FACULTY OF APPLIED SCIENCES
UNIVERSITY
OF WEST BOHEMIA

DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Bachelor's Thesis

Compression of connectivity for meshes with known geometry

Stanislav Kafara

PILSEN, CZECH REPUBLIC

2024

FACULTY OF APPLIED SCIENCES
UNIVERSITY
OF WEST BOHEMIA

DEPARTMENT OF
COMPUTER SCIENCE
AND ENGINEERING

Bachelor’s Thesis

Compression of connectivity for meshes with known geometry

Stanislav Kafara

Thesis advisor
Doc. Ing. Libor Váša, Ph.D.

PILSEN, CZECH REPUBLIC **2024**

Citation in the bibliography/reference list:
KAFARA, Stanislav. *Compression of connectivity for meshes with known geometry*. Pilsen, Czech Republic, 2024. Bachelor's Thesis. University of West Bohemia, Faculty of Applied Sciences, Department of Computer Science and Engineering. Thesis advisor Doc. Ing. Libor Váša, Ph.D.

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Stanislav KAFARA**
Osobní číslo:	**A21B0160P**
Studijní program:	**B0613A140015 Informatika a výpočetní technika**
Specializace:	**Informatika**
Téma práce:	**Komprese konektivity trojúhelníkových sítí se známou geometrií**
Zadávající katedra:	**Katedra informatiky a výpočetní techniky**

Zásady pro vypracování

1. Seznamte se s algoritmem pro kompresi konektivity trojúhelníkové sítě se známou geometrií, vyvíjeným na Katedře informatiky a výpočetní techniky.
2. Implementujte a otestujte vylepšení algoritmu založené na vyloučení nemanifoldních hran a hran vedoucích na neorientovatelný povrch.
3. Navrhněte, implementujte a otestujte automatickou volbu alespoň jednoho parametru (váhy) funkce pro vyhodnocení kvality kandidátního trojúhelníku na základě relevantní globální statistiky vyhodnocené nad vstupní trojúhelníkovou sítí (průměrný vnitřní úhel, průměrný dihedrální úhel atd.).
4. Na základě dosažených výsledků navrhněte a popište další možná vylepšení algoritmu.
5. Dosažené výsledky důkladně zdokumentujte.

Rozsah bakalářské práce:	**doporuč. 30 s. původního textu**
Rozsah grafických prací:	**dle potřeby**
Forma zpracování bakalářské práce:	**tištěná/elektronická**

Seznam doporučené literatury:

Dodá vedoucí bakalářské práce.

Vedoucí bakalářské práce:	**Doc. Ing. Libor Váša, Ph.D.** Katedra informatiky a výpočetní techniky
Datum zadání bakalářské práce:	**2. října 2023**
Termín odevzdání bakalářské práce:	**2. května 2024**

L.S.

Doc. Ing. Miloš Železný, Ph.D. děkan	**Doc. Ing. Přemysl Brada, MSc., Ph.D.** vedoucí katedry

V Plzni dne 25. října 2023

Declaration

I hereby declare that this Bachelor's Thesis is completely my own work and that I used only the cited sources, literature, and other resources. This thesis has not been used to obtain another or the same academic degree.

I acknowledge that my thesis is subject to the rights and obligations arising from Act No. 121/2000 Coll., the Copyright Act as amended, in particular the fact that the University of West Bohemia has the right to conclude a licence agreement for the use of this thesis as a school work pursuant to Section 60(1) of the Copyright Act.

In Pilsen, on 02 May 2024

. . . .
Stanislav Kafara

Abstract

This work aims to improve the compression ratio of the state-of-the-art single-rate priority-based connectivity compression algorithm for triangle meshes with known geometry developed at the Department of Computer Science and Engineering at the University of West Bohemia.

It enriches the general encoder and decoder capabilities to filter out additional inadmissible candidate vertices and to update gates' priorities as the mesh traversal proceeds. It proposes a way to automatically determine a better-than-default encoder's weight configuration per mesh based on the knowledge of its global surface statistics. The text describes the individual attempts to enhance the algorithm and presents experimental results, evaluating their actual impact on the resulting data rate.

Abstrakt

Tato práce se zabývá možnostmi vylepšení kompresního poměru prioritou řízeného algoritmu, vyvinutého na Katedře informatiky a výpočetní techniky Západočeské univerzity, který představuje aktuální stav poznání v oblasti jednostupňové komprese konektivity trojúhelníkových sítí se známou geometrií.

Práce rozšiřuje schopnosti kodéru a dekodéru o filtrování dalších nepřípustných kandidátních vrcholů a o aktualizaci priorit bran v průběhu průchodu trojúhelníkovou sítí. Navrhuje způsob, jak automaticky určit lepší než výchozí váhovou konfiguraci kodéru pro každou síť na základě znalosti globálních statistik jejího povrchu. Text práce popisuje jednotlivé pokusy o vylepšení algoritmu, prezentuje experimentální výsledky a hodnotí jejich skutečný dopad na výsledný datový tok.

Keywords

Triangle mesh • Connectivity compression • RBF approximation • Optimisation • Artifical neural network

Acknowledgement

I would like to thank my thesis supervisor, Doc. Ing. Libor Váša, Ph.D., for his willing guidance, valuable advice and enthusiasm. I also thank Ing. Filip Hácha for providing technical consultations.

Contents

Introduction 1

In computer graphics, we often need to work with three-dimensional objects. Representing these objects is a fundamental field of study. Its importance stems from its broad spectrum of applications in various domains, ranging from industrial to entertainment.

A triangle mesh is a commonly used discrete representation of an object, approximately capturing its surface as a set of vertices and faces. Compressing meshes becomes necessary to achieve a more detailed representation of the object or to represent more complex objects while maintaining reasonable storage and transmission costs, as the level of detail highly depends on the vertex and face count of the mesh.

This work explores further possibilities for improving the compression ratio of the priority-based connectivity coding algorithm [Dvo+22] developed by Dvořák et al. at the Department of Computer Science and Engineering at the University of West Bohemia. The performed experiments utilise the same datasets as in [Dvo+22], making it possible to compare the results directly to the previously achieved results.

The text first describes the existing algorithm in detail and provides its context in the field of mesh compression. Then, it introduces and discusses possible enhancements. Afterwards, it proposes and discusses the solutions and analyses the experimental results. Eventually, it gives directions and suggestions for further research.

Mesh Compression 2

Compression is a process and a set of techniques that ensure the transformation of information into another representation, i.e. the compressed representation, reducing its size while preserving the information. The *encoder* transforms the original representation into the compressed representation, and the *decoder* reconstructs the compressed representation back into the original representation.

Compression may be either *lossless* or *lossy*. Lossless compression exploits specific knowledge of the data and its redundancy and preserves all the information. On the other hand, lossy compression additionally removes some not-so-important information from the data.

The effectivity of a compression method may be expressed as a *compression ratio*, formulated as

$$\text{Compression Ratio} = \frac{|\text{Original Representation}|}{|\text{Compressed Representation}|},$$

defining it as a ratio of the size of the original representation to the size of the compressed representation.

2.1 Triangle Mesh

A *triangle mesh* is a data structure comprising a set of vertices V, i.e., mesh *geometry* and a set of faces F, i.e., mesh *connectivity*. The vertices are points in a three-dimensional space, and the faces are triangles formed by connecting these vertices with *edges*. An example triangle mesh, the well-known Stanford Bunny model, is visualised in Fig. 2.1.

