Modeling and Rendering Non-Euclidean Spaces approximated with Concatenated Polytopes

SEUNG-WOOK KIM, Korea University, South Korea
JAEHYUNG DOH, Korea University, South Korea
JUNGHYUN HAN, Korea University, South Korea

A non-Euclidean space is characterized as a manifold with a specific structure that violates Euclid's postulates. This paper proposes to approximate a manifold with polytopes. Based on the scene designer's specification, the polytopes are automatically concatenated and embedded in a higher-dimensional Euclidean space. Then, the scene is navigated and rendered via novel methods tailored to concatenated polytopes. The proof-of-concept implementation and experiments with it show that the proposed methods bring the virtual-world users unusual and fascinating experiences, which cannot be provided in Euclidean-space applications.

CCS Concepts: • **Computing methodologies** → **Ray tracing**; *Rasterization*; Mesh geometry models.

Additional Key Words and Phrases: Non-Euclidean space, Hypergraphics

ACM Reference Format:
Seung-wook Kim, Jaehyung Doh, and JungHyun Han. 2022. Modeling and Rendering Non-Euclidean Spaces approximated with Concatenated Polytopes. *ACM Trans. Graph.* 41, 4, Article 78 (July 2022), 13 pages. https://doi.org/10.1145/3528223.3530186

1 INTRODUCTION

Euclid, often referred to as the "founder of geometry," stated the following postulates:

- There is a unique straight line passing through two points.
- A straight line extends indefinitely in either direction.
- A circle may be drawn with any given center and radius.
- All right angles are equal.
- Given a point not on a given line, there exists a unique line through the point that is parallel to the given line.

In this paper, we call a space *non-Euclidean* if any of Euclid's postulates is violated in the space. Fig. 1-(a) shows a simplest non-Euclidean space example, where a room with a red cube and another room with a blue tetrahedron are repeatedly connected. This is often called a *periodic space*. A user starting in the red room walks straight through the blue room to return to the red room. As illustrated in Fig. 1-(b), the first postulate of Euclid is violated. (In the *curved spaces* such as hyperbolic and elliptic geometries, all Euclid's postulates hold except the fifth, which is called the parallel postulate.)

Authors' addresses: Seung-wook Kim, Korea University, Seoul, South Korea, wook0249@korea.ac.kr; Jaehyung Doh, Korea University, Seoul, South Korea, djhdjh@korea.ac.kr; JungHyun Han, Korea University, Seoul, South Korea, jhan@korea.ac.kr.

0730-0301/2022/7-ART78 $15.00
https://doi.org/10.1145/3528223.3530186

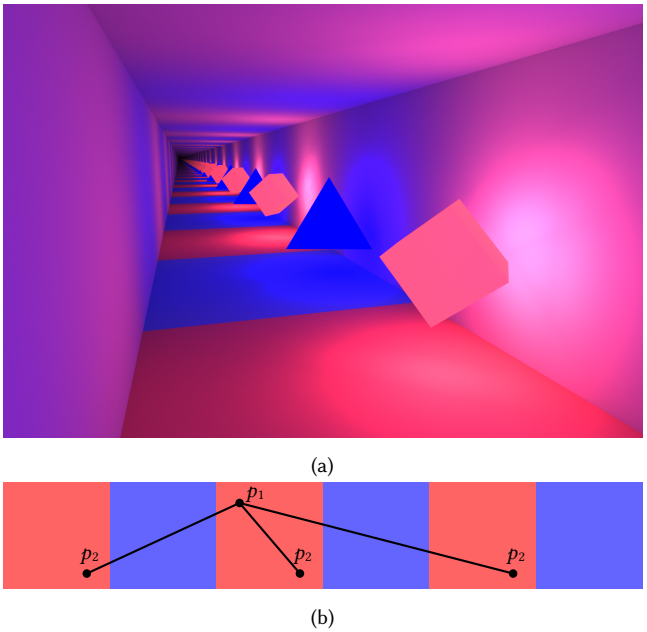


Fig. 1. A non-Euclidean space example: (a) A red room and a blue room are repeatedly connected. (b) The top view of the space shows that two points, p_1 and p_2, are connected by multiple lines.

Being unimplementable in the physical world, non-Euclidean spaces can bring strange but fascinating experiences to the users in the virtual world. Thus, a growing number of non-Euclidean games have attracted attention. A well-known example is *Portal* [Valve 2007]. As will be presented in Section 2, however, the modeling and rendering techniques adopted in such games are largely *ad hoc*, lacking mathematical backgrounds and methodologies.

In our study, an m-dimensional (simply, m-D) non-Euclidean space is characterized as the same dimensional *manifold* (simply, m-manifold) with a specific structure that violates Euclid's postulates. This paper first presents the general methods for rendering m-manifolds embedded in n-D Euclidean spaces, where $m < n$ ($m, n \in \mathbb{Z}^+$). Unfortunately, it is computationally infeasible to render the manifolds themselves. Therefore, we propose to approximate a manifold with a concatenation of *polytopes*. This paper presents rigorous methods for modeling and rendering such concatenated polytopes. As 3-manifolds allow us to create a variety of interesting virtual worlds, our experiments are focused on concatenated 3-polytopes embedded in 4D or higher-dimensional Euclidean spaces. The experiment results are promising and show the feasibility of modeling and rendering non-Euclidean spaces.

2 RELATED WORK

In computer graphics fields, there have been efforts for visualizing non-Euclidean 3D spaces. Silva [2018] proposed a technique for rendering 3D manifolds embedded in 4D Euclidean space. Novello et al. [2020a] implemented global illumination in non-Euclidean spaces, where the distance metric is analytically defined. On the other hand, several works for rendering with nonlinear rays have been reported. Weiskopf et al. [2004] and Ayush and Chaudhuri [2017] proposed rendering techniques using rays bent by gravitational forces from a black hole. Falk et al. [2007] generated panorama maps using curved rays to visualize various heights of terrain. Cavallo [2021] proposed successive conversions for dimension reduction in the context of generating a 2D screen for 4D Euclidean space. In immersive virtual environments, several methods were proposed for rendering non-Euclidean spaces, whose geometries are a subset of Thurston's geometries [Coulon et al. 2020a,b; Hart et al. 2017a,b, 2015]. Novello et al. [2020b] also proposed a rendering method in the spaces of Nil, Sol and $SL_2(\mathbb{R})$.

A few video games have been popular, which tried to visualize non-Euclidean spaces. They include *Portal* [Valve 2007] and *Antichamber* [Demruth 2013]. In reality, however, they are not non-Euclidean applications. Instead, they mimic connected virtual worlds using the so-called portals and mainly support teleportation between the worlds.

In the context of anisotropic meshing, many methods have been proposed to embed manifolds in higher-dimensional spaces. Zhong et al. [2013] proposed a particle-based meshing technique that utilizes the inter-particle energy optimization problem in a high dimensional embedding space. In contrast, explicit embedding methods have also been proposed [Dassi et al. 2014, 2015; Lévy and Bonneel 2013; Panozzo et al. 2014]. These methods suffer from self-intersection problems. Zhong et al. [2018] proposed to overcome the problems with a method for embedding continuous manifolds in high dimensional spaces using Riemannian metrics.

For reduction of embedding space's dimension, Baramiuk and Wakin [2009] showed an orthogonal linear projection of a manifold into a sufficiently high dimensional random subspace while approximately preserving all geodesic distances. Verma [2012] proposed the distance-preserving embedding method for any dimensional manifold using machine learning.

3 NAVIGATION AND RENDERING IN MANIFOLDS

Our methods support m-D non-Euclidean spaces embedded in n-D Euclidean spaces, where $m < n$ $(m, n \in \mathbb{Z}^+)$. An m-D non-Euclidean space is simply an m-manifold, which is defined as a topological space that locally resembles m-D Euclidean space.

We first present 2-manifold embedded in 3D Euclidean space in Section 3.1. It is easy to visualize and helps understand our methods for handling non-Euclidean spaces. In Section 3.2, it is extended to 3-manifold embedded in 4D or higher-dimensional Euclidean spaces.

3.1 Two-manifold in 3D Euclidean Space

Fig. 2-(a) shows an example of 2-manifold embedded in 3D Euclidean space, where the blue arrows represent the normals of the manifold. The basic *rendering primitive* in 2-manifolds is line segment. (That in 3-manifolds is polygon, as will be presented in Section 3.2.) Fig. 2-(b) shows a line segment in green (on the right). Suppose that the camera is located at **o**. For rendering the line segment, we first define the *camera space* with its origin at **o**.

Camera-space basis. In Fig. 2-(b), the red arrow represents the unit vector along the *view direction*, which we denote as **view**. The cross product of the manifold normal (in blue) and **view** defines what we denote as **left** (in black). Together with **view**, it composes the orthonormal basis of the 2D camera space.

Camera position. When a camera navigates a 2-manifold, its *moving direction* may be different from the view direction and is described by its *velocity*. For each frame, the camera is advanced by the velocity. Fig. 2-(c) shows the cross-section view of the manifold, and colored in red is the velocity. As the red arrow's tip is not necessarily on the manifold, it is *projected* onto the manifold.

Rendering. A set of camera-space *rays* are fired from **o** toward the 2-manifold scene (composed of a single line segment in the example). As depicted in Fig. 2-(d), each ray *marches*. For each marching step, the ray is tested if it hits the line segment. If it hits nothing, the ray is *projected* onto the manifold. Taking the projected point as the origin, a tangent space is computed, and the ray's direction is redefined in the tangent space for the next marching step.

Unless the marching step size is sufficiently small, however, the ray may move past the line segment. To resolve this problem, we extrude the line segment infinitely along the normal $\mathbf{n}$ and its opposite direction, as shown in Fig. 2-(d). If the ray of the 'current' marching step hits the infinite rectangle in the 3D Euclidean space, it is determined to hit the line segment in the 2-manifold.

