# ReLUs on Meshes

*Kerry Zou*


4th Year Project Report
Artificial Intelligence and Computer Science
School of Informatics
University of Edinburgh

2025

# Abstract

In the representation of 3D objects, meshes are commonly used to interpolate attributes like color, texture, or scalar fields at each vertex. However, the linear interpolation of these attributes often fails to adequately capture sharp transitions or abrupt changes between them. This issue is particularly pronounced in complex 3D shapes, where linear interpolation leads to insufficient representation of detailed features, requiring the use of higher-resolution meshes to maintain fidelity. This work presents a novel approach to address these challenges by leveraging activation functions such as ReLU and sharp sigmoid, enabling the mesh to become piecewise linear and effectively capturing sharp transitions without the need for remeshing.

The proposed method works by applying these activation functions to the attributes on the mesh faces, transforming the linear interpolation into piecewise linear behavior. This transformation allows the mesh to better represent non-smooth transitions while maintaining computational efficiency. Additionally, the method facilitates the generation of polycube maps for 3D objects, a crucial technique in computer graphics for creating simplified yet accurate representations of complex 3D structures. Polycube maps have applications in texture mapping, parameterization, and various simulation tasks, making this method highly relevant for a range of industries, including gaming, animation, and engineering.(more on polycube map)

The effectiveness of the approach is validated through extensive experiments on synthetic and real-world meshes, demonstrating improvements in both visual accuracy and numerical precision. By applying this method, the need for high-resolution meshes is significantly reduced, offering a more efficient solution for 3D object representation. The results suggest that this method not only enhances the quality of mesh-based representations but also provides a promising avenue for the generation of polycube maps, enabling more effective geometric processing in 3D modeling and simulation.

# Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

# Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Kerry Zou*)

# Acknowledgements

I would like to express my heartfelt gratitude to my supervisor, Amir Vaxman, for his unwavering support, patient guidance, and constant willingness to share his vast knowledge. From the earliest stages of my research, he has been not only a mentor but also a source of inspiration, offering clear explanations and insightful perspectives whenever I faced challenges. His openness to discussing my ideas, however tentative or exploratory, has encouraged me to think critically and grow more confident in my work. I greatly appreciate his patience in allowing me to learn from my mistakes and the genuine care he has shown for my academic and professional development.

I am also deeply indebted to my family, especially to my late father, whose faith in my abilities and steadfast support laid the foundation for my academic journey. He was a guiding light throughout my life, instilling in me the values of perseverance, curiosity, and hard work. Even though he is no longer here to share in my accomplishments, the memory of his encouragement and belief in my potential continues to motivate me. His unwavering love and guidance have shaped not only my educational pursuits but also the person I am today.

# Table of Contents

# Chapter 1

# Introduction

Three-dimensional surface models in computer graphics often need to be partitioned into simpler components for various tasks in geometry processing. This process, known as *mesh segmentation*, is fundamental to applications ranging from texture mapping and UV atlas creation to physical simulation and shape analysis. By dividing a complex mesh into meaningful or convenient patches, one can simplify subsequent processing: for example, separate patches can be parametrized with less distortion, assigned different material properties, or aligned with canonical shapes for meshing. In general, an effective segmentation enables a mapping from an intricate surface onto simpler domains, facilitating operations that would be difficult on the original, unpartitioned geometry. One particularly important use of mesh segmentation is in the construction of *polycube maps*. A polycube map is a global parameterization of a surface onto a network of axis-aligned cubes (a so-called *polycube* domain). Polycube mappings of triangle meshes are widely useful, including in surface texturing (where a regular grid texture can be mapped onto a complex shape) and in volumetric meshing (enabling all-hexahedral mesh generation in the interior) (1). Intuitively, a polycube map approximates a given shape by a voxel-like structure: each patch of the original surface corresponds to a face of an axis-aligned cube in the parametric domain. The result is a low-distortion parameterization that is aligned with the coordinate axes, which is advantageous for downstream tasks that require structured grids or alignment (for instance, simulations on a hexahedral grid or applying patterns that assume orthogonal directions). Achieving a high-quality polycube map, however, is challenging. The input mesh must be segmented into patches that are each roughly planar and can be aligned with one of the principal axes (X, Y, or Z). This often means identifying where the surface's orientation changes and inserting segmentation boundaries along those transitions. For simple shapes or those with obvious feature breaks, an artist or algorithm could manually define these cuts. But for detailed or organic shapes, determining an optimal segmentation that balances faithfulness to the original geometry with the requirement of axis alignment is non-trivial. The segmentation should neither introduce excessive distortion (which would make the mapping stretch or compress the surface) nor produce too many patches (which complicates the polycube structure). Early approaches to polycube mapping often relied on significant user guidance – for example, manually designing a polycube that roughly envelopes the shape and then

deforming the shape onto it – or on simplistic strategies like aligning with a shape's bounding box, which seldom yields low distortion for complex models. In recent years, more automated methods for polycube map construction have been developed. Notably, **(author?)** (1) present a variational approach to deform a detailed surface into a polycube while minimizing distortion. Their method optimizes an energy functional that includes a sparsity-promoting $\ell_1$ term on face normals, encouraging large regions of the mesh to align exactly with one of the coordinate axes. In essence, the $\ell_1$ regularization on surface normals causes most faces to snap to an axis-aligned orientation, yielding a piecewise-flat, orthogonal shape. At the same time, an as-rigid-as-possible volumetric distortion term is applied to preserve the mesh's original geometry as much as possible during the deformation. By balancing these terms, the algorithm deforms the input mesh into an axis-aligned polycube shape without requiring an initial alignment or predefined feature breaks. The resulting polycube map is of high quality, enabling low-distortion parameterization for applications like hexagonal remeshing and quadrangulation (1). This $\ell_1$-based method demonstrated that automatic polycube construction is feasible even for complex shapes, and it set a benchmark for both segmentation and mapping quality in this domain. While the variational approach of Huang et al. is effective, it involves solving a global optimization with potentially many variables (e.g. vertex positions of the deformed mesh) and relies on iterative solvers to converge. Moreover, the method conceptually operates by deforming the mesh into a polycube, rather than directly computing segmentation boundaries on the original shape. An alternative perspective is to focus explicitly on the segmentation problem: can we determine the patch layout and axis alignment directly, and in a way that is amenable to modern, differentiable optimization techniques? With the growing interest in using neural networks and differentiable representations for geometry, a promising direction is to represent the segmentation implicitly and optimize it using gradient-based methods. This is the direction explored in this project. Our approach draws inspiration from recent advances in implicit neural representations, in particular the idea of *ReLU fields* introduced by **(author?)** (2). ReLU fields demonstrated that a simple injection of non-linearity into a grid-based function representation can dramatically increase its expressive power. In their work, a regular spatial grid (with values defined at grid vertices) is combined with interpolation and a fixed Rectified Linear Unit (ReLU) activation to model complex spatially-varying data. In other words, after trilinear interpolation of the grid values, a ReLU is applied to produce the final output. This surprisingly simple change — essentially allowing a single-layer piecewise-linear transformation — was shown to produce high-fidelity implicit representations of images, radiance fields, and occupancy fields at a fraction of the memory and time cost of fully multi-layer perceptrons (2). The key insight is that the ReLU activation partitions the space into linear regions separated by hyperplane boundaries (where the ReLU "cuts off" the linear interpolation). With a coarse-to-fine optimization strategy, these piecewise-linear fields can capture detailed variations and sharp transitions in the data, effectively matching the quality of deeper networks while remaining computationally efficient. The piecewise-linear nature of ReLU-activated fields is particularly attractive for the mesh segmentation problem. A ReLU inherently creates a linear decision boundary in the field it is applied to: where the underlying interpolated value changes sign (from negative to positive at the ReLU threshold of 0), a flat interface is formed. If we interpret

one region as where the ReLU output is active (positive) and another where it is zero (inactive), the boundary between segments is a plane (or, in 3D surfaces, a piece of a plane intersecting the mesh). This means an implicit function using ReLU can naturally represent a segmentation patch boundary as a planar cut. By extension, a combination of several ReLU-based fields could represent multiple segmentation patches meeting along straight edges. In the context of polycube mapping, we desire exactly that: segmentation boundaries that are (approximately) planar and aligned with coordinate axes. By designing our implicit field representation carefully, we can encourage these automatically learned boundaries to align with the global axes. For example, we can set up multiple fields to represent distance along the X, Y, and Z directions and use their ReLU-defined transitions to carve the model into axis-aligned slabs. More generally, the parameters of the underlying linear functions (which the ReLUs operate on) can be optimized so that the normal of each segmentation plane is oriented along one of the principal axes. A direct use of ReLU functions for segmentation comes with a caveat: the ReLU function is not smooth at the threshold (its gradient is zero on one side and undefined exactly at the cutoff). In an optimization setting, especially one that relies on gradient descent, this lack of differentiability can impede progress or lead to instability. To address this, we incorporate *sharp sigmoid activations* as a differentiable proxy for the hard segmentation decision. In practice, this means that instead of an abrupt step change, we use a sigmoid function (or an equivalent smooth step) with a high slope that approximates a step function. During the early stages of optimization, the sigmoid is relatively gentle, so that the boundary between segments is fuzzy or soft – the model can smoothly adjust where the "cut" should be, and gradients flow across what will eventually become the dividing surface. As the optimization proceeds, we gradually increase the sharpness of the sigmoid (e.g. by increasing its gain or a temperature parameter), making the transition steeper. By the end of the process, the sigmoid is so sharp that it behaves almost like a binary step, yielding a crisp, piecewise-linear segmentation. This strategy is a form of continuation or annealing, allowing the optimizer to avoid poor local minima that a fixed hard segmentation might cause. The use of a sharp sigmoid ensures that our segmentation field remains fully differentiable throughout the optimization, which means we can seamlessly integrate various loss terms (for alignment, smoothness, distortion, etc.) and compute their gradients with respect to the segmentation parameters. Using activation-based implicit fields for mesh segmentation offers several potential advantages over traditional methods. First, it provides a unified, differentiable framework: the entire segmentation process can be cast as optimizing a set of continuous parameters (for example, the coefficients of linear functions or neural network weights) to minimize an objective. This is in contrast to many classical segmentation approaches that involve discrete decisions (e.g. assigning faces to clusters or cutting along edges) which can require heuristic tuning or combinatorial search. In our framework, any criterion we desire — alignment of patch normals to axes, contiguity of patches, low mapping distortion — can be added to the objective function and optimized jointly. Second, the implicit field representation is agnostic to the resolution of the mesh. Since the segmentation is defined in a continuous space (via an interpolated grid or neural network function), it can, in principle, handle shapes of different resolutions and even continuous deformations of a shape without needing to re-tune discrete parameters. This makes the approach flexible and general.

