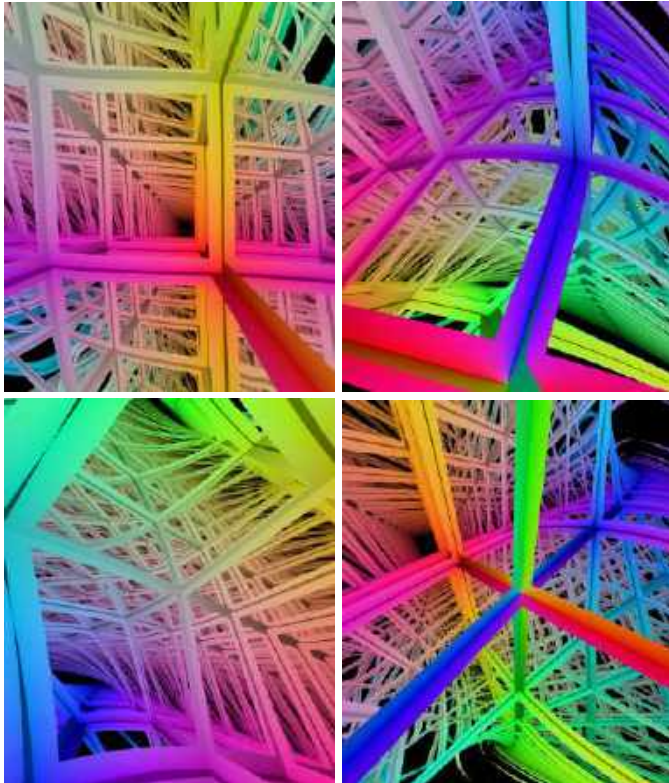
Figure 5.3: Inside views of a parameterization of $SL_2(\mathbb{R})$. The scene is composed of a regular grid in $\mathbb{R}^3$ deformed by $SL_2(\mathbb{R})$ metric. The RGB colors are given by the $x$, $y$, $z$-coordinates of the hit points. In the top left, we set the camera towards the $x$-axes (the red line); this is a geodesic and coincides with a straight line in the Euclidean sense. In the top right, the camera points towards $y$-axes, which coincides with the green line. In the bottom left, the observer points at the $z$-direction (the blue edge). In the bottom right, the camera is aligned with the grid's diagonal.

The experimental methodology consists of varying the number of ray bounces and the step size for the numerical integration and evaluating how performance and display quality are affected. Figure 5.4 shows the performance results. The image resolution for each eye is $1360 \times 1512$. It is important to emphasize that the scene is effectively ray-traced two times, one for each eye. The model succeeds in rendering high definition images in rates compatible with VR devices (90 fps) for Nil, Sol, and $SL_2(\mathbb{R})$, depending on proper parameter setup. The frame rate

of $SL_2(\mathbb{R})$ is slower than that of Sol because it performs more computations in the shader since $SL_2(\mathbb{R})$ has more non-zero Christoffel symbols.
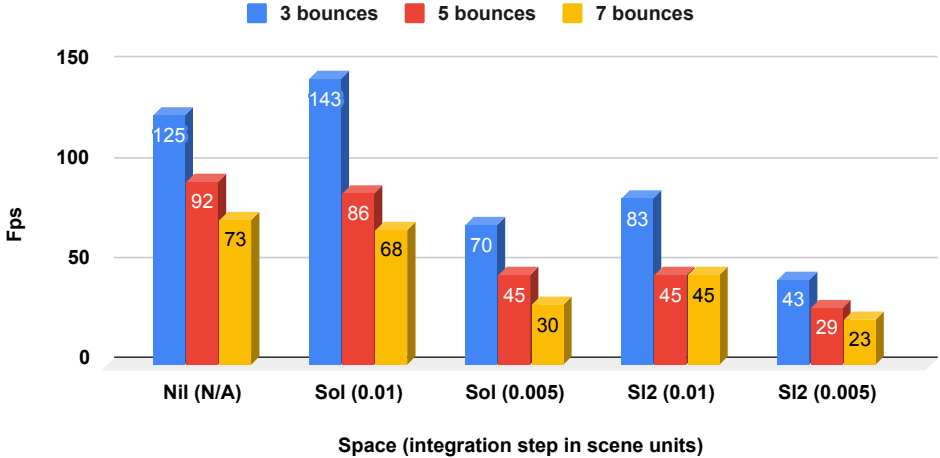
**Resolution: 1360x1512 (each eye)**



