Minimizing Self-Intersections of 3-dimensional Immersions of 5-dimensional Cubical Surfaces with Reinforcement Learning

Anonymous authors
Paper under double-blind review

Abstract

A closed cubical surface is a 2-dimensional cubical complex (analogous to a simplicial complex but with a cubical structure) such that each of its points has an open neighborhood homeomorphic to a disk. Aveni et al. (2024) proved that up to isomorphism 2690 connected closed cubical surfaces can be built from the faces of a 5-cube (sometimes called a penteract) and give a classification for closed orientable and non-orientable cubical surfaces. It is well known that non-orientable surfaces (of any kind) cannot be embedded in $\mathbb{R}^3$; their immersion will always have some self-intersection and in the context of cubical surfaces this also seems to be the case for some orientable surfaces. Therefore, given a cubical surface it is natural to ask: What is the smallest number of self-intersections it can have for any immersion in $\mathbb{R}^3$ using perspective projection and without deforming the 5-cube? Given an initial immersion, can we calculate a sequence of 5-dimensional rotations or perspective projections step-wise minimizing self-intersections efficiently? These questions are addressed using Reinforcement Learning and animation sequences are created to visualize the minimization strategies found by the agent.

1 Introduction

Compact Surfaces are 2-dimensional topological manifolds and are completely classified. There are two infinite families of compact surfaces, Σ_g $(g \geq 1)$, the orientable surface of genus g, which is a torus with g "handles", and N_k $(k \geq 1)$, the non-orientable surface of demigenus k. There is also the sphere which we denote by Σ_0 since it is orientable and has no "handles". Topological manifolds are often too general to work with them directly, for this reason it is often essential to assume the triangulability of manifolds. The problem of finding minimal triangulation of manifolds has been widely studied and has been completely solved for compact surfaces. In the context of simplicial surfaces, a minimal triangulation can be defined as a triangulation using the minimal number of vertices, edges, and faces. It can be shown using Euler characteristic that the minimal triangulation in each of these three meanings of minimality is realized by the same triangulation up to isomorphism. Aveni et al. (2024) consider Cubical Surfaces (see Section 2.1) instead of simplicial ones which can be thought of as two-dimensional sub-complexes of the n-dimensional cube homeomorphic to compact surfaces. They give a complete classification up to dimension $n = 6$ in terms of their genus g for orientable cubical surfaces and their demigenus k for the non-orientable; in particular they found that non-orientable cubical surfaces first appear for $n = 5$. One can ask what is the minimal cubical embedding of a given cubical surface but here minimality has to be specified more carefully since minimizing a cubical embedding can refer to vertices, edges, faces or the dimension of the cube. In this paper, the word **minimal** refers to minimality with respect to the number of faces needed to be realized in the 5-dimensional cube Q^5.

A (topological) **immersion** is a continuous function that is locally an homeomorphism while an **embedding** is an homeomorphism onto its image. From algebraic topology we know that all orientable surfaces can be embedded in $\mathbb{R}^3$, while non-orientable can be embedded in $\mathbb{R}^4$ but only immersed in $\mathbb{R}^3$. Consequently for any non-orientable cubical surface, any projection in $\mathbb{R}^3$ will have some positive number of face-intersections while for orientable cubical surfaces face-intersections can be completely cleared in some cases. Some examples of polyhedral immersions of simplicial surfaces have been studied by Cervone (2001).

Here, a cubical surface is immersed to $\mathbb{R}^3$ via perspective projections as explained in Section 2.3. We assume that for a given initial orientation of the cubical surface around the origin in $\mathbb{R}^5$; the **n-dimensional camera point** is set at a position $\boldsymbol{c}_5 \in \mathbb{R}^5$ from which projection rays extend to the 5-dimensional surface onto a **projection hyperplane** $\boldsymbol{p}_5 \in \mathbb{R}^5$. The result of this first projection is a surface in $\mathbb{R}^4$ so we can repeat the same process with a camera $\boldsymbol{c}_4 \in \mathbb{R}^4$ and a hyperplane $\boldsymbol{p}_4 \in \mathbb{R}^4$ to obtain a surface in $\mathbb{R}^3$.

The initial projection has some number of self-intersections; the number of pairings of projected faces intersecting in $\mathbb{R}^3$ which we simply call face-intersections. One can apply a 5-dimensional rotation or change the projection distances on the perspective projection and modify the number of face-intersections sequentially. The immersions here studied are always perspective projections of an unitary 5-cube; meaning the unitary 5-cube is never deformed. In this paper a Reinforcement Learning (RL) agent is trained to find a sequence of 5-dimensional rotation matrices or perspective projection modifications (see Section 2.2) which applied to the cubical surface sequentially minimize its number of face-intersections. This is addressed by formulating the face intersection minimization problem as a Markov Decision Process (MDP) as explained in Section 3.

Section 3.5 explains the algorithm for the environment and Section 3.6 the algorithm for the agent. Experiments and the resulting optimized immersions and embeddings of some cubical surfaces are shown in Section 4. A complete face intersection minimization sequence is shown in Section 4.2; however in Section 7 the reader can find a link to the 3-d animated models of the cubical surfaces here presented. Animations help as a tool to visualize the agent's learned strategy to minimize the face-intersections of each of the surfaces. The main objectives of this study are the following:

1. Given cubical surface, find an optimized immersion minimizing the number of self-intersections by applying a sequence of transformations consisting of either 5-d rotations or perspective projection camera modifications.

2. The sequence of actions transforming the initial into the optimized immersion must decrease the number of face-intersections as monotonically as possible and in the smallest number of steps possible.

2 Background

2.1 Cubical Surfaces

Cubical complexes have its origin in the beginning of the XX century with the work of mathematicians like Henry Poincare or Solomon Lefschetz. They have become an important tool in Topological Data Analysis (TDA) and applied topology since the 90's because squares or cubes are natural building blocks in areas like robotics, materials science and engineering, image analysis among others. Some of the main definitions of cubical complexes and cubical homology can be consulted in the work of Kaczynski et al. (2004). In particular, cubical surfaces introduced by Aveni et al. (2024) are two-dimensional cubical complexes topologically equivalent to a closed surface which are formally introduced below.

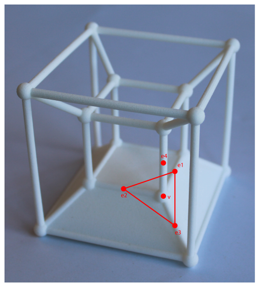


Figure 1: $\mathcal{F}_v$ on a 4-d Cubical Surface homeomorphic to a sphere.

Denote the n-dimensional unit cube by $Q^n = [0,1]^n = [0,1] \times \cdots \times [0,1]$ (n times), and its set of vertices by Q^n_0. Each vertex of Q^n can be represented by an element of the set of all n-tuples with binary entries $\{0,1\}^n$, for example the vertices of the unit square are represented by the tuples $\{(0,0),(0,1),(1,0),(1,1)\} \in \mathbb{R}^2$. The one-dimensional skeleton of Q^n is denoted by Q^n_1 and consists of the set of vertices v and edges e of Q^n. The one-skeleton Q^n_1 can also be regarded as a graph with vertex set Q^n_0 with an edge between two vertices if and only if they differ in exactly one coordinate. Similarly, the two-dimensional skeleton of Q^n is denoted by Q^n_2 and consists of the set of vertices Q^n_0, the one-dimensional skeleton Q^n_1, and all its two-dimensional faces $f \in Q^n$. We can continue this construction up to the n-cube Q^n_n and name the elements of all the preceding sets the cells of Q^n. Every cell of Q^n is a product of vertices and intervals, and therefore can be encoded combinatorially as an element of $\{0,1,2\}^n$. Here a 2 in an entry implies that in the product, the whole interval I is considered. Thus, every sub-complex of Q^n can be represented as a subset of $\{0,1,2\}^n$. We refer to a subset of Q_2^n as a two-dimensional cubical complex which in the following we denote by $\mathcal{C}$ and whose set of vertices, edges and faces are denoted by $\mathcal{C}_0$, $\mathcal{C}_1$, and $\mathcal{C}_2$ respectively. The **vertex figure** $\mathcal{F}_v$ of a vertex v is the graph whose nodes are the edges in $\mathcal{C}_1$ having v as an endpoint and where two nodes $e, e' \in C_1$ are joined by an edge if there is a face $f \in \mathcal{C}_2$ with e, e' as two of its edges. A **closed cubical surface** is a two-dimensional cubical complex $\mathcal{C}$ in which every point has an open neighborhood homeomorphic to an open disk. This condition is equivalent to requiring $\mathcal{C}$ to fulfill the following two conditions:

1. Every edge is shared by exactly two faces, i.e., for all $e \in \mathcal{C}_1, F_e = 2$.
2. The vertex figure $\mathcal{F}_v$ of any vertex $v \in \mathcal{C}_0$ is a cyclic graph.

For the case $n = 4$ only orientable cubical surfaces can exist, and the 3-dimensional embeddings of each cubical surface representative can be consulted in the paper by Estévez et al. (2023). In Section 4 some 3-d models of the perspective projections of minimal 5-dimensional cubical surfaces for genus g $(1 \le g \le 5)$ and demigenus k $(1 \le k \le 3)$ are presented; where in the non-orientable case $k = 1$ is equivalent to a Projective Plane and $k = 2$ to a Klein Bottle.

2.2 N-dimensional Rotations & Gimbal-Lock

Consider two canonical vectors $\boldsymbol{e}^{(i)}, \boldsymbol{e}^{(j)} \in \mathbb{R}^n$ and let $X_{i,j} \subset \mathbb{R}^n$ be their spanned plane. In $\mathbb{R}^n$ there are $\binom{n}{2} = n(n-1)/2$ possible pairing of canonical vectors $\boldsymbol{e}^{(i)}, \boldsymbol{e}^{(j)}$ or equivalently possible planes $X_{i,j}$. A rotation by an angle $\phi \in [0, 2\pi)$ in the plane $X_{i,j}$ is called an **elemental rotation**, and can be represented by an elemental rotation matrix $\boldsymbol{R}_{i,j}(\phi)$; elemental rotations will allow us to generate all possible rotations in $\mathbb{R}^n$. In $\mathbb{R}^3$, elemental rotations by angles $\phi_{0,1}, \phi_{0,2}$ and $\phi_{1,2}$ in planes $Z = 0$, $Y = 0$ and $X = 0$ correspond to elemental rotation matrices $\boldsymbol{R}_{0,1}(\phi_{0,1})$, $\boldsymbol{R}_{0,2}(\phi_{0,2}), \boldsymbol{R}_{1,2}(\phi_{1,2})$ respectively. For any angle $\phi \in [0, 2\pi)$, elemental rotation matrices satisfy the relationships $\boldsymbol{R}_{i,j}(\phi) = \boldsymbol{R}_{j,i}(-\phi)$, $\boldsymbol{R}_{i,j}^{-1}(\phi) = \boldsymbol{R}_{j,i}(\phi)$, and $\boldsymbol{R}_{i,j}(\theta)\boldsymbol{R}_{j,i}(\theta) = \boldsymbol{I}_n$ and they can be constructed as follows:

$$\boldsymbol{R}_{i,j}(\phi) := \begin{cases} R_{k,k} = \cos(\phi) & \text{if } k = i \\ R_{l,l} = \cos(\phi) & \text{if } l = j \\ R_{k,l} = -\sin(\phi) & \text{if } k = i \text{ and } l = j \\ R_{l,k} = \sin(\phi) & \text{if } k = i \text{ and } l = j \\ R_{k,k} = 1 & \text{if } k \neq i \text{ or } k \neq j \\ R_{k,l} = 0 & \text{otherwise.} \end{cases}$$

Consider a list of angles $\phi_n := (\phi_{i,j} : (i,j) \in \binom{n}{2})$ with the rotation angles in each plane $X_{i,j}$ ordered lexicographically, that is $(i,j) < (k,l)$ if $i < k$ or $(i = k$ and $j < l)$. A **general rotation matrix** $\boldsymbol{R} \in SO(n, \mathbb{R})$ by angles ϕ_n can be calculated by multiplying elemental rotation matrices $\boldsymbol{R}_{i,j}(\phi_{i,j})$ for each angle $\phi_{i,j} \in \phi_n$ on the left with respect to the order given by ϕ_n as shown in Equation 1. Since elemental rotation matrices generally do not commute, the order of the factors is crucial.

$$\boldsymbol{R}(\phi_n) := \boldsymbol{R}_{n-2,n-1}(\phi_{n-2,n-1}) \cdots \boldsymbol{R}_{0,2}(\phi_{0,2})\boldsymbol{R}_{0,1}(\phi_{0,1}). \tag{1}$$

However, when calculating general rotation matrices, some considerations must be taken into account. For example, lets consider the list of angles $\phi_3 = (\phi_{0,1}, \phi_{0,2}, \phi_{1,2}) = (\alpha, -\pi/2, \gamma)$ in $\mathbb{R}^3$. Computing the corresponding general rotation matrix results:

$$\boldsymbol{R}(\phi_n) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & \sin(\gamma) & \cos(\gamma) \end{bmatrix} \begin{bmatrix} \cos(\pi/2) & 0 & \sin(\pi/2) \\ 0 & 1 & 0 \\ -\sin(\pi/2) & 0 & \cos(\pi/2) \end{bmatrix} \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ \sin(\alpha+\gamma) & \cos(\alpha+\gamma) & 0 \\ -\cos(\alpha+\gamma) & \sin(\alpha+\gamma) & 0 \end{bmatrix}$$

Note that by setting $\beta = -\pi/2$, affecting α or γ yields the same change in the rotation matrix; moreover for any values of α and γ the matrix $\boldsymbol{R}(\phi_3)$ fixes the plane $Z = 0$ and a rotation in the plane $X = 0$ is no longer possible. We have then lost a degree of freedom, and in order for α and γ to have again different effects the values $\beta = \pm\pi/2$ should be avoided. This phenomenon is called Gimbal-Lock and appears in higher dimensions as well. Since in our algorithm, the rotations (see Section 3) performed by the agent step-wise are in just one plane $X_{i,j} \subset \mathbb{R}^n$ one can still use Euler angles and avoid Gimbal-Lock as explained by Shehata (2020) for cases $n = 3, 4$. The strategy is to use the action of $SO(n)$ in $\mathbb{R}^3$ when actualizing the rotation matrix at each step. By the compatibility axiom of group action, for any two rotation matrices $\boldsymbol{R}, \boldsymbol{S} \in SO(n)$ and any vector $\boldsymbol{x} \in \mathbb{R}^n$ the property $\boldsymbol{S}\cdot(\boldsymbol{R}\cdot\boldsymbol{x}) = (\boldsymbol{S}\cdot\boldsymbol{R})\cdot\boldsymbol{x}$ holds. Instead of adding the angle $\phi_{i,j}$ to the corresponding coordinate in the list ϕ_n and recalculating $\boldsymbol{R}(\phi_n)$, let $\boldsymbol{R} = \boldsymbol{R}(\phi_n)$ be the previous rotation matrix and $\boldsymbol{S} = \boldsymbol{R}_{i,j}(\phi_{i,j})$ be the elemental rotation matrix for the rotation at the current step and assigning $\boldsymbol{R} \leftarrow \boldsymbol{S}\cdot\boldsymbol{R}$. Algorithm 1 describes how to calculate a general rotation matrix $\boldsymbol{R}(\phi_n)$ given an ordered list of angles ϕ_n as in Equation 1. It will be used to construct the initial embedding of the surface $\mathcal{C}$ in Algorithm 9 and after each step in Algorithm 10, successfully avoiding Gimbal-Lock.