Third, the piecewise-linear nature of the ReLU field means that planar segmentation boundaries emerge naturally from the form of the representation, rather than having to explicitly enforce planarity. Traditional algorithms might need to detect or enforce that a cluster of faces lies in a plane or spend effort aligning a cut to be straight; in our case, the default behavior of the model is to produce linear boundaries due to the ReLU structure. Finally, by leveraging techniques from deep learning and automatic differentiation, our approach can potentially be accelerated on modern hardware (GPU/parallel processing) and can be integrated with learning-based pipelines. For instance, one could imagine training a network across a dataset of shapes to predict an initial segmentation field, which could then be fine-tuned with our differentiable method — something that would be much harder to accomplish with purely discrete segmentation methods. In summary, this dissertation investigates a novel approach to polycube mesh segmentation driven by activation functions. We propose to adapt the concept of ReLU fields to generate axis-aligned, piecewise-linear segmentation boundaries on 3D meshes, suitable for polycube map construction. By using sharp sigmoid activations, we maintain differentiability, enabling the use of gradient-based optimization to refine the segmentation. Our method aims to directly produce a partition of the input mesh into patches aligned with the principal axes, thus facilitating the subsequent creation of a polycube map without requiring heavy user intervention or complex global solvers. We will demonstrate how this activation-based segmentation can achieve results comparable to classical methods (such as the $\ell_1$-normal-based optimization of **(author?)** 1), while offering a more flexible and integrable framework. The remainder of this report is organized

as follows: Chapter 2 provides background on mesh segmentation and implicit field representations, including more details on polycube mapping and the ReLU fields technique. Chapter 3 describes the methodology of our approach in detail, from the formulation of the activation-based segmentation field to the design of the objective function and the optimization strategy. In Chapter 4, we present the implementation results, showcasing examples of segmented meshes and evaluating the quality of the resulting polycube maps against existing approaches. We discuss the advantages and any limitations observed in our method. Finally, Chapter 5 concludes the dissertation with a summary of our contributions and suggestions for future work, highlighting how activation-driven segmentation could be extended or applied to other problems in geometry processing.mmarizing our contributions and suggesting directions for future work.

# Chapter 2

# Single piecewise Linear Mesh in 2D

In order to express the mesh into a piecewise linear function, we first need to express how a single mesh cut be the discontinuity will express the piecewise linear function.

## 2.1 Introduction

In this approach, we aim to create a piecewise linear triangle mesh by using the ReLU activation function on the vertex values of the mesh. The method consists of three main steps: assuming the vertex values have different signs, interpolating the vertex values over the triangle, and applying the Heaviside function to the interpolated values to generate a piecewise linear structure. This ensures that the resulting mesh is linear within each piece while transitioning between 0 and 1.

## 2.2 Methodology

### Step 1: Assumption of Vertex Values

Let us consider a triangle mesh where the triangle has three vertices. Denote these vertices as $V_1, V_2, V_3$, with their associated scalar values $f(V_1), f(V_2), f(V_3)$. We assume that one of these values have different signs versus others, and this condition ensures that the triangle will transition from 0 to 1 across its surface.

### Step 2: Interpolation of Vertex Values

Next, we interpolate the values across the triangle. Given that the triangle has vertices $V_1, V_2, V_3$, we can express any point $P$ inside the triangle in terms of barycentric coordinates $\lambda_1, \lambda_2, \lambda_3$ with respect to the vertices $V_1, V_2, V_3$, such that:

$$\lambda_1 + \lambda_2 + \lambda_3 = 1, \quad 0 \leq \lambda_i \leq 1, \quad i = 1, 2, 3.$$

Let the interpolated value at a point $P$ be denoted by $f(P)$, which is the weighted sum of the vertex values:

$$f(P) = \lambda_1 f(V_1) + \lambda_2 f(V_2) + \lambda_3 f(V_3).$$

### Step 3: Application of the Activation Function

Finally, to create the piecewise linear mesh, we apply the activation function to the interpolated values $f(P)$. We can choose ReLU function $ReLU(x)$ or Heaviside Step function $H(x)$ which are:

$$ReLU(x) = \begin{cases} 0 & \text{if } x < 0, \\ x & \text{if } x \geq 0. \end{cases}$$

$$H(x) = \begin{cases} 0 & \text{if } x < 0, \\ 1 & \text{if } x \geq 0. \end{cases}$$

Thus, for any point $P$ inside the triangle, the final value after applying the activation function (use Heaviside step function as an example) is:

$$g(P) = H(f(P)) = \begin{cases} 0 & \text{if } f(P) < 0, \\ 1 & \text{if } f(P) \geq 0. \end{cases}$$

This ensures that the triangle is partitioned into two regions, with one region having values of 0 and the other having values of 1. Since the interpolation between vertices is linear, the resulting mesh is piecewise linear, with a sharp transition between the two regions at the boundary of the zero set.

## 2.3 Implementation

To test our approach, we applied the method to a triangle mesh, assigning values of 1, -1, and 0.5 to the vertices $V_1$, $V_2$, and $V_3$, respectively. We then compared two interpolation techniques: traditional linear interpolation and the activated interpolation using the ReLU and Heaviside functions. The results are shown in the following figure:

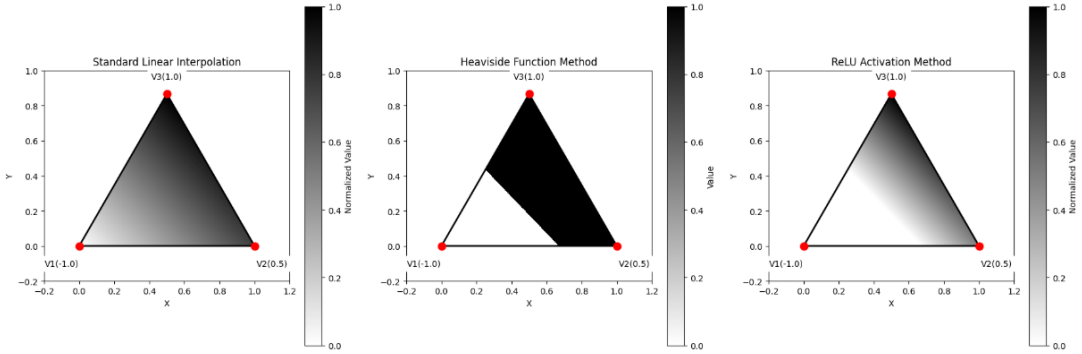



Figure 2.1: Comparison of standard interpolation and activated interpolation with different functions

The figure clearly demonstrates that the standard interpolation produces a smooth, linear variation in vertex values. In contrast, the activated interpolation introduces abrupt transitions between regions, as shown by the application of the Heaviside and ReLU functions. This confirms the effectiveness of our approach in generating sharp, piecewise linear transitions within the mesh.

## 2.4 Conclusion

In this work, we utilized activation functions, inspired by the ReLU field, to create a piecewise linear mesh representation. By applying the ReLU or Heaviside step function to the interpolated vertex values, we introduced sharp transitions in the mesh, which effectively divided the triangle into distinct regions. This approach ensures that the resulting mesh is piecewise linear, with clear boundaries between the regions.

# Chapter 3

# Piecewise Linear Meshes in 2D

## 3.1 The Question

Imagine a mesh with three pinned vertices valued at 1, 0, and -1, arranged so that the 0-valued vertex sits between the 1 and -1 vertices. How can we make a straight bisecting cut that guarantees one half of the mesh always retains the vertex value 1 while the other half always retains the vertex value 0?

## 3.2 The Gradient Alignment

One problem we first need to solve is the straightness of the level set, we can tackle this by introducing the gradient alignment loss.

The goal of the gradient alignment loss is to encourage triangles in the mesh that *cross zero* (i.e., have both negative and positive values in the scalar field $f$) to have similar gradient directions. Specifically, for each triangle, the gradient of $f$ is computed, and if the triangle crosses zero, the gradient directions of the triangles that also cross zero are compared.

The steps involved in computing the gradient alignment loss are as follows:

### 3.2.1 Gradient of each triangle

For each triangle $t$, with vertices $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ and scalar field values $f_1, f_2, f_3$, the gradient of the scalar field $f$ inside the triangle is computed using the following equation:

$$\nabla f \approx \left[ \frac{f_2 - f_1}{\mathbf{p}_2 - \mathbf{p}_1}, \frac{f_3 - f_1}{\mathbf{p}_3 - \mathbf{p}_1} \right].$$

This can be solved using the matrix equation:

$$\mathbf{M} \cdot \nabla f = \mathbf{b},$$

where $\mathbf{M}$ is the matrix formed by the edge vectors $\mathbf{p}_2 - \mathbf{p}_1$ and $\mathbf{p}_3 - \mathbf{p}_1$, and $\mathbf{b}$ is the vector of scalar field differences $[f_2 - f_1, f_3 - f_1]$. Solving this gives the gradient $\nabla f$.

### 3.2.2 Normalizing the gradient

For each triangle $t$, we normalize the gradient $\nabla f$ to obtain a unit vector $\hat{\nabla} f$:

$$\hat{\nabla} f_t = \frac{\nabla f_t}{\|\nabla f_t\|}.$$

This step ensures that we are comparing only the direction of the gradient, not its magnitude.

### 3.2.3 Computing the gradient alignment loss

For each pair of triangles $t_1$ and $t_2$ that both cross zero, we compute the squared difference between their normalized gradients:

$$\text{loss}_{\text{grad}} = \|\hat{\nabla} f_{t_1} - \hat{\nabla} f_{t_2}\|^2.$$

This is done by iterating over all pairs of triangles that cross zero and accumulating the squared differences in their normalized gradient directions.