Figure 5.4: Performance evaluation. We test two parameters: the number of ray bounces and the ray integration step size. Performance scales with number of bounces and the algorithm can be set to render all spaces at VR rates (near 90 fps). Resolution is 1360x1512 for each eye. Note that step size does not apply to Nil because we have used a closed form expression for the geodesics.

Given that performance is dependent on parameters, we evaluate how they affect the final image. Figure 5.5 shows the results for Sol. As expected, the number of bounces controls how far we see the tessellation of the covering space by replication of the fundamental domain. Even a small value is useful for giving an intuition of the manifold. As expected, varying the integration step size is a trade-off between performance and precision.

(a) Step:0.01. Bounces:3.   (b) Step:0.01; Bounces:5.   (c) Step:0.01; Bounces:7.

(d) Step:0.005; Bounces:3.   (e) Step:0.005; Bounces:5.   (f) Step:0.005; Bounces:7.

(g) Step:0.001; Bounces:3.   (h) Step:0.001; Bounces:5.   (i) Step:0.001; Bounces:7.
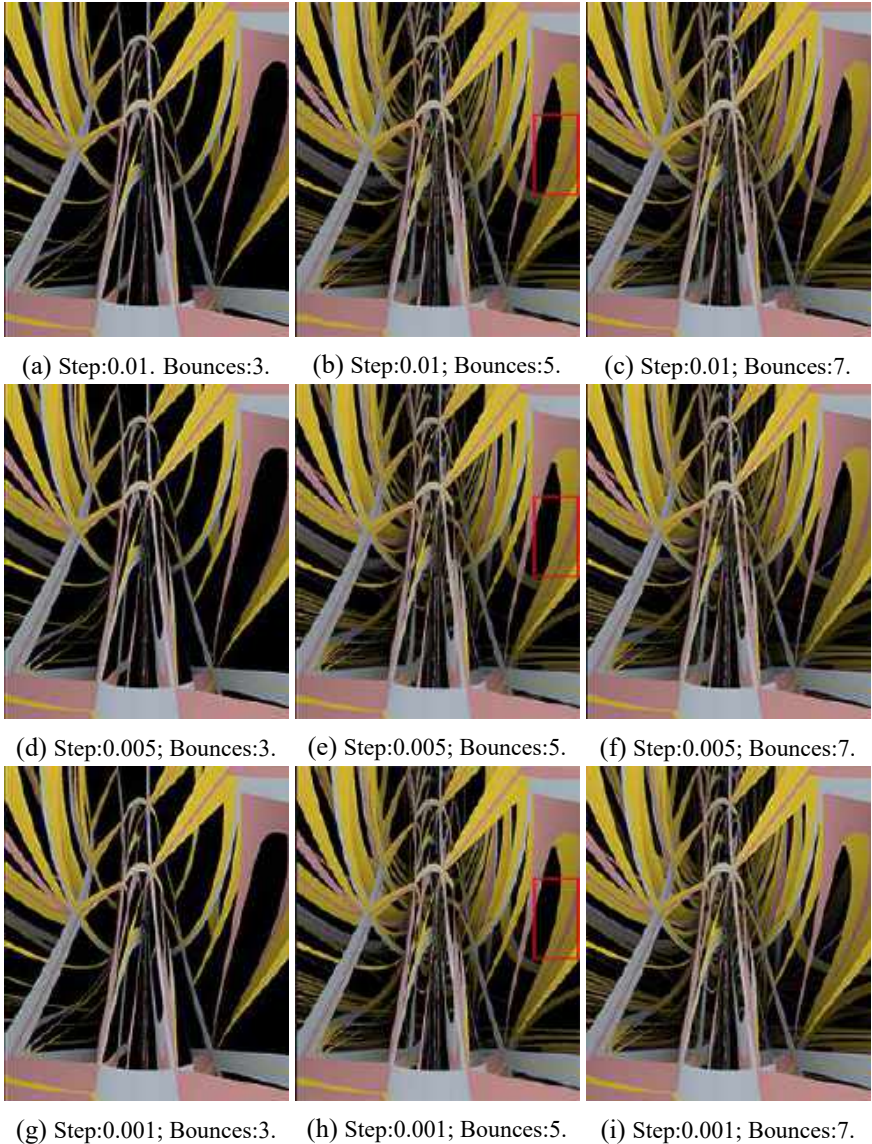
Figure 5.5: Rendering evaluation of Sol manifold. Figures show variations on the integration step (vertical) and number of bounces (horizontal). As bounces increase, more parallelepipeds tessellating space become visible. Even 3 bounces provide a good space intuition: (a), (d) and (g). The rectangles in (b), (e) and (h) highlight approximation improvements resulting from finer integration step.