The effectivity of a representation of a triangle mesh may be expressed as its required data rate in *bits per vertex* (bpv), formulated as

$$\text{Bits per Vertex} = \frac{|\text{Representation}|_{[\text{bits}]}}{|V|},$$

defining it as a ratio of the size of the representation in bits to the mesh's vertex count. Analogically, it may be expressed as its required data rate in *bits per face* (bpf),

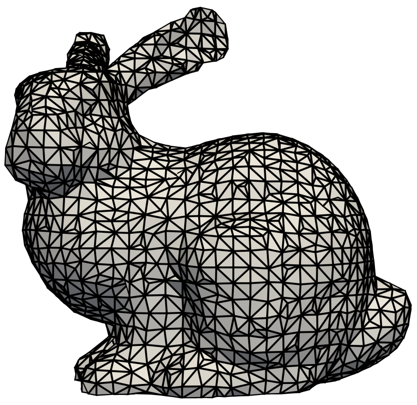

Figure 2.1: Triangulated Stanford Bunny model [TL94]

formulated as

$$\text{Bits per Face} = \frac{|\text{Representation}|_{[\text{bits}]}}{|F|},$$

defining it as a ratio of the size of the representation in bits to the mesh's face count.

Below are some terms and properties referred to later in the text. A *triangle fan* is a set of triangles incident to a common central vertex. A *boundary edge* is an edge incident to one face. A *boundary* is a set of boundary edges forming a cycle connecting their common vertices. A mesh has an *orientable surface* if its edges may be oriented so that each pair of adjacent triangles will have opposite orientations of the edges on their common side. The genus of a surface determines the number of *handles* it has. This text often mentions the property of *manifoldness* [Bot+10]. A *manifold vertex* is a vertex with one incident triangle fan. A *manifold edge* is an edge incident to at most two faces. A *manifold mesh* is a mesh with a surface that is locally homeomorphic to a disc or a half-disc at the boundary for every point of the surface. Equivalently, a manifold mesh contains neither non-manifold vertices nor non-manifold edges nor self-intersections. Fig. 2.2 illustrates examples of non-manifold meshes and an unorientable surface.

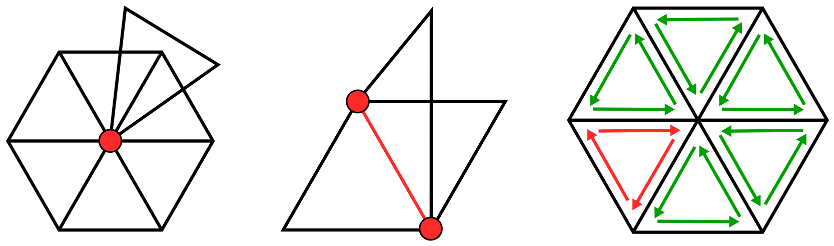

Figure 2.2: Non-manifold vertex, non-manifold edge and unorientable surface

2.2 Related Algorithms

This section briefly introduces the existing and related mesh compression techniques and algorithms. Several [PKJ05] have been developed in the past, each with its advantages and disadvantages in certain situations, making each suitable for a different practical application. They will be classified based on their principal differences.

Earlier, it was necessary to compress *static meshes*, i.e., meshes that do not change in time. Later, with the advancements in the fields of object animation and scanning, compressing *dynamic meshes*, i.e., meshes that are part of temporal sequences and change over time, was needed.

Single-rate compression algorithms [PKJ05] encode and decode a mesh as a whole. These may be practical for efficient representation, exploiting as much information about the whole mesh as possible and allowing freedom of order in encoding the mesh. In contrast, *progressive* compression algorithms [PKJ05] proceed by gradually simplifying the mesh and coding consecutive steps to reconstruct the mesh, with each step getting closer to the original shape. They thus may be very efficient for streaming, as they enable immediate work with a coarse representation of the mesh, which is gradually refined on the go as the decoder receives further instructions.

Another fundamental property is whether they compress connectivity and geometry simultaneously or independently. Methods that compress the connectivity and the geometry simultaneously primarily encode the connectivity, alongside encoding the geometry by utilising the knowledge of the connectivity. The decoder then decodes the geometry alongside the connectivity. Methods that compress connectivity and geometry independently of each other may utilise the knowledge of the geometry at both the encoder and the decoder to achieve a better compression ratio. The decoder then decodes the geometry and, exploiting its knowledge, decodes the connectivity. Most algorithms do not utilise this approach. However, it may be convenient in certain situations, e.g., if a mesh is a part of a temporal sequence and its geometry can be predicted efficiently.

Most current single-rate mesh compression methods build on a canonical *mesh traversal* that begins with selecting and encoding an *initial triangle*, splitting the mesh into the *processed part* and the *unprocessed part* divided by border edges called *gates*. It progresses by selecting a gate and expanding the processed part by one triangle, the one incident to the gate from the unprocessed part, encoding the information required to identify the triangle later by the decoder. It repeats this process until it processes all the triangles. The traversal is unambiguously defined, and the encoder utilises only the information the decoder also possesses, ensuring they stay synchronised. The decoder then mimics the traversal the encoder performs and reconstructs the mesh utilising the encoded information. These algorithms [PKJ05] compress connectivity losslessly with a 1-4 bpv data rate and geometry lossy with a 9-12 bpv data rate.

The following sections describe two well-known representatives of single-rate manifold mesh compression algorithms in more detail. One presents a connectivity-first approach, encoding geometry alongside connectivity, and the other a separate connectivity and geometry approach to mesh compression. They provide more context for the algorithm this work builds upon later.

2.2.1 **Edgebreaker**

Edgebreaker [Ros99], developed by Rossignac, is a manifold mesh compression algorithm that encodes connectivity and geometry simultaneously. It traverses the mesh, and based on the local state of the processed part of the mesh in each iteration, it encodes a particular symbol from the *CLERS alphabet* into a compressed connectivity stream and, if necessary, encodes a correction of a *parallelogram prediction* to the actual triangle's tip vertex into a compressed geometry stream. The CLERS symbols represent all the possible actions the algorithm can take that drive the traversal based on the processed or unprocessed state of the triangles in a fan formed around the encoded triangle's tip vertex. Fig. 2.3 illustrates these possible cases. The blue edge represents the active gate, and X is the currently processed triangle with its tip vertex v. The blue triangles belong to the unprocessed part, while the already-processed triangles are transparent.