### 3.2.4 Total gradient alignment loss

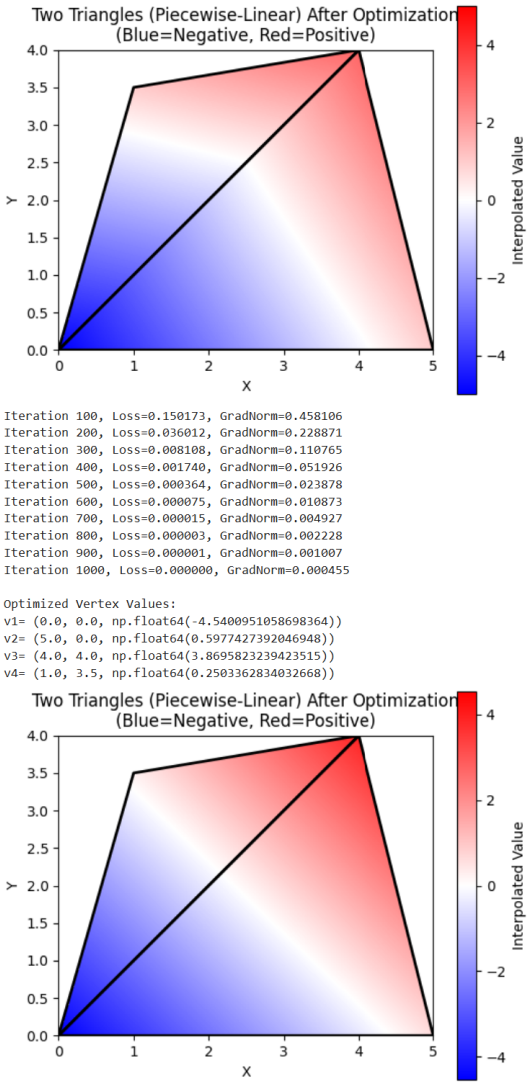
The total gradient alignment loss is the sum of these squared differences for all pairs of triangles that cross zero:

$$L_{\text{grad}} = \sum_{t_1, t_2 \in \text{triangles}} \|\hat{\nabla} f_{t_1} - \hat{\nabla} f_{t_2}\|^2.$$

This loss encourages the gradients of triangles that cross zero to align with each other in terms of direction, thereby promoting smoother transitions across the mesh.

### 3.2.5 Test Implementation

We test our method in meshes made of 2 triangles to prove its correctness:




```
Iteration 100, Loss=0.150173, GradNorm=0.458106
Iteration 200, Loss=0.036012, GradNorm=0.228871
Iteration 300, Loss=0.008108, GradNorm=0.110765
Iteration 400, Loss=0.001740, GradNorm=0.051926
Iteration 500, Loss=0.000364, GradNorm=0.023878
Iteration 600, Loss=0.000075, GradNorm=0.010873
Iteration 700, Loss=0.000015, GradNorm=0.004927
Iteration 800, Loss=0.000003, GradNorm=0.002228
Iteration 900, Loss=0.000001, GradNorm=0.001007
Iteration 1000, Loss=0.000000, GradNorm=0.000455

Optimized Vertex Values:
v1= (0.0, 0.0, np.float64(-4.5400951058698364))
v2= (5.0, 0.0, np.float64(0.5977427392046948))
v3= (4.0, 4.0, np.float64(3.8695823239423515))
v4= (1.0, 3.5, np.float64(0.2503362834032668))
```




Figure 3.1: Test and its result

In the two-triangle test, we observe that after applying the gradient alignment loss, the level set becomes significantly straighter where it crosses the triangles. This outcome verifies that the proposed method effectively aligns the scalar field gradients in adjacent triangles, thereby ensuring a smoother transition across the mesh.

## 3.3 Mesh partition Using Activation Function

The mesh partitioning problem traditionally generates linear bisected meshes, but we aim to apply an activation function to create piecewise linear meshes. This approach allows for more flexible and non-linear mesh partitions. To solve this problem, we will design a loss function that leverages machine learning techniques for optimization.

### Problem Setup

We consider a two-dimensional mesh consisting of randomly placed points in the unit square, which are then connected into triangles. We will refer to the entire collection of triangular faces as the *mesh*. Our goal is to learn a scalar function $f$ defined at each vertex of the mesh. In other words, each vertex $i$ has a scalar value $f_i$, and these values are treated as parameters we wish to optimize.

From the set of all vertices, we designate three special ones, labeled $A$, $B$, and $C$. We *pin* their values so that:

$$f(A) = 1, \quad f(B) = -1, \quad f(C) = 0.$$

These assignments effectively impose boundary conditions at those three vertices. Geometrically, $A$ and $B$ are chosen at the extremal leftmost and rightmost parts of the mesh, and $C$ is chosen near the midpoint between them.

## 3.4 Loss Functions

We define three distinct losses that, when combined, shape the behavior of the scalar field $f$:

### 3.4.1 Area Balance Loss

We interpret the function $f$ as splitting the domain into regions where $f \geq 0$ and $f < 0$. A smooth approximation to these regions is obtained by applying a *sigmoid* function to $f$. Concretely, we define:

$$\sigma_\beta(x) = \frac{1}{1 + e^{-\beta x}},$$

where $\beta$ is a parameter controlling the sharpness of the transition. Integrating $\sigma_\beta(f)$ over the entire mesh gives us a notion of how much "area" is covered by the positive side of $f$.

We want this to be roughly half of the total area of the mesh, so we define:

$$\mathcal{L}_{\text{area}} = \Big( \frac{1}{\text{MeshArea}} \int_{\text{mesh}} \sigma_\beta\big(f(\mathbf{x})\big)\, dA \; - \; 0.5 \Big)^2.$$

If the integral is greater than half the area, or less than half, this loss will be non-zero and drive the solution to balance the areas on each side of $f = 0$.

### 3.4.2 Gradient Alignment Loss

We further encourage the level set $f = 0$ to form a coherent curve through the mesh. Specifically, if two adjacent triangles both contain the zero-crossing of $f$, we want their gradients to align rather than abruptly change direction. For each triangle, we compute the gradient of $f$ by viewing $f$ as linearly interpolated across that triangle's vertices.

Adjacent triangles that both cross $f = 0$ will have gradients $\nabla f_1$ and $\nabla f_2$. We normalize them to unit vectors, say $\hat{g}_1$ and $\hat{g}_2$. The alignment loss then penalizes large differences:

$$\mathcal{L}_{\text{grad}} = \sum_{(t_1,t_2)\in\text{adjacent}} \|\hat{g}_1 - \hat{g}_2\|^2 \quad \text{(only summing if both triangles cross } f = 0\text{)}.$$

### 3.4.3 Smoothness Loss

Lastly, for each edge connecting vertices $i$ and $j$, we penalize squared differences $(f_i - f_j)^2$. This encourages a smooth field overall:

$$\mathcal{L}_{\text{smooth}} = \sum_{(i,j)\in\text{edges}} (f_i - f_j)^2.$$

## 3.5 Optimization

We combine these three losses into a single objective function

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{area}} + \lambda_{\text{grad}}\,\mathcal{L}_{\text{grad}} + \lambda_{\text{smooth}}\,\mathcal{L}_{\text{smooth}},$$

where $\lambda_{\text{grad}}$ and $\lambda_{\text{smooth}}$ are user-chosen coefficients that control the relative importance of gradient alignment and smoothness. A stochastic gradient-based optimizer (like Adam) is used to iterate over all trainable parameters $\{f_i\}$. At each iteration, we:

1. Compute the individual losses and their weighted sum, $\mathcal{L}_{\text{total}}$.
2. Perform a backward pass to calculate gradients $\nabla f_i$.
3. Update each $f_i$ in the direction that reduces $\mathcal{L}_{\text{total}}$.
4. Re-impose boundary conditions $f(A) = 1$, $f(B) = -1$, $f(C) = 0$ to keep them strictly pinned.

Over many iterations, the function $f$ stabilizes into a configuration that:

- Splits the mesh so the sigmoid-applied values cover about half the area at or above 0,
- Has smoothly varying values across the vertices,
- And has consistent gradient directions along the zero-crossing.

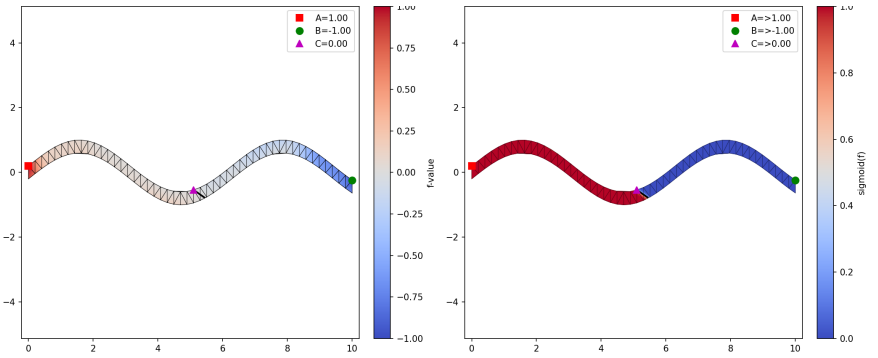



Figure 3.3: Experiment on non-convex Mesh

## 3.6 Results and Visualizations

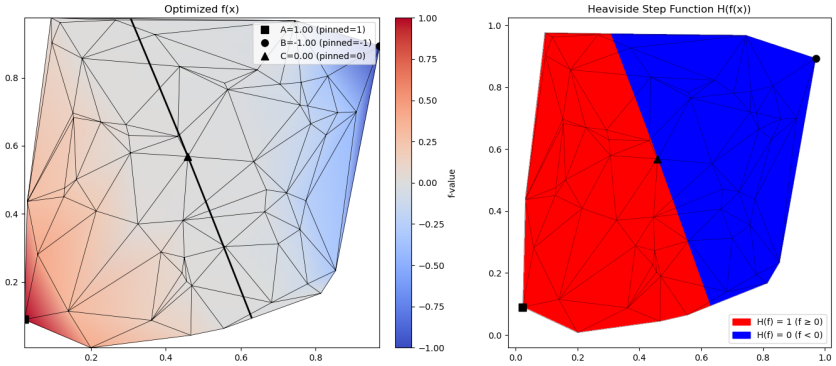



Figure 3.2: The linearly interpolated mesh and activated mesh

The code then visualizes the results in two subplots. The *left* subplot shows the optimized scalar field $f$, interpolated across the triangles. Its color bar indicates the values of $f$, and a black contour highlights $f = 0$. The pinned vertices $A, B, C$ are marked with distinct shapes and carry fixed values $1, -1$, and $0$, respectively.

The *right* subplot applies the Heaviside function $H$ to the learned scalar field. The red region corresponds to $f \geq 0$, and the blue region indicates $f < 0$. An image-based rasterization method checks, for each pixel, which triangle it falls into and then computes the pixel's color by sampling $f$ at that location. The zero-level set becomes a crisp boundary dividing the domain into two subregions. Despite the discrete sampling, the final result remains consistent with the analytically computed $\nabla f$. The same pinned vertices are indicated for reference in the Heaviside image.