# *Bibliography*

J. Arvo and D. Kirk (1989). "A Survey of Ray Tracing Acceleration Techniques."
In: *An Introduction to Ray Tracing*. GBR: Academic Press Ltd., pp. 201–262
(cit. on p. 25).

M. Bailey (2018). "Introduction to the Vulkan Graphics API." In: *ACM SIG-
GRAPH 2018 Courses*. Ed. by C. Kaplan. SIGGRAPH '18. Vancouver, British
Columbia, Canada: ACM, 3:1–3:146 (cit. on p. 28).

A. H. Barr (1986). "Ray tracing deformed surfaces." *ACM SIGGRAPH Computer
Graphics* 20.4, pp. 287–296.

N. Benty, K.-H. Yao, T. Foley, M. Oakes, C. Lavelle, and C. Wyman (May 2018a).
*The Falcor Rendering Framework* (cit. on p. 66).

— (2018b). *The Falcor Rendering Framework* (cit. on p. 28).

P. Berger, A. Laier, and L. Velho (2014). "An image-space algorithm for immersive
views in 3-manifolds and orbifolds." *Visual Computer* 31.1, pp. 93–104 (cit.
on pp. 38, 40).

M. Boileau, S. Maillot, and J. Porti (2003). *Three-dimensional orbifolds and
their geometric structures*. Vol. 15. Panoramas et Synthèses [Panoramas
and Syntheses]. Paris: Société Mathématique de France, pp. viii+167. MR:
2060653(2005b:57030). Zbl: 1058.57009.

A. Bölcskei and B. Szilágyi (2007). "Frenet formulas and geodesics in Sol ge-
ometry." *Contributions to Algebra and Geometry* 48.2, pp. 411–421. MR:
2364799. Zbl: 1167.53021 (cit. on p. 20).

M. R. Bridson and A. Haefliger (1999). *Metric spaces of non-positive curvature*.
Vol. 319. Grundlehren der Mathematischen Wissenschaften [Fundamental

Principles of Mathematical Sciences]. Berlin: Springer-Verlag, pp. xxii+643. MR: 1744486(2000k:53038). Zbl: 0988.53001.

A. Burnes (Mar. 2019). *Accelerating The Real-Time Ray Tracing Ecosystem: DXR For GeForce RTX and GeForce GTX* (cit. on pp. 26, 34).

B. Burton (2004). "Introducing Regina, the 3-manifold topology software." *Experimental Mathematics* 13.3, pp. 267–272. MR: 2103324. Zbl: 1090.57003.

M. P. d. Carmo (1992). *Riemannian geometry*. Birkhäuser. MR: 1138207. Zbl: 0752.53001 (cit. on pp. 9, 14, 15, 18, 47).

X.-W. Chen and X. Lin (2014). "Big data deep learning: challenges and perspectives." *IEEE access* 2, pp. 514–525.