In Fig. 2-(e), $\mathbf{p}_0$ and $\mathbf{p}_1$ denote the end points of the line segment. The normals at $\mathbf{p}_0$ and $\mathbf{p}_1$ are interpolated to define $\mathbf{n}$. Let $\mathbf{s}$ denote the start position of the current ray and $\mathbf{r}$ denote the unit vector along the ray. Then, solving the following equations, we obtain α, β and γ:

$$\mathbf{p}_0 + \alpha\,(\mathbf{p}_1 - \mathbf{p}_0) + \beta\mathbf{n} = \mathbf{s} + \gamma\mathbf{r} \tag{1}$$

If $\alpha \in [0, 1]$ and $\gamma > 0$, the ray is determined to hit the rectangle and equivalently the line segment. (Illuminating the point hit by the ray will be presented in Section 4.2.)

The above equations are not always solvable. For example, if $\mathbf{r}$ and $\mathbf{n}$ are parallel, the equations become singular. Then, the least squares method is used to obtain approximated solutions.

3.2 Three-manifold in 4D or Higher-dimensional Euclidean Spaces

Even though 2-manifolds are easy to understand, they leave little room for creating meaningful content. In contrast, 3-manifolds are not easy to understand but allow us to create a variety of interesting virtual worlds. Therefore, the experiments reported in Section 6 focus on 3-manifolds.

Before diving into 3-manifolds, let us review how the camera pose is updated in Euclidean-space applications. In PC games, for example, the camera's view directions are updated typically using mouses whereas its moving directions are updated typically using the arrow keys of the keyboard.

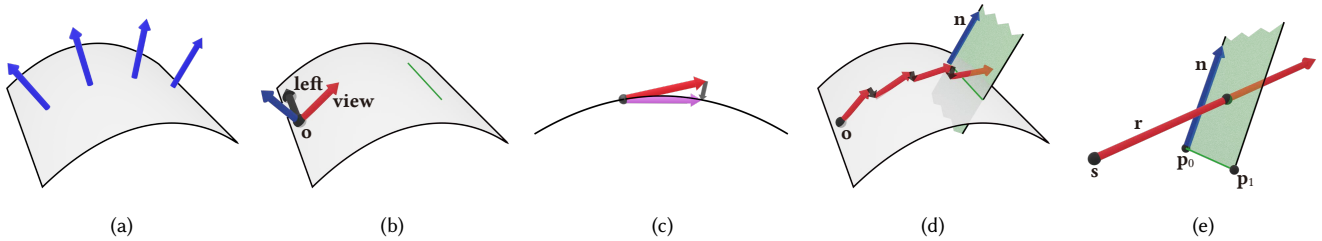


Fig. 2. Two-manifold embedded in 3D Euclidean space: (a) Two-manifold and its normal vectors. (b) The camera-space basis, {**left**, **view**}, and the rendering primitive, line segment. (c) The camera is advanced using its velocity (red vector) and then projected onto the manifold, i.e., purple vector = red vector + gray vector, so that the camera is confined to the manifold. (d) Marching ray and the infinite rectangle extruded from the line segment. (e) In the current marching step, the ray hits the infinite rectangle.

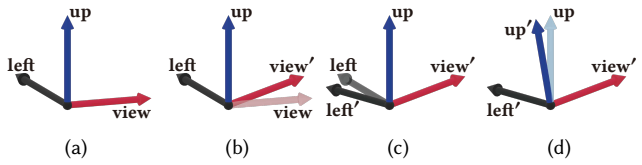


Fig. 3. Steps for updating the camera-space basis: (a) The original camera-space basis is composed of three orthonormal vectors, **left**, **up** and **view**. (b) Small fractions of **left** and **up** are added to **view** to define **view′**. (c) Then, **left** is changed to **left′**: **left′** = *normalize*(**up** × **view′**). (d) Finally, **up** is changed to **up′**: **up′** = **view′** × **left′**.

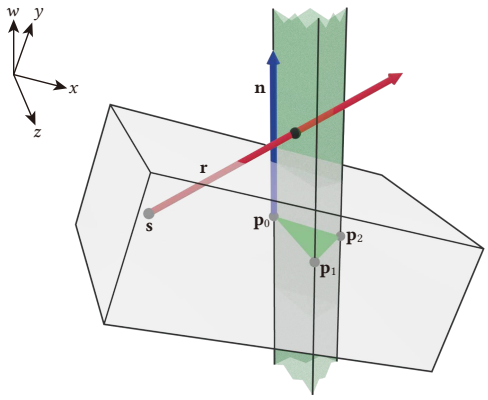


Fig. 4. Every ray marches in 3-manifold embedded in 4D Euclidean space. In each marching step, the ray is tested if it hits the infinite prisms.

The camera-space basis is composed of three orthonormal vectors, which we name **left**, **up** and **view**. See Fig. 3-(a). If the user looks slightly to the left and upward (using mouses), for example, small fractions of **left** and **up** are added to **view** to update it, as shown in Fig. 3-(b), where the updated vector is normalized to define **view′**. Then, the cross product of **up** and **view′** defines **left′**. See Fig. 3-(c). Finally, the cross product of **view′** and **left′** is taken as **up′**, as shown in Fig. 3-(d), so that **left′**, **up′** and **view′** make up the new basis. On the other hand, the camera's moving direction provided by the user (using arrow keys) is mapped to a velocity vector, by which the camera is advanced.

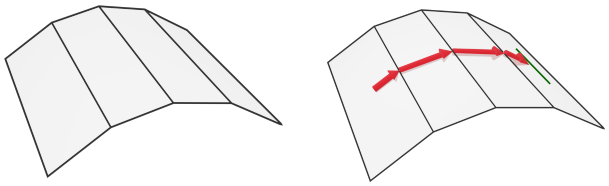


Fig. 5. Two-manifold embedded in 3D Euclidean space: (a) Concatenated 2-polytopes. (b) Ray marching in the polytopes.

Now, we will present camera pose update and rendering in "3-manifold embedded in 4D Euclidean space." Recall that all operations for handling "2-manifold embedded in 3D Euclidean space," e.g., the cross-product operation used to define 2D camera-space basis, are made in the embedding space, which is 3D. Similarly, all operations for 3-manifold will be made in the 4D Euclidean space.

Camera-space basis. Just as **left**, **up** and **view** are initialized with some default vectors in most Euclidean-space applications, so are those in "3-manifold embedded in 4D Euclidean space." The difference is that **left**, **up** and **view** are now 4D vectors. If the user looks slightly to the left and upward (e.g., using mouses), we add small fractions of **left** and **up** to **view** to define **view′**. Next, we should compute **left′** and **up′**. For this, we use the cross-product operation for 4D vectors, as in Blinn's work [2003]. The cross product of three 4D vectors (**a**, **b** and **c**) is defined as follows:

$$cross(\mathbf{a}, \mathbf{b}, \mathbf{c}) = (d_x, d_y, d_z, d_w) \tag{2}$$

where

$$d_x = \begin{vmatrix} a_y & a_z & a_w \\ b_y & b_z & b_w \\ c_y & c_z & c_w \end{vmatrix}, \quad d_y = \begin{vmatrix} a_z & a_w & a_x \\ b_z & b_w & b_x \\ c_z & c_w & c_x \end{vmatrix},$$

$$d_z = \begin{vmatrix} a_w & a_x & a_y \\ b_w & b_x & b_y \\ c_w & c_x & c_y \end{vmatrix}, \quad d_w = \begin{vmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{vmatrix}$$

Let **normal** denote the normal of 3-manifold at the current position. Then, *cross*(**up**, **view′**, **normal**) defines **left′**, and *cross*(**view′**,

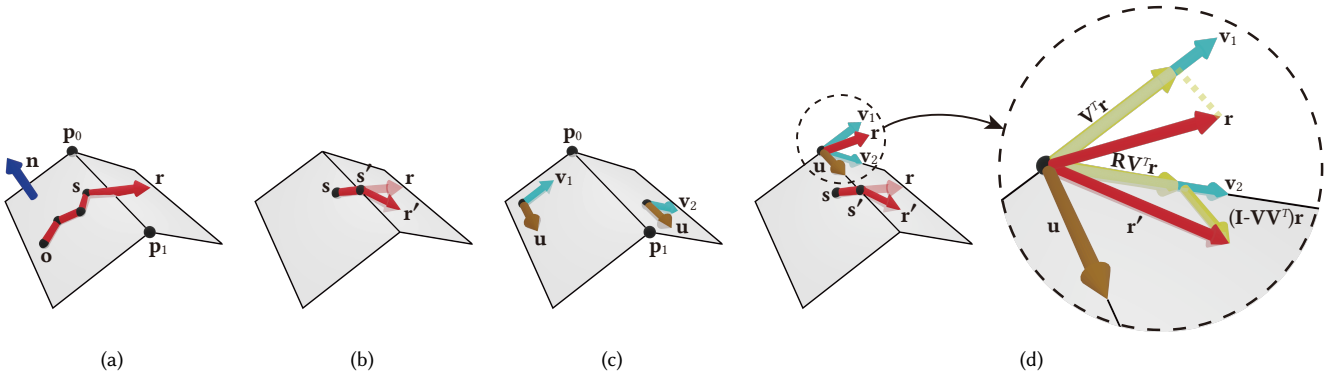


Fig. 6. Crossing over polytopes: (a) The camera at $\mathbf{s}$ moves with $\mathbf{r}$ to cross over the polytopes. (b) The new position, $\mathbf{s}'$, and the new moving direction, $\mathbf{r}'$, are computed for navigating the right polytope. (c) The bases of two polytopes share $\mathbf{u}$. (d) When crossing over the polytopes, $\mathbf{r}$ is rotated to $\mathbf{r}'$ (Equation (6)).

normal, **left**′) defines **up**′. The new camera-space basis is composed of **left**′, **up**′ and **view**′.

Camera position. As in general Euclidean-space applications, the moving direction provided by user is mapped to a velocity vector. As the basis vectors are all 4D, the velocity is also 4D. The camera is advanced by the 4D linear velocity and then projected onto the 3-manifold.