These results demonstrate that the learned piecewise linear function effectively splits the mesh into two subdomains of roughly equal area, with smooth transitions and coherent gradient directions where $f = 0$. The direct control over area balance, gradient consistency, and smoothness reveals the flexibility of the method for mesh partitioning via an activation-function-based scalar field.

# Chapter 4

# Piecewise Linear Meshes in 3D

## 4.1 Problem Setup

We are tasked with optimizing a scalar field over a 3D mesh, aiming to partition the surface of the mesh into six distinct regions. Each of these regions should correspond to one of the principal directions in 3D space: top, bottom, front, back, left, and right. These regions are defined based on the positions of certain vertices, which will be pinned to specific coordinates in 3D space. The optimization process must ensure the following:

- The scalar field is used to create six regions that represent the top, bottom, front, back, left, and right sides of the mesh.
- The mesh needs to be clearly partitioned with straight boundaries between adjacent regions.
- The regions should be able to be defined by users, by the properties of area balance, normal similarities and etc.
- The scalar field must be smooth within each region but show sharp transitions at the boundaries between adjacent regions.

## 4.2 New Data Structure and New Activation

### 4.2.1 Multi-channel Field Representation

To address the challenges in mesh segmentation, we extend the traditional scalar field to a vector-valued field $\mathbf{f} : V \to R^C$, where $V$ is the set of mesh vertices and $C = 6$ is the number of channels. Formally, for each vertex $v_i \in V$, we define:

$$\mathbf{f}(v_i) = [f_1(v_i), f_2(v_i), \ldots, f_6(v_i)]^T \tag{4.1}$$

This vector-valued field allows for a richer representation at each vertex and enables the encoding of multiple region affiliations simultaneously. The values in each channel

represent the ”strength” of association between a vertex and a particular region.

### 4.2.2 Region Assignment via Argmax

To determine the region assignment for any point, we apply the argmax operation to the interpolated vector:

$$\text{region}(p) = \underset{c \in \{1,2,\ldots,6\}}{\text{argmax}} \; f_c(p) \tag{4.2}$$

This corresponds to selecting the channel with the highest value, effectively partitioning the mesh into regions where different channels dominate.

While the argmax operation is not differentiable, during optimization we use a differentiable approximation such as softmax with a temperature parameter $\beta$:

$$\text{softmax}_\beta(\mathbf{f}(p))_c = \frac{e^{\beta \cdot f_c(p)}}{\sum_{c'=1}^{6} e^{\beta \cdot f_{c'}(p)}} \tag{4.3}$$

As $\beta \to \infty$, the softmax approaches the one-hot encoding produced by argmax.

### 4.2.3 Boundary Formation and Piecewise Linearity

The boundary between regions $i$ and $j$ occurs at points where $f_i(p) = f_j(p)$ and these values are greater than all other channel values. Since each $f_c$ is linearly interpolated within triangles, the equation $f_i(p) = f_j(p)$ defines a line within each triangle.

Formally, let $\mathbf{h}_{ij}(p) = f_i(p) - f_j(p)$. Then the boundary between regions $i$ and $j$ within a triangle is the zero level set:

$$\mathcal{B}_{ij} = \{p \in \triangle \mid \mathbf{h}_{ij}(p) = 0 \text{ and } i = \underset{c \neq j}{\text{argmax}}\, f_c(p) \text{ and } j = \underset{c \neq i}{\text{argmax}}\, f_c(p)\} \tag{4.4}$$

Since $\mathbf{h}_{ij}$ is linear within each triangle, its zero level set is either empty, a single point, or a line segment, making the boundary piecewise linear across the mesh.

### 4.2.4 Vertex Pinning for Boundary Control

To guide the segmentation process and ensure distinct regions, we pin specific vertices to predefined values. Let $V_p \subset V$ be the set of pinned vertices. For each channel $c \in \{1,2,\ldots,6\}$, we designate a vertex $v_c \in V_p$ that represents the ”center” of region $c$. We constrain:

$$\mathbf{f}(v_c) = [r_1, r_2, \ldots, r_6]^T \quad \text{where} \quad r_i = \begin{cases} 1 & \text{if } i = c \\ -1 & \text{if } i \neq c \end{cases} \tag{4.5}$$

This creates a contrast of 2 between the target channel and all other channels, effectively anchoring each region to a specific location on the mesh.

### 4.2.5 Mathematical Implications for Optimization

The existence of these three triangle types has profound implications for the optimization process. Our contour alignment loss specifically accounts for both edge intersections (relevant for dual-region triangles) and triple intersections (relevant for triple-region triangles). The area balance loss ensures a fair distribution of triangle types, preventing any single region type from dominating the segmentation.

By leveraging this multi-channel field representation with appropriate boundary modeling and vertex constraints, we create a framework that can produce high-quality mesh segmentations with well-defined, geometrically coherent boundaries between regions.

## 4.3 Loss Functions

### 4.3.1 Contour Alignment Loss

The contour alignment loss is designed to create smooth, well-defined boundaries between different regions in mesh segmentation. When segmenting a 3D surface into multiple regions using a multi-channel scalar field, this loss function encourages the boundaries between regions to form geometrically coherent curves.

For a mesh with vertices $V = \{v_1, \ldots, v_N\}$ where $v_i \in R^3$ and faces $F = \{f_1, \ldots, f_T\}$, we define a multi-channel scalar field $\mathbf{f} : V \to R^C$. The boundary between regions corresponding to channels $i$ and $j$ occurs where $f_i(v) - f_j(v) = 0$.

The contour alignment loss consists of several key components:

#### 4.3.1.1 Edge Intersections

For each edge $(v_a, v_b)$ in the mesh and each pair of channels $(i, j)$ where $i < j$, we detect intersection points where $f_i(v) - f_j(v) = 0$.

Let $d_{ij}^a = f_i(v_a) - f_j(v_a)$ and $d_{ij}^b = f_i(v_b) - f_j(v_b)$. An intersection is detected when $d_{ij}^a \cdot d_{ij}^b < 0$. Since a hard threshold isn't differentiable, we use a logistic function to soften the detection:

$$w_{ij} = \sigma(-\beta \cdot d_{ij}^a \cdot d_{ij}^b) \tag{4.6}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function and $\beta > 0$ controls the sharpness.

The intersection point is computed through linear interpolation:

$$\alpha = \frac{|d_{ij}^a|}{|d_{ij}^a| + |d_{ij}^b| + \varepsilon} \tag{4.7}$$

$$p_{ij} = v_a + \alpha(v_b - v_a) \tag{4.8}$$

#### 4.3.1.2 Triple Intersections

For triangle faces, we also consider points where three different region boundaries intersect. For a triplet of channels $(c_0, c_1, c_2)$ where $c_0 < c_1 < c_2$, we seek points within triangles satisfying:

$$f_{c_0}(v) - f_{c_1}(v) = 0 \tag{4.9}$$
$$f_{c_0}(v) - f_{c_2}(v) = 0 \tag{4.10}$$

Using barycentric coordinates for a triangle with vertices $(v_0, v_1, v_2)$, any point $p$ can be expressed as:

$$p = \alpha v_0 + \beta v_1 + \gamma v_2 \tag{4.11}$$

where $\alpha + \beta + \gamma = 1$ and $\alpha, \beta, \gamma \geq 0$.

The field values are interpolated as:

$$f_c(p) = \alpha f_c(v_0) + \beta f_c(v_1) + \gamma f_c(v_2) \tag{4.12}$$

To find triple intersections, we solve for $(\alpha, \beta, \gamma)$ that satisfy our constraints. After substitution and simplification, we obtain a linear system:

$$A_{xy}\alpha + B_{xy}\beta = X_{xy} \tag{4.13}$$
$$A_{xz}\alpha + B_{xz}\beta = X_{xz} \tag{4.14}$$

where:

$$A_{xy} = [f_{c_0}(v_0) - f_{c_1}(v_0)] - [f_{c_0}(v_2) - f_{c_1}(v_2)] \tag{4.15}$$
$$B_{xy} = [f_{c_0}(v_1) - f_{c_1}(v_1)] - [f_{c_0}(v_2) - f_{c_1}(v_2)] \tag{4.16}$$
$$X_{xy} = -[f_{c_0}(v_2) - f_{c_1}(v_2)] \tag{4.17}$$
$$A_{xz} = [f_{c_0}(v_0) - f_{c_2}(v_0)] - [f_{c_0}(v_2) - f_{c_2}(v_2)] \tag{4.18}$$
$$B_{xz} = [f_{c_0}(v_1) - f_{c_2}(v_1)] - [f_{c_0}(v_2) - f_{c_2}(v_2)] \tag{4.19}$$
$$X_{xz} = -[f_{c_0}(v_2) - f_{c_2}(v_2)] \tag{4.20}$$

The solution using Cramer's rule is:

$$\alpha = \frac{X_{xy}B_{xz} - B_{xy}X_{xz}}{A_{xy}B_{xz} - B_{xy}A_{xz} + \varepsilon} \tag{4.21}$$
$$\beta = \frac{A_{xy}X_{xz} - X_{xy}A_{xz}}{A_{xy}B_{xz} - B_{xy}A_{xz} + \varepsilon} \tag{4.22}$$
$$\gamma = 1 - \alpha - \beta \tag{4.23}$$

The weight for a triple intersection uses softmax probabilities for each vertex:

$$\pi_i(c) = \frac{e^{f_c(v_i)}}{\sum_{c'} e^{f_{c'}(v_i)}} \tag{4.24}$$

Combined with a smoothed step function $S(x) = \sigma(k \cdot x)$ to penalize points outside the triangle:

$$w_{triple} = \pi_0(c_0) \cdot \pi_1(c_1) \cdot \pi_2(c_2) \cdot S(\alpha) \cdot S(\beta) \cdot S(\gamma) \tag{4.25}$$

#### 4.3.1.3 Plane Fitting via Weighted Covariance

For each pair of channels $(i, j)$, we collect all intersection points and fit a plane to represent the ideal smooth boundary. The weighted mean of the points is:

$$\mu_{ij} = \frac{\sum_k w_k p_k}{\sum_k w_k} \tag{4.26}$$

The weighted covariance matrix is:

$$\text{Cov}_{ij} = \frac{\sum_k w_k p_k p_k^T}{\sum_k w_k} - \mu_{ij}\mu_{ij}^T \tag{4.27}$$

Using Singular Value Decomposition (SVD), we find the plane normal $n_{ij}$ as the eigenvector corresponding to the smallest eigenvalue of $\text{Cov}_{ij}$. The plane equation is $n_{ij} \cdot p + d_{ij} = 0$ where $d_{ij} = -n_{ij} \cdot \mu_{ij}$.