M. Cicconet (2007). "Visualização Relativística Usando Ray Tracing."

S. Cook (2013). *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

R. Coulon, E. A. Matsumoto, H. Segerman, and S. Trettel (2020a). "Non-Euclidean Virtual Reality III: Nil." arXiv: 2002.00513 (cit. on p. 13).

— (2020b). "Non-Euclidean Virtual Reality IV: Sol." arXiv: 2002.00369 (cit. on pp. 13, 64).

— (2020c). "Ray-marching Thurston geometries." arXiv: 2010.15801 (cit. on p. 13).

B. Divjak, Z. Erjavec, B. Szabolcs, and B. Szilágyi (2009). "Geodesics and geodesic spheres in SL(2,R) geometry." *Mathematical communications* 14.2, pp. 413–424. MR: 2743187. Zbl: 1193.53103 (cit. on p. 60).

e. Fernando R., ed. (2004). *GPU gems: programming techniques, tips, and tricks for real-time graphics*. Vol. 590. Addison-Wesley Reading (cit. on p. 26).

A. T. Fomenko and S. V. Matveev (2013). *Algorithmic and computer methods for three-manifolds*. Vol. 425. Springer Science & Business Media. MR: 1486574 (cit. on p. 5).

G. K. Francis, C. M. A. Goudeseune, H. J. Kaczmarski, B. J. Schaeffer, and J. M. Sullivan (2003). "ALICE on the Eightfold Way: Exploring Curved Spaces in an Enclosed Virtual Reality Theatre." In: *Visualization and Mathematics III*. Ed. by H.-C. Hege and K. Polthier. Berlin, Germany: Springer Berlin Heidelberg, pp. 305–315. Zbl: 1097.68650.

M. H. Freedman et al. (1982). "The topology of four-dimensional manifolds." *J. Differential Geom* 17.3, pp. 357–453. MR: 0679066. Zbl: 0528.57011 (cit. on p. 2).

M. von Gagern and J. Richter-Gebert (n.d.). "Hyperbolization of Euclidean Orna-
    ments." *The Electronic Journal of Combinatorics* 16.2 (). MR: 2515775. Zbl:
    1168.51310.

E. Games (Oct. 2017). *Unreal Engine Sun Temple, Open Research Content Archive
    (ORCA)*.

É. Ghys (2017). *A singular mathematical promenade*. ENS Éditions Lyon. MR:
    3702027. Zbl: 06797578 (cit. on p. 2).

R. Gilmore (2008). *Lie groups, physics, and geometry: an introduction for physi-
    cists, engineers and chemists*. Cambridge University Press. MR: 2398213. Zbl:
    1157.00009 (cit. on pp. 15, 17).

E. Gröller (1995). "Nonlinear ray tracing: Visualizing strange worlds." *The Visual
    Computer* 11.5, pp. 263–274.

F. Guimarães, V. Mello, and L. Velho (Aug. 26, 2015). "Geometry independent
    game encapsulation for Non-Euclidean geometries." In: *Proceedings of the
    28th SIBGRAPI Conference on Graphics, Patterns and Images: Workshop of
    Works in Progress*. Ed. by R. Rios and A. Paiva. Salvador, BA, Brazil (cit. on
    p. 45).

C. Gunn (1993). "Discrete groups and visualization of three-dimensional mani-
    folds." In: *Proceedings of the 20th annual conference on Computer graphics
    and interactive techniques*. Ed. by M. Whitton. SIGGRAPH '93. Anaheim,
    CA: ACM, pp. 255–262 (cit. on pp. 12, 23, 53).

—  (2010). "Advances in Metric-neutral Visualization." In: *GraVisMa 2010 Pro-
    ceedings*. Ed. by V. Skala and E. Hitzer. Eurographics. Brno, Czech Republic:
    GraVisMa, pp. 17–26.

C. Gunn and D. Maxwell (1991). *Not Knot*. MR: 1176795.

E. Haines and T. Akenine-Möller, eds. (2019a). *Ray Tracing Gems*. Apress (cit. on
    p. 44).

—  eds. (2019b). *Ray Tracing Gems*. Apress (cit. on p. 28).