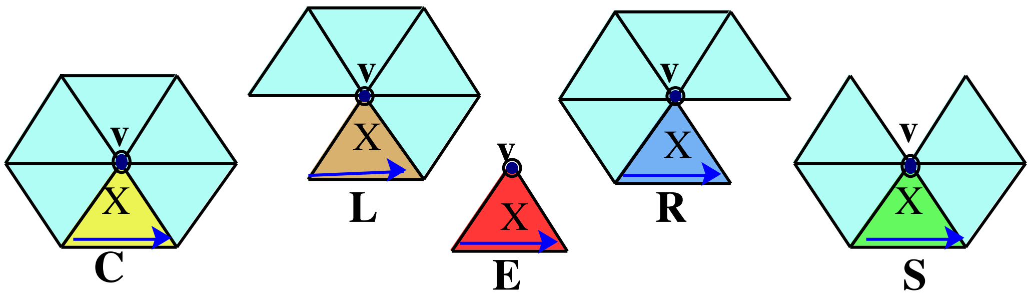

Figure 2.3: Edgebreaker CLERS alphabet [Ros99]

It begins with selecting an initial triangle, encoding the positions of its vertices and marking the vertices and the triangle as encoded. Afterwards, the encoding loop begins with the triangle opposite to the tip vertex of the initial triangle. It marks the triangle as encoded and, based on the states of its tip vertex and triangles to the left and right, proceeds as follows:

- C: encodes the correction of a prediction, encodes symbol C, marks the triangle's tip vertex as encoded and continues with the triangle to the right,

- L: encodes symbol L and continues with the triangle to the right,
- R: encodes symbol R and continues with the triangle to the left,
- S: encodes symbol S, recursively continues with the triangle to the right and after returning, continues with the triangle to the left,
- E: encodes symbol E and breaks from the current encoding loop.

The resulting CLERS string unambiguously describes the mesh's connectivity and defines the geometry associating it with the encoded C symbols. During decompression, it loops over the CLERS string in the encoded connectivity stream, simulating the connectivity traversal. Traversing the mesh for the first time, it decodes the connectivity, knowing the common colocated vertices of consecutive triangles. Then, it decodes the positions of the vertices of the initial triangle and, traversing the mesh again, reconstructs the geometry from the separate stream of predictions' corrections by adding the correction to the prediction of the tip vertex in the currently decoded triangle when it encounters the symbol C in the CLERS string.

The algorithm guarantees encoding the connectivity of a mesh homeomorphic to a sphere at a data rate of 4 bpv. However, if it additionally passes the CLERS string to an *entropy coder*, it may be less than 3 bpv for large meshes. The algorithm may be modified to support encoding meshes with boundaries and handles. However, additional storage is required for each boundary or handle.

2.2.2 Distance-Ranked Connectivity Coding

The algorithm presented in [MGS07], developed by Marais et al., is a manifold mesh compression algorithm that encodes connectivity and geometry separately. It exploits the knowledge of geometry at both the encoder and decoder to achieve better results in compressing connectivity. It traverses the mesh, gradually extending the processed part with triangles from the unprocessed part. In each iteration, it takes one gate and constructs a prediction of the expansion triangle's actual tip vertex. It encodes the *rank*, i.e. the index in a list of vertices sorted by their Euclidean distance from the prediction, of the actual tip vertex into the compressed connectivity stream. The list of these *candidate vertices* is additionally filtered so as not to consider *inadmissible candidates*, i.e., vertices having complete fans of triangles formed around them, all of which belong to the already processed part of the mesh. Fig. 2.4 illustrates ranking the candidate vertices c_i based on their distance from the prediction p.

It begins with selecting an initial triangle and encoding its actual vertex indices. Afterwards, it initialises a double-ended queue of gates with the initial triangle's

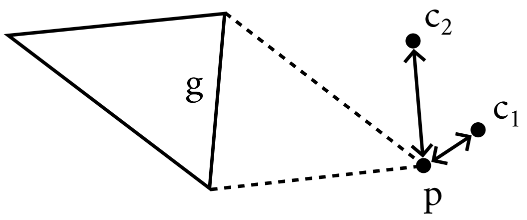

Figure 2.4: Distance rank

edges. Then, it loops until the queue is empty. In every iteration, it pops a gate from the front of the queue. If the gate is marked as completed, the next iteration follows. Otherwise, if it is a boundary edge, it encodes 0, signalising the boundary edge. Otherwise, it constructs a prediction and encodes the rank of the actual tip vertex, the best being 1. Finally, it marks the gate as completed, preventing it from being processed if popped from the queue again and pushes the newly formed left and right gates to the queue, one to the front and the other to the back, for improved traversal.

The decompression mirrors the compression process. First, it reads the initial triangle's vertices and initialises the queue with its edges. Then, it continues looping until the queue is empty. During every iteration, it pops a gate from the queue. If the gate is marked as completed, it proceeds to the next iteration. Otherwise, if it is a boundary edge, no triangle is attached to the processed part of the mesh. Otherwise, it constructs a prediction and extends the processed part of the mesh with a triangle whose tip vertex is the one with the encoded rank. Finally, it marks the gate as completed and pushes the newly formed left and right gates to the queue.

The algorithm's effectiveness highly depends on the quality of predictions. The method may and does beforehand determine which predictor it will use to exploit the geometry better to describe the connectivity. A simple parallelogram predictor works well for regular meshes. However, it performs poorly on irregularly triangulated meshes. In [MGS07], Marais et al. discuss a few other predictors, exploiting more information about the geometry, e.g., surface curvature or triangle sizes.

If the utilised predictor constructs good predictions, the actual tip vertex will lie close to the prediction, yielding small ranks with low entropy, making it a good input for the used entropy coder. The algorithm performs well on smooth and regular meshes. However, it may perform worse than other mesh compression algorithms on meshes on the opposite side of the spectrum. Section 2.3.4, respectively Table 2.2, discusses its effectiveness of compression.

2.3 Priority-Based Connectivity Coding

The algorithm presented in [Dvo+22], developed by Dvořák et al., is a manifold mesh compression algorithm that encodes connectivity and geometry separately. It builds on the distance-ranked approach described in [MGS07], as summarised in Section 2.2.2, with the fundamental difference being the *priority-driven mesh traversal*.

It traverses the mesh, prioritising expanding the processed part of the mesh with a triangle, with its base being the best priority gate. A *gate's priority* represents the probability that the actual tip vertex can be well distinguished from the other candidate vertices. A *candidate's quality* represents the heuristic estimate of its likelihood to be the actual tip vertex of the triangle extending from the gate.

The algorithm additionally takes a *weight configuration* as an input, defining the function to evaluate a candidate's quality, as discussed later in Section 2.3.1. Algorithm 2.1 summarises the process of encoding a mesh. It shares the fundamental basis

Algorithm 2.1: Priority-based connectivity encoding

```
Input: Mesh, Initial triangle, Weight configuration
Output: List of candidate indices
1  Initialise data structures
2  UpdateFilter(initial triangle)
3  priority queue ⟵ initial triangle's edges
4  while priority queue is not empty do
5      priority queue ⟶ gate
6      if gate is completed then continue
7      if gate is boundary then
8          candidate indices ⟵ EncodeBoundary(gate)
9          UpdateFilter(gate)
10     else
11         candidates ⟵ QueryCandidates(gate, Quality(tip vertex))
12         candidates ⟵ Filter(candidates)
13         candidate indices ⟵ EncodeIndex(tip vertex, candidates)
14         UpdateFilter(triangle formed by gate and tip vertex)
15         priority queue ⟵ newly formed left and right gates
16     end
17     mark gate as completed
18 end
```

of the distance-ranked approach but has some significant differences. Instead of a queue, it utilises a *priority queue*, preferring gates with greater priority. Section 2.3.2 provides more information on how the algorithm utilises gate priorities. Next, it does not determine a candidate's rank by Euclidean distance but by a candidate quality metric discussed in Section 2.3.1. During encoding, if it encounters a completed gate, it proceeds to the next iteration. If the gate is a boundary edge, it handles it specifically, encoding a special index, as described in [Dvo+22]. Otherwise, it queries all the candidates with a quality greater than or equal to the quality of the actual tip vertex, ensuring the completeness of the list of candidates and unambiguity of the selected candidate. Next, it filters out the inadmissible candidates, as described in Section 2.3.3 and encodes the rank of the actual tip vertex, i.e. its index in the list of the candidates sorted by their respective qualities, the best being 0. Then, it computes the priority of the newly formed left and right gates and pushes them to the priority queue. Finally, it updates the *candidate filtering mechanism* accordingly and marks the gate as completed. Fig. 2.5 illustrates a typical situation the algorithm handles in every iteration. The green edge g represents the active gate, and the edges