#### 4.3.1.4 Final Loss Computation

The contour alignment loss is defined as the weighted mean squared distance from intersection points to their respective planes:

$$L_{\text{contour}} = \sum_{i<j} \frac{\sum_k w_k (n_{ij} \cdot p_k + d_{ij})^2}{\sum_k w_k + \varepsilon} \tag{4.28}$$

This formulation is fully differentiable, allowing for gradient-based optimization. It encourages geometric coherence of boundaries without requiring discrete operations, and properly handles both simple edge intersections and complex triple intersections in 3D mesh segmentation.

### 4.3.2 Area Balance Loss

The area balance loss enforces an even distribution of the mesh surface area across the different regions in the segmentation. Without this constraint, the optimization might result in some regions dominating the mesh while others become vanishingly small.

For a mesh with vertices $V = \{v_1, \ldots, v_N\}$, faces $F = \{f_1, \ldots, f_T\}$, and a multi-channel scalar field $\mathbf{f} : V \to R^C$, the area balance loss uses a softmax-based approach to measure and equalize region sizes.

#### 4.3.2.1 Triangle Area Computation

First, we compute the area of each triangle face in the mesh. For a triangle with vertices $v_0$, $v_1$, and $v_2$, the area is:

$$A_t = \frac{1}{2}\|(v_1 - v_0) \times (v_2 - v_0)\| \tag{4.29}$$

where $\times$ denotes the cross product and $\|\cdot\|$ is the Euclidean norm.

#### 4.3.2.2 Barycentric Sampling for Robust Evaluation

To capture the field distribution across the entire face rather than just at vertices, we sample the field at multiple points within each triangle using barycentric coordinates. We define a set of sample points $\{b_s\}$ in barycentric coordinates:

$$b_1 = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3}) \quad \text{(center)} \tag{4.30}$$

$$b_2 = (\frac{1}{2}, \frac{1}{2}, 0) \quad \text{(edge midpoint 1)} \tag{4.31}$$

$$b_3 = (\frac{1}{2}, 0, \frac{1}{2}) \quad \text{(edge midpoint 2)} \tag{4.32}$$

$$b_4 = (0, \frac{1}{2}, \frac{1}{2}) \quad \text{(edge midpoint 3)} \tag{4.33}$$

Each sample point is assigned equal weight $w_s = \frac{1}{S}$, where $S$ is the number of sample points (typically 4).

#### 4.3.2.3 Field Interpolation and Softmax

For each triangle $t$ and each sample point $s$ with barycentric coordinates $b_s = (\alpha_s, \beta_s, \gamma_s)$, we interpolate the field values:

$$\mathbf{f}_{t,s} = \alpha_s \mathbf{f}(v_{t,0}) + \beta_s \mathbf{f}(v_{t,1}) + \gamma_s \mathbf{f}(v_{t,2}) \tag{4.34}$$

where $v_{t,0}$, $v_{t,1}$, and $v_{t,2}$ are the three vertices of triangle $t$.

We then apply the softmax function with temperature parameter $\beta$ to obtain a probability distribution over channels:

$$\mathbf{p}_{t,s,c} = \frac{e^{\beta \cdot \mathbf{f}_{t,s,c}}}{\sum_{c'=1}^{C} e^{\beta \cdot \mathbf{f}_{t,s,c'}}} \tag{4.35}$$

This converts the vector of field values into a probability distribution, with the temperature parameter $\beta$ controlling the sharpness of the distribution.

#### 4.3.2.4 Area-Weighted Channel Probabilities

For each triangle, we compute the mean probabilities over all sample points:

$$\mathbf{p}_{t,c} = \sum_{s=1}^{S} w_s \cdot \mathbf{p}_{t,s,c} \tag{4.36}$$

We then weight these probabilities by the triangle's area to get the effective area contribution for each channel:

$$A_{t,c} = A_t \cdot \mathbf{p}_{t,c} \tag{4.37}$$

The total area attributed to each channel is the sum over all triangles:

$$A_c = \sum_{t=1}^{T} A_{t,c} \tag{4.38}$$

#### 4.3.2.5 Area Fraction Computation

We compute the fraction of the total mesh area attributed to each channel:

$$f_c = \frac{A_c}{A_{total}} \tag{4.39}$$

where $A_{total} = \sum_{t=1}^{T} A_t$ is the total mesh area.

#### 4.3.2.6 Loss Formulation

The area balance loss measures the deviation from the ideal equal distribution where each channel should receive $\frac{1}{C}$ of the total area:

$$L_{\text{area}} = \sum_{c=1}^{C} \left| f_c - \frac{1}{C} \right| \tag{4.40}$$

This is the L1 norm of the difference between the actual area fractions and the target uniform distribution. Alternative formulations might use the L2 norm or KL-divergence.

#### 4.3.2.7 Interpretations and Properties

This formulation has several important properties:

1. **Continuous and Differentiable**: By using softmax and barycentric sampling, the loss is fully differentiable with respect to the field values.
2. **Robust to Discretization Artifacts**: Sampling at multiple points within each triangle provides a more accurate representation of the field distribution than using only vertex values.

3. **Adaptive to Mesh Geometry**: By weighting probabilities by triangle areas, the method accounts for non-uniform mesh tessellation.
4. **Balanced Segmentation**: The loss directly encourages equal distribution of the mesh area across channels, preventing any one region from dominating.

The area balance loss complements the contour alignment loss by ensuring that while boundaries are sharp and geometrically coherent, no region becomes excessively large or small during optimization.

### 4.3.3 Smoothness Loss

The smoothness loss encourages gradual transitions in the multi-channel scalar field across the mesh surface, preventing excessive oscillations while still allowing for the formation of sharp boundaries where needed. This regularization term balances the contour alignment loss, which focuses on boundary geometry, and complements the area balance loss, which addresses regional proportions.

For a mesh with vertices $V = \{v_1, \ldots, v_N\}$ and a multi-channel scalar field $\mathbf{f} : V \to R^C$, the smoothness loss penalizes large variations in field values between adjacent vertices.

#### 4.3.3.1 Mesh Connectivity Graph

We represent the connectivity of the mesh as an undirected graph $G = (V, E)$, where $V$ is the set of vertices and $E$ is the set of edges. Each edge $e_{ij} \in E$ connects vertices $v_i$ and $v_j$ that share a face in the mesh.

The set of edges $E$ can be derived from the faces $F$ of the mesh by extracting all unique pairs of vertices that form the edges of triangles:

$$E = \{(i, j) \mid \exists f \in F \text{ such that } i \in f \text{ and } j \in f \text{ and } i \neq j\} \tag{4.41}$$

To ensure uniqueness and consistent ordering, we typically represent each edge as an ordered pair $(i, j)$ where $i < j$.

#### 4.3.3.2 Field Gradient Approximation

The smoothness loss implicitly approximates the gradient of the scalar field along mesh edges. For an edge $e_{ij}$ connecting vertices $v_i$ and $v_j$, the discrete gradient of the field for channel $c$ can be approximated as:

$$\nabla \mathbf{f}_c(e_{ij}) \approx \frac{\mathbf{f}_c(v_j) - \mathbf{f}_c(v_i)}{\|v_j - v_i\|} \tag{4.42}$$

For simplicity and computational efficiency, we often omit the normalization by edge length, assuming relatively uniform edge lengths across the mesh:

$$\nabla \mathbf{f}_c(e_{ij}) \approx \mathbf{f}_c(v_j) - \mathbf{f}_c(v_i) \tag{4.43}$$

#### 4.3.3.3 Channel-wise Formulation

For a multi-channel scalar field, we compute the loss separately for each channel and sum the results. The squared magnitude of the gradient approximation for channel $c$ along edge $e_{ij}$ is:

$$\|\nabla \mathbf{f}_c(e_{ij})\|^2 \approx (\mathbf{f}_c(v_j) - \mathbf{f}_c(v_i))^2 \tag{4.44}$$

#### 4.3.3.4 Complete Loss Formulation

The total smoothness loss is the sum of squared differences across all edges and all channels:

$$L_{\text{smooth}} = \sum_{c=1}^{C} \sum_{(i,j)\in E} (\mathbf{f}_c(v_j) - \mathbf{f}_c(v_i))^2 \tag{4.45}$$

This can also be expressed in vector form for computational efficiency:

$$L_{\text{smooth}} = \sum_{(i,j)\in E} \|\mathbf{f}(v_j) - \mathbf{f}(v_i)\|^2 \tag{4.46}$$

where $\|\cdot\|$ denotes the Euclidean norm in $R^C$.

## 4.4 Optimizing the Multi-Channel Field

The optimization process aims to find the optimal values for the multi-channel field that produce well-defined region boundaries while maintaining equal area distribution and appropriate smoothness. This section details the mathematical formulation of the optimization process and the strategies employed to ensure efficient convergence.

### 4.4.1 Optimization Objective

We formulate mesh segmentation as an energy minimization problem over the space of possible field values. Let $\mathbf{F} \in R^{N\times C}$ be the matrix of field values for all vertices, where $N$ is the number of vertices and $C = 6$ is the number of channels. Our objective is to find:

$$\mathbf{F}^* = \underset{\mathbf{F}}{\operatorname{argmin}} L_{\text{total}}(\mathbf{F}) \tag{4.47}$$

where $L_{\text{total}}$ is the weighted sum of our loss components:

$$L_{\text{total}}(\mathbf{F}) = \lambda_{\text{contour}}(t) \cdot L_{\text{contour}}(\mathbf{F}, \beta(t)) + \lambda_{\text{smooth}} \cdot L_{\text{smooth}}(\mathbf{F}) + \lambda_{\text{area}}(t) \cdot L_{\text{area}}(\mathbf{F}, \beta(t)) \tag{4.48}$$

Here, $\lambda_{\text{contour}}(t)$, $\lambda_{\text{smooth}}$, and $\lambda_{\text{area}}(t)$ are weighting coefficients, potentially varying with iteration $t$. The parameter $\beta(t)$ controls the sharpness of the softmax functions used in the loss calculations and may also vary with iteration.

### 4.4.2 Initialization with Pinned Vertices

We initialize the field with small random values and set specific values at pinned vertices:

$$\mathbf{F}_{ij}^{(0)} = \begin{cases} \delta_{ij} & \text{if vertex } i \text{ is pinned to channel } j \\ -\delta_{ij} & \text{if vertex } i \text{ is pinned but not to channel } j \\ \eta_{ij} & \text{otherwise} \end{cases} \tag{4.49}$$

where $\delta_{ij}$ is the Kronecker delta, and $\eta_{ij} \sim \mathcal{N}(0, \sigma^2)$ are small random values drawn from a normal distribution with standard deviation $\sigma = 0.01$. Typically we set $\delta_{ij} = 1$, creating a contrast of 2 between the positive and negative values at pinned vertices.