A. J. Hanson, T. Munzner, and G. Francis (July 1994). "Interactive methods for
    visualizable geometry." *Computer* 27.7, pp. 73–83.

V. Hart, A. Hawksley, E. Matsumoto, and H. Segerman (2017). "Non-euclidean
    Virtual Reality I: Explorations of $H^3$." In: *Proceedings of Bridges 2017: Math-
    ematics, Art, Music, Architecture, Education, Culture*. Ed. by D. Swart, C. H.
    Séquin, and K. Fenyvesi. Phoenix, Arizona: Tessellations Publishing, pp. 33–
    40. MR: 3702013.

J. F. P. Hudson (1969). *Piecewise linear topology*. University of Chicago Lecture
    Notes prepared with the assistance of J. L. Shaneson and J. Lees. New York–

Amsterdam: W. A. Benjamin, Inc., pp. ix+282. MR: 0248844(40\#2094). Zbl: 0189.54507.

W. Jaco and P. B. Shalen (1979). "Seifert fibered spaces in 3-manifolds." In: *Geometric topology*. Ed. by J. C. Cantrell. Elsevier, pp. 91–99. MR: 0537728. Zbl: 0471.57001 (cit. on p. 6).

A. M. Jaffe (2006). "The millennium grand challenge in mathematics." *Notices Amer. Math. Soc.* 53.6, pp. 652–660. MR: 2235325. Zbl: 1130.01012 (cit. on p. 3).

K. Johannson (1979). *Homotopy equivalences of 3-manifolds with boundaries*. Vol. 761. Lecture Notes in Mathematics. Springer, Berlin, pp. ii+303. MR: 551744. Zbl: 0412.57007 (cit. on p. 6).

C. V. Johnson (2002). *D-branes*. Cambridge university press. MR: 1960004. Zbl: 1026.81047.

J. T. Kajiya (1986). "The Rendering Equation." In: *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*. Ed. by D. C. Evans and R. J. Athay. SIGGRAPH '86. New York, NY, USA: ACM, pp. 143–150.

S. Klimenko, I. Nikitin, V. Burkin, V. Semenov, O. Tarlapan, and H. Hagen (2000). "Visualization in string theory." *Computers & Graphics* 24, pp. 23–30.

E. Kopczyński and D. Celińska-Kopczyńska (2020). "Real-Time Visualization in Non-Isotropic Geometries." arXiv: 2002.09533 (cit. on p. 13).

J. Lee (2010). *Introduction to topological manifolds*. Vol. 202. Springer Science & Business Media. MR: 2766102 (cit. on p. 5).

A. Lumberyard (July 2017). *Amazon Lumberyard Bistro, Open Research Content Archive (ORCA)*.

F. Luna (2016). *Introduction to 3D Game Programming with DirectX 12*. USA: Mercury Learning & Information (cit. on p. 28).

V. Marenitch (2008). "Geodesic Lines in SL2(R) and Sol." *Novi Sad J. Math* 38.2, pp. 91–104. MR: 2526032. Zbl: 1289.53094 (cit. on p. 60).

B. Martelli (2016). "An introduction to geometric topology." arXiv: 1610.02592 (cit. on pp. 8, 10–13, 15, 19, 20, 39, 66).

C. T. McMullen (2011). "The evolution of geometric structures on 3-manifolds." *Bull. Amer. Math. Soc. (N.S.)* 48.2, pp. 259–274. MR: 2774092. Zbl: 1214.57017 (cit. on p. 6).

J. Milnor (1962). "A unique decomposition theorem for 3-manifolds." *American Journal of Mathematics* 84.1, pp. 1–7. MR: 0142125. Zbl: 0108.36501 (cit. on p. 6).

U. of Minnesota (1994). *The Geometry Center*.

E. Molnár (1997). "The Projective Interpretation of the Eight 3-dimensional Homogeneous Geometries." *Contributions to Algebra and Geometry* 38.2, pp. 261–288. MR: 1473106. Zbl: 0889.51021 (cit. on pp. 22, 23).