2.3 N-dimensional Perspective Projection

Projection maps allow us to visualize n-dimensional objects in $\mathbb{R}^3$ conserving particular properties from the original n-dimensional object. One could use for example a Stereographic Projection which is an example example of a conformal map, a mapping preserving angles but not necessary lengths. The main goal of this study is to find immersions in $\mathbb{R}^3$ which minimize the number face and edge intersections, therefore a good starting point is mapping this projected edges to line segments $L \subset \mathbb{R}^3$ and similarly projected faces to plane segments $T \subset \mathbb{R}^3$. A good candidate for such a map is Perspective Projection 2, which maps a point $\boldsymbol{a} \in \mathbb{R}^n$ from a camera position $\boldsymbol{c} \in \mathbb{R}^n$ (also called a vanishing point) to an orthogonal hyperplane $\boldsymbol{p} \in \mathbb{R}^n$, returning a projected point $\boldsymbol{b} \in \mathbb{R}^{n-1}$. Perspective projection mapping then is denoted as

$$\begin{aligned} Proj_n(\boldsymbol{a}_n, \boldsymbol{c}_n, \boldsymbol{e}_n) : \mathbb{R}^n &\rightarrow \mathbb{R}^{n-1} \\ (\boldsymbol{a}_n, \boldsymbol{c}_n, \boldsymbol{e}_n) &\mapsto \boldsymbol{b}. \end{aligned}$$

Consider an edge $e \in Q_1^n$ (resp. a face $f \in \mathcal{C}_2$) and a rotation matrix $\boldsymbol{R} \in SO(n)$ as in Algorithm 1; we apply to every rotated edge $\boldsymbol{R}e \subset \mathbb{R}^n$ (resp. face $\boldsymbol{R}f \in \mathbb{R}^n$) a sequence of perspective projections

$$Proj_4(\cdots(Proj_{n-1}((Proj_n(\boldsymbol{a}_n, \boldsymbol{c}_n, \boldsymbol{e}_n), \boldsymbol{c}_{n-1}, \boldsymbol{e}_{n-1}), \cdots), \boldsymbol{c}_4, \boldsymbol{e}_4) : \mathbb{R}^n \rightarrow \mathbb{R}^3.$$

After each mapping the points $\boldsymbol{b} \in Proj_i(\boldsymbol{a}_i, \boldsymbol{c}_i, \boldsymbol{e}_i) \subset \mathbb{R}^{i-1}$ should always map to the same side of the camera $\boldsymbol{c}_{i-1} \in \mathbb{R}^{i-1}$ in the next perspective projection; otherwise they would be inverted in the next projection. Cubical surfaces are faces on the 5-dimensional cube Q^5 centered at the origin with unit-length edges. It's vertices are of the form $(\pm1/2, \pm1/2, \pm1/2, \pm1/2, \pm1/2) \in \mathbb{R}^5$ and lay on the boundary of the 5-dimensional sphere $\mathcal{S}^5$ of radius $r_5 = \sqrt{5(\pm1/2)^2} = \sqrt{5/4} = \sqrt{5}/2$; or for the n-dimensional case in the boundary of the sphere $\mathcal{S}^n$ of radius $r_n = \sqrt{n}/2$. The camera position $\boldsymbol{c}_5 \in \mathbb{R}^5$ is limited to move in the line segment $(0, 0, 0, 0, d_5) \in \mathbb{R}^5$ with $d_5 \in [-15, -2]$ so the furthest a point can be projected to $\mathbb{R}^4$ occurs when the camera position is $\boldsymbol{c}_5 = (0, 0, 0, 0, -2)$. Consider the 4-dimensional sphere $\mathcal{S}^5$, one must know how far from the origin can any point $\boldsymbol{a} \in \mathcal{S}^5$ project so the range within $\boldsymbol{c}_4 \in \mathbb{R}^4$ can move can be determined. Assume the projection line $L \subset \mathbb{R}^5$ is contained in the plane $X_{1,2}$ spanned by axes $X_1, X_2 \subset \mathbb{R}^5$ (therefore $X_3, X_4, X_5 = 0$) and has equation $X_2 - mX_1 + 2 = 0$. The plane $X_{1,2}$ intersects the 5-sphere in a circle of radius $r_5 = \sqrt{5}/2$ with equation $X_1^2 + X_2^2 - r_5^2 = 0$; we want to minimize the projection of the line L parametrized by m onto the line $X_2 = 0$. For any point $x = (r_5\cos(\theta), r_5\sin(\theta))$ in this circle with $\theta \in (-\pi, \pi)$, the line passing through x and $(0, -2)$ has slope $m = (r_5\sin(\theta) - (-2))/(r_5\cos(\theta) - 0)(r_5\sin(\theta) + 2)/r_5\cos(\theta) = \tan(\theta) + 2/(r_5\cos(\theta))$, and substituting the value of m in the equation of L yields $X_2 - (\tan(\theta) + 2/r_5\cos(\theta))X_1 + 2 = 0$. This line intersects the line $X_2 = 0$ at the point $X_1 = 2/(\tan(\theta) + 2/r_5\cos(\theta))$. As a function defined in the interval $(-\pi, \pi) \subset \mathbb{R}$ it achieves a maximum value $X_1 = 1.348$ at $\theta = -.5932 \approx -\pi/5$ radians, so the projected points $a \in \mathbb{R}^4$ would not map behind the camera $\boldsymbol{c}_4 = (0, 0, 0, -2) \in \mathbb{R}^4$; the 4-dimensional camera can be set as $\boldsymbol{c}_4 = (0, 0, 0, d_4)$ with $d_4 \in [-15, -2] \in \mathbb{R}$.

After applying the sequence of perspective projections as in Section 2.3, the resulting line segments (resp. plane segments) are stored in a list $SegList$ (resp. $FaceList$). This process is detailed in Algorithm 3 and in Appendix A the process of determining whether whether edges in $SegList$ (resp. faces in $FaceList$) intersect in $\mathbb{R}^3$ is explained. For the list of projected edges $SegList$, we assign its elements an edge-radius $w > 0 \in \mathbb{R}$ and determine if the resulting cylindrical segments intersect in pairs.

2.4 Reinforcement Learning

Reinforcement Learning (RL) is a machine learning technique first proposed by Richard Bellman in the 60's and further developed by Sutton et al. (2018) in the recent years. In RL an algorithm called an agent interacts with its environment $\mathcal{E}$ by performing a sequence of actions maximizing a cumulative reward based on feedback received for each action taken. More specifically, at each time-step t the agent takes as input information from the environment called a state $s_t \in \mathcal{S}$ (this is the information that the agent knows about $\mathcal{E}$) and outputs an action $a_t \in \mathcal{A}$ which is passed to $\mathcal{E}$; returning a new state $s_{t+1} \in \mathcal{S}$ and a reward r_t for taking a_t at s_t. Future rewards are multiplied by a **discount factor** $\gamma \in [0, 1)$ at each step. The future discounted return at step t is defined as $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_t$. A policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ is a map from the states to the set of probability distributions over actions mapping $s \mapsto \pi(a|s)$, where $\pi(a|s)$ is the conditional probability of selection the action a at the state s. The action-value function following a policy π is defined as

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi} \left[R_t | s_t = s, a_t = a, \pi \right], a_t \sim \pi(\cdot|s_t). \tag{2}$$

Letting Π be the set of all policies, we define the **optimal action-value function** as the maximum expected return achievable by following any $\pi \in \Pi$ after performing some action $a \in \mathcal{A}$ at a state $s \in \mathcal{S}$; that is $Q^*(s, a) = max_{\pi \in \Pi} \{ \mathbb{E}_{\pi} \left[R_t | s_t = s, a_t = a, \pi \right] \}, a_t \sim \pi(\cdot|s_t)$.

2.4.1 Proximal Policy Optimization

Proximal Policy Optimization Algorithms (PPO) are a family of Reinforcement Learning algorithms proposed by Schulman et al. (2017) which compute an estimation of the policy gradient and plug it into a stochastic gradient ascent algorithm. Among this family of algorithms we use the Clipped Surrogate Objective algorithm which attempts to maximize the objective function $L^{CLIP}(\theta)$ with weights θ of the actor network and weights φ of the critic network. The actor network takes as input a state s_t and outputs an action a_{t+1} and the critic network takes as input a state s_t and outputs the value $V_{\varphi}(s_t)$ of the state s_t. Let $V_{\varphi}(s_t)$ be the this state-value function, and $r_t = \pi_{\theta}(a_t|s_t)/\pi_{\theta_{old}}(a_t|s_t)$ the probability ratio, the **advantages** at state s_t are calculated as

$$\hat{A}_t = Q^{\pi_{\theta_{old}}}(s_t, a_t) - V_{\theta_{old}}(s_t), \tag{3}$$

and the clip objective function to maximize is

$$L^{CLIP}(\theta) = \mathbb{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)], \tag{4}$$

where epsilon is a small real number usually taken as $\epsilon = 0.2$. The actor and critic network weights are updated independently by stochastic gradient descent; in this case ADAM optimizer by Kingma & Ba (2017).

2.5 Neural Network Architecture

In PPO (clip) it is common practice to use two feed-forward neural networks namely the actor network with weights θ_k and the critic network with weights φ_k changing at each iteration k. The actor network takes as input a state s_t and returns an action $a \in \mathcal{A}$. To achieve exploration, an action $a_t \in \mathcal{A}$ is sampled from a multivariate normal distribution with mean a and a specified covariance matrix. To minimize the surrogate loss $-L^{CLIP}(\theta_k)$ (Equation 3) at each iteration k, the actor weights are actualized via stochastic gradient descent. On the other hand, the critic network takes as input a state s_t and estimates the value $V_{\varphi}(s_t)$ which is then used to estimate the advantages $\hat{A}_t$ in Equation 4. To minimize the the mean squared error $L(\varphi) = (R_t - V_{\varphi}(s_t))^2$ at each iteration k, the critic weights are actualized via stochastic gradient descent. We optimize both the actor and critic weights using ADAM optimizer by Kingma & Ba (2017) with a fixed **learning rate** $lr = 3e^{-4}$. Both neural networks have the same architecture consisting on an input layer with 27 neurons (the length of a vector s_t as in Equation 5), two fully connected hidden layers with 64 neurons each, and an output layer with 18 neurons (one per every action $a \in Action$). The output of each layer passes through a RELU activation function.

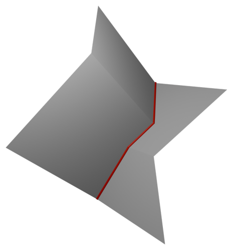


Figure 2: Four plane segments in $\mathbb{R}^3$ intersecting along $FaceInt = 3$ line-segments shown in red.

Figure 3: A pair of overlapping edges in Q_1^5 for edge radius $w > 0 \in \mathbb{R}$.

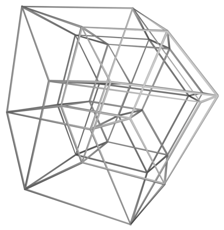


Figure 4: 3-d perspective projection of Q_1^5 with $Overlap = 0$ for edge radius $w = 0.5$.

3 Cubical Surface Self-Intersections as a Finite MDP

A Markov Decision Process (MDP) is a tuple $M = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma)$ where $\mathcal{S}$ is a state space, $\mathcal{A}$ an action space, $\mathcal{P}$ a transition probability function, $\mathcal{R}$ a reward function and $\gamma \in [0, 1)$ a future reward discount factor. Solving M means finding a policy π over $\mathcal{A}$ yielding the supremum of R_t, that is finding π maximizing at each state the *state-value function* $v_\pi(s) = \mathbb{E}_\pi \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...|s_t = s\right]$. Finding immersions of n-dimensional cubical surfaces minimizing face-intersections can be formulated as a (MDP). For the rest of this article, fix $n = 5$ and consider a 5-dimensional cubical surface $\mathcal{C}$. When immersing a non-orientable cubical surface $\mathcal{C}$ in $\mathbb{R}^3$ one always expects to find **face-intersections**, that is pairs of faces in the embedding that intersect along a line segment. To minimize them, at each state s_t the number of pairs of faces intersecting is counted and denoted by $FaceInt(s_t) \geq 0 \in \mathbb{Z}$ also simply called $FaceInt$. To determine whether two faces $f_1, f_2 \subset \mathbb{R}^3$ one can triangulate the projection of each face into two triangles $\{T_1^i, T_2^i\}$ for each f_i and use triangle collision detection for each possible pairing of triangles T_i^1, T_j^2. Appendix A.7 explains the Algorithm by Möller (1997) determining whether two triangles $T_1, T_2 \subset \mathbb{R}^3$ intersect and their intersection line or point. A Python implementation of this algorithm NeonRice (2020) is used in this paper. In general for $\mathcal{C}$ it is unknown how few face-intersections one can achieve, but an **expected minimum number of face-intersections** $Exp \geq 0 \in \mathbb{Z}$ can be set as a goal. At each state s_t the agent attempts to modify $FaceInt(s_t)$ with respect to the **precious number of face-intersections** $PrevFaceInt(s_t) := FaceInt(s_{t-1})$ from two different approaches depending on the **exactness parameter** $Exact$. If the parameter $Exact = True$ the agent attempts to set strictly $FaceInt = Exp$, otherwise it attempts to set $FaceInt \leq Exp$.