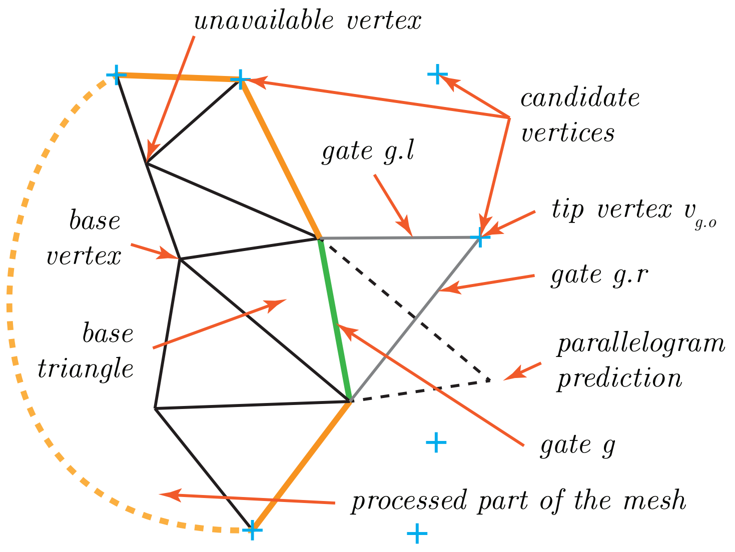

Figure 2.5: Typical iteration of the algorithm [Dvo+22]

$g.l$ and $g.r$ represent the newly formed left and right gates.

The decoder performs inverse operations to the encoder, starting with reconstructing the initial triangle and initialising the priority queue. It continues with traversing the mesh in a priority-driven manner. In each iteration, it pops a gate from the priority queue. If the gate is marked as completed, it proceeds with the next iteration. Otherwise, it reads the next encoded rank r. If it represents a boundary edge, the next iteration follows. Otherwise, it queries at least $r + 1$ already filtered best candidates, ensuring the completeness of the list of candidates. The actual tip vertex is on the index r in the list of these candidates sorted by their respective qualities. Then, it puts the newly formed left and right gates into the priority queue.

Finally, it updates the candidate filtering mechanism accordingly and marks the gate as completed.

2.3.1 Determining Quality

A significant enhancement this algorithm implements over the distance-ranked approach is how it ranks the candidate vertices. Instead of only considering the *distance* from the prediction, it considers three additional factors. It considers the *inner angle* at the tip vertex of the *candidate triangle*, i.e., the triangle having the gate as its base and the candidate vertex as its tip vertex. It also considers the *dihedral angle* of the candidate triangle and the *base triangle*, i.e. the triangle sharing the gate edge with the candidate triangle. It also considers the *similarity* of the base and candidate triangles.

The individual factors are weighted by their importance in contributing to the resulting quality q_c of the candidate vertex c, formulated as

$$q_c = \theta_{Cc} - \frac{w_1}{l_{avg}} \cdot d_{cp} + w_2 \cdot \phi_{BC} + w_3 \cdot S_{BC}, \tag{2.1}$$

defining it as a linear combination of these factors. The symbol θ_{Cc} represents the inner angle at the tip vertex of the candidate triangle, computed as in Eq. (2.2); d_{cp} represents the distance of the candidate from a prediction, computed as in Eq. (2.3); ϕ_{BC} represents the dihedral angle between the base triangle and candidate triangle, computed as in Eq. (2.4), and S_{BC} represents the similarity of the base and candidate triangle, as computed in Eq. (2.5). The symbols w_1, w_2 and w_3 represent the *weight* of the respective terms, and l_{avg} is the average edge length of the mesh used to ensure mesh scale invariance.

Let B be a base triangle defined by the vertices g_2, g_1 and b. Let C be a candidate triangle defined by the vertices g_1, g_2 and c. The vertices g_1 and g_2 are the vertices of the gate g. The point p is a constructed prediction, and b and c are the base and the candidate vertices. Fig. 2.6 illustrates this situation. Then, the inner angle θ_{Cc} at

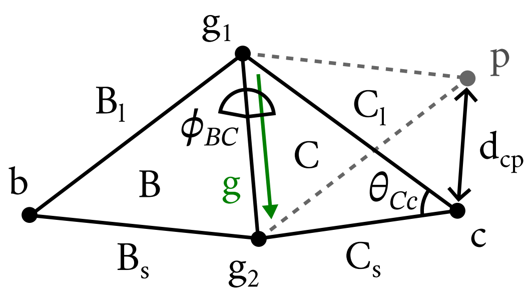

Figure 2.6: Candidate's quality metric components

the vertex c of a candidate triangle C is formulated as

$$\theta_{Cc} = \arccos\left(\frac{\mathbf{cg_1} \cdot \mathbf{cg_2}}{\|\mathbf{cg_1}\| \cdot \|\mathbf{cg_2}\|}\right), \tag{2.2}$$

where $\mathbf{cg_1}$ and $\mathbf{cg_2}$ are the vectors pointing from the vertex c to the vertices g_1 and g_2. The distance d_{cp} of the candidate vertex c from the prediction p is formulated as

$$d_{cp} = \|\mathbf{c} - \mathbf{p}\|. \tag{2.3}$$

The dihedral angle ϕ_{BC} between the base and the candidate triangle B and C is formulated as

$$\phi_{BC} = \pi - \arccos\left(\mathbf{n}_B \cdot \mathbf{n}_C\right), \tag{2.4}$$

where $\mathbf{n}_B$ and $\mathbf{n}_C$ are the unit normal vectors of the base and the candidate triangles. The similarity S_{BC} of the base and the candidate triangle B and C is formulated as

$$\begin{aligned} r_s &= \frac{\|B_s\|}{\|C_s\|}, r_l = \frac{\|B_l\|}{\|C_l\|}, r = \frac{r_s + r_l}{2}, \\ S_{BC} &= -\frac{|r - r_s| + |r - r_l|}{2}, \end{aligned} \tag{2.5}$$

where B_s and B_l are the shorter and the longer non-gate edges of the base triangle B, and C_s and C_l are the shorter and the longer non-gate edges of the candidate triangle C.

The weight for the inner angle θ_{Cc} is implicitly set to 1 to reduce the number of degrees of freedom of the weights of the quality metric function and relate other weights to this one. The goal is to set these weights so that the actual tip vertices have the best quality compared to other candidates. The weights are non-negative numbers, assuming that a greater inner angle at the tip vertex of a candidate triangle, a smaller distance of a candidate from prediction, a greater dihedral angle and a greater similarity of base and candidate triangles contribute positively to a candidate's quality.