A. Munshi, B. Gaster, T. G. Mattson, and D. Ginsburg (2011). *OpenCL programming guide*. Pearson Education.

H. Nguyen (2007). *Gpu gems 3*. Addison-Wesley Professional.

J. Nickolls, I. Buck, M. Garland, and K. Skadron (2008). "Scalable Parallel Programming with CUDA." *Queue* 6.2, pp. 40–53.

T. Novello, V. da Silva, and L. Velho (2020a). "Design and visualization of Riemannian metrics." arXiv: 2005.05386.

— (2020b). "Global illumination of non-Euclidean spaces." *Computers & Graphics* 93, pp. 61–70.

— (2020c). "How to see the eight Thurston geometries." arXiv: 2005.12772.

— (2020d). "Immersive Visualization of the Classical Non-Euclidean Spaces using Real-Time Ray Tracing in VR." In: *Proceedings of Graphics Interface 2020*. Ed. by D. I. Levin, F. Chevalier, and A. Jacobson. GI 2020. University of Toronto, 423–4f30 (cit. on p. 20).

— (2020e). "Visualization of Nil, Sol, and SL2 geometries." *Computers & Graphics* 91, pp. 219–231 (cit. on pp. 13, 20, 37).

S. G. Parker et al. (2010). "OptiX: A General Purpose Ray Tracing Engine." *ACM Trans. Graph.* 29.4 (cit. on p. 28).

G. Perelman (2002). "The entropy formula for the Ricci flow and its geometric applications." arXiv: math/0211159 (cit. on pp. 2, 3).

— (2003a). "Finite extinction time for the solutions to the Ricci flow on certain three-manifolds." arXiv: math.DG/0307245 (cit. on pp. 2, 3).

— (2003b). "Ricci flow with surgery on three-manifolds." arXiv: math/0303109 (cit. on pp. 2, 3).

M. Pharr and R. Fernando (2005). *Gpu gems 2: programming techniques for high-performance graphics and general-purpose computation*. Addison-Wesley Professional.

M. Pharr, W. Jakob, and G. Humphreys (2016a). *Physically based rendering: From theory to implementation*. Morgan Kaufmann.

— (2016b). *Physically based rendering: From theory to implementation*. Morgan Kaufmann (cit. on pp. 31, 33).

M. Phillips and C. Gunn (1992). "Visualizing hyperbolic space: unusual uses of 4x4 matrices." In: *Proceedings of the 1992 symposium on Interactive 3D graphics*. Ed. by M. Levoy, E. Catmull, and D. Zeltzer. I3D '92. Cambridge, Massachusetts, United States: ACM, pp. 209–214 (cit. on p. 23).

H. Poincaré (1904). "Cinquième complément à l'analysis situs." *Rendiconti del Circolo Matematico di Palermo (1884-1940)* 18.1, pp. 45–110 (cit. on p. 1).

H. Poincaré (1895). *Analysis situs*. Translation to English by John Stillwell in Poincaré (2010). Gauthier-Villars (cit. on pp. 1, 75).

— (2010). *Papers on topology*. Vol. 37. History of Mathematics. Translation from the original in French in Poincaré (1895) by John Stillwell. American Mathematical Society, Providence, RI; London Mathematical Society, London, pp. xx+228. MR: 2723194 (cit. on p. 75).

C. Y. Ren and I. Reid (2011). "gSLIC: a real-time implementation of SLIC superpixel segmentation." *University of Oxford, Department of Engineering, Technical Report*.

E. Riegel, T. Indinger, and N. A. Adams (2009). "Implementation of a Lattice–Boltzmann method for numerical fluid mechanics using the nVIDIA CUDA technology." *Computer Science-Research and Development* 23.3–4, pp. 241–247.

Z. Rogue (2020). *Experiments with geometry* (cit. on p. 13).

J. Sanders and E. Kandrot (2010). *CUDA by example: an introduction to general-purpose GPU programming, portable documents*. Addison-Wesley Professional.