3.1 State Space

In this problem setting, there are two types of parameters needed to determine all the information necessary to immerse $\mathcal{C}$ in $\mathbb{R}^3$:

1. The 5-d (resp. 4-d) projection camera distance $d_5 \in [-15, -2] \in \mathbb{R}$ (resp. $d_4 \in [-15, -2] \in \mathbb{R}$) is a scalar parameter representing the 5-d (resp. 4-d) camera position $\boldsymbol{c}_5 = d_5\boldsymbol{e}^{(5)} \in \mathbb{R}^5$ (resp. $\boldsymbol{c}_4 = d_4\boldsymbol{e}^{(4)} \in \mathbb{R}^4$). The cubical surface $\mathcal{C}$ is projected to $\mathbb{R}^3$ as in Algorithm 2, fixing the projection hyperplane at the position $\boldsymbol{e}_5 = (0, 0, 0, 0, 0) \in \mathbb{R}^5$ (resp. $\boldsymbol{e}_4 = 10\boldsymbol{e}^{(4)} \in \mathbb{R}^4$ which serves as a scaling factor) with respect to the origin.

2. Section 2.2 explains how the orientation of $\mathcal{C}$ around the origin in $\mathbb{R}^n$ can be described by a general 5 by 5 rotation matrix $\boldsymbol{R} = (\boldsymbol{R}_{i,j})$ with entries $(-1 \leq \boldsymbol{R}_{i,j} \leq 1)$.

Therefore, at any time-step t the embedding of $\mathcal{C}$ is parameterized by a state (in this case also a vector)

$$s_t = (d_5, d_4, \boldsymbol{R}_{1,1}, \boldsymbol{R}_{1,2}, ..., \boldsymbol{R}_{5,4}, \boldsymbol{R}_{5,5}) \in \mathbb{R}^{27} \tag{5}$$

where the first two entries give the agent information about the camera distances and the rest are the entries of the general rotation matrix $\boldsymbol{R} = (\boldsymbol{R}_{i,j})$ at step t.

3.2 Action Space

The action space $\mathcal{A}$ describes how the agent can interact with its environment $\mathcal{E}$. The agent receives a state $s_t \in \mathcal{S}$ from $\mathcal{E}$ and selects one of the possible actions $a \in \mathcal{A}$ which then takes it to a new state s_{t+1}. This action space used here consists on the discrete set $\mathcal{A} := \{0, 1, ..., 17\}$; each number executing either a camera modification or a 5-dimensional rotation. The following dictionary identifies each action $a \in \mathcal{A}$ with a particular vector:

$$ActVec = \{0 : \delta \boldsymbol{e}^{(1)}, 1 : -\delta \boldsymbol{e}^{(1)}, 2 : \delta \boldsymbol{e}^{(2)}, 3 : -\delta \boldsymbol{e}^{(2)}, 4 : \epsilon \boldsymbol{e}^{(5)}, 5 : -\epsilon \boldsymbol{e}^{(5)}, \cdots, 16 : \epsilon \boldsymbol{e}^{(12)}, 17 : -\epsilon \boldsymbol{e}^{(12)}\}, \quad (6)$$

where $\boldsymbol{e}^{(i)}, (1 \leq i \leq 12, i \neq 3, 4, 7)$ are the canonical basis vectors in $\mathbb{R}^{12}$, and $\delta, \epsilon > 0 \in \mathbb{R}$ are small positive real numbers. We eliminate the actions corresponding to $i = 3, 4, 7$ because they correspond to rotations in planes $X_{0,1}, X_{0,2}, X_{1,2}$ (equivalently Z, Y, X) respectively around which rotating doesn't yield any change on $FaceInt$ or $Overlap$. The roles of parameters δ, ϵ are the following:

1. Actions $a \in \{0, 1\}$ (resp. $a \in \{2, 3\}$) modify the distance d_5 (resp. d_4) of the 5-d (resp. 4-d) camera by a small distance $\delta > 0 \in \mathbb{R}$. To modify the 5-d (resp. 4-d) camera position we simply add $\pm\delta$ to the last coordinate of the camera vector, i.e. $\boldsymbol{c}_5[4] \leftarrow \boldsymbol{c}_5[4] + ActVec[a][0]$ (resp. $\boldsymbol{c}_4[3] \leftarrow \boldsymbol{c}_4[3] + ActVec[a][1]$).
2. Actions $a \in \{4, ..., 17\}$ apply a small rotation-step $\pm\epsilon > 0 \in \mathbb{R}$ in one of the planes $X_{i,j} \subset \mathbb{R}^5$. The rotation-step is taken as $\epsilon = \alpha\pi/180$ (α degrees) if at the current state s_t it holds that $FaceInt(s_t) > Exp$ and $\epsilon = \pi/180$ (one degree) if at the current state s_t it holds that $FaceInt(s_t) \leq Exp$. Section 4 shows training plots for $\alpha = 1, 2, 5$ degrees which allow to determine the best rotation-step size for each of the surfaces we study here. This technique is intended to explore the environment in a wider extent but switching to small steps when the agent is close to a solution. To rotate a surface $\mathcal{C}$ by an action $a \in \{4, ..., 17\}$ one must take the entries describing the rotation $ActVec[a][2, :] \in \mathbb{R}^{10}$, calculate the elemental rotation matrix $\boldsymbol{S} = RotMat(ActVec[a][2, :])$ as in Algorithm 1, multiply $\boldsymbol{S}$ on the left of the previous rotation matrix $\boldsymbol{R}$, and finally assigning $\boldsymbol{R} \leftarrow \boldsymbol{S} \cdot \boldsymbol{R}$. The new entries $(\boldsymbol{R}_{i,j})$ are passed to the next state $s_{t+1} \in \mathcal{S}$ as in Equation 5.

3.3 Reward Functions

Dense rewards are received by the agent after achieving some goal while sparse rewards are received at each step. In this paper a combination of both is used and the reward function is the sum of the following:

1. Reward 5 prevents the agent from taking two consecutive inverse actions a_t, a_{t+1} that yield no change in the state; for example rotating $+\epsilon$ after rotating $-\epsilon$ on the same plane $X_{i,j}$. If this is the case the agent takes a reward $r_1 = -1$. This is an example of a sparse reward.
2. The second reward function has to do with the **observation space**. The camera parameters d_4 and d_5 can range in the closed interval $[-15, -2] \in \mathbb{R}$. Similarly any rotation matrix $\boldsymbol{R} = (\boldsymbol{R}_{i,j}) \in SO(5)$ has entries within the interval $[-1, 1]$ (see Section 2.2). We define the Observation Space as

$$ObsSpace := [-15, -2] \times [-15, -2] \times [-1, 1] \times \cdots \times [-1, 1] \subset \mathbb{R}^{27}. \quad (7)$$

   Whenever the agent performs an action a_t such that that the next state $s_{t+1} \notin ObsSpace$, then it receives a reward $r_2 = -1$. The agent receives no reward for staying within these bounds (see Algorithm 6); this is an example of a sparse reward.
3. At each step, the agent should attempt to reduce $FaceInt(s_t)$ (see Algorithm 14) with respect to $PrevFaceInt(s_t)$. The reward in Algorithm 7 is dense because it is given to the agent at every step if at the current state $FaceInt > Exp$ (resp. $FaceInt \neq Exp$) for $Exact = False$ (resp. for $Exact = True$). On the other hand, if $FaceInt \leq Exp$ (resp. $FaceInt = Exp$) for $Exact = False$ (resp. $Exact = True$) the agent should not get this reward. This reward is intended to incentivize the agent to transition into states at which $FaceInt$ is closer to Exp by monotonically decreasing $FaceInt$ during the exploration stage; although in some cases $FaceInt$ may need to increase in order to reach new minima like in the first row in Figure 23.

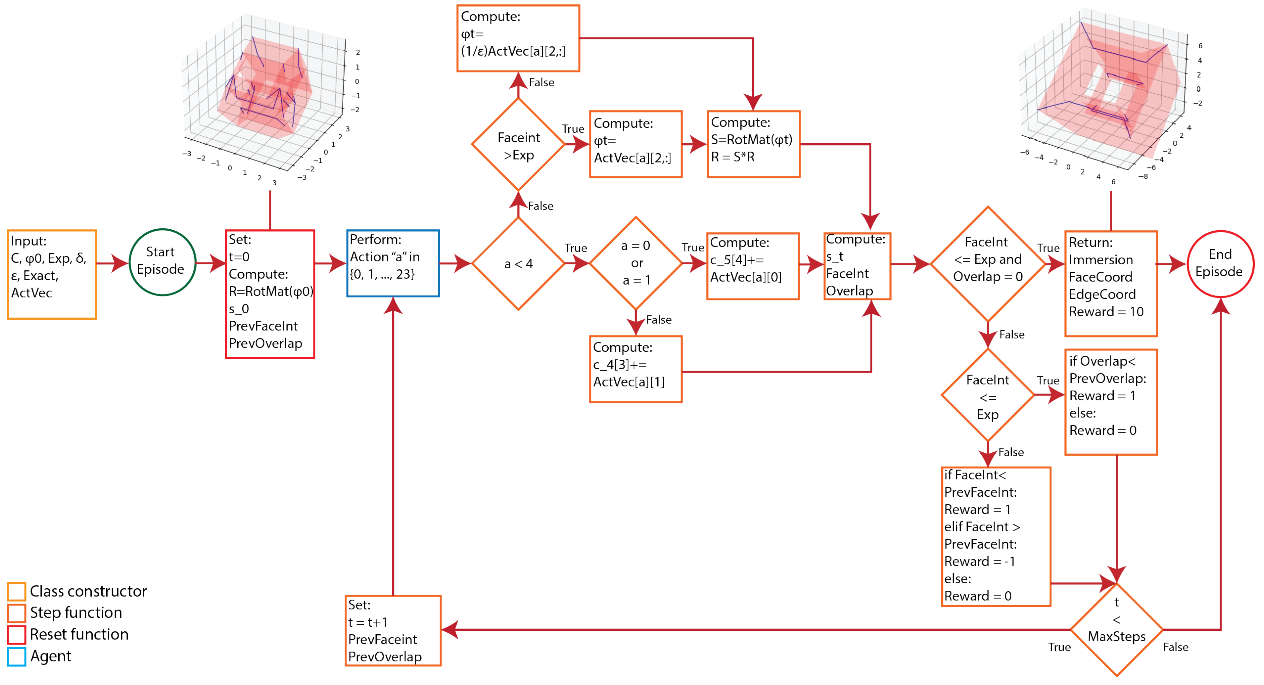
4. For a cubical surface $\mathcal{C}$ the task has two types of solutions depending on the $Exact$ parameter. If $Exact = False$ (resp. $Exact = True$), the agent's task is to set $Overlap = 0$ (see Algorithm 13) if $FaceInt \leq Exp$ (resp. $FaxeInt = Exp$) (see Algorithm 8). If the agent finds a solution in any of these situations then it receives $r_4 = 10$. Note that the agent will tend to find solutions (terminal states s_T) requiring less steps because the penalization given by the future discount reward $\gamma^{t'-t}$ will have a smaller effect on the future discounted return $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_t$ throughout the intermediate steps s_t.

3.4 Termination Criteria

There are two termination criteria. The first one has to do with finding a solution of the problem. Given a cubical surface $\mathcal{C}$ the task has two types of solutions depending on the $Exact$ parameter. If $Exact = True$, then the objective of the agent is arriving to a terminal state s_T at which $Overlap(s_T) = 0$ and $FaceInt(s_T) = Exp$; otherwise will be arriving to a terminal state s_T at which $Overlap(s_T) = 0$ and $FaceInt(s_T) \leq Exp$. This is formalized in Algorithm 4 (1). The second one is an episode truncation which ends the episode once the agent exceeds a **maximum number of steps** $MaxSteps > 0 \in \mathbb{Z}$. For some surfaces we must allow the agent to explore the environment further by increasing the allowed $MaxSteps$ but in this work a $MaxSteps = 100$ is tested.

3.5 The Environment

The environment $\mathcal{E}$ is a class containing a constructor, a step function describing how the agent and the environment interact, and a reset function which runs if $Done \leftarrow True$. The constructor passes the parameters from the user and assigns them to the environment class. The step function takes an action a_t given by the agent at a state s_t and outputs the resulting state s_{t+1} and the resulting reward r_t. The reset function simply sets the environment into its initial state s_0 after $Done \leftarrow True$. Figure 5 explains the algorithm controlling the environment. A more detailed pseudo-code with each of its components can be consulted in in Section A.5, Algorithms 9, 10 and 11.