Rendering. The basic rendering primitive in 3-manifold is polygon or triangle. A set of camera rays are fired toward the 3-manifold scene. Each ray *marches* and is tested if it hits the rendering primitives. If not, the ray is *projected* onto the manifold and marches again.

Fig. 4 visualizes a 3-manifold as a box, where a triangle in it (with three vertices, $\mathbf{p}_0$, $\mathbf{p}_1$ and $\mathbf{p}_2$) is extruded along $\mathbf{n}$ (and its opposite direction) to define an infinite prism. The 'current' ray is fired from $\mathbf{s}$ with the direction $\mathbf{r}$. We solve the following equations of 4D points and 4D vectors, which are extended from Equation (1):

$$\mathbf{p}_0 + \alpha_1 (\mathbf{p}_1 - \mathbf{p}_0) + \alpha_2 (\mathbf{p}_2 - \mathbf{p}_0) + \beta\mathbf{n} = \mathbf{s} + \gamma\mathbf{r} \tag{3}$$

If both α_1 and $\alpha_2 \in [0, 1]$, $\alpha_1+\alpha_2 \leq 1$ and $\gamma > 0$, the ray is determined to hit the prism and equivalently the triangle.

The embedding space is not restricted to 4D. For example, a few 3-manifolds reported in Section 6 are embedded in 5D Euclidean spaces. When m-manifolds are embedded in n-D Euclidean spaces, $n - m$ normals are defined in the n-D space. On the other hand, the rendering primitives in 3-manifolds are not restricted to triangles but include 0-, 1- and 2-*simplices*, which correspond to a point, a line segment and a triangle, respectively. The rendering primitive in m-manifold is generally l-simplex, where l is either 0, 1, .. , or $m - 1$. For the ray hit test, the l-simplex is extruded along $n - m$ normals (and their opposite directions) to make an $n - m$ dimensional higher object. Then, the ray hit test expressed in Equation (3) is extended as follows:

$$\mathbf{p}_0 + \sum_{i=1}^{l} \alpha_i(\mathbf{p}_i - \mathbf{p}_0) + \sum_{j=1}^{n-m} \beta_j \mathbf{n}_j = \mathbf{s} + \gamma\mathbf{r} \tag{4}$$

where $\mathbf{p}_i$ represent the vertices of the l-simplex, and $\mathbf{n}_j$ represent $n - m$ normals. If every $\alpha_i \in [0, 1]$, $\sum_{i=1}^{l} \alpha_i \leq 1$ and $\gamma > 0$, the ray is determined to hit the l-simplex. (Appendix A gives more discussions on m-manifolds embedded in n-D spaces.)

4 NAVIGATION AND RENDERING IN POLYTOPES

The major computational bottleneck of the methods presented in Section 3 lies in *ray marching* (as shown in Fig. 2-(d) for 2-manifolds). For each marching step, we solve Equation (4) for every rendering primitive, and we also project the ray onto the manifold if it hits nothing. In order to reduce the computational cost, we approximate an m-manifold with a concatenation of m-D *polytopes* (henceforth, m-polytopes). Fig. 5-(a) shows a 2-manifold approximated with four concatenated 2-polytopes. Given the polytopes, a single step of ray marching is made per polytope, as illustrated in Fig. 5-(b). No projection is needed. Unlike in Fig. 2-(c), advancing the camera is not followed by projection either. This section presents camera navigation and rendering in concatenated polytopes, and Section 5 presents how the polytopes are concatenated.

4.1 Camera Navigation

Fig. 6-(a) depicts a camera's path on concatenated 2-polytopes. Currently, the camera located at $\mathbf{s}$ is moving to the right polytope with the velocity $\mathbf{r}$. The intersection between $\mathbf{r}$ and the left polytope's edge can be computed using Equation (1), where $\mathbf{p}_0$ and $\mathbf{p}_1$ are the edge's end points and $\mathbf{n}$ is the current polytope's normal. Among α, β and γ obtained by solving Equation (1), β may not be zero due to computational errors but must be very close to zero.

Once the intersection is computed, it is taken as new position of the camera (denoted as $\mathbf{s}'$ in Fig. 6-(b)) and camera navigation is resumed in the right polytope. For this, the velocity needs to be recomputed unless the adjacent polytopes are parallel. It is denoted as $\mathbf{r}'$ in Fig. 6-(b).

Fig. 6-(c) shows the orthonormal bases of two polytopes, $\{\mathbf{u}, \mathbf{v}_1\}$ and $\{\mathbf{u}, \mathbf{v}_2\}$. They share $\mathbf{u}$, which is obtained by normalizing the vector connecting $\mathbf{p}_0$ and $\mathbf{p}_1$. In Fig. 6-(d), the basis vectors and $\mathbf{r}$ are relocated for exposition purposes. See the dotted circle. A 2D space is spanned by $\mathbf{v}_1$ and $\mathbf{v}_2$. Let $\mathbf{V} = (\mathbf{v}_1\ \mathbf{v}_2)$. It is a *basis-change* matrix from the 2D space to the 3D embedding space.

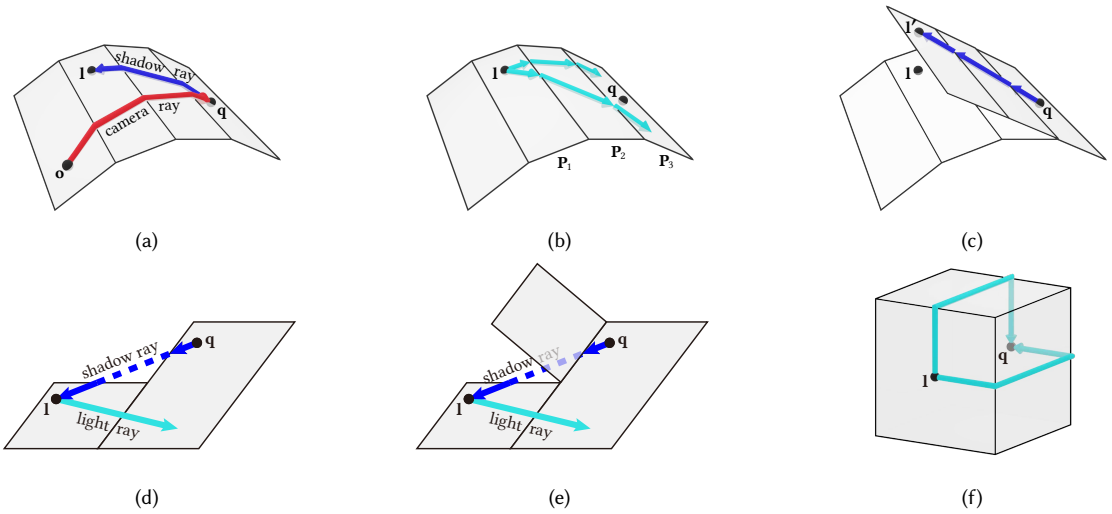


Fig. 7. Illumination in non-Euclidean spaces: (a) In general, the path between a surface point, $\mathbf{q}$, and a light source, $\mathbf{l}$, is not straight. (b) The light rays from $\mathbf{l}$ classify the polytope, $\mathbf{P}_3$, as lightable using $\mathbf{l}$. (c) Before flattening the polytopes, the shadow ray is not straight. It is hard to define such a shadow ray. By flattening the polytopes and transforming the light source from $\mathbf{l}$ to $\mathbf{l}'$, the shadow ray becomes straight. (d) Even if a polytope is lightable, the shadow ray may not reach the light source but proceed to an empty space. (e) The shadow ray may enter an invalid polytope. (f) In this example, $\mathbf{l}$ and $\mathbf{q}$ are located in the opposite sides of a cube, and there are multiple paths between them.

The dotted circle of Fig. 6-(d) shows that $\mathbf{r}$ is projected onto $\mathbf{v}_1$ by $\mathbf{V}^T$. The projected, $\mathbf{V}^T\mathbf{r}$, is rotated by $\mathbf{R}$ presented below so that it is aligned with $\mathbf{v}_2$:

$$\mathbf{R} = \begin{pmatrix} \cos\frac{\pi}{2} & -\sin\frac{\pi}{2} \\ \sin\frac{\pi}{2} & \cos\frac{\pi}{2} \end{pmatrix} \tag{5}$$

Finally, the rotated, $\mathbf{R}\mathbf{V}^T\mathbf{r}$, is given the "3D representation" by $\mathbf{V}$, i.e., $\mathbf{V}\mathbf{R}\mathbf{V}^T\mathbf{r}$ represents the 3D vector aligned with $\mathbf{v}_2$. Let us call it $\mathbf{r}'_0$.

On the other hand, note that $\mathbf{V}\mathbf{V}^T\mathbf{r}$ is the "3D representation" of $\mathbf{r}$'s projection onto $\mathbf{v}_1$. Then, the vector connecting it and the original $\mathbf{r}$ is represented as $(\mathbf{I}-\mathbf{V}\mathbf{V}^T)\mathbf{r}$, where $\mathbf{I}$ is the identity matrix. Adding it to $\mathbf{r}'_0$, we obtain $\mathbf{r}'$:

$$\begin{aligned} \mathbf{r}' &= \mathbf{r}'_0 + (\mathbf{I} - \mathbf{V}\mathbf{V}^T)\mathbf{r} \\ &= \mathbf{V}\mathbf{R}\mathbf{V}^T\mathbf{r} + (\mathbf{I} - \mathbf{V}\mathbf{V}^T)\mathbf{r} \end{aligned} \tag{6}$$

It is the velocity (for camera's moving direction) in the right polytope.

It is straightforward to extend the above methods (for computing $\mathbf{s}'$ and $\mathbf{r}'$) to m-polytopes embedded in n-D Euclidean space. The boundary of an m-polytope is composed of $(m-1)$-polytopes. We call them *facets*. Two m-polytopes are connected at an $(m-1)$-D facet. It is decomposed into $(m-1)$-simplices. (For example, a quad facet connecting two cubes is decomposed into two triangles.) The intersection between $\mathbf{r}$ and each simplex is computed using Equation (4). It defines the n-D position, $\mathbf{s}'$.