P. Scott (1983). "The geometries of 3-manifolds." *Bulletin of the London Mathematical Society* 15.5, pp. 401–487. MR: 0705527. Zbl: 0561.57001 (cit. on pp. 5, 8).

G. Sellers and J. Kessenich (2016). *Vulkan Programming Guide: The Official Guide to Learning Vulkan*. Always learning. Addison-Wesley (cit. on p. 28).

L. J. S. Silva, D. L. S. da Silva, A. B. Raposo, L. Velho, and H. C. V. Lopes (2019). "Tensorpose: Real-time pose estimation for interactive applications." *Computers & Graphics* 85, pp. 1–14 (cit. on pp. 57, 58).

V. da Silva and L. Velho (2020). "Ray-VR: Ray Tracing Virtual Reality in Falcor." arXiv: 2006.11348.

M. Skrodzki (2020). "Illustrations of non-Euclidean geometry in virtual reality." arXiv: 2008.01363 (cit. on p. 13).

S. Smale (2007). "Generalized Poincaré's conjecture in dimensions greater than four." In: *Topological Library: Part 1: Cobordisms and Their Applications*. Ed. by S. P. Novikov and I. A. Taymanov. World Scientific, pp. 251–268 (cit. on p. 2).

J. E. Stone, D. Gohara, and G. Shi (2010). "OpenCL: A parallel programming standard for heterogeneous computing systems." *Computing in science & engineering* 12.3, p. 66.

B. Szilágyi and D. Virosztek (2003). "Curvature and torsion of geodesics in three homogeneous Riemannian 3-geometries." *Studies of the University of Žilina, Math. Ser* 16, pp. 1–7 (cit. on pp. 15, 60).

W. Thurston (1979). *The geometry and topology of three-manifolds*. Princeton University (cit. on pp. 17, 39).

— (1982). "Three dimensional manifolds, Kleinian groups and hyperbolic geometry." *Bulletin of the American Mathematical Society* 6.3, pp. 357–381. MR: 0648524. Zbl: 0496.57005 (cit. on pp. 2, 5–8).

— (1997). *Three-dimensional geometry and topology*. Vol. 35. Princeton university press. MR: 1435975. Zbl: 0873.57001 (cit. on p. 64).

— (1998). "How to see 3-manifolds." *Classical and Quantum Gravity* 15.9, p. 2545. MR: 1649658. Zbl: 0932.57017 (cit. on p. 39).

M. Troyanov (1998). "L'horizon de SOL." *Exposition. Math.* 16.5, pp. 441–479. MR: 1656902. Zbl: 0939.53042 (cit. on p. 60).

L. Velho, L. Carvalho, and D. Lucio (2018). *VR Tour: Guided Participatory Meta-Narrative for Virtual Reality Exploration*. Technical Report TR-06-2018. VISGRAF Lab - IMPA.

L. Velho, T. Novello, V. da Silva, and D. Lucio (2019). *Visualization of Non-Euclidean Spaces using Ray Tracing*. Tech. Report TR-09-2019. Visgraf-Impa.

J. Weeks (2021). "Body coherence in curved-space virtual reality games." *Computers & Graphics* (cit. on p. 57).

J. Weeks (1995). *The Shape of Space*. MR: 1875835. Zbl: 0858.57021.

— (1999). *Snappea*.

— (Nov. 2002a). "Real-Time Rendering in Curved Spaces." *IEEE Comput. Graph. Appl.* 22.6, pp. 90–99 (cit. on p. 47).

— (2002b). "Real-time rendering in curved spaces." *IEEE Computer Graphics and Applications* 22.6, pp. 90–99 (cit. on p. 22).

— (2006). "Real-Time Animation in Hyperbolic, Spherical, and Product Geometries." In: *Non-Euclidean Geometries*. Ed. by A. Prékopa and E. Molnár. Vol. 581. Mathematics and Its Applications. Springer US, pp. 287–305. MR: 2191253. Zbl: 1116.68653 (cit. on p. 12).