Input: C, φ0, Exp, δ, ε, Exact, ActVec
Start Episode
Set: t=0 Compute: R=RotMat(φ0) s_0 PrevFaceInt PrevOverlap
Perform: Action "a" in {0, 1, ..., 23}
a < 4
True
False
Faceint >Exp
True
False
Compute: φt= (1/ε)ActVec[a][2,:]
Compute: φt= ActVec[a][2,:]
Compute: S=RotMat(φt) R = S*R
a = 0 or a = 1
Compute: c_5[4]+= ActVec[a][0]
Compute: c_4[3]+= ActVec[a][1]
Compute: s_t FaceInt Overlap
FaceInt <= Exp and Overlap = 0
Return: Immersion FaceCoord EdgeCoord Reward = 10
End Episode
FaceInt <= Exp
if Overlap< PrevOverlap: Reward = 1 else: Reward = 0
if FaceInt< PrevFaceInt: Reward = 1 elif FaceInt > PrevFaceInt: Reward = -1 else: Reward = 0
t < MaxSteps
Set: t = t+1 PrevFaceint PrevOverlap
Class constructor
Step function
Reset function
Agent

Figure 5: Episode flow diagram

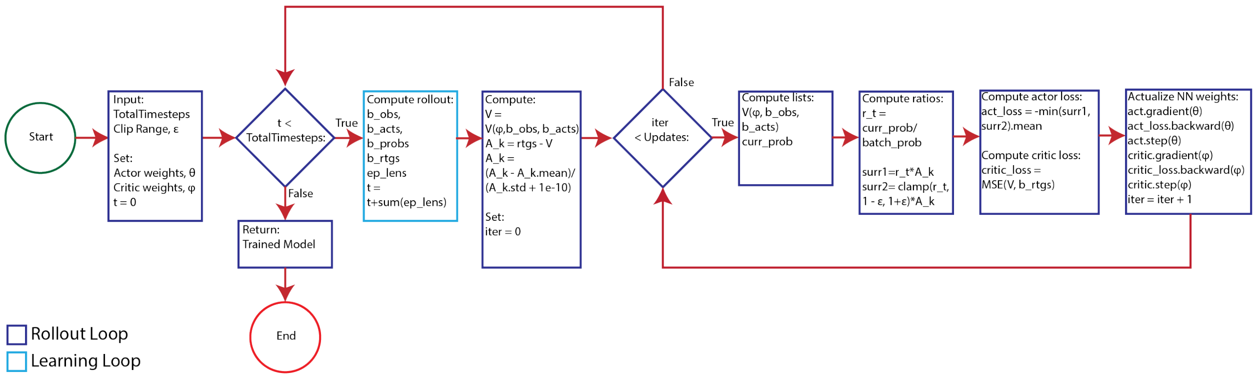


Figure 6: PPO (clip) agent flow diagram.

3.6 The Agent

Proximal Policy Optimization algorithms samples a batch of size $N > 0$ of data $s_t, a_t, \pi(a_t|s_t), r_t$ following an initial policy π_{old}. When $done = True$ the future discounted reward R_t is calculated as in Section 3 the episode length is saved. The R_t estimate the action-value functions $Q^{\pi_{\theta_{old}}}(s, a)$ used to calculate the advantages $\hat{A}_t$ in Equation 3. In practice, a $MinibatchSize > 0$ of elements is sampled form the memory of size N. According to Keskar et al. (2016) a larger $MinibatchSize$ tends to find sharper minima (leading to poor generalization), while small batch sizes tend to find flat minima (allowing better generalization). Here a $MinibatchSize = 32$ is used to achieve some generalization. Algorithm 1 in Schulman et al. (2017) shows the PPO (clip) algorithm workflow. The models were trained using Stable-Baselines 3, an implementation by Raffin et al. (2021) which follows the logic of Figure 6.

4 Experiments

Recall from Section 3 that the rotation steps the agent is allowed to perform via an elementary rotation matrix are specified by the user. A small step size will not be the best way to explore the entire configuration space but will be efficient to explore near a particular configuration. On the other hand, a large step size will be better to explore the whole configuration space but can miss good configurations between steps. Each of the eight minimal cubical surfaces here presented is trained for 204800 steps with different rotation-step sizes, namely $\epsilon = 1$ (red), $\epsilon = 2$ (blue) and $\epsilon = 5$ (orange) degrees to find the best ϵ parameter for each surface. We set the number of steps sampled on the rollout function to $Updates = 2048$ (see Figure 6) from which a $MiniBatchSize = 32$ is sampled. During training, the initial state s_0 is fixed for each cubical surface and specified in Tables 4.3, 4.4. The expected number of face-intersections $Exp \geq 0$ is specified for each surface depending on the minimum $FaceInt$ observed in previous runs; if during training a $FaceInt < Exp$ is reached then the training is repeated with the smallest Exp found. The rest of the training parameters are fixed to: $\delta = .5$, $Exact = False$, $w = .05$, $MaxSteps = 100$, and $\gamma = .99$.

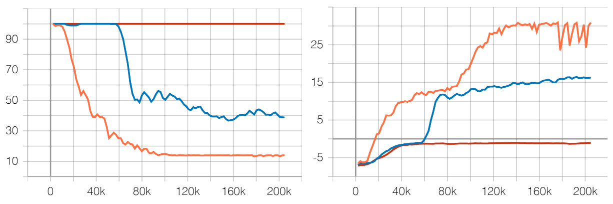


Figure 7: Genus-1 torus with $Exp = 0$. Left: Episode length mean. Right: Episode reward mean.

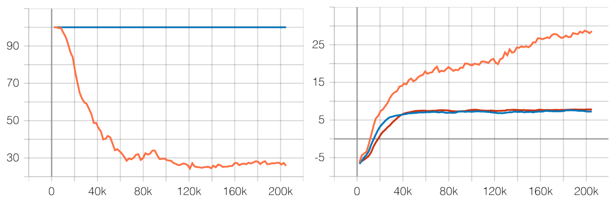


Figure 8: Genus-2 torus with $Exp = 0$. Left: Episode length mean. Right: Episode reward mean.

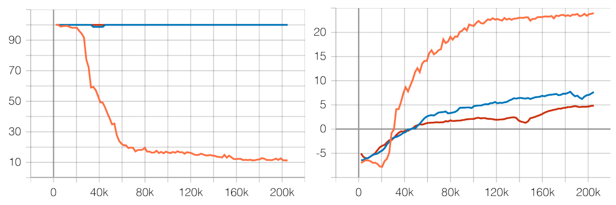


Figure 9: Genus-3 torus with $Exp = 9$. Left: Episode length mean. Right: Episode reward mean.

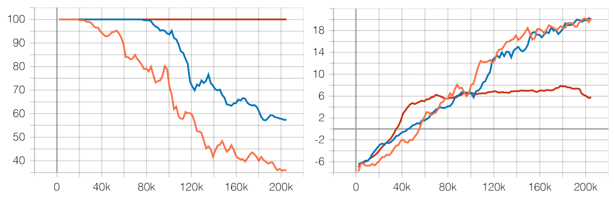


Figure 10: Genus-4 torus with $Exp = 12$. Left: Episode length mean. Right: Episode reward mean.

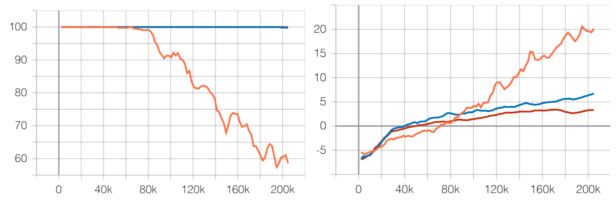


Figure 11: Genus-5 torus with $Exp = 16$. Left: Episode length mean. Right: Episode reward mean.

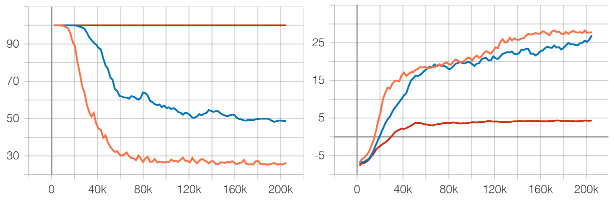


Figure 12: Projective Plane with $Exp = 3$. Left: Episode length mean. Right: Episode reward mean.

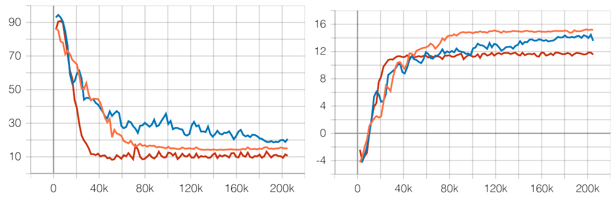


Figure 13: Klein Bottle with $Exp = 3$. Left: Episode length mean. Right: Episode reward mean.

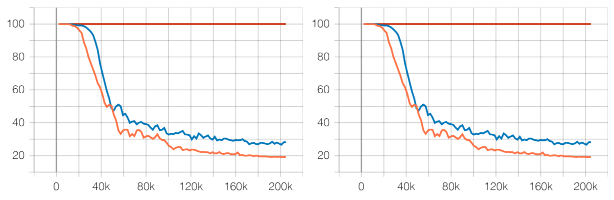


Figure 14: K-3 surface with $Exp = 6$. Left: Episode length mean. Right: Episode reward mean.

4.1 Initial and Final Immersions of some Cubical Surfaces

In the following plots, lines in red represent face-intersections, and the edges of the 5-cube are not rendered for better understanding of the surface. Recall from Section 2.2 that the initial orientation of a cubical surface is determined from the list of angles $\phi_5 = (\phi_{0,1}, \phi_{0,2}, \phi_{0,3}, \phi_{0,4}, \phi_{1,2}, \phi_{1,3}, \phi_{1,4}, \phi_{2,3}, \phi_{2,4}, \phi_{3,4})$ sorted in lexicographic order. For each cubical surface and choice of rotation-step $\epsilon \in \{1, 2, 3\}$ the last trained model saved is tested. Having a high mean episode reward translates into finding a solution in less steps. Table 1 presents the number of steps each trained model requires to find a solution.

$Surface$	$\epsilon = 1$	$\epsilon = 2$	$\epsilon = 5$
Genus-1	> 100	35	**13**
Genus-2	> 100	> 100	**25**
Genus-3	> 100	> 100	**11**
Genus-4	> 100	56	**39**
Genus-5	> 100	> 100	**57**
Projective Plane	> 100	45	**25**
Klein Bottle	**9**	15	14
K-3	> 100	39	**19**

Table 1: Number of time-steps required to find a solution.

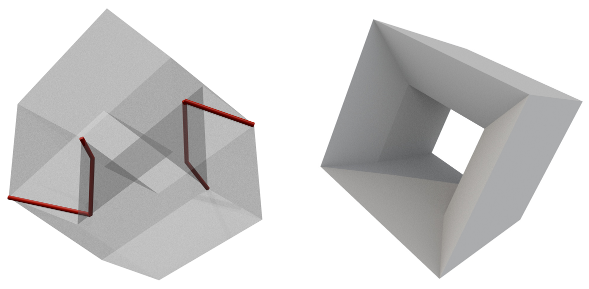


Figure 15: Genus-1 torus (16 faces). Left: Initial with $FaceInt = 6$. Right: Optimized with $Faceint = 0$.

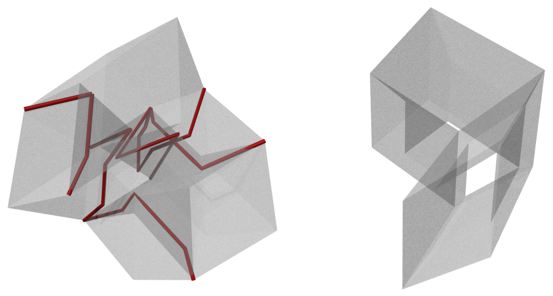


Figure 16: Genus-2 torus (26 faces). Left: Initial with $FaceInt = 23$. Right: Optimized with $Faceint = 0$.

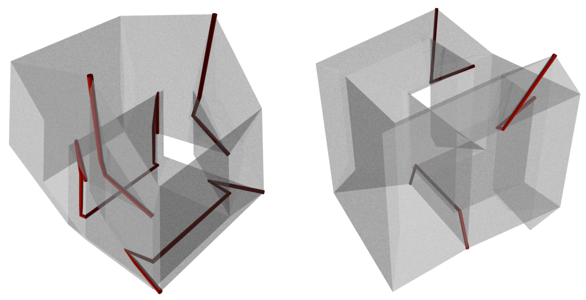


Figure 17: Genus-3 torus (36 faces). Left: Initial with $FaceInt = 20$. Right: Optimized with $FaceInt = 9$.

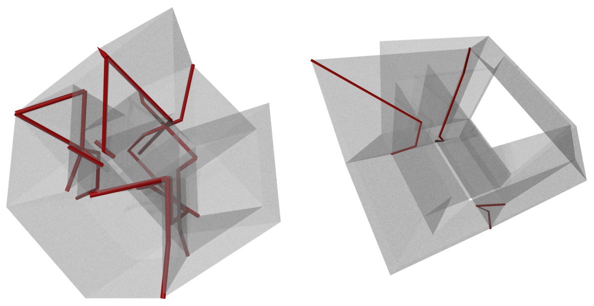


Figure 18: Genus-4 torus (38 faces). Left:Initial with $FaceInt = 36$. Right: Optimized with $FaceInt = 10$.

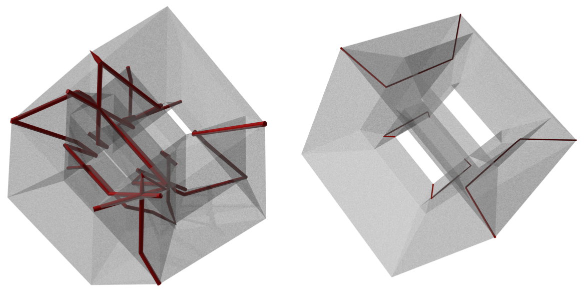


Figure 19: Genus-5 torus (40 faces). Left: Initial with $FaceInt = 48$. Right: Optimized with $FaceInt = 16$.

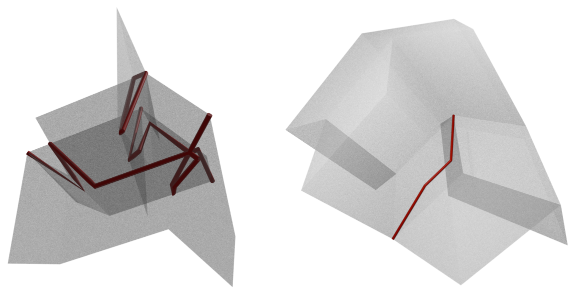


Figure 20: Projective Plane (20 faces). Left: Initial with $FaceInt = 17$. Right: Optimized with $FaceInt = 3$.