When two m-polytopes are connected at an $(m-1)$-D facet, the basis of each m-polytope is composed of "the basis of the facet" plus an additional vector. Let us denote the additional vectors as $\mathbf{v}_1$ (of the current polytope) and $\mathbf{v}_2$ (of the next polytope). Then, Equations (5) and (6) are used to compute $\mathbf{r}'$. It is an n-D vector.

4.2 Rendering

In principle, rendering is done via *ray tracing*. For each ray, only a single step of ray marching is made per polytope, as illustrated in Fig. 5-(b). When a camera ray crosses over polytopes, its direction is changed in the same manner as the camera's moving direction (presented in Fig. 6).

Suppose that a camera ray hits a rendering primitive. In 7-(a), $\mathbf{q}$ denotes the point hit by the ray. To illuminate $\mathbf{q}$, the *shadow ray* must be fired from $\mathbf{q}$ to the light source, denoted as $\mathbf{l}$. In non-Euclidean spaces, however, it is a challenge to define the shadow ray because its geodesic is not straight in general.

In order to tackle the challenge, we pre-process the light sources in the scene. Assuming that they are all static, uniformly-sampled omnidirectional *light rays* are fired from each light source. Fig. 7-(b) depicts two light rays from $\mathbf{l}$. (When crossing over the polytopes, the light ray's direction is changed in the same manner as the camera's moving direction.)

When a light ray from $\mathbf{l}$ visits a polytope, it is classified as "*lightable* using $\mathbf{l}$" and the previously visited polytopes are stored in a list which we call the *light-ray path* to the lightable polytope. In Fig. 7-(b), $\mathbf{P}_3$ is lightable using $\mathbf{l}$ and the light-ray path is the list of $\mathbf{P}_1$, $\mathbf{P}_2$ and $\mathbf{P}_3$. Only when the point hit by the camera ray, $\mathbf{q}$, resides in the polytope that is lightable using $\mathbf{l}$, the shadow ray is defined between $\mathbf{q}$ and $\mathbf{l}$.

As all polytopes have zero Gaussian curvatures, we can *flatten* the polytopes included in the light-ray path by using the rotation method presented in Section 4.1. Fig. 7-(c) shows that the light source $\mathbf{l}$ is transformed to $\mathbf{l}'$ through successive rotations, making the shadow ray straight. It is $\mathbf{l}' - \mathbf{q}$.

On the way to $\mathbf{l}'$, the shadow ray visits the flattened polytopes one by one. If it intersects any object in a polytope, we judge that $\mathbf{q}$

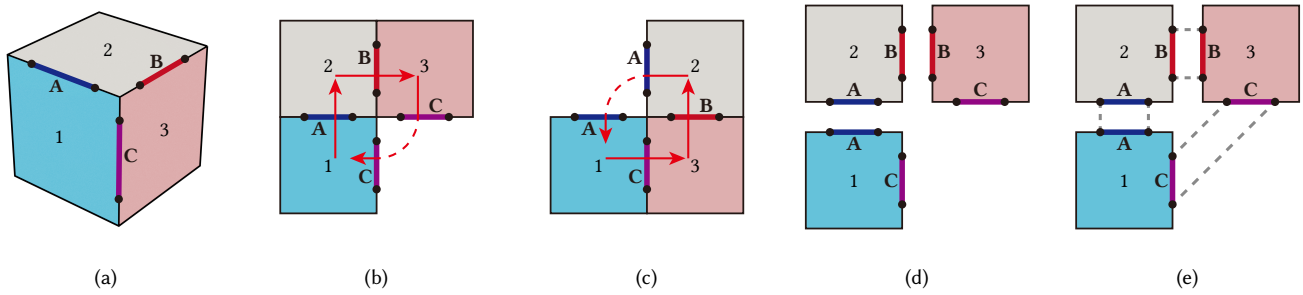


Fig. 8. Concatenating 2-polytopes (Example 1): (a) A 2-manifold house is composed of three 2-polytopes, each of which is considered a 2D room. (b) User's walking path. (c) The reverse path. (d) Three pairs of mating facets (corresponding to doors) are specified manually. (e) The corresponding vertices of the mating facets are connected with dotted links, and their lengths are minimized so that the 2-manifold in (a) is automatically created.

is not lit by **l** and stop the shadow ray. Otherwise, it proceeds to the next connected polytope.

There may be no connected polytopes. Fig. 7-(d) shows an example, where the right polytope is lightable using **l** but the shadow ray leaving the polytope enters an empty space. Then, we stop the shadow ray. Even though there exists a connected polytope, it may not belong to the light-ray path to **q**, as shown in Fig. 7-(e). Then, we judge that the polytope is invalid and stop the shadow ray.

Note that there may be multiple light-ray paths from a single light source to a single point, as illustrated in Fig. 7-(f). Then, each path is processed separately, leading to an independent shadow ray. This makes multiple shadow rays contribute to illuminating **q** with a single light source, **l**. It is a distinguished feature of non-Euclidean spaces.

5 SCENE MODELING: POLYTOPE CONCATENATION

Fig. 8-(a) shows a 2-manifold composed of three 2-polytopes. Assume that each polytope represents a 2D "square" room. Then, the manifold can be considered as a house with three rooms, where Rooms 1 and 2 are connected by a door denoted as **A**, Rooms 2 and 3 are by **B**, and Rooms 1 and 3 are by **C**.

Suppose that a user starts walking from Room 1 and passes through **A**, **B** and **C** in sequence to return to Room 1. Fig. 8-(b) depicts the user's path on the conceptually-unfolded 2-manifold. The user would be surprised to return to Room 1 without walking through an additional room which would be expected to be located at the lower-right quadrant in Fig. 8-(b). Non-Euclidean spaces can bring such novel experiences to the users. Fig. 8-(c) depicts the reverse path, which may also give similar experiences to the users. The three-room house is an example of elliptic geometry, where the interior angles of the triangular path do not sum to 180°.

This section presents the current implementation for concatenating polytopes. It is a proof-of-concept implementation. Its possible extensions will be discussed in Section 7.4.

5.1 Spring Forces and Collision Resolution

As a polytope corresponds to a small world such as a room, we assume that all polytopes are created by scene designers. Fig. 8-(d) shows separate polytopes. When concatenating two polytopes, the scene designer selects a *mating facet* per polytope. The mating facets

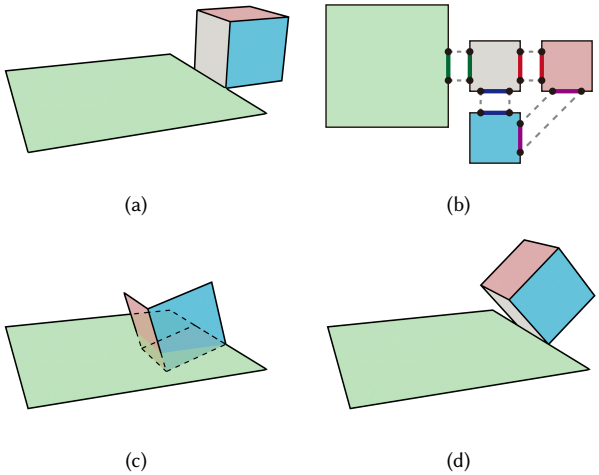


Fig. 9. Concatenating 2-polytopes (Example 2): (a) A three-room house + a front yard. (b) Vertex correspondences for the mating facets, where those for the three rooms are copied from Fig. 8-(e). (c) Spring forces often result in interpenetrating polytopes. (d) Penetration-resolved polytopes.

of two polytopes should be congruent so that they are converted into a door connecting two worlds. Three pairs of mating facets are shown in Fig. 8-(d). The designer also specifies the correspondences between the vertices of the mating facets. In Fig. 8-(e), they are depicted as dotted links.

Given the vertex correspondences, our goal is to find automatically the poses of the rigid polytopes which minimize the links' lengths. An efficient and easy-to-implement way to achieve the goal is to adopt the *mass-spring model*, where the vertex-connecting links are taken as springs so that the polytopes are transformed by the spring forces. The polytopes are assumed to be rigid bodies, and the spring forces follow simple Hooke's law, i.e., the springs' potential energies are defined by the distances between the mating facets' vertices. Using the mass-spring model, we obtain the three-room house given in Fig. 8-(a).

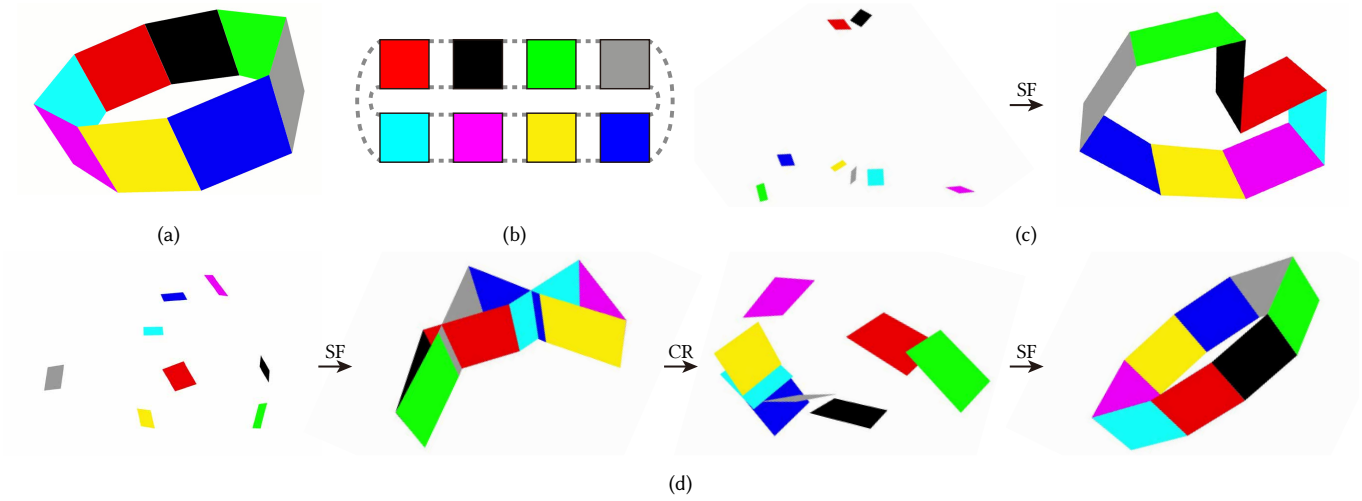


Fig. 10. Spring forces and collision resolution: (a) The 2-manifold band is composed of eight 2-polytopes. (b) Vertex correspondences for the mating facets. (c) The polytopes are successfully concatenated only by the spring forces (SF). (d) Given the initial configuration, which is different from that in (c), the spring forces make the polytopes interpenetrate, as shown in the second image. Collision resolution (CR) separates them, as shown in the third image. Then, the spring forces concatenate them with no interpenetration.