— (2020a). "Non-Euclidean billiards in VR." In: *Proceedings of Bridges 2020: Mathematics, Art, Music, Architecture, Education, Culture*. Ed. by C. Yackel, R. Bosch, E. Torrence, and K. Fenyvesi. Phoenix, Arizona: Tessellations Publishing, pp. 1–8 (cit. on p. 20).

— (2020b). *The shape of space*. 3rd ed. CRC Press. MR: 0806764. Zbl: 1432.57001 (cit. on p. 8).

— (2020c). "Virtual reality simulations of curved spaces." arXiv: 2011.00510 (cit. on p. 20).

D. Weiskopf (2001). "Visualization of four-dimensional spacetimes."

T. Whitted (June 1980). "An Improved Illumination Model for Shaded Display." *Commun. ACM* 23.6, pp. 343–349 (cit. on pp. 37, 40).

Wikipedia (2010). *4K resolution standard*.

C. Wyman and A. Marrs (2019). "Introduction to DirectX Raytracing." In: *Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other APIs*. Ed. by E. Haines and T. Akenine-Möller. Berkeley, CA, USA: Apress, pp. 21–47 (cit. on pp. 27, 28, 44).

# *Index*

# Títulos Publicados — 33º Colóquio Brasileiro de Matemática

**Geometria Lipschitz das singularidades** – *Lev Birbrair e Edvalter Sena*

**Combinatória** – *Fábio Botler, Maurício Collares, Taísa Martins, Walner Mendonça, Rob Morris e Guilherme Mota*

**Códigos geométricos, uma introdução via corpos de funções algébricas** – *Gilberto Brito de Almeida Filho e Saeed Tafazolian*

**Topologia e geometria de 3-variedades, uma agradável introdução** – *André Salles de Carvalho e Rafał Marian Siejakowski*

**Ciência de dados: algoritmos e aplicações** – *Luerbio Faria, Fabiano de Souza Oliveira, Paulo Eustáquio Duarte Pinto e Jayme Luiz Szwarcfiter*

**Discovering Poncelet invariants in the plane** – *Ronaldo A. Garcia e Dan S. Reznik*

**Introdução à geometria e topologia dos sistemas dinâmicos em superfícies e além** – *Víctor León e Bruno Scárdua*

**Equações diferenciais e modelos epidemiológicos** – *Marlon M. López-Flores, Dan Marchesin, Vítor Matos e Stephen Schecter*

**Differential Equation Models in Epidemiology** – *Marlon M. López-Flores, Dan Marchesin, Vítor Matos e Stephen Schecter*

**A friendly invitation to Fourier analysis on polytopes** – *Sinai Robins*

**PI-álgebras: uma introdução à PI-teoria** – *Rafael Bezerra dos Santos e Ana Cristina Vieira*

**First steps into Model Order Reduction** – *Alessandro Alla*

**The Einstein Constraint Equations** – *Rodrigo Avalos e Jorge H. Lira*

**Dynamics of Circle Mappings** – *Edson de Faria e Pablo Guarino*

**Statistical model selection for stochastic systems with applications to Bioinformatics, Linguistics and Neurobiology** – *Antonio Galves, Florencia Leonardi e Guilherme Ost*

**Transfer operators in Hyperbolic Dynamics - an introduction** – *Mark F. Demers, Niloofar Kiamari e Carlangelo Liverani*

**A course in Hodge Theory: Periods of Algebraic Cycles** – *Hossein Movasati e Roberto Villaflor Loyola*

**A dynamical system approach for Lane-Emden type problems** – *Liliane Maia, Gabrielle Nornberg e Filomena Pacella*

**Visualizing Thurston's geometries** – *Tiago Novello, Vinícius da Silva e Luiz Velho*

**Scaling problems, algorithms and applications to Computer Science and Statistics** – *Rafael Oliveira e Akshay Ramachandran*

**An introduction to Characteristic Classes** – *Jean-Paul Brasselet*

impa

Instituto de
Matemática
Pura e Aplicada