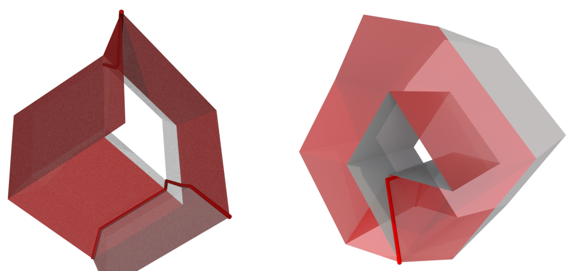


Figure 21: Klein Bottle (24 faces). Left: Initial with $FaceInt = 9$. Right: Optimized with $FaceInt = 3$.

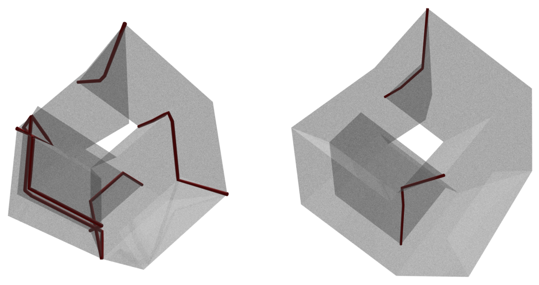
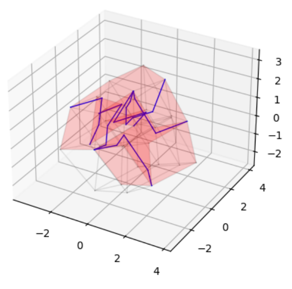
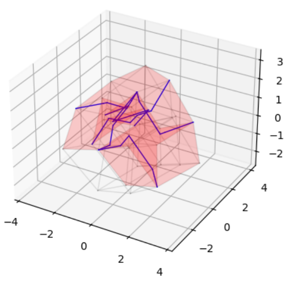
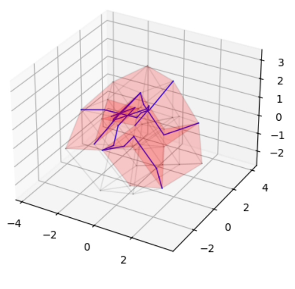
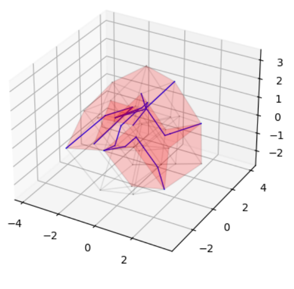


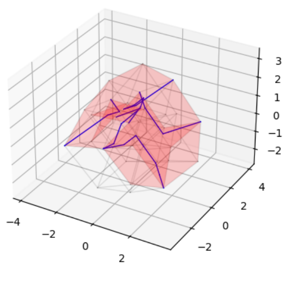
Figure 22: K-3 surface (30 faces). Left: Initial with $FaceInt = 23$. Right: Optimized with $FaceInt = 6$.

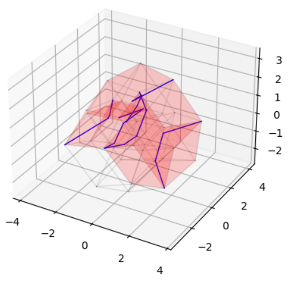

(a) $FaceInt = 23, Overlap = 2$

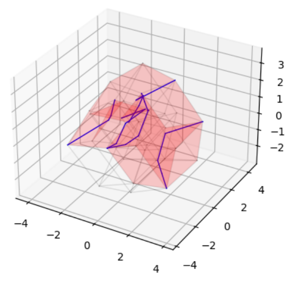

(b) $FaceInt = 22, Overlap = 7$

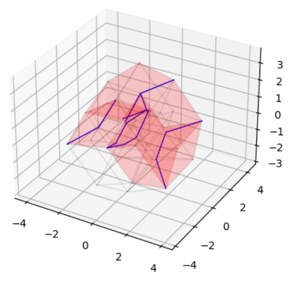

(c) $FaceInt = 26, Overlap = 2$

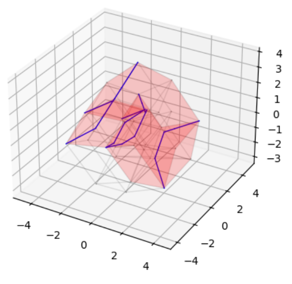

(d) $FaceInt = 21, Overlap = 6$

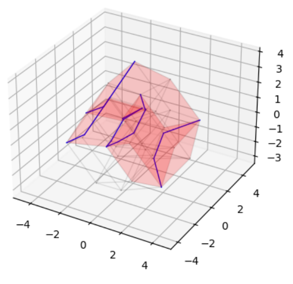

(e) $FaceInt = 20, Overlap = 4$

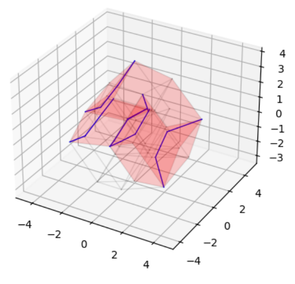

(f) $FaceInt = 19, Overlap = 8$

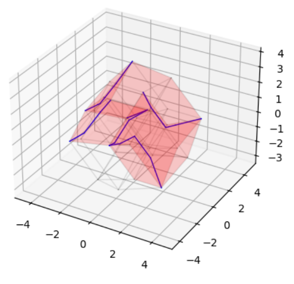

(g) $FaceInt = 19, Overlap = 6$

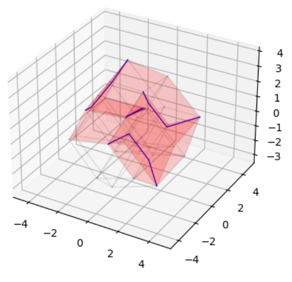

(h) $FaceInt = 19, Overlap = 6$

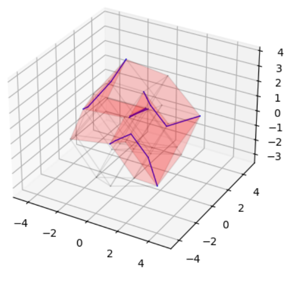

(i) $FaceInt = 18, Overlap = 6$

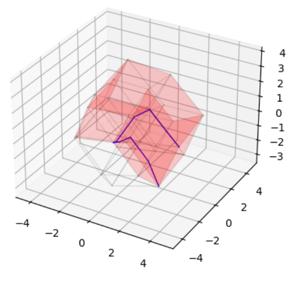

(j) $FaceInt = 18, Overlap = 4$

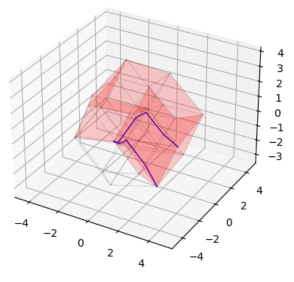

(k) $FaceInt = 18, Overlap = 4$

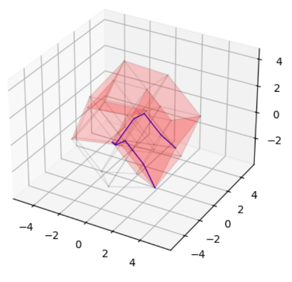

(l) $FaceInt = 17, Overlap = 4$

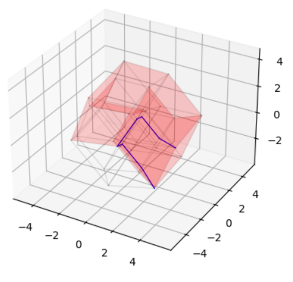

(m) $FaceInt = 13, Overlap = 6$

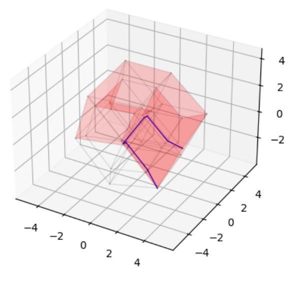

(n) $FaceInt = 13, Overlap = 4$


(o) $FaceInt = 9, Overlap = 7$


(p) $FaceInt = 9, Overlap = 2$


(q) $FaceInt = 9, Overlap = 2$


(r) $FaceInt = 7, Overlap = 2$


(s) $FaceInt = 7, Overlap = 0$

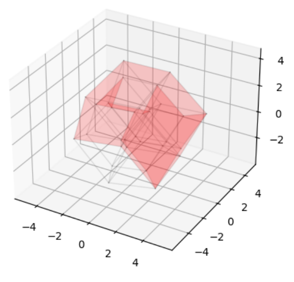

(t) $FaceInt = 0, Overlap = 3$

Figure 23: Face optimization stage for the orientable genus-2 cubical surface in Figure 16 with $Exp = 0$ and $\epsilon = 5$ degrees rotation-step. Faces are shown in red and face-intersections in blue. The 5-cube's 1-skeleton is rendered in light gray.

4.2 An Optimization Sequence

Figure 23 shows the face-intersection minimization sequence for the genus-2 cubical surface in Figure 16 and rotation-step $\epsilon = 5$ degrees is presented. Here, $FaceInt$ is increased only once in Figure 23c, in all other steps $FacInt(s_{t+1}) \leq FaceInt(s_t)$ holds. The edge-overlap minimization stage is not presented since it is hard to appreciate in a plot. Each of the 25 steps the agent performs is a frame in an animation sequence; this allows capturing the effects of the 5-d rotations and perspective projections needed to transform s_0 into s_T in a realistic way. Creating this animation only with the initial and final immersions (without any intermediate one) would only yield a linear vertex displacement from their initial to the final locations which would not corresponding to the effect of a 5-d rotation.

4.3 Orientable Cubical Surfaces

The following table summarizes the results for orientable cubical surfaces.

g	Initial Orientation (ϕ_5)	$FaceInt(s_0)$	$FaceInt(s_T)$	$Overlap(s_0)$	$Overlap(s_T)$
1	$(0, 0, \frac{\pi}{6}, \frac{\pi}{6}, 0, \frac{\pi}{6}, \frac{\pi}{6}, \frac{\pi}{6}, \frac{\pi}{6}, \frac{\pi}{6})$	6	0	2	0
2	$(0, 0, \frac{\pi}{6}, \frac{\pi}{6}, 0, \frac{\pi}{6}, \frac{\pi}{6}, \frac{\pi}{6}, \frac{\pi}{6}, \frac{\pi}{6})$	23	0	2	0
3	$(0, 0, 0, 0, 0, 0, \frac{\pi}{6}, \frac{\pi}{6}, \frac{\pi}{6}, \frac{\pi}{6})$	20	9	0	0
4	$(0, 0, \frac{\pi}{6}, \frac{\pi}{6}, 0, \frac{\pi}{6}, \frac{\pi}{6}, \frac{\pi}{6}, \frac{\pi}{6}, \frac{\pi}{6})$	36	10	2	0
5	$(0, 0, \frac{\pi}{6}, \frac{\pi}{6}, 0, \frac{\pi}{6}, \frac{\pi}{6}, \frac{\pi}{6}, \frac{\pi}{6}, \frac{\pi}{6})$	48	16	2	0

4.3.1 Genus-1 Cubical Surface

Figure 15 shows the initial (left) and optimized (right) immersions of the $g = 1$ cubical surface:

$$\{ (0,0,0,2,2), (0,0,1,2,2), (0,0,2,0,2), (0,0,2,1,2), (0,1,0,2,2), (0,1,1,2,2), (0,1,2,0,2), (0,1,2,1,2), (0,2,0,2,0), (0,2,0,2,1), (0,2,1,2,0), (0,2,1,2,1), (0,2,2,0,0), (0,2,2,0,1), (0,2,2,1,0), (0,2,2,1,1)\}$$

4.3.2 Genus-2 Cubical Surface

Figure 16 shows the initial (left) and optimized (right) immersions of the $g = 2$ cubical surface:

$$\{ (0,0,0,2,2), (0,0,1,2,2), (0,0,2,0,2), (0,0,2,1,2), (0,1,0,2,2), (0,1,1,2,2), (0,2,0,2,0), (0,2,0,2,1), (0,2,1,2,0), (0,2,1,2,1), (0,2,2,0,0), (0,2,2,0,1), (0,2,2,1,0), (0,2,2,1,1), (1,1,0,2,2), (1,1,1,2,2), (1,1,2,2,0), (1,1,2,2,1), (2,1,0,0,2), (2,1,0,1,2), (2,1,1,0,2), (2,1,1,1,2), (2,1,2,0,0), (2,1,2,0,1), (2,1,2,1,0), (2,1,2,1,1)\}$$

4.3.3 Genus-3 Cubical Surface

Figure 17 shows the initial (left) and optimized (right) immersions of the $g = 3$ cubical surface:

$$\{ (0,0,0,2,2), (0,0,2,0,2), (0,0,2,1,2), (0,0,2,2,0), (0,1,0,2,2), (0,1,2,0,2), (0,1,2,1,2), (0,2,0,2,1), (0,2,1,0,2), (0,2,1,1,2), (0,2,1,2,0), (0,2,1,2,1), (1,0,0,2,2), (1,0,2,0,2), (1,0,2,1,2), (1,0,2,2,0), (1,1,0,2,2), (1,1,2,0,2), (1,1,2,1,2), (1,2,0,2,1), (1,2,1,0,2), (1,2,1,1,2), (1,2,1,2,0), (1,2,1,2,1), (2,0,1,2,1), (2,0,2,0,1), (2,0,2,1,1), (2,1,0,2,0), (2,1,1,2,0), (2,1,1,2,1), (2,1,2,0,0), (2,1,2,0,1), (2,1,2,1,0), (2,1,2,1,1), (2,2,0,0,1), (2,2,0,1,1)\}$$

This surface's initial orientation is different from the rest so the agent could find some solution within the $MaxSteps = 100$ specified. With this modification the agent was able to find a solution only with a rotation-step $\epsilon = 5$ degrees.