Unfortunately, as the target manifold becomes complicated, the spring forces may make the polytopes interpenetrate. Fig. 9-(a) depicts a 2-manifold for a three-room house (the same as Fig. 8-(a)) with a green front yard. Suppose that as shown in Fig. 9-(b), the scene designer creates four 2-polytopes and specifies the vertex correspondences for the mating facets. Then, the spring forces may generate the result shown in Fig. 9-(c), where the rooms are concatenated to make a house but the house penetrates the front yard.

The interpenetrating polytopes are separated via *collision resolution*. Fig. 9-(d) shows a penetration-free state generated by collision resolution. Even though the manifolds in Fig. 9-(a) and -(d) have different geometries "from the viewpoint of the 3D embedding space," they have the same topology, and therefore the camera navigating the manifolds cannot perceive any differences between them.

Due to collision resolution, however, the polytopes might be separated. Then, we again exert the spring forces on the vertex-connecting links and then invoke collision resolution. Alternating between spring forces and collision resolution is repeated until either the polytopes are concatenated with no interpenetration or the predefined count of iterations is reached.

Fig. 10-(a) shows a target manifold composed of eight 2-polytopes, and Fig. 10-(b) depicts the vertex correspondences for their mating facets. Shown on the left in Fig. 10-(c) is the randomly-generated initial configuration of the polytopes. (The polytopes appear to be small because they are captured from a distance.) The spring forces (abbreviated to SF in the figure) concatenate the polytopes with no interpenetration, as shown on the right in Fig. 10-(c), making it unnecessary to invoke collision resolution. In contrast, Fig. 10-(d) shows that a different initial configuration leads to the sequence of SF, CR (standing for collision resolution) and again SF. As already discussed with Fig. 9-(a) and -(d), the result shown in Fig. 10-(d) is identical to that in Fig. 10-(c) from the non-Euclidean viewpoint.

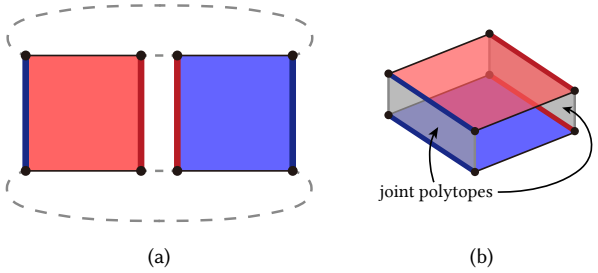


Fig. 11. Concatenating 2-polytopes (Example 3): (a) Mating facets and vertex correspondences. (b) Two joint polytopes are generated to concatenate the polytopes given in (a).

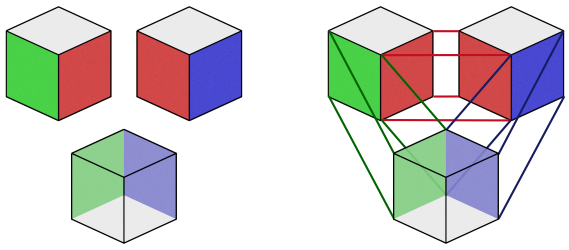
5.2 Joint Polytopes

Concatenated penetration-free polytopes may not be obtained within the predefined iteration count. Consider the example given in Fig. 11-(a). Our goal is to concatenate the red and blue rooms so that a user in the red room walks straight through the blue room to return to the red room. It is a one-dimensional lower version of the periodic space presented in Fig. 1-(a). The spring forces make the polytopes overlap, which are then separated by collision resolution. Unfortunately, this alternation is repeated indefinitely.

When a pair of m-polytopes is not concatenated within the predefined iteration count, the corresponding vertices of two mating facets are connected to create a new m-polytope. We call it a *joint polytope*. Fig. 11-(b) depicts two joint polytopes.

5.3 Embedding in Higher-dimensional Spaces

The joint polytopes may penetrate other polytopes. Then, abandoning the n-D embedding space, we embed the given polytopes in



(a) (b)

Fig. 12. Concatenating 3-polytopes: (a) The mating facets are painted in the same color. (b) Vertex correspondences.

$(n+1)$-D Euclidean space and start the concatenation procedure presented in Sections 5.1 and 5.2. Due to the additional dimension, the chances of interpenetration are significantly reduced. Section 6 shows specific examples that demonstrate how higher-dimensional embedding spaces help polytope concatenation.

If embedding in the $(n+1)$-D Euclidean space does not work, we move to $(n+2)$-D space and resume the concatenation procedure. This is repeated until we obtain concatenated penetration-free polytopes.

The difference between the run-time performance in n-D embedding space and that in a higher-dimensional space is negligible. For example, consider the dot product operations used in illumination and the cross product operations used in camera navigation. There is no big difference between the operations with n-D vectors and those with higher-dimensional ones.

5.4 Concatenating m-polytopes in n-D Euclidean Spaces

Our concatenation procedure was presented with 2-polytopes just because they are easy to visualize. The same procedure can be used for concatenating arbitrary m-polytopes. Fig. 12-(a) shows 3-polytopes, which represent three cuboid rooms. Once the mating facets and vertex correspondences are specified, the concatenation procedure will generate a 3-manifold, which represents a three-room house embedded in 4D Euclidean space. Section 6 will present such a 3-manifold house.

Recall that the rigid motions caused by the spring forces are made "in the embedding space." For example, the rigid motions made in Fig. 12 are 4D. In contrast, the rigid motions in Fig. 9 are 3D. If the embedding space were changed to 4D, however, the rigid motions would also be 4D. These imply that we need a generalized n-D rigid-body simulator that works in n-D embedding space. In the current implementation, we use the n-D simulation method proposed by Bosch [2020].

In order for an m-polytope to be moved by such an n-D rigid-body simulator, its dimension should be extended to n-D. The number of normals of the m-polytope is $n-m$ (as discussed in Section 3.2), and therefore the m-polytope is slightly extruded along the $n-m$ normals to become n-D.

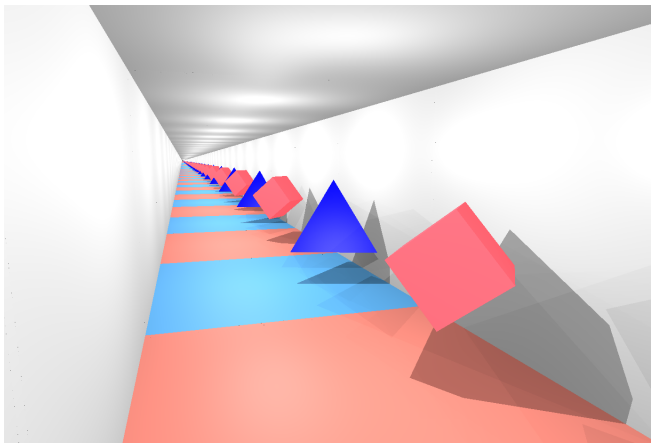


Fig. 13. The scene of Fig. 1-(a) is re-rendered using Phong lighting.

Fig. 14. Three-room house (3-manifold embedded in 4D Euclidean space).

6 EXAMPLES AND EXPERIMENTAL RESULTS

This section reports the results of our experiments made on an AMD Ryzen 7 3800X 3.9 GHz processor and an NVIDIA GeForce RTX 3090 with 32 GB memory. We also use CUDA for hardware acceleration. We will present five scenes, and three among them (presented in Fig. 14, 16 and 18) are designed to be similar to those on YouTube [CodeParade 2018], which motivated our research together with the portal-based games introduced in Section 2. The differences in the methods will be presented in Section 7.5.

For rendering, we use three techniques. *Phong lighting* and *ray tracing* are for real-time rendering whereas *path tracing* is not.

- Phong lighting: For each surface point hit by a camera ray, lighting is computed with the Phong model.
- Ray tracing: The ray tree's depth ranges from three to five, depending on the scene complexities.
- Path tracing: Hundreds of rays are fired toward a pixel, and every ray tree's depth is five. The scene is assumed to be composed of Lambertian surfaces.

The red-&-blue rooms introduced in Fig. 1-(a) are rendered with path tracing whereas Fig. 13 shows the same scene rendered with Phong lighting.

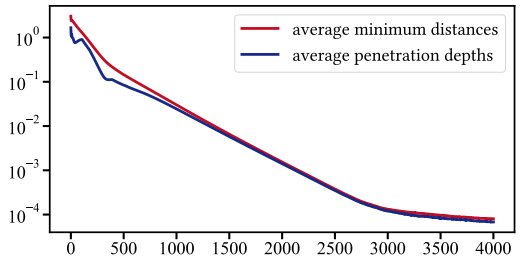


Fig. 15. Minimum distances between mating facets and penetration depths measured over the time steps of rigid-body simulation.