### 4.4.3 Temporal Parameter Scheduling

To enhance convergence, we employ strategic scheduling of several parameters over the course of optimization:

#### 4.4.3.1 Softmax Temperature $\beta$

The temperature parameter $\beta$ controls the sharpness of the softmax function. We gradually increase it from an initial value $\beta_0$ to a target value $\beta_{\text{target}}$ using a linear schedule:

$$\beta(t) = \beta_0 + (\beta_{\text{target}} - \beta_0) \cdot \min\left(\frac{t}{T_{\max}}, 1\right) \tag{4.50}$$

where $t$ is the current iteration and $T_{\max}$ is the total number of iterations. Typical values are $\beta_0 = 1.0$ and $\beta_{\text{target}} = 40.0$ or higher.

#### 4.4.3.2 Loss Coefficient Ramping

We gradually increase the weights of the contour alignment and area balance losses:

$$\lambda_{\text{contour}}(t) = \lambda_{\text{contour}}^{\text{initial}} + (\lambda_{\text{contour}}^{\text{final}} - \lambda_{\text{contour}}^{\text{initial}}) \cdot \min\left(\frac{t}{T_{\text{ramp}}}, 1\right) \tag{4.51}$$

$$\lambda_{\text{area}}(t) = \lambda_{\text{area}}^{\text{initial}} + (\lambda_{\text{area}}^{\text{final}} - \lambda_{\text{area}}^{\text{initial}}) \cdot \min\left(\frac{t}{T_{\text{ramp}}}, 1\right) \tag{4.52}$$

where $T_{\text{ramp}} = \gamma \cdot T_{\max}$ with $\gamma$ typically set to 0.2. This ramping strategy allows the field to develop smoothly first before enforcing stricter boundary and area constraints.

### 4.4.4 Optimization Algorithm

We employ the AdamW algorithm, a variant of Adam with decoupled weight decay, for gradient-based optimization. Let $\theta_t = \text{vec}(\mathbf{F}_t)$ be the vectorized parameters at iteration $t$. The update rule is:

$$\mathbf{g}_t = \nabla_\theta L(\theta_t) \tag{4.53}$$
$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t \tag{4.54}$$
$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)\mathbf{g}_t^2 \tag{4.55}$$
$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-\beta_1^t} \tag{4.56}$$
$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\beta_2^t} \tag{4.57}$$
$$\theta_{t+1} = \theta_t - \alpha_t \left( \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \varepsilon} + \lambda_{\text{wd}} \theta_t \right) \tag{4.58}$$

where $\beta_1 = 0.9$, $\beta_2 = 0.99$, $\varepsilon = 10^{-8}$, and $\lambda_{\text{wd}} = 10^{-4}$ are the Adam hyperparameters and weight decay coefficient.

### 4.4.5 Learning Rate Scheduling

We utilize the OneCycleLR scheduler, which varies the learning rate over time according to:

$$\alpha_t = \begin{cases} \alpha_{\text{base}} + (\alpha_{\max} - \alpha_{\text{base}}) \cdot \sin\left(\frac{\pi}{2} \cdot \frac{t}{p \cdot T_{\max}}\right) & \text{if } t \le p \cdot T_{\max} \\ \alpha_{\max} + (\alpha_{\text{final}} - \alpha_{\max}) \cdot \sin\left(\frac{\pi}{2} \cdot \frac{t - p \cdot T_{\max}}{(1-p) \cdot T_{\max}}\right) & \text{if } t > p \cdot T_{\max} \end{cases} \tag{4.59}$$

where $p = 0.3$ is the fraction of iterations spent increasing the learning rate, $\alpha_{\text{base}} = \frac{\alpha_{\max}}{d}$ with $d = 25$ is the initial learning rate, $\alpha_{\max} = 2 \times 10^{-3}$ is the maximum learning rate, and $\alpha_{\text{final}} = \frac{\alpha_{\text{base}}}{10^4}$ is the final learning rate.

This cyclical schedule helps escape local minima and saddle points while ensuring fine-grained optimization toward the end.

### 4.4.6 Gradient Handling

To enhance numerical stability, we employ several techniques:

#### 4.4.6.1 Gradient Clipping

To prevent exploding gradients, we clip the L2 norm of the gradient:

$$\mathbf{g}_t \leftarrow \mathbf{g}_t \cdot \min\left(1, \frac{\tau}{\|\mathbf{g}_t\|_2}\right) \tag{4.60}$$

where $\tau = 5.0$ is the clipping threshold.

### 4.4.7 Pinning Constraints

After each optimization step, we explicitly enforce the pinning constraints:

$$\mathbf{F}_{v_c,j} = \begin{cases} 1 & \text{if } j = c \\ -1 & \text{if } j \neq c \end{cases} \tag{4.61}$$

for each pinned vertex $v_c$ associated with channel $c$. This ensures that the region anchors remain fixed throughout optimization.

### 4.4.8 Early Stopping

To avoid unnecessary computation, we implement early stopping based on the lack of improvement in the loss function:

$$\text{stop at iteration } t \text{ if } L_{\text{best}} < L_{\text{total}}(\mathbf{F}_s) \text{ for all } s \in \{t-P+1, t-P+2, \ldots, t\} \tag{4.62}$$

where $L_{\text{best}}$ is the best loss observed so far, and $P$ is the patience parameter (typically set to 2000 iterations).

### 4.4.9 Convergence Properties

The optimization process exhibits several important properties:

1. **Gradual Boundary Formation**: The progressive increase in $\beta$ allows boundaries to form gradually, preventing premature convergence to suboptimal configurations.
2. **Region Stability**: The pinned vertices ensure that each of the six regions remains present throughout optimization.
3. **Balance Between Competing Objectives**: The scheduled loss weights maintain a dynamic balance between smooth field values, clean boundaries, and equal area distribution.
4. **Efficient Exploration-Exploitation**: The cyclical learning rate schedule enables efficient exploration of the parameter space followed by exploitation of promising regions.

### 4.4.10 Post-Processing and Evaluation

After optimization, the final segmentation is determined by applying the argmax operation to the field values:

$$\text{region}(v_i) = \underset{c \in \{1,2,\ldots,6\}}{\text{argmax}} \ \mathbf{F}_{i,c} \tag{4.63}$$

The quality of the segmentation can be evaluated by examining:

1. **Boundary Smoothness**: The geometric coherence of region boundaries
2. **Area Distribution**: The relative areas of the six regions
3. **Field Smoothness**: The gradual variation of field values within regions

These metrics provide a comprehensive assessment of the optimization's success in creating a high-quality mesh segmentation.

## 4.5 Implementation Details

The implementation of our piecewise linear mesh segmentation framework encompasses several interconnected components designed to partition 3D surfaces into distinct regions defined by sharp boundaries. This section details the technical aspects of the implementation, focusing on the data structures, algorithms, and mathematical foundations.

### 4.5.1 Mesh Representation and Preprocessing

For the mesh representation, we implemented both synthetic mesh generation and loading capabilities: For synthetic models, we created an icosphere generator that starts with a regular icosahedron and recursively subdivides its faces, projecting new vertices onto the unit sphere. This approach allows for controlled mesh resolution through a parameter that specifies either the target number of vertices or subdivision level. The subdivision process was implemented using a half-edge-like data structure to efficiently track newly created midpoints during refinement. For real-world models, we implemented functions to load volumetric tetrahedral meshes from VTK/VTU files and extract their boundary surfaces. The implementation ensures proper triangulation of the boundary, maintaining consistent orientation of faces. The mesh connectivity is represented as:

- Vertices: An array of shape $(N, 3)$ containing 3D coordinates
- Faces: An array of shape $(T, 3)$ containing vertex indices for each triangle
- Triangle adjacency: A graph structure that maps each triangle to its neighboring triangles that share an edge
- Vertex edges: A list of vertex pairs that form the edges of the mesh

We also calculated and stored the face areas using the cross product formula:

$$A_t = \frac{1}{2}|(v_1 - v_0) \times (v_2 - v_0)| \tag{4.64}$$

where $v_0$, $v_1$, and $v_2$ are the vertices of triangle $t$.

### 4.5.2 Multi-Channel Field Representation

The core of our implementation is a multi-channel scalar field defined over the mesh vertices. For 3D meshes, we used a 6-channel field $\mathbf{f} : V \to R^6$, represented as a matrix of shape $(N, 6)$ where $N$ is the number of vertices. To establish region anchors, we developed an algorithm to automatically identify six vertices that correspond to the principal directions (top, bottom, front, back, left, right). This algorithm:

Computes the bounding box of the mesh Establishes a coordinate system based on the mesh's principal axes Creates anchor points at fixed distances along these axes For each anchor point, finds the nearest mesh vertex to serve as the "pinned" vertex for that region

For each pinned vertex $v_c$ associated with channel $c$, we constrained the field values to:

$$\mathbf{f}(v_c)_i = \begin{cases} 1 & \text{if } i = c \ -1 \\ \text{if } i \neq c \end{cases} \tag{4.65}$$

We also computed six plane normals based on the positions of the pinned vertices:

$$\mathbf{n}Z = \text{normalize}(\text{postop} - \text{posbottom}) \ \mathbf{n}Y \quad = \text{normalize}(\text{posfront} - \text{posback}) \ \mathbf{n}X = \text{normalize}(\text{posright} - \tag{4.66}$$

The six normals were then defined as $\mathbf{n}_Z$, $-\mathbf{n}_Z$, $\mathbf{n}_Y$, $-\mathbf{n}_Y$, $\mathbf{n}_X$, $-\mathbf{n}_X$, corresponding to the top, bottom, front, back, right, and left directions.