2.3.2 Determining Priority

Another significant difference between this algorithm and the distance-ranked approach is how it traverses the mesh. Instead of just popping the gate from the front of the queue, it pops the gate with the best priority. The algorithm evaluates a gate's priority once, at the moment when it is about to put it into the priority queue. The priority is interpreted as the probability that the actual tip vertex is well distinguishable in the list of candidates, assuming that the quality metric function, defined by Eq. (2.1), suits the mesh well.

The algorithm utilises the approximated priority p_g of gate g, defined as

$$p_g = q_{c1} - q_{c2}, \tag{2.6}$$

where q_{c1} and q_{c2} are the best and the second-best candidate qualities. Fig. 2.7 illustrates the property of priority. Suppose there are the red and blue gates, and the

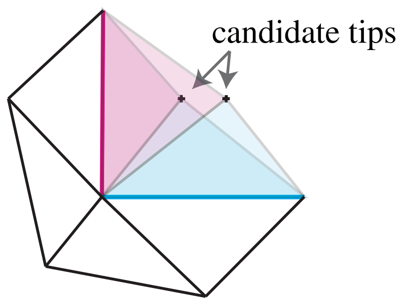

Figure 2.7: Gate priority [Dvo+22]

two vertices are the two best candidates for both gates. In the case of the blue gate, assume that the candidates have identical qualities, resulting in the priority being zero. In the case of the red gate, assume that one candidate has greater quality than the other, resulting in priority being greater than the blue gate's priority, thus being preferred over the blue one to extend the processed part of the mesh with a triangle, since it is more probable to distinguish the actual tip vertex well in the list of candidates than the blue gate.

2.3.3 Filtering Candidates

In every iteration, the algorithm needs to query possible candidates for the actual tip vertex forming a triangle extending from the active gate, either to determine its rank, to determine it by its rank, or to evaluate the gate's priority before putting it into the priority queue. However, based on the particular weights of the utilised candidate's quality metric function (Eq. (2.1)) and possible local distribution of the mesh vertices for which it is not well-suited, many candidates may have greater quality than the actual tip vertex. Therefore, to yield smaller ranks and improve accuracy by not considering inadmissible candidates when evaluating the gate's priority, the algorithm implements a candidate filtering mechanism that filters out the inadmissible candidates, i.e. the candidates that cannot possibly form a triangle extending from the gate.

The algorithm implements the same candidate filtering mechanism utilised by the distance-ranked approach [MGS07]. It filters out every vertex that has a complete fan of triangles formed around it, all of which belong to the already processed part of the mesh. Such a vertex may not be considered since expanding the processed part of the mesh with such a triangle violates the assumption that the mesh is manifold. Of course, it also filters out the gate vertices and the base triangle's tip vertex. During the traversal, the candidate filtering mechanism is regularly updated to reflect the state of the already processed part of the mesh.

2.3.4 Achieved Results

In [Dvo+22], Dvořák et al. performed experiments to evaluate the effectiveness of their proposed method. They tested it against six datasets, providing a general overview of its performance on various meshes and comparing it to the distance-ranked approach. These datasets include *abc_regular* and *abc_irregular*, which contain regular and irregular meshes of the CAD models from the ABC [Koc+19] dataset. The *thingi10k* [ZJ16] dataset comprises 3D-printed models with varying regularity. The *tosca* [BBK08] and *mcgill* [Zha+05] datasets contain synthetic models in different poses. Finally, the *casual_man* dataset consists of a proprietary 3D-scanned sequence of meshes of varying connectivity representing a moving human. Fig. 2.8 visualises a representative mesh from each dataset, from left to right: *tosca*, *casual_man*, *abc_irregular*, *abc_regular*, *mcgill* and *thingi10k*.

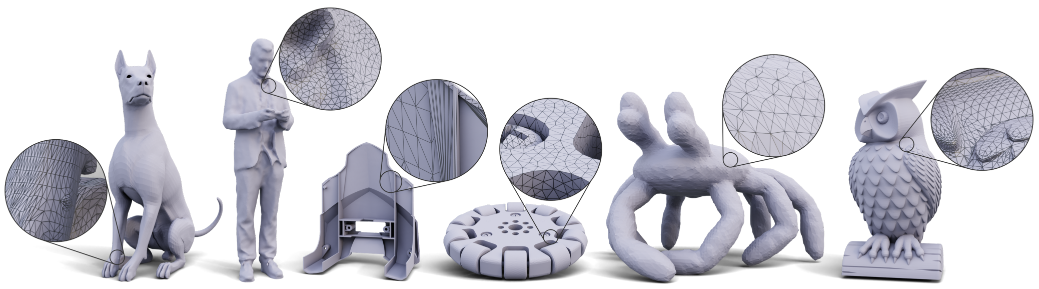
Figure 2.8: Representative meshes of the experimental datasets [Dvo+22]

A weight configuration refers to the weights w_1, w_2 and w_3 of the respective components of the candidate's quality metric function defined by Eq. (2.1). Dvořák et al. determined the *dataset-optimised weight configuration* for each dataset by randomly selecting 10 meshes from each dataset and optimising the total sum of the data rates. Additionally, they determined a *default weight configuration* by optimising the data rates of 40 selected meshes of varying properties. Table 2.1 shows the particular weight configurations.

They evaluated the compression performance for the default and the dataset-optimised weight configurations. Table 2.2 summarises the results and compares them with the distance-ranked approach. They both use a *CABAC* [MSW03] entropy coder to encode candidate indices. The mentioned data rate is the average data rate weighted by the respective mesh vertex count.

Table 2.1: Dataset-optimised weight configurations [Dvo+22]

dataset	w_1	w_1	w_3
default	0.4094	0.7920	0.1350
abc_regular	0.252	0.654	0.151
abc_irregular	0.84	1.8125	0.089
thingi10k	0.24	0.905	0.000015
tosca	2.18	1.05	0.0005
mcgill	0.362	0.116	0.845
casual_man	0.1406	0.8781	0.0272

Table 2.2: Compression performance summary [Dvo+22]

		bpf		
dataset	**# meshes**	$\mathbf{w}_{\text{default}}$	$\mathbf{w}_{\text{dataset}}$	[MGS07]
abc_regular	10000	0.191	0.181	0.331
abc_irregular	10000	1.059	1.050	2.261
thingi10k	8133	0.988	0.919	1.523
tosca	80	1.129	1.112	1.343
mcgill	458	0.487	0.448	0.708
casual_man	546	0.165	0.152	0.264

Enhancements 3

The priority-based connectivity coding algorithm [Dvo+22], described in Section 2.3, leaves room for improvement. This work attempts to further improve the data rate of the *reference implementation* by filtering out additional inadmissible candidate vertices, continuously updating priorities of unprocessed gates and estimating *mesh-optimal weight configurations*.

As described in Section 2.3.3, the reference implementation already filters the candidate vertices utilising a candidate filtering mechanism. While it generally reduces the encoded candidate ranks and entropy, additional inadmissible candidates may be identified and filtered out. This work implements a candidate filtering mechanism that additionally filters out the candidates, which would lead to a mesh with a non-manifold edge or an unorientable surface if the candidate triangle extended the processed part of the mesh, violating the assumption of an orientable manifold mesh on input.