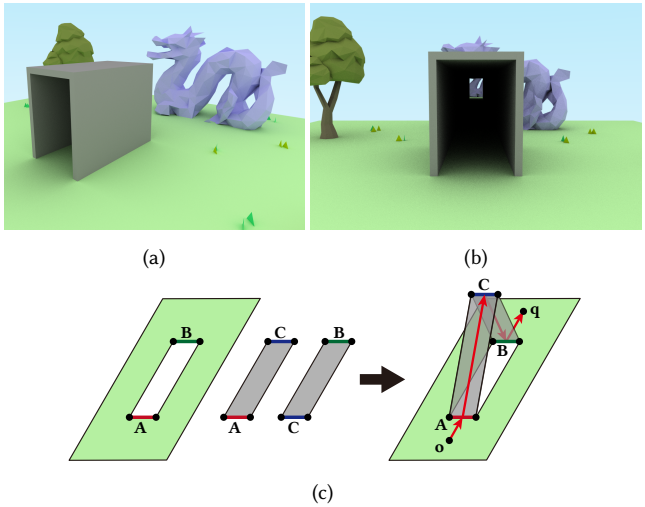


Fig. 16. Tunnel (3-manifold embedded in 4D Euclidean space).

Fig. 14 shows a 3-manifold house with three cuboid rooms. Each room is a 3-polytope, as presented in Fig. 12, and the concatenated 3-polytopes are embedded in 4D Euclidean space. The camera is located at the red room, which is connected to the blue room by a left doorway (henceforth, simply door) and to the gray room by a right door. The blue and gray rooms are also connected by a door, which we call the third door. Observe that Mona Lisa in the blue room is visible through the right and third doors as well as through the left door. Similarly, the outdoor scene (with sky and green yard) visible through the gray room's window is also visible through the left and third doors. These would bring users unusual experiences, which cannot be provided in Euclidean spaces.

The three-room house in Fig. 14 is a one-dimensional higher version of that in Fig. 8-(a). Therefore, the novel user-experiences presented in Fig. 8-(b) and -(c) can be reiterated: Starting from the red room, the user passes through only three doors, i.e., the left, third and right doors in sequence, to return to the red room.

When the polytope concatenation procedure presented in Section 5 is used for creating the three-room house shown in Fig. 14, the spring forces concatenate the 3-polytopes with no interpenetration, making it unnecessary to invoke collision resolution. In Fig. 15, the red curve depicts the average minimum distances between mating facets over the time steps whereas the blue curve depicts the average penetration depths of the polytopes. As the rigid-body simulation proceeds, both distances and penetration depths keep decreasing.

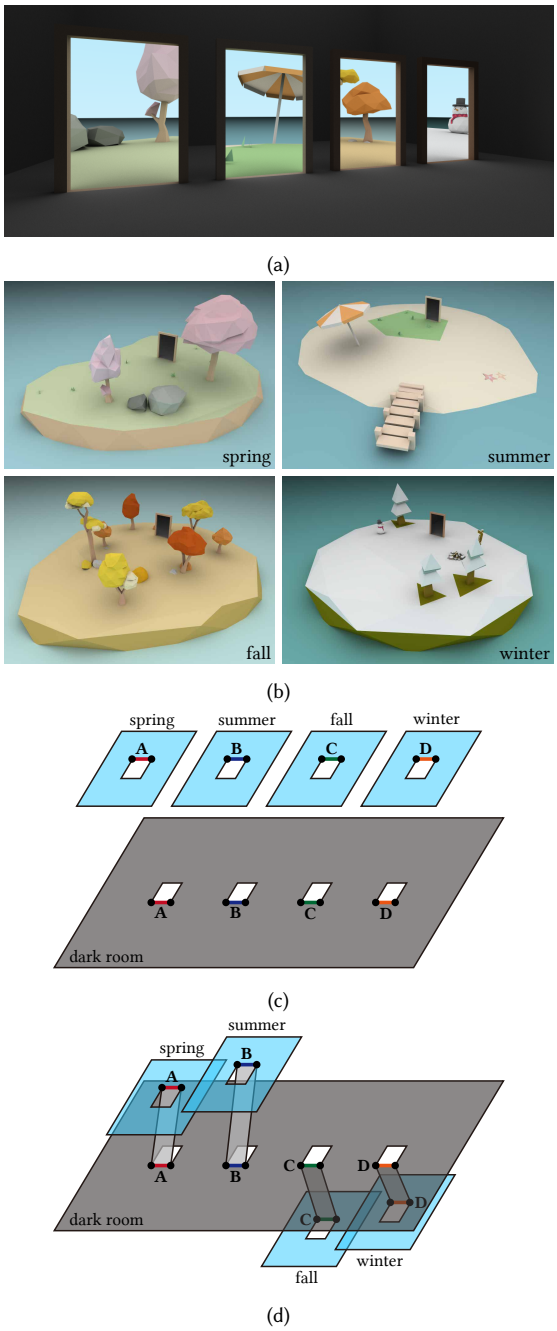


Fig. 17. Four seasons (3-manifold embedded in 5D Euclidean space).

All experiments reported in this section are made with 3-manifolds, but it is hard to visualize their structures. Therefore, as done with the three-room house in Fig. 8-(a), the structure of a 3-manifold will be visualized and discussed using the corresponding 2-manifold.

Table 1. Illumination statistics: Elapsed times per frame (in milliseconds).

	Red-&-blue rooms (Fig. 13)	Three-room house (Fig. 14)	Tunnel (Fig. 16)	Four seasons (Fig. 17)	Pillared rooms (Fig. 18)
Phong lighting	6.9	4.3	3.9	10.8	8.1
Ray tracing	24.7	15.9	18.1	24.8	27.6
Path tracing	18416.3	108719.4	5514.5	7493.2	301770.5

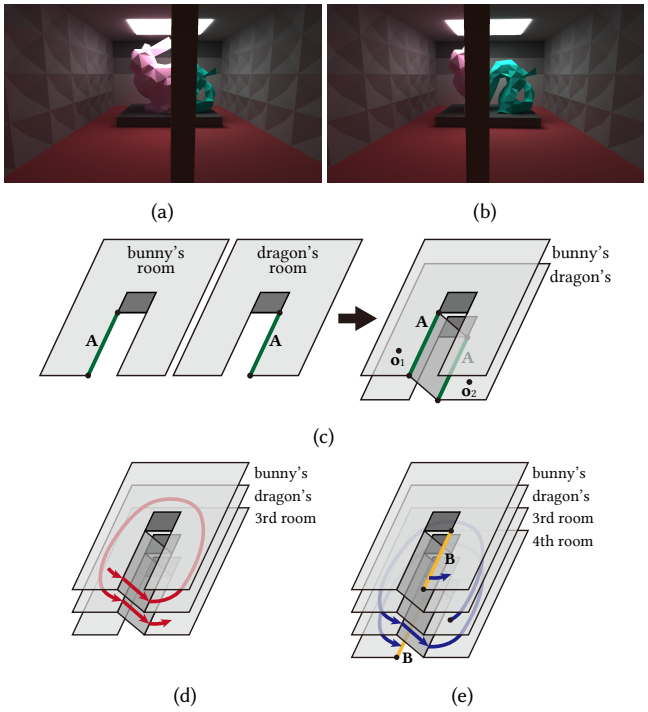


Fig. 18. Pillared rooms (3-manifold embedded in 5D Euclidean space).

Let us see another example of 3-manifold embedded in 4D Euclidean space. Fig. 16-(a) shows a tunnel seen from outside. It is short. If we look inside the tunnel, however, we realize it is quite long, as shown in Fig. 16-(b). Fig. 16-(c) explains how this non-Euclidean space is created using three 2-polytopes, where **A**, **B** and **C** represent the mating facets. As depicted by red arrows, the camera ray fired from **o** travels a long way to hit a surface point, **q**.

Fig. 17-(a) shows a 3-manifold scene, where the camera is located in a dark room connected to four different outdoor spaces. The spaces seen through the dark room's windows are 3-polytopes that represent four seasons. See Fig. 17-(b), where the small black upright rectangles represent the facets that mate to the windows of the dark room.

Fig. 17-(c) shows five 2-polytopes, which correspond to the dark room and outdoor spaces, together with the mating facets. Without *joint polytopes*, they cannot be concatenated in a penetration-free state. In Fig. 17-(d), the four seasons are linked to the dark room by joint polytopes. All polytopes are embedded in 3D Euclidean space. Unfortunately, this is not valid. The four-season polytopes are illustrated as small rectangles just for visualization purposes, but they are wide open in reality. Therefore, for example, the spring polytope penetrates the joint polytope connected to the summer polytope.

We may not be able to escape from such an unwanted structure even if we alternate many times between spring forces and collision resolution. This can be resolved by increasing the dimension of the embedding space to 4D. The similar discussion applies to the original 3-polytopes shown in Fig. 17-(a) and -(b): They are embedded in 5D Euclidean space, not in 4D space, to ensure non-penetration between 3-polytopes.

Let us see another example of 3-manifold embedded in 5D Euclidean space. Fig. 18-(a) would appear to be a pillared room, but a bunny is on the left whereas a dragon is on the right. If the camera translates to the right, we obtain the image shown in Fig. 18-(b) to find that the space is not Euclidean.

Using 2-polytopes, Fig. 18-(c) presents how the non-Euclidean space is generated. The mating facets are denoted as **A**, and the small dark rectangles represent the areas occupied by pillars. The bunny's room and the dragon's are linked by a joint polytope. Note that $\mathbf{o}_1$ represents the camera's position located "at the bunny's room" and is used for capturing Fig. 18-(a) whereas $\mathbf{o}_2$ is located "at the dragon's room" and is used for capturing Fig. 18-(b).

Fig. 18-(d) shows that a new room, named the 3rd room, is concatenated to the existing 2-manifold by a joint polytope. Suppose that a user moves along the red curve, i.e., from the bunny's room to the dragon's and then to the 3rd room. Walking around the pillars that appear to be a single one located in a flat floor, the user can visit the rooms one by one.

In Fig. 18-(e), another room, named the 4th room, is added to the 2-manifold. Due to the mating facets denoted as **B**, the 4th room is concatenated to the bunny's room by a joint polytope. (The joint polytope is not illustrated in the figure as it would make Fig. 18-(e) overly cluttered.) Note that the four rooms make a circular list. The red curve depicted in Fig. 18-(d) is followed by the blue curve in Fig. 18-(e), making the user return to the bunny's room.