### 4.5.3 Boundary Detection and Characterization

A critical component of our implementation is the detection of region boundaries. Within each triangle, the boundary between regions associated with channels $i$ and $j$ occurs where $f_i(p) - f_j(p) = 0$. For edge boundaries, we implemented an algorithm that:

For each edge $(v_a, v_b)$ in the mesh and each pair of channels $(i, j)$:

Computes differences $d_a = f_i(v_a) - f_j(v_a)$ and $d_b = f_i(v_b) - f_j(v_b)$ Detects a potential intersection when $d_a \cdot d_b < 0$ Uses a sigmoid function to assign a weight to this potential intersection:

$$w = \sigma(-\beta \cdot d_a \cdot d_b) \tag{4.67}$$

Computes the intersection point through linear interpolation:

$$\alpha = \frac{|d_a|}{|d_a| + |d_b| + \varepsilon} \tag{4.68}$$

$$p = v_a + \alpha(v_b - v_a) \tag{4.69}$$

For triple intersections (where three different region boundaries meet), we implemented a more sophisticated algorithm that:

For each triangle with vertices $(v_0, v_1, v_2)$ and each triplet of channels $(c_0, c_1, c_2)$:

Formulates a linear system to find the barycentric coordinates $(\alpha, \beta, \gamma)$ of the point where:

$$f_{c_0}(p) - f_{c_1}(p) = 0 \; f_{c_0}(p) - f_{c_2}(p) \qquad = 0 \tag{4.70}$$

Solves this system using Cramer's rule Assigns a weight based on softmax probabilities and how well the point stays within the triangle

### 4.5.4 Field Interpolation and Argmax

For visualization and evaluation, we implemented field interpolation and classification methods:

Within each triangle, the field value at point $p$ with barycentric coordinates $(\alpha, \beta, \gamma)$ is:

$$\mathbf{f}(p) = \alpha \mathbf{f}(v_0) + \beta \mathbf{f}(v_1) + \gamma \mathbf{f}(v_2) \tag{4.71}$$

Region assignment is determined by the argmax operation:

$$\text{region}(p) = \underset{c \in 1,2,\ldots,6}{\text{argmax}}, f_c(p) \tag{4.72}$$

For differentiable optimization, we use softmax with temperature parameter $\beta$:

$$\mathbf{p}c(p) = \frac{e^{\beta \cdot f_c(p)}}{\sum c' = 1^6 e^{\beta \cdot f_{c'}(p)}} \tag{4.73}$$

### 4.5.5 Data Structures for Efficient Computation

To enable efficient computation, particularly for the contour alignment loss, we implemented several specialized data structures:

Edge-to-midpoint map: A dictionary that maps each mesh edge to its midpoint vertex index during subdivision Triangle adjacency list: A sparse representation of which triangles share edges Channel pair indexing: A system to efficiently map between channel pairs $(i, j)$ and a linear index, enabling vectorized operations Weighted covariance accumulation: Data structures to efficiently gather intersection points and their weights for plane fitting

These structures allowed us to implement fully vectorized operations that avoid Python loops, significantly accelerating computation on modern hardware.

### 4.5.6 Mesh Subdivision for Visualization

For visualization purposes, we implemented a mesh subdivision algorithm that:

Subdivides each triangle in the original mesh into smaller triangles Interpolates the scalar field values at the new vertices Applies argmax to determine the region assignment for each vertex Renders the mesh with different colors for each region

This approach allows for a smoother visualization of region boundaries without changing the underlying optimization.

### 4.5.7 Implementation Variants and Experimentation

Throughout the development process, we implemented and tested several variants of the contour alignment loss:

A channel-wise version that treats each channel separately and fits planes to its boundaries A pairwise version that considers all pairs of channels and fits planes to their intersection points A version that uses learnable plane offsets as parameters A version that focuses on aligning gradient directions across adjacent triangles

These variants were evaluated based on convergence behavior, boundary quality, and computational efficiency. The final implementation uses a hybrid approach that balances these considerations.

### 4.5.8 Sinusoidal Learning Rate Scheduling

A key innovation in our implementation is the use of a sinusoidal, multi-phased learning rate scheduler. This scheduler:

Divides the total optimization process into multiple phases Within each phase, varies the learning rate according to a sine wave Progressively reduces the amplitude of the sine wave with each phase Allows for both exploration (during high learning rate periods) and exploitation (during low learning rate periods)

This approach helps the optimization escape local minima while still achieving fine-grained convergence in the final stages.

### 4.5.9 Mixed Precision Training and Gradient Handling

To enhance numerical stability and performance, we implemented several advanced techniques:

Mixed precision training: Performs forward passes in lower precision (e.g., FP16) while maintaining master weights in FP32 Gradient scaling: Applies a scaling factor to the loss before backpropagation to prevent underflow Gradient clipping: Limits the maximum norm of gradients to prevent exploding gradients Gradient scaler: A custom implementation that handles gradient scaling and unscaling

These techniques were particularly important for complex meshes where the optimization process could otherwise become unstable.

### 4.5.10 Visualization Framework

The implementation includes a comprehensive visualization framework that:

Renders the mesh with colors based on the argmax of the scalar field Highlights the pinned vertices and their associated regions Optionally shows the boundaries between regions Provides both PyVista-based 3D visualization and matplotlib-based fallbacks

For more detailed visualization, we implemented a subdivision approach that applies the argmax operation after interpolating the field values, resulting in smoother region boundaries.

### 4.5.11 Technical Challenges and Solutions

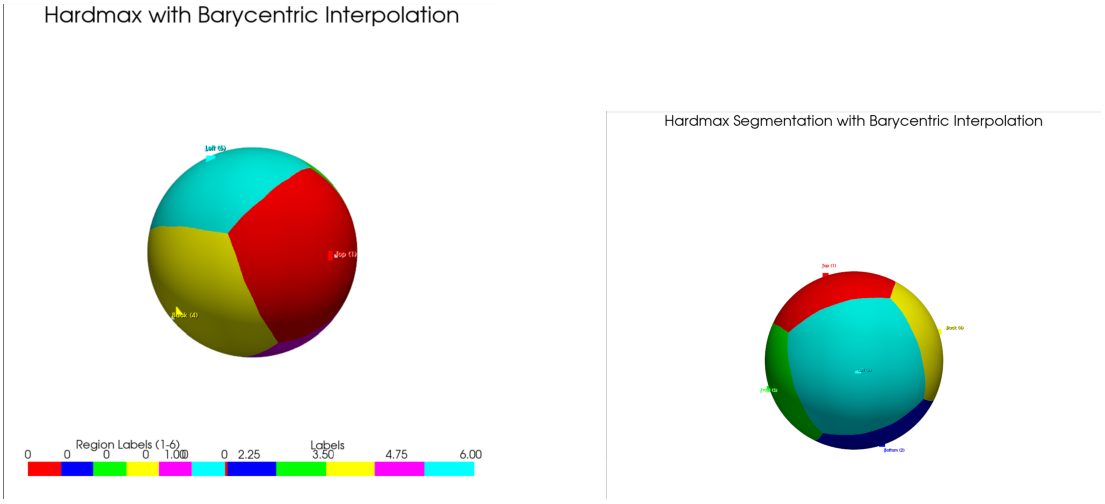
Several technical challenges were addressed during implementation:

Numerical stability in SVD: We implemented a fallback to float32 for SVD computations to enhance stability Triple intersection detection: We developed a robust algorithm for finding and weighting triple points where three regions meet Early stopping criteria: We implemented a patience-based early stopping mechanism to avoid unnecessary computation Memory efficiency: For large meshes, we optimized memory usage through strategic tensor allocation and deallocation

These solutions enabled the framework to handle a wide range of mesh complexities and sizes while maintaining reasonable computational requirements.

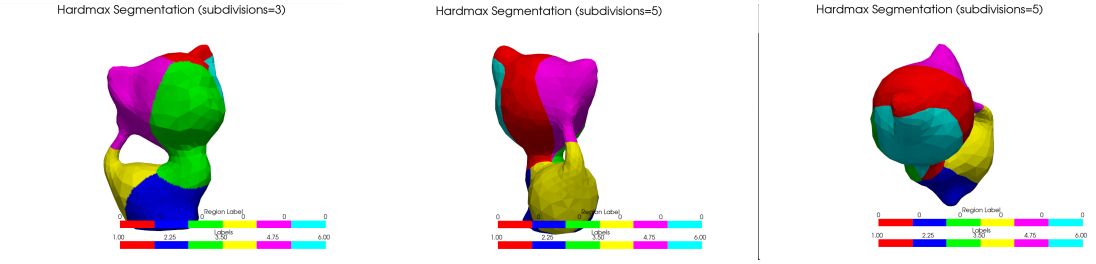
## 4.6 Experimental Results

We firstly test out our method on a 5000 vertice sphere, after 1M rounds of training, we successfully gained the sphere with 6 axis-aligned patches with relatively straight boundaries. Our optimization approach consistently reduced the error metric during training, leading to a stable partition of the mesh surface. That proved our method is effective for simple-geometry and high-resolution mesh.



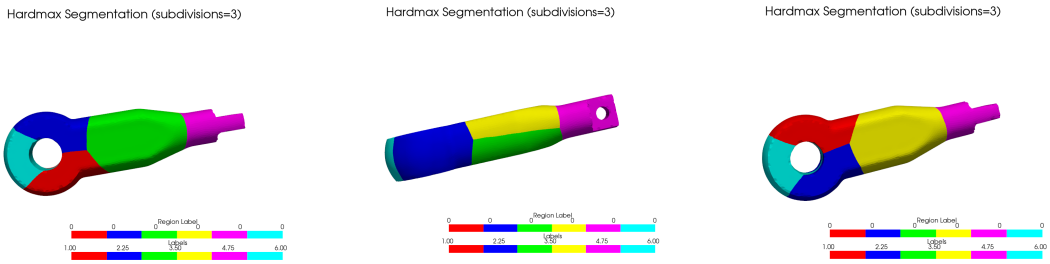

Partitioned Sphere Partitioned Sphere

We then test out our mesh from the dataset in hexalab, it can still create patches on relatively simple meshes but the loss started to refuse to converge, and the cut become non-planar. The optimization process exhibited different behavior compared to the sphere test, suggesting geometry complexity affects performance. These results indicate a limitation in our current approach when handling non-trivial shapes.




(a) (b) (c)

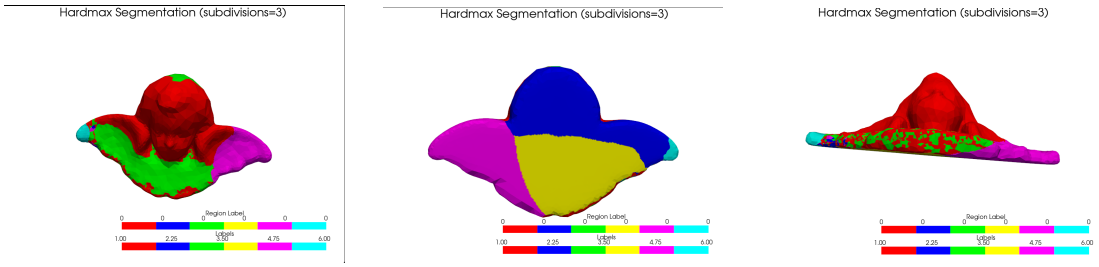
Figure 4.1: Partitioned Kitty Mesh




(a) (b) (c)

Figure 4.2: Partitioned Rod Mesh

For even more complicated mesh with our contour loss stayed in $5 \times 10^1$, and unable to converge further. Despite attempts to extend training time and adjust hyperparameters, these complex geometries remained challenging. The results suggest that future work should focus on developing more robust optimization strategies for handling intricate mesh structures.