4.3.4 Genus-4 Cubical Surface

Figure 18 shows the initial (left) and optimized (right) immersions of the $g = 4$ cubical surface:

$$\{ (0,0,0,2,2), (0,0,1,2,2), (0,0,2,0,2), (0,0,2,1,2), (0,1,0,2,2), (0,1,1,2,2), (0,1,2,1,2), (0,2,0,2,0), (0,2,0,2,1), (0,2,1,2,0), (0,2,1,2,1), (0,2,2,0,0), (0,2,2,0,1), (1,0,0,2,2), (1,0,1,2,2), (1,0,2,0,2), (1,0,2,1,2), (1,1,0,2,2), (1,1,1,2,2), (1,1,2,1,2), (1,2,0,2,0), (1,2,0,2,1), (1,2,1,2,0), (1,2,1,2,1), (1,2,2,0,0), (1,2,2,0,1), (2,0,2,1,0), (2,0,2,1,1), (2,1,0,0,2), (2,1,1,0,2), (2,1,2,0,0), (2,1,2,0,1), (2,1,2,1,0), (2,1,2,1,1), (2,2,0,1,0), (2,2,0,1,1), (2,2,1,1,0), (2,2,1,1,1)\}$$

4.3.5 Genus-5 Cubical Surface

Figure 19 shows the initial (left) and optimized (right) immersions of the $g = 5$ cubical surface:

$$\{ \quad (0,0,0,2,2),(0,0,1,2,2),(0,0,2,0,2),(0,0,2,1,2),(0,1,0,2,2),(0,1,1,2,2),(0,1,2,0,2),(0,1,2,1,2),(0,2,0,2,0),(0,2,0,2,1),$$
$$(0,2,1,2,0),(0,2,1,2,1),(1,0,0,2,2),(1,0,1,2,2),(1,0,2,0,2),(1,0,2,1,2),(1,1,0,2,2),(1,1,1,2,2),(1,1,2,0,2),(1,1,2,1,2),$$
$$(1,2,0,2,0),(1,2,0,2,1),(1,2,1,2,0),(1,2,1,2,1),(2,0,2,0,0),(2,0,2,0,1),(2,0,2,1,0),(2,0,2,1,1),(2,1,2,0,0),(2,1,2,0,1),$$
$$(2,1,2,1,0),(2,1,2,1,1),(2,2,0,0,0),(2,2,0,0,1),(2,2,0,1,0),(2,2,0,1,1),(2,2,1,0,0),(2,2,1,0,1),(2,2,1,1,0),(2,2,1,1,1)$$

4.4 Non-Orientable Cubical Surfaces

The following table summarizes the results for non-orientable cubical surfaces.

k	Initial Orientation (ϕ_5)	$FaceInt(s_0)$	$FaceInt(s_T)$	$Overlap(s_0)$	$Overlap(s_T)$
1	$(0,0,\frac{\pi}{6},\frac{\pi}{6},0,\frac{\pi}{6},\frac{\pi}{6},\frac{\pi}{6},\frac{\pi}{6},\frac{\pi}{6})$	17	3	2	0
2	$(0,0,\frac{\pi}{6},\frac{\pi}{6},0,\frac{\pi}{6},\frac{\pi}{6},\frac{\pi}{6},\frac{\pi}{6},\frac{\pi}{6})$	9	3	2	0
3	$(0,0,\frac{\pi}{6},\frac{\pi}{6},0,\frac{\pi}{6},\frac{\pi}{6},\frac{\pi}{6},\frac{\pi}{6},\frac{\pi}{6})$	23	6	2	0

4.4.1 Projective Plane

Figure 20 shows the initial (left) and optimized (right) immersions of the $k = 1$ cubical surface:

$$\{ \quad (0,0,0,2,2),(0,0,1,2,2),(0,0,2,0,2),(0,0,2,2,0),(0,1,2,1,2),(0,2,0,1,2),(0,2,1,2,1),(0,2,2,1,0),(0,2,2,1,1),(1,0,2,2,1),$$
$$(1,2,1,1,2),(1,2,1,2,1),(2,0,0,2,1),(2,0,1,1,2),(2,0,2,0,1),(2,0,2,1,1),(2,1,1,1,2),(2,1,1,2,1),(2,2,1,0,1),(2,2,1,1,0)\}$$

4.4.2 Klein Bottle

Figure 21 shows the initial (left) and optimized (right) immersions of the $k = 2$ cubical surface:

$$\{ \quad (0,0,0,2,2),(0,0,2,0,2),(0,0,2,1,2),(0,0,2,2,0),(0,1,2,0,2),(0,1,2,1,2),(0,1,2,2,0),(0,1,2,2,1),$$
$$(0,2,0,2,1),(0,2,1,0,2),(0,2,1,1,2),(0,2,1,2,0),(0,2,1,2,1),(1,0,2,2,1),(1,1,0,2,2),(1,2,0,2,1),$$
$$(2,0,1,2,1),(2,0,2,0,1),(2,0,2,1,1),(2,1,0,0,2),(2,1,0,1,2),(2,1,0,2,0),(2,2,0,0,1),(2,2,0,1,1)\}.$$

The faces in red are equivalent to a Möbius Strip. In this minimal immersion of a cubical Klein Bottle there exist two parallel Möbius Strips joined by the gray set of faces in the figure on the right.

4.4.3 K-3 surface

Figure 22 shows the initial (left) and optimized (right) immersions of the $k = 3$ cubical surface:

$$\{ \quad (0,0,0,2,2),(0,0,2,0,2),(0,0,2,1,2),(0,0,2,2,0),(0,1,2,0,2),(0,1,2,2,0),(0,1,2,2,1),(0,2,0,2,1),(0,2,1,0,2),(0,2,1,1,2),$$
$$(0,2,1,2,0),(0,2,1,2,1),(1,0,2,1,2),(1,1,0,2,2),(1,1,2,2,1),(1,2,0,1,2),(1,2,1,1,2),(1,2,1,2,1),(1,2,2,0,1),(1,2,2,1,0),$$
$$(2,0,1,2,1),(2,0,2,0,1),(2,0,2,1,1),(2,1,0,0,2),(2,1,0,2,0),(2,1,1,1,2),(2,1,2,1,0),(2,1,2,1,1),(2,2,0,0,1),(2,2,0,1,1)\}$$

5 Conclusions

This article presents the first setting to minimize the face-intersections of any 5-dimensional cubical surface immersed in $\mathbb{R}^3$ by perspective projection. The agent modifies an initial immersion by sequentially applying the surface a 5-dimensional rotation or modifying the projection-distance parameters. For orientable cubical surfaces with genus $g = 1, 2$ the algorithm removes all face-intersections, while for orientable with $g = 3, 4, 5$ and non-orientable with $k = 1, 2, 3$ they are reduced to a minimum possible known so far. Moreover, the strategy found by the agent is such that a small number of steps is required and the face-intersections are decreased as monotonically as possible. This is particularly useful when animating the agent's strategy because the animations evidence an efficient rotation strategy to transform a visually complicated immersion into one whose topological features can be visualized and understood easier.

6 Further directions

For orientable cubical surface with genus $g = 1, 2$ the RL algorithm here introduced is able to find an embedding using perspective projections by performing a sequence of actions sequentially minimizing $FaceInt$.

However, for genus $g = 3, 4, 5$ the RL algorithm can not reduce $FaceInt$ below the results shown in Section 4 even though these surfaces are orientable and could be embedded allowing the unitary 5-cube to be deformed in some way. A possible solution can be adapting the RL algorithm to a continuous action-space framework eliminating the need of perspective projection and rotation-step size parameters δ and ϵ, but some considerations regarding Gimbal-Lock and non-commutativity of rotation matrices should be made.

For non-orientable cubical surfaces, a possible direction of interest could be finding immersions of cubical surfaces whose self-intersections have special properties. For example, the Boy's surface is an immersion in $\mathbb{R}^3$ of the projective plane whose intersection curve has a triple point (and no other singularities). It is natural to ask whether some non-orientable cubical surface homeomorphic to a Projective plane can be immersed in $\mathbb{R}^3$ with its intersection lines having these same properties and what is the least amount of faces required.

The RL algorithm is being tested with the 6-dimensional cubical surfaces found in the GitHub repository by Govc (2024). Here there exist representatives of orientable surfaces with genus $g = 3$ that can be embedded in $\mathbb{R}^3$. For non-orientable cubical surfaces with $k = 1, 2$ we still observe that $FaceInt \geq 3$.

7 Supplements

If the reader wishes to explore the cubical surfaces here presented more closely, their **3-d models and animation sequences** can be consulted and downloaded from Sketchfab Estevez (2025a).

If the reader wishes to minimize a particular cubical surface or take a deeper look into the implementations visit the following GitHub repository Estevez (2025b).

Some **3-d prints** of 5-dimensional cubical surfaces can be consulted in Estévez et al. (2024).

Acknowledgments

In loving memory of Prof. Dr. Sayan Mukherjee. This work was supported by Prof. Dr. Sayan Mukherjee's Humboldt Research Fellowship, awarded by the Alexander von Humboldt Foundation in 2023.

Thanks to Nikola Milosevic for the fruitful conversations and advices on this project and in general in RL.

Thanks to the Blender Team for such a robust 3-d modeling and animation software.

References

Aveni, Govc, and Roldan. Cubical surfaces (in preparation). *arxiv*, 1:1–10, 2024.

Paul Bourke. The shortest line between two lines in 3d. *Disponível na internet via WWW. URL: http://local. wasp. uwa. edu. au/˜ pbourke/geometry/lineline3d*, 1998.

Davide P. Cervone. Traditional research, 2001. URL `https://www.math.union.edu/~dpvc/professional/research/traditional.html`.

Manuel Estevez. Cubical surface sketchfab. https://sketchfab.com/Mane.Estevez/collections/5-d-cubical-surfaces-6e9512c3597d4a85aca657dd8a33e917, 2025a. Accesed: 2025-7-9.

Manuel Estevez. Cubical surface immersions rl. https://github.com/mestevez88/Cubical-Surface-Immersions-RL, 2025b. Accesed: 2025-7-9.

Manuel Estévez, Érika Roldán, and Henry Segerman. Surfaces in the tesseract. In Judy Holdener, Eve Torrence, Chamberlain Fong, and Katherine Seaton (eds.), *Proceedings of Bridges 2023: Mathematics, Art, Music, Architecture, Culture*, pp. 441–444, Phoenix, Arizona, 2023. Tessellations Publishing. ISBN 978-1-938664-45-8. URL `http://archive.bridgesmathart.org/2023/bridges2023-441.html`.

Manuel Estévez, Érika Roldán, and Henry Segerman. Oriented and non-oriented cubical surfaces in the penteract. In Helena Verrill, Karl Kattchee, S. Louise Gould, and Eve Torrence (eds.), *Proceedings of*

Bridges 2024: Mathematics, Art, Music, Architecture, Culture, pp. 381–384, Phoenix, Arizona, 2024. Tessellations Publishing. ISBN 978-1-938664-49-6. URL `http://archive.bridgesmathart.org/2024/bridges2024-381.html`.

Dejan Govc. 6-dimensional cubical surfaces. https://github.com/DejanGovc/Surfaces6Cube, 2024. Accesed: 2025-6-11.

Tomasz Kaczynski, Konstantin Mischaikow, and Marian Mrozek. Computational homology. *Appl Math Sci*, 157:482, 1 2004. doi: 10.1007/b97315.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL `https://arxiv.org/abs/1412.6980`.

Tomas Möller. A fast triangle-triangle intersection test. *Journal of graphics tools*, 2(2):25–30, 1997.

NeonRice. 3d-triangle-intersection-detection. https://github.com/NeonRice/3D-triangle-intersection-detection, 2020. Accessed: 2024-10-15.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL `http://jmlr.org/papers/v22/20-1364.html`.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Omar Shehata. How to fix gimbal lock in n-dimensions. https://omar-shehata.medium.com/how-to-fix-gimbal-lock-in-n-dimensions-f2f7baec2b5e, 2020. Accessed: 2024-10-30.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction.* MIT press Cambridge, 2018.

A Appendix

A.1 High-dimensional rotations

Algorithm 1: $RotMat(\phi_n)$

```
Data: Dimension $n \in \mathbb{N}$ and angle list $\phi_n := (\phi_{i,j} : (i,j) \in \binom{n}{2})$.
Result: General rotation matrix $\boldsymbol{R} \in SO(n, \mathbb{R})$.
$Comb_2(n) \leftarrow \binom{n}{2}$;
$\boldsymbol{R} \leftarrow \boldsymbol{I}_n$;
for $(i,j) \in Comb_2(n)$ do
    $\boldsymbol{S} \leftarrow \boldsymbol{I}_n$;
    $\boldsymbol{S}_{i,i} \leftarrow \cos(\phi_{i,j})$;
    $\boldsymbol{S}_{j,j} \leftarrow \cos(\phi_{i,j})$;
    $\boldsymbol{S}_{i,j} \leftarrow -\sin(\phi_{i,j})$;
    $\boldsymbol{S}_{j,i} \leftarrow \sin(\phi_{i,j})$;
    $\boldsymbol{R} \leftarrow \boldsymbol{S} \cdot \boldsymbol{R}$
end
```

A.2 Perspective Projection

Algorithm 2: $Proj_n(\boldsymbol{a}, \boldsymbol{c}, \boldsymbol{p})$

```
Data: Point to project $\boldsymbol{a} \in \mathbb{R}^n$, Camera position $\boldsymbol{c} \in \mathbb{R}^n$, Projection Plane position $\boldsymbol{p} \in \mathbb{R}^n$.
Result: Projected point $\boldsymbol{b} \in \mathbb{R}^{n-1}$.
$\boldsymbol{d} \leftarrow \boldsymbol{a} - \boldsymbol{c}$;
$\boldsymbol{M} \leftarrow \boldsymbol{I}_n$;
for $i, (0 \le i < n-1)$ do
    $\boldsymbol{M}_{i,n} \leftarrow \boldsymbol{p}[i]/\boldsymbol{p}[n]$;
end
$\boldsymbol{M}_{n,n} \leftarrow 1/\boldsymbol{p}[n]$;
$\boldsymbol{f} \leftarrow \boldsymbol{M} \cdot \boldsymbol{d}$;
$\boldsymbol{f} \leftarrow \boldsymbol{f}/\boldsymbol{f}[n]$;
$\boldsymbol{b} \leftarrow (\boldsymbol{f}[1], ..., \boldsymbol{f}[n-1])$;
```