The reference implementation evaluates a gate's priority once, as described in Section 2.3.2. The priority remains unchanged throughout the gate's existence in the priority queue. However, as the mesh traversal proceeds, the local area determining one's gate priority may change as the processed part of the mesh expands, and so may its priority. Assuming that the candidate's quality metric function is well-suited and the priority represents the probability of distinguishing the actual tip vertex well in the list of candidates, continuously updating priorities should further improve the traversal and, thus, the data rate, as it would utilise additional more accurate recent information. This work implements a mechanism that synchronises the gate priorities with the changes in the unprocessed part of the mesh.

The data rate depends primarily on the choice of a weight configuration, as discussed in Section 2.3.4, as it determines the encoded candidate ranks and defines the priority, as described in Section 2.3.2, that drives the mesh traversal. However, achieving this algorithm's optimal compression ratio for different meshes requires different utilised weight configurations. Assuming it is possible to obtain complete descriptive statistics of a mesh, such as a mean inner of its triangles or a mean dihedral angle of adjacent pairs of its triangles, and that there exists a relation between the mesh statistics and the respective optimal weight configurations, it is possible to create a dataset of many sample pairs of mesh statistics and the related optimal weight configurations and approximate the other optimal weight configurations, assuming a well-behaved relation between the mesh statistics and the optimal weight configurations. This work implements a mechanism that automatically determines a weight configuration per mesh based on its evaluated global surface statistics.

Background Theory 4

This chapter provides the necessary information and context to implement the enhancements presented in Chapter 3 and defines the terminology and techniques used later in the text.

4.1 Data Standardisation

Data standardisation [SHH96] is a technique widely used to transform original data into other data with desired properties. It may often be a necessary step in *data preprocessing*, as various applications and algorithms require standardised input data to operate or improve performance. This section mentions two well-known methods used to transform the data range.

Linear transformation is a method for transforming data $X \subset \mathbb{R}$ with values in the range $\langle \min(X), \max(X) \rangle$ into other data X' with values in the range $\langle 0, 1 \rangle$. The data transformation function $f : \mathbb{R} \to \langle 0, 1 \rangle$ for data $x \in X$ is defined as

$$f(x) = \frac{x - \min(X)}{\max(X) - \min(X)}. \tag{4.1}$$

Statistical standardisation is a method used to transform data $X \subset \mathbb{R}$ into other data X', which has a mean $\overline{X'} = 0$ and variance $\mathrm{var}(X') = 1$. The new data x'_i represents the multiple of the standard deviation $\mathrm{std}(X)$ the data x_i deviates from the mean $\overline{X}$. The data transformation function $f : \mathbb{R} \to \mathbb{R}$ for data $x \in X$ is defined as

$$f(x) = \frac{x - \overline{X}}{\mathrm{std}(X)}. \tag{4.2}$$

4.2 RBF Approximation

A *radial basis function* (RBF) [MS17] is a function $\phi : \langle 0, \infty) \to \mathbb{R}$. Generally, a radial basis function $\Phi : \mathbb{E}^\mathrm{d} \to \mathbb{R}$ is a function in d-dimensional Euclidean space centred around a point $\mathbf{c} \in \mathbb{E}^\mathrm{d}$. This function has a property that values $\Phi(\mathbf{x})$ depend only on the radial distance $r = \|\mathbf{c}-\mathbf{x}\|$ of the point $\mathbf{x} \in \mathbb{E}^d$ from the centre $\mathbf{c}$, as illustrated in Fig. 4.1. It is defined as

$$\Phi(\mathbf{x}) = \phi(r) = \phi(\|\mathbf{c} - \mathbf{x}\|). \tag{4.3}$$

Radial basis functions are popular in applications involving data interpolation and approximation, as they enable efficient work with higher-dimensional and scattered data. Let $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\} \subset \mathbb{E}^d$ be data of size $|X| = \mathrm{N}$. Then, the RBF interpolant f is formulated as

$$f(\mathbf{x}) = \sum_{i=1}^{N} \lambda_i \Phi_i(\mathbf{x}) = \sum_{i=1}^{N} \lambda_i \phi(\|\mathbf{c}_i - \mathbf{x}\|), \tag{4.4}$$

defining it as a linear combination of N radial basis functions Φ_i centred around the points $\mathbf{c}_i$, weighted by their respective weights λ_i. This leads to a system of linear equations, written as $\mathbf{A}\boldsymbol{\lambda} = \mathbf{y}$ in the matrix form, where $\mathbf{A}_{ij} = \Phi_j(\mathbf{x}_i) = \phi(\|\mathbf{c}_j - \mathbf{x}_i\|)$, $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_N)^T$ and $\mathbf{y} = (f(\mathbf{x}_1), f(\mathbf{x}_2), ..., f(\mathbf{x}_N))^T$.

However, depending, e.g., on the size, type, and quality of data X, it may be more appropriate to use *RBF approximation*, as it may be more effective or yield better overall results. Let $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\} \subset \mathbb{E}^d$ be data of size $|X| = \mathrm{N}$. Then, the RBF approximant f is formulated as

$$f(\mathbf{x}) = \sum_{i=1}^{M} \lambda_i \Phi_i(\mathbf{x}) = \sum_{i=1}^{M} \lambda_i \phi(\|\mathbf{c}_i - \mathbf{x}\|), \tag{4.5}$$

defining it as a linear combination of $\mathrm{M} \ll \mathrm{N}$ radial basis functions Φ_i centred around the points $\mathbf{c}_i$, weighted by their respective weights λ_i. This leads to an overdetermined system of linear equations, written as $\mathbf{A}\boldsymbol{\lambda} = \mathbf{y}$ in the matrix form, where $\mathbf{A}_{ij} = \Phi_j(\mathbf{x}_i) = \phi(\|\mathbf{c}_j - \mathbf{x}_i\|)$, $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_M)^T$ and $\mathbf{y} = (f(\mathbf{x}_1), f(\mathbf{x}_2), ..., f(\mathbf{x}_N))^T$. This system may be solved as $\mathbf{A}^T\mathbf{A}\boldsymbol{\lambda} = \mathbf{A}^T\mathbf{y}$, using the *least squares method*.

Choosing an appropriate *RBF kernel*, i.e. the radial basis function used in the interpolation or approximation, is essential to ensure desired results. Different RBF kernels have different properties, so the choice of RBF kernel depends on the specific application. The *thin plate spline* and Gaussian functions are well-known RBF kernels. The thin plate spline function, defined as $f(r) = r^2 \log r$, diverges with an increasing radius and has no free parameters that must be explicitly set. The Gaussian function, defined as $f(r) = e^{-\epsilon r^2}$, on the other hand, has a shape parameter ϵ that needs to be tuned to fit the problem well. It is infinitely differentiable and converges to zero with an increasing radius. Fig. 4.1 illustrates its behaviour for various shape parameters.