The 2-manifolds shown in Fig. 18-(c) and -(d) can be embedded in 3D Euclidean space, but the 2-manifold shown in Fig. 18-(e) cannot. We need 4D embedding space. The similar discussion applies to the original 3-polytopes shown in Fig. 18-(a) and -(b). They are embedded in 5D Euclidean space, not in 4D space, to ensure non-penetration.

Table 1 enumerates the elapsed times per frame (in milliseconds) for rendering all test scenes reported in this section with three techniques. Recall that *Phong lighting* and *ray tracing* are for real-time rendering whereas *path tracing* is not.

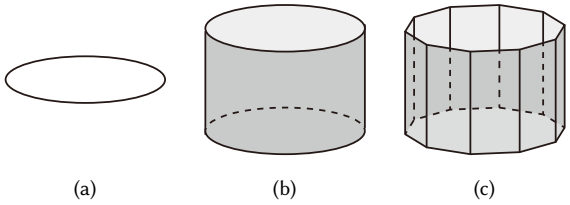


Fig. 19. Two-manifold and its approximation: (a) A 1-manifold. (b) This 2-manifold is obtained by extruding vertically the 1-manifold in (a). If this is extruded along an additional axis, we obtain a 3-manifold. (c) The 2-manifold in (b) is approximated with ten 2-polytopes.

Table 2. Times (in milliseconds) consumed by the major components in illumination. In the concatenated 3-polytopes, the ray is not projected at all but it crosses over the adjacent polytopes.

original 3-manifold	1M ray marching steps	0.6851
	1M ray hit tests × 1K prisms	31914.8
	1M projections	20.0344
concatenated 3-polytopes	1M ray marching steps	0.7689
	1M ray hit tests × 100 triangles	1565.5
	1M crossings over polytopes	2.8176

7 DISCUSSION AND LIMITATIONS

7.1 Performance Analysis

Fig. 19-(a) shows a circle, which is a 1-manifold. If it is extruded along the vertical axis, we obtain a 2-manifold shown in Fig. 19-(b). Fig. 19-(c) shows that the 2-manifold is approximated with ten 2-polytopes.

If the 2-manifold shown in Fig. 19-(b) is extruded along an additional axis, we obtain a 3-manifold. It is then approximated with ten 3-polytopes. We made a simple experiment to compare the performance of illumination in the concatenated 3-polytopes and that in the original 3-manifold.

In total, a thousand (1K) triangles are uniformly distributed in the 3-manifold so that each 3-polytope is inhabited by 100 triangles. The major components of illuminating the original 3-manifold are ray marching and projection (as presented in Fig. 2-(d)) and ray hit test with the *prisms* that are extruded from the triangles (using Equation (4)). Just for test purposes, we fire only a single ray, which makes a million (1M) marching steps even if it intersects some prisms along its way. Then, both ray hit test and projection are also made 1M times. In a single marching step, the ray is tested for intersection with "1K prisms." The upper row in Table 2 enumerates the accumulated times in milliseconds.

In the test with the concatenation of ten 3-polytopes, the single ray is forced to make 1M marching steps for comparison purposes. As presented in Section 4, only a single step of ray marching is made per polytope, and therefore the total number of polytope visits is 1M. (Each polytope is visited 100K times.) In a polytope, the ray is tested for intersection with "100 triangles." See the accumulated times in the lower row of Table 2. The big difference in the ray hit tests is explained by "1K prisms *vs.* 100 triangles."

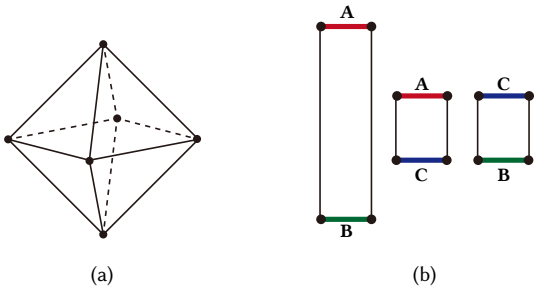


Fig. 20. Challenging cases: (a) A 3-polytope is located at each vertex of the octahedron. (b) The long 2-polytope's length is three whereas the short 2-polytopes' lengths are one. It is impossible to concatenated them.

7.2 Embedding Space

In computer graphics fields, virtually all methods for handling m-D non-Euclidean spaces take $(m+1)$-D Euclidean space as the embedding space. Good examples include not only the three-room houses presented in Fig. 8-(a) and Fig. 14 but also the work of Silva et al. [2018], which deals with embedding 3-manifolds in 4D Euclidean spaces only. On the contrary, Fig. 17 and Fig. 18 show that the embedding space's dimension often must be higher than $m+1$.

According to the Whitney embedding theorem [Whitney et al. 1992], it must be higher than $2m-1$ so as to be able to maintain the geodesic distances of "all kinds" of continuous manifolds. For example, it is 6 for 3-manifolds. In Section 6, all 3-manifolds are embedded in 4D or 5D Euclidean spaces. However, it is not hard to imagine a 3-manifold that cannot be embedded in 4D or 5D Euclidean spaces but requires 6D embedding space. Suppose that we have six 3-polytopes. Taking each 3-polytope as a point, let us connect them using joint polytopes to construct an octahedral structure, as shown in Fig. 20-(a). It can be embedded only in 3D or higher-dimensional Euclidean spaces. As each vertex of the octahedron is a 3-polytope, the embedding space's dimension should be accordingly increased to 6D or higher. (Recall that our polytope concatenation procedure can successfully embed such a 3-manifold in 6D space due to its capability of increasing the embedding space's dimensions.)

7.3 Scalability in Illumination

A single step of ray marching is made per polytope. Therefore, the more polytopes, the higher cost. However, the cost does not increase linearly because a ray (either camera ray or shadow ray) crosses over polytopes only if there exists a mating facet on its way. This results in sublinear complexity.

The red-&-blue rooms presented in Fig. 13 is rendered in real time (without explosion). In the current implementation, we stop a ray if its marching distance exceeds a pre-defined threshold. If the ray marched further, it might hit a simplex. However, such a simplex will appear quite tiny in the image space, contributing little to the final image. On the other hand, our ray tree's depth ranges from three to five, as stated in Section 6. It also helps avoid explosion.

(a) (b) (c)

Fig. 21. Polytope concatenation: (1) A large number of 1-polytopes are concatenated successfully in 2D Euclidean space. (b) A concatenation failure case in 2D Euclidean space. (c) A successful concatenation in 3D Euclidean space.

7.4 Polytope Concatenation

The experimental results show that our polytope concatenation procedure works successfully unless the given polytopes are inherently impossible to concatenate, for example, as illustrated in Fig. 20-(b). However, it is often observed that joint polytopes are unnecessarily created in cases where the given polytopes can be concatenated without them. They are usually tiny but might make users annoying. They also degrade run-time performances slightly.

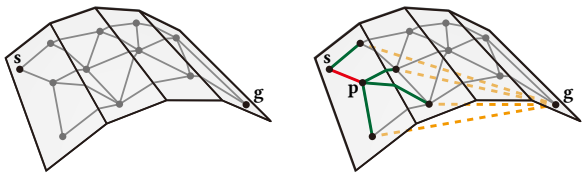
For the sake of *scalability* of our concatenation procedure, consider a 1-manifold shown in Fig. 21-(a). It is a one-dimensional lower version of the band presented in Fig. 10 but is approximated with a much larger number of polytopes (line segments). Tested with many different initial configurations of the line segments, our procedure successfully concatenates the line segments into closed curves in a few iterations. Assume however that an incorrect concatenation is produced after a pre-defined number of iterations, as shown in Fig. 21-(b). Then, the embedding space's dimension is increased to three. In 3D Euclidean space, a correct concatenation is produced almost always in a single iteration, as shown in Fig. 21-(c), where the line segments are made 'volumetric' and 'shaded' just for visualization purposes.

In the current implementation, the embedding space's dimension is increased after ten iterations without success. The count is heuristically selected. It would be worth seeking a rigorous method to find an optimal count. In principle, the concatenation procedure's goal is to solve a *constrained* optimization problem, where the objective is to minimize the lengths of the vertex-connecting links and the constraint is "non-penetration between polytopes." However, it is challenging to find a clever way to solve the optimization problem especially because the Minkowski difference between two rigid bodies produces a number of linear inequalities.

On the other hand, it would be desirable to provide a user interface for the scene designers to post-process the concatenated polytopes. However, this is not easy due to the difficulties visualizing 3-polytopes in a higher-dimensional embedding space. We leave the task of developing the optimal concatenation procedure and user-friendly interface as a future work.

7.5 Differences from Portal-based Methods

Our method is clearly distinguished from existing portal-based methods [CodeParade 2018; Demruth 2013; Valve 2007]. They provide users with *discontinuous* movements between 3D Euclidean spaces through *teleportation* at the portals, whereas our method supports users' continuous movements within a single non-Euclidean space.



(a) (b)

Fig. 22. Calculating the optimal path from **s** to **g** in concatenated polytopes: (a) A number of nodes are located at the concatenated polytopes. (b) Given the current path from **s** to **p** (in red), the heuristic costs of the fringe nodes (connected with green edges) are estimated using their L^2 distances to **g** (colored in orange).

In addition, they render the spaces beyond the portals using the secondary cameras located at those spaces, whereas our method uses ray tracing with a single camera.

A notable advantage of embedding m-polytopes in an n-D Euclidean space is that all polytopes and simplices are defined in the 'shared' n-D space. Then, for example, we can compute the L^2 distance between any pair of simplices even if they reside in different polytopes. Taken as the lower bound of their geodesic distance that is not straight in general, it can be used for many distance-based applications. In collision detection, for example, the potentially colliding set can be quickly identified using the L^2 distances.