Algorithm 3: $ProjSegList(n, \boldsymbol{R}, Q^n_k, \boldsymbol{c}_n, \boldsymbol{p}_n, ..., \boldsymbol{c}_4, \boldsymbol{p}_4)$.

```
Data: Point to project $\boldsymbol{a}_i \in \mathbb{R}^i$, $n$-dimensional Rotation Matrix $\boldsymbol{R}$, $n$-dimensional Edges $Q^n_1$, Camera
      position $\boldsymbol{c}_i \in \mathbb{R}^i$, Orthogonal distance from origin to projection plane $p > 0 \in \mathbb{R}$.
Result: $list(tuple : \boldsymbol{b} \in \mathbb{R}^3)$
$SegList \leftarrow list()$;
for $e \in Q^n_1$ do
    $VtxList \leftarrow list()$;
    for $v \in e$ do
        $\boldsymbol{a} \leftarrow \boldsymbol{R} \cdot v$;
        $N \leftarrow n$;
        while $N > 3$ do
            $\boldsymbol{a} \leftarrow Proj_N(\boldsymbol{a}, \boldsymbol{c}_N, \boldsymbol{p}_N)$;
            $N \leftarrow N - 1$;
        end
        $VtxList.append(\boldsymbol{a})$;
    end
    $SegList.append(VtxList)$;
end
```

A.3 Termination Criteria

```
Algorithm 4: $Done(FaceInt, Exp, Overlap, Counter, Exact)$
Data: $(int, int, int, int, bool); (FaceInt, Exp, Overlap, Counter, Exact)$
Result: $bool : Done$
Done ← False;
if Exact;                                                    // (1)
 then
    if $Overlap = 0$ and $FaceInt = Exp$ then
        $Done \leftarrow True$;
    end
else
    if $Overlap = 0$ and $FaceInt \leq Exp$ then
        $Done \leftarrow True$;
    end
end
if $Counter = MaxSteps$;                                     // (2)
 then
    $Done \leftarrow True$;
end
```

A.4 Reward Functions

```
Algorithm 5: $r_1(PrevAction, Action)$
Data: $(int, int) : (PrevAction, Action)$
Result: $r_1 \in \mathbb{R}$
$r_1 \leftarrow 0$;
if $Counter > 1$ then
    if $ActVec[PrevAction] + ActVec[Action] = 0$ then
        $r_1 \leftarrow r_1 - 1$
    end
end
```

```
Algorithm 6: $r_2(State, Action)$
Data: $(tuple, int) : (State, Action)$
Result: $r_2 \in \mathbb{R}$
$r_2 \leftarrow 0$;
if $State[0] + ActVec[Action][0] \notin ObsSpace[0]$ and $State[1] + ActVec[Action][1] \notin ObsSpace[1]$ then
    $r_2 \leftarrow r_2 - 1$;
end
```

Algorithm 7: $r_3(FaceInt, PrevFaceInt, Exact)$

```
Data: $(int, int, bool) : (FaceInt, PrevFaceInt)$
Result: $r_3 \in \mathbb{R}$
if $Exact$ then
    if $FaceInt = Exp$ then
        $r_3 \leftarrow 0$;
    else
        if $|FaceInt - Exp| < |PrevFaceInt - Exp|$ then
            $r_3 \leftarrow 1$
        end
        if $|FaceInt - Exp| = |PrevFaceInt - Exp|$ then
            $r_3 \leftarrow 0$
        end
        if $|FaceInt - Exp| > |PrevFaceInt - Exp|$ then
            $r_3 \leftarrow -1$
        end
    end
else
    if $FaceInt > Exp$ then
        if $FaceInt < PrevFaceInt$ then
            $r_3 \leftarrow 1$
        end
        if $FaceInt = PrevFaceInt$ then
            $r_3 \leftarrow 0$
        end
        if $FaceInt > PrevFaceInt$ then
            $r_3 \leftarrow -1$
        end
    else
        $r_3 \leftarrow 0$;
    end
end
```

Algorithm 8: $Reward_4(FaceInt, PrevFaceInt, Overlap, PrevOverlap, Exact)$

```
Data: $(int, int, int, int, bool) : (FaceInt, PrevFaceInt, Overlap, PrevOverlap, Exact)$
Result: $r_4 \in \mathbb{R}$
if $Exact$ then
    if $FaceInt = Exp$ then
        if $Overlap = 0$ or $PrevOverlap = 0$ then
            $r_4 \leftarrow 10;$
        else
            if $Overlap < PrevOverlap$ then
                $r_4 \leftarrow 1$
            end
            if $Overlap = PrevOverlap$ then
                $r_4 \leftarrow 0$
            end
            if $Overlap < PrevOverlap$ then
                $r_4 \leftarrow -1$
            end
        end
    end
else
    if $FaceInt \leq Exp$ then
        if $Overlap = 0$ or $PrevOverlap = 0$ then
            $Reward_4 \leftarrow 10;$
        else
            if $Overlap < PrevOverlap$ then
                $r_4 \leftarrow 1$
            end
            if $Overlap = PrevOverlap$ then
                $r_4 \leftarrow 0$
            end
            if $Overlap > PrevOverlap$ then
                $r_4 \leftarrow -1$
            end
        end
    end
end
```

A.5 Environment Logic

Algorithm 9: Hypercube Environment

Data: Camera distances $d_5, d_4 \in [-15, -2]$, $\phi_i \in [0, \pi/2]$; Step size $\delta, \epsilon > 0 \in \mathbb{R}$; $Exp \geq 0 \in \mathbb{Z}$; $Exact = bool$; $w > 0 \in \mathbb{R}$; Cubical surface $\mathcal{C} = list(tuple)$.

```
class Hypercube5{
    Constructor($d_5, d_4, \phi_1, ..., \phi_{10}, \delta, \epsilon, Exp, Exact, \mathcal{C}_2, w$)
        $Q_1^5 \leftarrow EdgeCoordinates(Q_1^5)$;
        $\boldsymbol{f} \leftarrow FaceCoordinates(\mathcal{C})$;
        $ActVec \leftarrow \{0 : \delta\boldsymbol{e}^{(1)}, 1 : -\delta\boldsymbol{e}^{(1)}, 2 : \delta\boldsymbol{e}^{(2)}, 3 : -\delta\boldsymbol{e}^{(2)}, 4 : \epsilon\boldsymbol{e}^{(3)}, 5 : -\epsilon\boldsymbol{e}^{(3)}, ..., 23 : -\epsilon\boldsymbol{e}^{(12)}\}$;
        $Action \leftarrow (0, ..., 17)$;
        $ObsSpace \leftarrow ([-15, -2], [-15, -2], [-1, 1], ..., [-1, 1])$;
        $\boldsymbol{R} \leftarrow RotMat(\phi_1, ..., \phi_{10})$;                          // Algorithm 1
        $State \leftarrow (d_5, d_4, \boldsymbol{R}_{1,1}, \boldsymbol{R}_{1,2}, \cdots, \boldsymbol{R}_{5,4}, \boldsymbol{R}_{5,5})$;
        $\boldsymbol{c}_5 \leftarrow State[0]\boldsymbol{e}^{(5)}$;                            // Camera positions.
        $\boldsymbol{c}_4 \leftarrow State[1]\boldsymbol{e}^{(4)}$;
        $\boldsymbol{p}_5 \leftarrow -State[0]\boldsymbol{e}^{(5)} + \boldsymbol{e}^{(5)}$ ;       // Hyperplane positions relative to camera.
        $\boldsymbol{p}_4 \leftarrow -State[1]\boldsymbol{e}^{(4)} + 10\boldsymbol{e}^{(4)}$;
        $SegList \leftarrow ProjSegList(5, \boldsymbol{R}, Q_1^5, \boldsymbol{c}_5, \boldsymbol{p}_5, \boldsymbol{c}_4, \boldsymbol{p}_4)$;      // Algorithm 3
        $FaceList \leftarrow ProjSegList(5, \boldsymbol{R}, \boldsymbol{f}, \boldsymbol{c}_5, \boldsymbol{p}_5, \boldsymbol{c}_4, \boldsymbol{p}_4)$;     // Algorithm 3
        $Overlaps \leftarrow EdgeIntersections(w, SegList)$;                // Algorithm 13
        $PrevOverlaps \leftarrow Overlaps$;
        $FaceInt \leftarrow FaceInt(FaceList)$;                              // Algorithm 14
        $PrevFaceInt \leftarrow FaceInt$;
        $MinFaceInt \leftarrow PrevFaceInt$;
```

Algorithm 10: Step function

```
...
  $Step(Action)$ :
    $Counter \leftarrow Counter + 1$;
    $PrevOverlap \leftarrow Overlap$;
    $PrevFaceInt \leftarrow FaceInt$;
    $R_1 \leftarrow Reward_1(PrevAction, Action)$;                          // Algorithm 5
    $R_2 \leftarrow Reward_1(State, Action)$;                               // Algorithm 6
    if $Action \in \{0,1\}$ and $\boldsymbol{c}_5[4] + ActVec[Action][0] \in [-15,-2]$ then
      $\boldsymbol{c}_5 \leftarrow \boldsymbol{c}_5 + ActVec[Action][0]\boldsymbol{e}^{(5)}$;        // 5-dimensional camera.
    else if $Action \in \{2,3\}$ and $\boldsymbol{c}_4[3] + ActVec[Action][1] \in [-15,-2]$ then
      $\boldsymbol{c}_4 \leftarrow \boldsymbol{c}_4 + ActVec[Action][1]\boldsymbol{e}^{(4)}$;        // 4-dimensional camera.
    else
      $\boldsymbol{S} \leftarrow RotMat(ActDir[Action][2,:])$;                  // Algorithm 1
      $\boldsymbol{R} \leftarrow \boldsymbol{S} \cdot \boldsymbol{R}$;
    $State \leftarrow (\boldsymbol{c}_5[4], \boldsymbol{c}_4[3], \boldsymbol{R}_{1,1}, \boldsymbol{R}_{1,2}, \cdots, \boldsymbol{R}_{5,4}, \boldsymbol{R}_{5,5})$;
    $\boldsymbol{p}_5 \leftarrow -State[0]\boldsymbol{e}^{(5)} + \boldsymbol{e}^{(5)}$ ;       // Hyperplane positions relative to camera.
    $\boldsymbol{p}_4 \leftarrow -State[1]\boldsymbol{e}^{(4)} + 10\boldsymbol{e}^{(4)}$;
    $SegList \leftarrow ProjSegList(5, \boldsymbol{R}, Q_1^5, \boldsymbol{c}_5, \boldsymbol{p}_5, \boldsymbol{c}_4, \boldsymbol{p}_4)$;        // Algorithm 3
    $FaceList \leftarrow ProjSegList(5, \boldsymbol{R}, \boldsymbol{f}, \boldsymbol{c}_5, \boldsymbol{p}_5, \boldsymbol{c}_4, \boldsymbol{p}_4)$;        // Algorithm 3
    $Overlaps \leftarrow EdgeIntersections(w, SegList)$;                    // Algorithm 13
    $FaceInt \leftarrow FaceInt(FaceList)$;                                 // Algorithm 14
    $MinFaceInt \leftarrow min(MinFaceInt, FaceInt)$;
    $R_3 \leftarrow Reward_3(FaceInt, PrevFaceInt)$;                         // Algorithm 7
    $R_4 \leftarrow Reward_4(FaceInt, PrevFaceInt, Overlap, PrevOverlap)$;    // Algorithm 8
    $Reward \leftarrow R_1 + R_2 + R_3 + R_4$;
    $Done \leftarrow Done(FaceInt, Exp, Overlap, Counter, Exact)$;          // Algorithm 4
```

Algorithm 11: Reset function

```
...
  $Reset(Done)$ :
    $\boldsymbol{R} \leftarrow RotMat(\phi_1, ..., \phi_{10})$;                          // Algorithm 1
    $State \leftarrow (d_5, d_4, \boldsymbol{R}_{1,1}, \boldsymbol{R}_{1,2}, \cdots, \boldsymbol{R}_{5,4}, \boldsymbol{R}_{5,5})$;
    $\boldsymbol{c}_5 \leftarrow State[0]\boldsymbol{e}^{(5)}$;                              // Camera positions.
    $\boldsymbol{c}_4 \leftarrow State[1]\boldsymbol{e}^{(4)}$;
    $\boldsymbol{p}_5 \leftarrow -State[0]\boldsymbol{e}^{(5)} + \boldsymbol{e}^{(5)}$ ;       // Hyperplane positions relative to camera.
    $\boldsymbol{p}_4 \leftarrow -State[1]\boldsymbol{e}^{(4)} + 10\boldsymbol{e}^{(4)}$;
    $SegList \leftarrow ProjSegList(5, \boldsymbol{R}, Q_1^5, \boldsymbol{c}_5, \boldsymbol{p}_5, \boldsymbol{c}_4, \boldsymbol{p}_4)$;        // Algorithm 3
    $FaceList \leftarrow ProjSegList(5, \boldsymbol{R}, \boldsymbol{f}, \boldsymbol{c}_5, \boldsymbol{p}_5, \boldsymbol{c}_4, \boldsymbol{p}_4)$;        // Algorithm 3
    $Overlaps \leftarrow EdgeIntersections(w, SegList)$;                    // Algorithm 13
    $PrevOverlaps \leftarrow Overlaps$;
    $FaceInt \leftarrow FaceInt(FaceList)$;                                 // Algorithm 14
    $PrevFaceInt \leftarrow FaceInt$;
    $MinFaceInt \leftarrow PrevFaceInt$;
```