(a) (b) (c)

Figure 4.3: Unsuccessful Try On Angel Mesh

## 4.7 Other tried Contour Alignment Loss

### 4.7.1 Variant 1: Learnable Plane Offsets

This implementation uses predefined plane normals (pinned axes) but learns the offset for each plane:

$$\text{For each edge } (v_a, v_b) \text{ and channel pair } (i, j): \tag{4.74}$$
$$w_{edge} = \sigma(-\beta_{edge} \cdot d_a \cdot d_b) \tag{4.75}$$
$$\alpha = \frac{|d_a|}{|d_a| + |d_b| + \varepsilon} \tag{4.76}$$
$$p_{intersection} = v_a + \alpha(v_b - v_a) \tag{4.77}$$
$$\text{Loss} = \sum_{m,c} w_{m,c} \cdot (p_m \cdot n_c + \text{plane\_offsets}[c])^2 \tag{4.78}$$

This approach requires maintaining a global parameter `plane_offsets` as an `nn.Parameter`, increasing implementation complexity.

### 4.7.2 Variant 2: Adjacency Direction Alignment

This variant focuses on aligning gradient directions across adjacent triangles:

$$\text{For each triangle } t: \tag{4.79}$$
$$\nabla f_c(t) = \text{compute gradient of channel } c \text{ in triangle } t \tag{4.80}$$
$$\text{For each adjacent triangle pair } (t_1, t_2) \text{ and channel pair } (c, i): \tag{4.81}$$
$$\text{dir\_diff} = \nabla(f_c - f_i)(t_1) - \nabla(f_c - f_i)(t_2) \tag{4.82}$$
$$\text{adjacency\_loss} = \sum \text{intersection\_weight} \cdot \|\text{dir\_diff}\|^2 \tag{4.83}$$
$$\text{plane\_loss} = \sum \lambda_{plane} \cdot s_{min} \tag{4.84}$$
$$\text{Total loss} = \text{adjacency\_loss} + \text{plane\_loss} \tag{4.85}$$

This approach explicitly enforces consistency in gradient directions between adjacent triangles but adds computational complexity.

### 4.7.3 Variant 3: Compact Vectorized Implementation

A more compact but equally vectorized approach focusing on computational efficiency:

$$\text{Build all channel pairs } (i < j) \text{ via triu\_indices} \tag{4.86}$$
$$\text{Compute differences } d = f_i - f_j \text{ for all pairs} \tag{4.87}$$
$$\text{Edge intersection weight: } w = \sigma(-\beta \cdot d_a \cdot d_b) \tag{4.88}$$
$$\text{Weighted covariance: } \text{Cov}_{ij} = \frac{\sum_k w_k p_k p_k^T}{\sum_k w_k} - \mu_{ij}\mu_{ij}^T \tag{4.89}$$
$$\text{Plane normal via SVD, MSE distance computation} \tag{4.90}$$
$$\text{Loss} = \sum_{i<j} \text{mean squared distance to plane} \tag{4.91}$$

This variant eliminates all Python loops and focuses on maximum computational efficiency using PyTorch's batched operations.

### 4.7.4 Convergence Challenges

Despite implementation correctness, all variants faced convergence difficulties:

- High sensitivity to hyperparameters ($\beta$, $\lambda$)
- Numerical instability in SVD calculations
- Complex loss landscape with many local minima
- Interference with other loss terms
- Scale dependence requiring careful normalization

While theoretically sound, practical applications required extensive tuning and regularization strategies, limiting the immediate utility of these loss functions without further refinement.

# Chapter 5

# Conclusions

## 5.1 Overview

This work explored how activation functions, particularly ReLU and sharpened sigmoid, can be used to create piecewise linear representations on mesh surfaces. By assigning scalar values to vertices and applying nonlinear activations, we induced sharp transitions in an otherwise linearly interpolated attribute field. This approach addresses a persistent challenge in geometry processing: how to represent abrupt changes (e.g., sharp edges or boundaries) without resorting to dense remeshing or higher-order interpolation schemes.

We demonstrated our method both in two-dimensional and three-dimensional contexts, highlighting how a carefully designed loss function—encompassing gradient alignment, area balance, and smoothness—can yield crisp boundaries and well-defined partitions on a mesh. In the three-dimensional case, we further introduced a multi-channel field representation that segments the surface into multiple axis-aligned or user-defined patches. Throughout the project, our technique aimed to streamline the generation of piecewise linear partitions suitable for polycube maps and related downstream applications.

## 5.2 Key Contributions

### 5.2.1 Activation-Driven Piecewise Linearity

A central contribution was proving and experimentally verifying that simple activations, such as ReLU or a steepened sigmoid (Heaviside-like), effectively transform a linearly interpolated field into a piecewise linear one with abrupt changes across faces. This idea is intuitive in neural implicit representations but less common in traditional mesh-processing pipelines. Our implementation in 2D meshes illustrated how a single triangle could be "cut" along a chosen boundary by merely changing the signs of vertex values, then activating them to enforce piecewise-constant subregions.

### 5.2.2 Loss Formulations for Sharp Boundaries

We introduced a gradient alignment loss to encourage boundaries to remain straight (or at least consistently oriented) across adjacent faces. By penalizing large discrepancies in gradient directions among triangles that cross the zero level set, our method ensures that transitions between "positive" and "negative" regions do not meander unnecessarily. This concept was extended to multi-region segmentation by examining pairwise differences between channel values and aligning the resulting iso-contours.

### 5.2.3 Multi-Channel Segmentation on 3D Meshes

For three-dimensional surfaces, we moved from a single scalar function to a vector-valued field, with each channel corresponding to a desired region or "label." Applying an argmax argmax or softmax at each vertex naturally partitions the mesh into multiple pieces. This multi-channel extension shows promise for more complex tasks such as polycube mapping, where each channel can represent a distinct axis-aligned face or patch. By building a single optimization problem that balances smoothness, contour sharpness, and area coverage, we managed to produce multiple well-defined regions without prohibitive remeshing.

### 5.2.4 Pathway to Polycube Maps

One of the main motivations for introducing piecewise linear cuts is to facilitate polycube map construction. Polycube maps provide important advantages, like globally regular structures and simpler parameter domains, but they can be difficult to construct for arbitrary shapes. Our work contributes to this goal by creating clear axis-aligned partitions on a mesh. Although we have not fully integrated these partitions into a complete volumetric or hexahedral meshing pipeline, the results offer a compelling step toward generating less distorted parameterizations for complex surfaces.

## 5.3 Limitations and Challenges

### 5.3.1 Complex Geometry Convergence Issues

While our method converged reliably on simpler or moderately complex shapes (e.g., spheres, low-detail objects), it encountered difficulties on intricate meshes with rich geometric features (e.g., models containing multiple protrusions or high curvature regions). The contour alignment loss sometimes plateaued, indicating the optimization could get stuck in local minima or require more specialized scheduling.

### 5.3.2 Non-Planar Boundaries

Our framework aims to produce "straight" or piecewise-linear boundaries, but for very complex shapes, boundaries may deviate from the desired plane or axis-aligned cut. The gradient alignment loss encourages planarity, yet extremely detailed models can

still yield slight undulations or misalignments, particularly when the pinned vertices or channel assignments conflict with natural shape features.

### 5.3.3 Parameter Sensitivity

Another challenge is the sensitivity of results to hyperparameters such as learning rate, softmax temperature schedule, and relative weighting of the loss terms (area balance, gradient alignment, and smoothness). Achieving optimal performance often required careful tuning, suggesting that a more adaptive or automated approach to parameter selection could be beneficial.

## 5.4 Future Work

### 5.4.1 Robust and Adaptive Optimization

One avenue for improvement is adopting more advanced optimization algorithms—e.g., multi-scale (coarse-to-fine) methods, or incorporating second-order information—to tackle local minima on complex meshes. Adaptive gradient methods with dynamic parameter schedules might prevent stagnation and improve boundary sharpness.

### 5.4.2 Feature-Driven Constraints

We could integrate more shape-aware constraints (e.g., curvature-based weighting, geodesic distances) into the loss formulation to anchor boundaries at significant surface features. Such an approach might better align partitions with natural edges or ridges, enhancing the final segmentation quality.

### 5.4.3 Extension to Volumetric Meshing and Polycubes

While our work primarily addressed surface segmentation, extending these concepts volumetrically could unlock advanced all-hex or hex-dominant meshing strategies. By defining multi-channel fields throughout a volume, it might be possible to achieve consistent piecewise-linear structures that form the foundation of a full 3D polycube decomposition.

### 5.4.4 Interactive Segmentation Tools

Many applications in computer graphics and modeling benefit from user control. An interactive interface allowing direct manipulation of pinned vertices or region constraints could guide the segmentation in real time, balancing automated optimization with artistic or engineering demands.

## 5.5 Concluding Remarks

Overall, this work shows the feasibility and promise of using activation functions to achieve sharp, piecewise linear partitioning on 2D and 3D meshes. By leveraging simple but powerful concepts—activation-driven boundary formation, gradient alignment, and multi-channel fields—we opened a pathway toward more flexible polycube map generation and other structured meshing tasks. While there are clear hurdles in scaling to very intricate shapes, the results on simpler or moderately complex models demonstrate the core idea's effectiveness. With further refinements in optimization strategies and geometric constraints, activation-based segmentation can become a valuable technique in the broader geometry-processing toolbox.

# Bibliography

# Bibliography

[1] HUANG, JINGWEI AND WANG, KANG AND LIU, LIGANG AND TONG, XIN (2014).
*Boundary Aligned Polycube Construction from Cross Fields*.
*ACM Transactions on Graphics (SIGGRAPH Asia)*, 33(6), Article 224 (pp. 1–11).
ACM. https://doi.org/10.1145/2661229.2661238

[2] KARNEWAR, ADITYA AND EGGERT, LUCAS AND HENZLER, PHILIPP AND NIESSNER, MATTHIAS (2022).
*ReLU Fields: The Little Non-linearity That Could*.
arXiv preprint https://arxiv.org/abs/2206.10771.