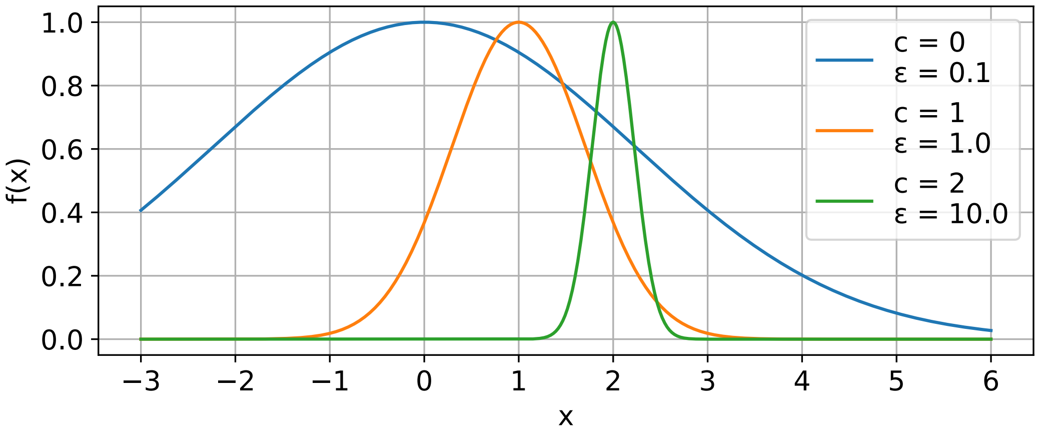

Figure 4.1: Gaussian RBF kernel

4.3 Artificial Neural Network

An *artificial neural network* [Dre05] is a computational model used in *machine learning* originating from the structure and function of biological neural networks. It comprises *layers* of interconnected nodes, the *neurons*, that process and transmit information. The model can learn to recognise complex patterns and make predictions based on input data.

Artificial neural networks vary in organisation, the simplest being a *fully connected feed-forward network*, in which every neuron in a layer processes its input and passes its output to all the neurons in the next layer. The network's input proceeds from the *input layer* through all the *hidden layers* to the *output layer* as the network's output, as illustrated in Fig. 4.2. The computation performed by the j-th neuron in

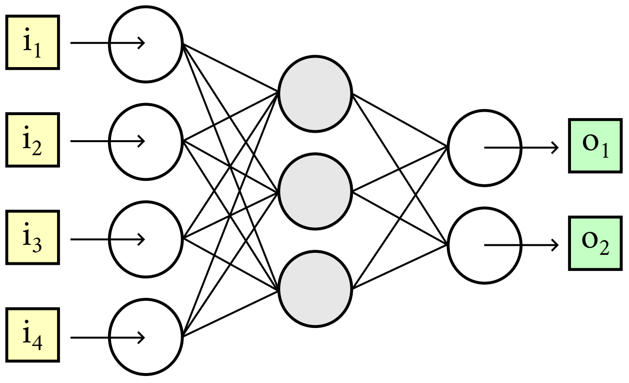

Figure 4.2: Artificial neural network

the i-th hidden layer is formulated as

$$\text{out}_{ij} = f_i\left(\mathbf{w}_{ij} \cdot \mathbf{out}_{i-1} + b_{ij}\right), \tag{4.6}$$

where out_{ij} is the neuron's output, $\mathbf{w}_{ij}$ is the vector of the *neuron's weights*, $\mathbf{out}_{i-1}$ is the vector of all neuron outputs of the previous layer, b_{ij} is the *neuron's output bias*, and f_i is the i-th layer's *activation function*.

The neuron determines its output by applying the activation function to the biased sum of its inputs, each multiplied by its respective weight. The neuron weights and utilised activation functions define the function modelled by the neural network. Artificial neural networks use specific activation functions to model general non-linear functions. Several are widely used, including hyperbolic tangent or *ReLU*, defined as $f(x) = \max\{0, x\}$.

Artificial neural network training is the iterative process of adjusting neuron weights to improve a model's performance. During training, the model predicts outputs given a set of inputs. The predictions are then compared to the target values to determine the error of predictions, defined by a *loss function*. The goal is to minimise the error. The *model optimiser* minimises the error by computing its gradient with respect to a neuron's weights, adjusting its weights by performing a *gradient descent*, and propagating the error backwards through the network using the *backpropagation algorithm*.

Filtering Candidates 5

This chapter discusses the enhanced candidate filtering mechanism. It also describes the implementation and provides experimental results and their analysis, comparing it to the original mechanism [MGS07], described in Section 2.3.3.

5.1 Definition

Generally, filtering candidate vertices leads to encoding smaller ranks with lower entropy and improving a gate's priority evaluation accuracy, resulting in exploiting more information during the priority-based traversal. Both these factors contribute to reducing the overall data rate.

The algorithm assumes an orientable manifold mesh on input. It divides the mesh into processed and unprocessed parts and starts to traverse the mesh. In every iteration, it extends the processed part of the mesh with a triangle from the unprocessed part. The processed part is always an orientable manifold. Thus, when considering extending the processed part from a gate, triangles that make the extended processed part unorientable or non-manifold cannot be considered candidate triangles, making their tip vertices inadmissible candidates for the gate.

Extending the processed part from the gate g with a triangle having an edge in common with the processed part, where it already has two incident triangles, forms a non-manifold edge in the extended processed part, as it increases this edge's triangle incidence count to three. On the other hand, extending the processed part with a triangle with an opposite orientation forms an unorientable surface in the extended processed part, as it violates the surface orientation. Fig. 5.1 illustrates these scenarios. The processed part is represented by the black triangles and the vertex c is inadmissible to extend the processed part with the blue triangle.

The original candidate filtering mechanism partially checks whether the already processed part, when extended with a candidate triangle, violates the assumption of being manifold. However, it only checks for the formation of a non-manifold vertex in the extended processed part. An enhanced mechanism filtering out additional

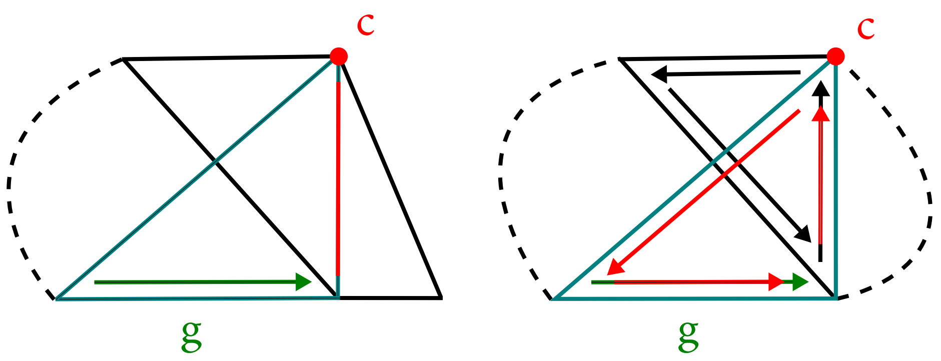

Figure 5.1: Inadmissible candidate vertices

candidate vertices is introduced to ensure the orientability and manifoldness of the extended processed part of the mesh.

5.2 Implementation

The elimination of candidates leading to the formation of a non-manifold edge in the extended processed part may be implemented by storing the edges of triangles in the processed part and keeping track of their triangle incidence count. Then, a candidate triangle is inadmissible if its left or right edge or both have a triangle incidence count of two in the processed part. Similarly, filtering out candidates that lead to the formation of an unorientable surface in the extended processed part may be implemented by storing the *half-edges* [Bot+10], i.e. oriented edges of triangles in the processed part. Then, a candidate triangle is inadmissible if its left or right half-edge or both exist in the processed part.