A.6 Edge Collision Detection

Here, the algorithm by Bourke (1998) to compute the shortest line between two line segments in $\mathbb{R}^3$ is presented. Two lines $L_{1,2} \subset \mathbb{R}^3$ and $L_{3,4} \subset \mathbb{R}^3$ passing through points $P_1, P_2 \in \mathbb{R}^3$ and $P_3, P_4 \in \mathbb{R}^3$ respectively generally do not intersect. If $L_{1,2}$ and $L_{3,4}$ are not parallel and they are co-planar, then they must intersect. However, if they are not co-planar, they can be connected by a unique shortest line segment $L_{a,b} \subset \mathbb{R}^3$ perpendicular to both lines with $P_a \in L_{1,2}$ and $P_b \in L_{3,4}$. The algorithm calculates the points P_a and P_b defining $L_{a,b}$, and determines whether the point P_a (resp. P_b) lies between the points P_1 and P_2 (resp. P_3 and P_4) or not. First note that any point $P \in L_{1,2}$ (resp. $P' \in L_{3,4}$) between P_1 and P_2 (resp. P_3, P_4) is of the form

$$P = P_1 + m_a(P_2 - P_1),$$

(resp. $P' = P_3 + m_b(P_4 - P_3)$) for some real number $0 \leq m_a \leq 1$ (resp. $0 \leq m_b \leq 1$). Since the shortest line segment $L_{a,b}$ between two lines $L_{1,2}$ and $L_{3,4}$ is perpendicular to both of them, the dot product must satisfy $(P_b - P_a) \cdot (P_2 - P_1) = 0$ (resp. $(P_b - P_a) \cdot (P_4 - P_3) = 0$). By taking $P_i - P_j = (x_i - x_j, y_i - y_j, z_i - z_j)$, and setting

$$d_{ijkl}(P_i, P_j, P_k, P_l) := (P_i - P_j) \cdot (P_k - P_l) = (x_i - x_j)(x_k - x_l) + (y_i - y_j)(y_k - y_l) + (z_i - z_j)(z_k - z_l),$$

expanding the dot product we get

$$\begin{aligned}(P_a - P_b) \cdot (P_2 - P_1) &= (P_1 + m_a(P_2 - P_1) - P_3 - m_b(P_4 - P_3)) \cdot (P_2 - P_1) \\ &= ((P_1 - P_3) + m_a(P_2 - P_1) - m_b(P_4 - P_3)) \cdot (P_2 - P_1) \\ &= d_{1321} + m_a d_{2121} - m_b d_{4321},\end{aligned}$$

therefore we get the equality

$$d_{1321} + m_a d_{2121} - m_b d_{4321} = 0, \tag{8}$$

and similarly for segments $L_{3,4}$ and $L_{a,b}$

$$d_{1343} + m_a d_{4321} - m_b d_{4343} = 0. \tag{9}$$

Solving for m_b in Equation 9 yields $m_b = (d_{1343} + m_a d_{4321})/d_{4343}$, and substituting the expression for m_b in Equation 8 yields $m_a = (d_{1343}d_{4321} - d_{1321}d_{4343})/(d_{2121}d_{4343} - (d_{4321})^2)$. Note that $d_{4343} \neq 0$ if and only if $P_4 \neq P_3$, therefore lets analyze when can $d_{2121}d_{4343} - (d_{4321})^2 = 0$. Let $\boldsymbol{u} = P_2 - P_1$ and $\boldsymbol{v} = P_4 - P_3$, and recall that $\boldsymbol{u} \cdot \boldsymbol{v} = ||\boldsymbol{u}||||\boldsymbol{v}|| \cos(\theta)$, where θ is the angle between $\boldsymbol{u}$ and $\boldsymbol{v}$. Substituting in the denominator on Equation A.6, and since $\boldsymbol{u} \neq 0$ and $\boldsymbol{v} \neq 0$ we get

$$\begin{aligned}d_{2121}d_{4343} - (d_{4321})^2 &= (\boldsymbol{u} \cdot \boldsymbol{u})(\boldsymbol{v} \cdot \boldsymbol{v}) - (\boldsymbol{u} \cdot \boldsymbol{v})^2 \\ &= ||\boldsymbol{u}||^2||\boldsymbol{v}||^2 - (||\boldsymbol{u}||||\boldsymbol{v}|| \cos(\theta))^2 \\ &= ||\boldsymbol{u}||^2||\boldsymbol{v}||^2 - ||\boldsymbol{u}||^2||\boldsymbol{v}||^2 \cos^2(\theta) \\ &= ||\boldsymbol{u}||^2||\boldsymbol{v}||^2(1 - cos^2(\theta)) \\ &= 0,\end{aligned}$$

if and only if $\theta = 0$ or $\theta = \pi$, that is if and only if $\boldsymbol{u}$ and $\boldsymbol{v}$ are parallel. Algorithm 12 determines whether line segments $L_{1,2}$ and $L_{3,4}$ intersect for a given edge-with w.

Algorithm 12: *IntersectionLine*(w, P_1, P_2, P_3, P_4)

```
Data: Edge radius $w > 0 \in \mathbb{R}$ and points $P_1, P_2, P_3, P_4 \in \mathbb{R}^3$ with $(P_2 - P_1), (P_4 - P_3) \neq 0$.
Result: $(bool : Intersects, float : distance, tuple : P_a, tuple : P_b)$
1  $Intersects \leftarrow False$;
2  $\boldsymbol{u} = P_2 - P_1$;
3  $\boldsymbol{v} = P_4 - P_3$;
4  $d_{1343} \leftarrow d_{ijkl}(P_1, P_3, P_4, P_3)$;
5  $d_{4321} \leftarrow d_{ijkl}(P_4, P_3, P_2, P_1)$;
6  $d_{1321} \leftarrow d_{ijkl}(P_1, P_3, P_2, P_1)$;
7  $d_{4343} \leftarrow d_{ijkl}(P_4, P_3, P_4, P_3)$;
8  $b \leftarrow (d_{2121} d_{4343} - d_{4321} d_{4321})$;
9  if $b = 0$ ;                                              // Lines are parallel.
10  then
11     $\boldsymbol{w} = P_3 - P_1$;
12     $\boldsymbol{z} = P_4 - P_1$;
13     $pr_{\boldsymbol{u}}(P_3) = \frac{\boldsymbol{u}}{||\boldsymbol{u}||} \cdot \boldsymbol{w}$;
14     $pr_{\boldsymbol{u}}(P_4) = \frac{\boldsymbol{u}}{||\boldsymbol{u}||} \cdot \boldsymbol{z}$;
15     if $!(||\boldsymbol{u}|| < pr_{\boldsymbol{u}}(P_3) \wedge ||\boldsymbol{u}|| < pr_{\boldsymbol{u}}(P_4) \vee pr_{\boldsymbol{u}}(P_3) < 0 \wedge pr_{\boldsymbol{u}}(P_4) < 0)$ // $L_{3,4}$ projects out of $L_{1,2}$.
16      then
17        $distance = \frac{\boldsymbol{u}}{||\boldsymbol{u}||} \times \boldsymbol{w}$;
18        if $distance < 2w$ then
19           $Intersects \leftarrow True$
20        end
21     end
22 else
23     $m_a \leftarrow (d_{1343} d_{4321} - d_{1321} d_{4343})/b$;
24     $m_b \leftarrow (d_{1343} + m_a d_{4321})/d_{4343}$;
25     $P_a \leftarrow P_1 + m_a \boldsymbol{u}$;
26     $P_b \leftarrow P_3 + m_b \boldsymbol{v}$;
27     if $(0 < m_a < 1)$ and $(0 < m_b < 1)$ then
28        $distance \leftarrow ||P_b - P_a||$;
29        if $distance < 2w$ then
30           $Intersects \leftarrow True$;
31        end
32     end
33 end
```

Algorithm 13: *EdgeIntersections*$(w, SegList)$

```
Data: Edge radius $w > 0 \in \mathbb{R}, list : SegList$.
Result: $int : Overlaps$
1  $EdgeCombinations \leftarrow Combinations(SegList, 2)$;
2  $Overlaps \leftarrow 0$;
3  for Edge1, Edge2 in EdgeCombinations do
4     $Intersection, Distance, P_a, P_b \leftarrow IntersectionLine(w, Edge1[0], Edge1[1], Edge2[0], Edge2[1])$;
5     if Intersection then
6        $Overlaps \leftarrow Overlaps + 1$;
7     end
8  end
```

A.7 Face Collision Detection

Here, the algorithm by Möller (1997) to calculate whether two triangles $T_1, T_2 \subset \mathbb{R}^3$ intersect is presented. If they intersect, it also returns the coordinates of their intersection line or point. Denote the vertices of T_1 and T_2 by V_0^1, V_1^1, V_2^1 and V_0^2, V_1^2, V_2^2 respectively; and the planes they lie in by π_1 and π_2 respectively. Consider vectors $\boldsymbol{u}_2 = (V_1^2 - V_0^2)$ and $\boldsymbol{v}_2 = (V_2^2 - V_0^2)$, then for any point $x \in \pi_2$ the plane equation satisfies

$$\pi_2 : N_2 \cdot x + d_2 = 0, \tag{10}$$

where $N_2 = \boldsymbol{u}_2 \times \boldsymbol{v}_2$ and $d_2 = -N_2 \cdot V_0^2$ the projection distance of the vertex V_0^2 over the vector $-N_2$. The signed (perpendicular) distances from the vertices $V_i^1, i = 0, 1, 2$ of the triangle T_1 to the plane π_2 can be computed by inserting the vertices into the equation 10, yielding

$$d_{V_i^1} = N_2 \cdot V_i^1 + d_2, i = 0, 1, 2.$$

For triangle T_1 and plane π_2, two possible situations can occur:

1. If $d_{V_i^1} \neq 0$ for some $i \in \{0, 1, 2\}$ then the possible sub-cases can occur:
   (a) If all $d_{V_i^1} \neq 0, i \in \{0, 1, 2\}$ have the same sign, then all vertices of T_1 lie on the same side of the plane π_2, so in particular T_1 and $T_2 \subset \pi_2$ don't intersect.
   (b) If any of the $d_{V_i^1} = 0, i \in \{0, 1, 2\}$, or has a different sign with respect to the other $d_{V_j^1}, j \neq i$, then T_1 and the plane π_2 intersect.
2. If all $d_{V_i^1} = 0, i = 0, 1, 2$, then T_1 and T_2 are co-planar.

Suppose both pairs (T_1, π_2) and (T_2, π_1) are on the situation 1(b) described above, then there exist a line $L \subset \mathbb{R}^3$ in the direction of $D := N_1 \times N_2$ such that $L \cap T_1 \neq \emptyset$ and $L \cap T_2 \neq \emptyset$ with equation $L = O + tD$, where O is some point on L and $t \in \mathbb{R}$. Moreover, for triangle T_1 there must be a vertex V_i^1 lying on the other side of π_2 (or in π_2) with respect to the remaining vertices $V_j^1, j \neq i$ (otherwise we would have $T_1 \cap \pi_2 = \emptyset$, which we already discarded). To keep notation simple we suppose this vertex is V_0^1 (resp. V_0^2) for T_1 (resp. for T_2), and we consider the edges $E_{0,1}^1$ and $E_{0,2}^1$ of T_1 (resp. $E_{0,1}^2$ and $E_{0,2}^2$ of T_2). The goal is to compute a scalar parameter value t_1 for $B = E_{0,1}^1 \cap L = O + t_1 D$. First consider the projections of the vertices onto L, that is

$$p_{V_i^1} = D \cdot (V_i^1 - O), i = 0, 1, 2.$$

Let K_i^1 be the projection of V_i^1 onto π_2 and note that the triangles $\Delta V_0^1 B K_0^1$ and $\Delta V_1^1 B K_1^1$ are similar, therefore we get the following equation:

$$t_1 = p_{V_0^1} + (p_{V_1^1} - p_{V_0^1}) \frac{d_{V_0^1}}{d_{V_0^1} - d_{V_1^1}}. \tag{11}$$

Similar calculations are done to compute a scalar parameter t_2 for $E_{0,2}^1 \cap L = O + t_2 D$. Without loss of generality we can suppose $t_1 \leq t_2$ and therefore these two parameters yield a closed interval $[t_1, t_2] \subset \mathbb{R}$ describing the intersection of T_1 with L. By computing the corresponding interval for T_2, the intersection between T_1 and T_2 is computed by the intersection of both intervals.

On the other hand, if both pairs (T_1, π_2) and (T_2, π_1) are on the situation 2, start by projecting the triangles onto the axis where their area is maximized. A 2-dimensional triangle-triangle intersection is performed, that is checking if any edge of T_1 intersects some edge of T_2; if any intersection is found then T_1 and T_2 intersect. Otherwise it only remains to check if T_i is totally contained in T_j by checking if some point of T_i lies inside the triangle T_j; then all the vertices of T_i should lie inside T_j otherwise T_i and T_j should have an edge to edge intersection which we had already discarded. This Algorithm has the following Python implementation NeonRice (2020), which we name here as **TriTriIntersect** and use in Algorithm 14 which counts the number of face-intersections of the projected faces.

Algorithm 14: $FaceInt(FaceList)$

```
Data: list : FaceList.
Result: (int : FaceInt, list : IntersectionLine)
FaceCombinations ← Combinations(FaceList, 2);
FaceInt ← 0;
VtxList ← list();
IntersectionLine ← list();
for Face1, Face2 in FaceCombinations do
    FaceTriangles1 ← Combinations(Face1, 3);          // Combinations of triangles in Face1.
    FaceTriangles2 ← Combinations(Face2, 3);          // Combinations of triangles in Face2.
    TriangleIntersections ← 0;
    Count ← 0;                                        // Avoid co-planar face-intersections.
    for vertex in Face1 do
        if vertex not in Face2 then
            Count ← Count + 1;
        end
    end
    if Count = 3 and vertex not in VtxList then
        VtxList.append(vertex);
    end
    for T1 in FaceTriangles1 do
        for T2 in FaceTriangles2 do
            if Count ≥ 3 then
                Res1, Res2, Intersects ← TriTriIntersect(T1[0], T1[1], T1[2], T2[0], T2[1], T2[2]);
                if Intersects and |Res2[0] − Res2[1]| ≥ 1e − 10 then
                    TriangleIntersections ← TriangleIntersections + 1;
                    IntersectionLine.append([Res2[2], Res2[3]]);
                end
            end
        end
    end
    if TriangleIntersections > 0 then
        FaceInt ← FaceInt + 1;
    end
end
```