In games, path planning algorithms, such as A* [Hart et al. 1968] and jump point search algorithms [Harabor and Grastien 2011], are widely used. For path planning in non-Euclidean spaces, the L^2 distances can be used to estimate the *heuristic costs*. See Fig. 22-(a), where **s** and **g** represent the start and goal nodes respectively. They reside in different polytopes. Suppose that the red edge in Fig. 22-(b) is the currently expanded path. Then, we should select one of the fringe nodes connected with green edges, and the heuristic costs can be estimated using the L^2 distances from the fringe nodes to **g**, which are visualized with dotted orange lines.

Consider a game world with a huge number of polytopes. It may not be loaded into memory at once. Then, we can load only the polytopes, whose L^2 distances from the current polytope are shorter than the threshold of the ray marching distance presented in Section 7.3.

8 CONCLUSION AND FUTURE WORK

In this paper, we proposed a method for designing non-Euclidean spaces of arbitrary dimensions. The spaces are represented as concatenated polytopes embedded in Euclidean spaces of higher dimensions. We also proposed camera navigation and rendering methods tailored to the concatenated polytopes. The experiment results are promising and show the feasibility of the proposed methods.

However, our methods also have drawbacks. The rendering technique assumes static light sources. Processing dynamic light sources would prevent the proposed system from being executed at real time even on a high-end computer. As is the case for traditional 3D applications such as games, it will be crucial to provide a user interface for designing various non-Euclidean scenes. Our future work

will be directed toward developing an intuitive tool for modeling manifolds. This is also driven by the need to generate an optimal concatenation.

ACKNOWLEDGMENTS

This research was supported by the Ministry of Science and ICT, Korea, under the ICT Creative Consilience Program (IITP-2022-2020-0-01819), ITRC (Information Technology Research Center) Support Program (IITP-2022-2020-0-01460) and the grant No.2020-0-00861, which are all supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation).

REFERENCES

K Ayush and P Chaudhuri. 2017. Rendering curved spacetime in everyday scenes. In *ACM SIGGRAPH 2017 Posters*. 1–2.

RG Baraniuk and MB Wakin. 2009. Random projections of smooth manifolds. *Foundations of computational mathematics* 9, 1 (2009), 51–77.

DS Bernstein. 2009. *Matrix mathematics*. Princeton university press.

J Blinn. 2003. *Jim Blinn's corner: notation, notation, notation*. Morgan Kaufmann.

MT Bosch. 2020. N-dimensional rigid body dynamics. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 55–1.

M Cavallo. 2021. Higher Dimensional Graphics: Conceiving Worlds in Four Spatial Dimensions and Beyond. In *Computer Graphics Forum*, Vol. 40. Wiley Online Library, 51–63.

CodeParade. 2018. Non-Euclidean Worlds Engine. Video. https://www.youtube.com/watch?v=kEB11PQ9Eo8. (Accessed January 27, 2022).

R Coulon, EA Matsumoto, H Segerman, and S Trettel. 2020a. Non-euclidean virtual reality III: Nil. *Proceedings of Bridges 2020: Mathematics, Music, Art, Architecture, Culture* (2020), 153–160.

R Coulon, EA Matsumoto, H Segerman, and S Trettel. 2020b. Non-euclidean virtual reality IV: Sol. *Proceedings of Bridges 2020: Mathematics, Music, Art, Architecture, Culture* (2020), 161–168.

F Dassi, A Mola, and H Si. 2014. Curvature-adapted remeshing of CAD surfaces. *Procedia Engineering* 82 (2014), 253–265.

F Dassi, H Si, S Perotto, and T Streckenbach. 2015. Anisotropic finite element mesh adaptation via higher dimensional embedding. *Procedia Engineering* 124 (2015), 265–277.

Demruth. 2013. Antichamber. Video Game. (31 January 2013). http://www.antichamber-game.com/. Retrieved January 27, 2022.

M Falk, T Schafhitzel, D Weiskopf, and T Ertl. 2007. Panorama maps with non-linear ray tracing. In *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*. 9–16.

D Harabor and A Grastien. 2011. Online graph pruning for pathfinding on grid maps. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 25. 1114–1119.

PE Hart, NJ Nilsson, and B Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107.

V Hart, A Hawksley, EA Matsumoto, and H Segerman. 2017a. Non-euclidean virtual reality I: explorations of $\mathbb{H}^3$, 2017. , 33–40 pages.

V Hart, A Hawksley, EA Matsumoto, and H Segerman. 2017b. Non-euclidean virtual reality II: explorations of $\mathbb{H}^2 \times \mathbb{E}$. *Proceedings of Bridges 2017: Mathematics, Music, Art, Architecture, Culture* (2017), 41–48.

V Hart, A Hawksley, H Segerman, and MT Bosch. 2015. Hypernom: Mapping VR Headset Orientation to $\mathbb{S}^3$. *Proceedings of Bridges 2015: Mathematics, Music, Art, Architecture, Culture* (2015), 387–390.

B Lévy and N Bonneel. 2013. Variational anisotropic surface meshing with Voronoi parallel linear enumeration. In *Proceedings of the 21st international meshing roundtable*. Springer, 349–366.

A Macdonald. 2010. *Linear and geometric algebra*. CreateSpace Independent Publishing Platform.

T Novello, V da Silva, and L Velho. 2020a. Global illumination of non-Euclidean spaces. *Computers & Graphics* 93 (2020), 61–70.

T Novello, V da Silva, and L Velho. 2020b. Visualization of Nil, Sol, and $SL_2(\mathbb{R})$ geometries. *Computers & Graphics* 91 (2020), 219–231.

D Panozzo, E Puppo, M Tarini, and O Sorkine-Hornung. 2014. Frame fields: Anisotropic and non-orthogonal cross fields. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 1–11.

JR Silva et al. 2018. *Ray Tracing in Non-Euclidean Spaces*. Ph. D. Dissertation.

Valve. 2007. Portal. Video Game. (10 October 2007). https://store.steampowered.com/app/400/Portal/. Retrieved January 27, 2022.

N Verma. 2012. Distance preserving embeddings for general n-dimensional manifolds. In *Conference on Learning Theory*. JMLR Workshop and Conference Proceedings, 32–1.

D Weiskopf, T Schafhitzel, and T Ertl. 2004. GPU-based nonlinear ray tracing. In *Computer graphics forum*, Vol. 23. Wiley Online Library, 625–633.

H Whitney, J Eells, and D Toledo. 1992. *Collected Papers of Hassler Whitney*. Vol. 1. Nelson Thornes.

Z Zhong, X Guo, W Wang, B Lévy, F Sun, Y Liu, W Mao, et al. 2013. Particle-based anisotropic surface meshing. *ACM Trans. Graph.* 32, 4 (2013), 99–1.

Z Zhong, W Wang, B Lévy, J Hua, and X Guo. 2018. Computing a high-dimensional euclidean embedding from an arbitrary smooth riemannian metric. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–16.

A m-MANIFOLDS IN n-D EUCLIDEAN SPACES

Given m-manifolds embedded in n-D Euclidean spaces, where $m < n$, the camera is advanced using the n-D velocity vector and then projected onto the m-manifold. The camera-space basis is composed of m unit vectors including **view**. They are n-D and orthonormal. Based on the user input to change the view direction, **view** is updated first. Then, the remaining $m-1$ vectors are updated one by one, as done in Section 3.2. Recall that the basis vectors in "3-manifolds embedded in 4D Euclidean space" were updated via *cross* presented in Equation (2) and **normal** was involved in it. We now have $n-m$ normals. They are all involved in the basis update procedure, together with $m-1$ basis vectors, i.e., in total, $n-1$ vectors are involved. For this, *cross* in Equation (2) is extended to take $n-1$ vectors:

$$cross(\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_{n-1}) = \star(\mathbf{v}_1 \wedge \mathbf{v}_2 \wedge ... \wedge \mathbf{v}_{n-1}) \tag{7}$$

where $\star$ represents Hodge star operator [Bernstein 2009] and $\wedge$ represents the exterior product operator.

Hodge star operation is described using the language of *geometry algebra* [Macdonald 2010]. The exterior product, $\wedge$, of $n-1$ 1-vectors returns an $(n-1)$-vector. On the other hand, Hodge Star operator in n-D space is a linear mapping between m-D subspace and $(n-m)$-D subspace. Each $(n-1)$-blade is mapped to the corresponding 1-blade. For example, in 3D space with orthonormal basis vectors, $\mathbf{e}_1$, $\mathbf{e}_2$ and $\mathbf{e}_3$, the exterior product of two 1-vectors, $a\mathbf{e}_1 + b\mathbf{e}_2 + c\mathbf{e}_3$ and $d\mathbf{e}_1 + e\mathbf{e}_2 + f\mathbf{e}_3$, is $(ae-bf)\mathbf{e}_1\mathbf{e}_2 + (bf-ce)\mathbf{e}_2\mathbf{e}_3 + (cd-af)\mathbf{e}_3\mathbf{e}_1$, which is a 2-vector. Applying Hodge star operator to the 2-vector, we obtain $(bf-ce)\mathbf{e}_1 + (cd-af)\mathbf{e}_2 + (ae-bf)\mathbf{e}_3$. This is the cross product of the two 1-vectors. In the same manner, we can compute the cross product of any dimensional vectors using Hodge Star operation.

In theory, the screen space in m-manifold is not limited to 2D but may be up to $(m-1)$-D. The screen space is regularly sampled to define what we call *hyperpixels*. Each hyperpixel is assigned the camera-space coordinates, i.e., the coordinates with respect to m basis vectors of the camera space. The basis vectors are all n-D and are *linearly combined* using the coordinates of each hyperpixel to define its n-D position. Toward each n-D hyperpixel, a ray is fired.

In typical display devices such as PC monitor or VR HMD, however, the screen is 2D. For rendering an m-manifold scene into such a 2D screen, a 2D rectangle is defined in the m-D camera space such that **view** is orthogonal to it and points to its center. The rectangle is regularly sampled to define a 2D array of pixels, and each pixel is assigned the coordinates with respect to m basis vectors of the camera space. The n-D basis vectors are linearly combined using the coordinates of each pixel to define its n-D position. Toward each n-D pixel, a ray is fired.