The mechanism that filters out the candidates leading to the formation of an unorientable surface in the extended processed part also excludes the candidates leading to the formation of a non-manifold edge. If a non-manifold edge shall be formed, there must already be an edge with a triangle incidence count of two in the processed part. The half-edge representation of this edge with two incident triangles corresponds to two half-edges with opposite orientations, as the processed part is orientable. Suppose a candidate triangle is considered inadmissible by the mechanism checking the formation of a non-manifold edge because its left or right edge already has a triangle incidence count of two in the processed part. In that case, it is also considered inadmissible by the mechanism checking the formation of an unorientable surface because the half-edges of both possible orientations corresponding to the particular edge already exist in the processed part.

The original candidate filtering mechanism considered only the tip vertex of the candidate triangle to determine its admissibility. However, to determine this enhanced candidate admissibility, the left and right edges of the candidate triangle are also considered. The overall algorithm remains the same as summarised in Algorithm 2.1 with the enhanced procedure `UpdateFilter`, updating the state of the candidate filtering mechanism and the function `Filter`, filtering out the inadmissible candidate vertices.

Procedure 5.1 describes the enhanced process of updating the mechanism's state. As the algorithm iteratively extends the processed part of the mesh with a triangle, it adds the half-edges forming the triangle to the set of half-edges of triangles in the processed part.

Procedure 5.1: Updating the state of the candidate filtering mechanism

```
   Input: Processed triangle or boundary edge
 1 if boundary edge was processed then
 2     ... reference implementation ...
 3     processed half-edges ⟵ boundary edge
 4 else
 5     ... reference implementation ...
 6     processed half-edges ⟵ triangle's half-edges
 7 end
```

Function 5.2 describes the enhanced process of filtering the candidate vertices. It evaluates each candidate for inadmissibility by checking whether any of the candidate triangle's left or right half-edges already exists in the set of half-edges of triangles in the processed part.

Function 5.2: Filtering out the inadmissible candidates

```
   Input: List of candidates, Candidate triangle's gate
   Output: List of admissible candidates
 1 inadmissible candidates ⟵ base triangle's vertices
 2 foreach candidate in candidates do
 3     ... reference implementation ...
 4     left half-edge ⟵ HalfEdge(candidate, V1(gate))
 5     right half-edge ⟵ HalfEdge(V2(gate), candidate)
 6     if left half-edge is in processed half-edges then
 7         inadmissible candidates ⟵ candidate
 8     else if right half-edge is in processed half-edges then
 9         inadmissible candidates ⟵ candidate
10     end
11 end
12 admissible candidates ⟵ complement of inadmissible candidates
13 return admissible candidates
```

It stores the half-edges of triangles belonging to the processed part of the mesh in a hash set to enable the constant-time addition and look-up operations required by Procedure 5.1 and Function 5.2. As a result, Procedure 5.1 requires additional constant time to update the mechanism, as it performs at most three addition operations. On the other hand, Function 5.2 performs at most two look-up operations for every candidate. Therefore, it requires additional linear time with respect to the number of filtered candidates. As discussed in Section 2.3, the algorithm queries a complete filtered list of candidates every time it determines the rank of the actual tip vertex or evaluates a gate's priority. The number of these candidates depends on the ranks the utilised candidate's quality metric function yields for the actual tip vertices for respective gates. Therefore, the additional time required by Function 5.2 equivalently depends on the mesh type and the appropriateness of the utilised weight configuration. Assuming that the weight configuration is well-suited for a particular mesh, the additional time required to filter the candidates is constant with respect to the mesh size, as the number of candidates can be bounded by a small constant independent of the mesh vertex count.

5.3 Experimental Results

The impact of the enhanced candidate filtering mechanism was assessed by measuring the data rate and encoding and decoding time across all the datasets used in [Dvo+22], described in Section 2.3.4. The experiment was performed on the *tosca*, *mcgill*, and *casual_man* datasets without any changes. However, the *abc_regular*, *abc_irregular*, and *thingi10k* datasets are represented by different random mesh samples with triangle counts ranging from 1000 to 20000. Table 5.1 presents the results, comparing the mean per-mesh change in data rate and encoding and decoding time to the original candidate filtering mechanism.

Table 5.1: Enhanced candidate filtering mechanism performance summary

		mean per-mesh change [%]					
		$\mathbf{w}_{\text{default}}$			$\mathbf{w}_{\text{dataset}}$		
dataset	**# meshes**	**bpf**	$\mathbf{t}_\text{e}$	$\mathbf{t}_\text{d}$	**bpf**	$\mathbf{t}_\text{e}$	$\mathbf{t}_\text{d}$
abc_regular	1000	-0.68	37.0	38.9	-0.75	40.6	40.0
abc_irregular	1000	-0.79	51.9	49.5	-0.68	52.4	48.2
thingi10k	1000	-1.13	44.8	44.7	-1.01	49.3	46.7
tosca	80	-0.49	49.0	45.9	-0.99	42.0	39.4
mcgill	458	-0.81	37.3	38.1	-0.70	36.4	35.7
casual_man	546	-0.55	39.4	39.4	-0.41	44.8	44.4

Depending on the weight configuration, utilising the enhanced candidate filtering mechanism decreases the data rate by 0.41 to 1.13 per cent per-mesh on average on the experimental datasets. It does not provide a significant improvement, especially considering that the computational overhead of the enhanced candidate filtering mechanism costs an additional 35.7 to 52.4 per cent of the encoding and decoding time.

5.4 Validation

Another experiment was performed on one specific mesh to validate the results. This mesh, the *delaunay_plane*, comprises one million vertices randomly uniformly distributed on a plane restricted by a rectangle. The Delaunay triangulation [Bot+10] defines the connectivity for this geometry. The experiment evaluated the mechanism's performance for different weight configurations that determine the candidate's quality metric function, as discussed in Section 2.3.1. This experiment should provide nearly error-unbiased results, as it amortises the randomness over almost two million triangles.

The experiment considers four different weight configurations. The first tested weight configuration maximises the quality of a candidate triangle based on the inner angle at its tip vertex, minimises the influence of the distance from the prediction, and omits other quality metric components. It is considered optimal for this particular mesh as the Delaunay triangulation favours large inner angles. The other weight configurations gradually introduce errors to the respective candidate's quality metric functions by considering other quality metric components. The second weight configuration considers the distance from the prediction, and the third additionally considers the triangle similarity, giving them all equal weights. The last tested weight configuration is the default weight configuration discussed in Section 2.3.4. The weight configurations are introduced in the order of their expected performance to suit the quality metric function sufficiently well for this particular mesh, as each next introduces some error by considering other components, and the last additionally introduces the weighting of the components. However, the Delaunay triangulation optimises the inner angles but does not directly consider other properties of the triangles.

Table 5.2 summarises the experimental results. Depending on the appropriateness of the weight configuration, the enhanced candidate filtering mechanism improves the data rate by 2.5 to 19.2 per cent over the original candidate filtering mechanism. Assuming that the observed relation between the sufficiently well-suited weight configuration for a particular mesh and the data rate improvement the enhanced candidate filtering mechanism provides applies in general, then the data rate improvement is more significant the more sufficiently well the weight configuration is suited for a particular mesh.

Table 5.2: Enhanced candidate filtering mechanism performance validation

	change [%]		
w	**bpf**	$\mathbf{t}_e$	$\mathbf{t}_d$
optimal	-19.18	57.1	79.1
$w_1 = 1$	-9.21	35.2	35.1
$w_{1,3} = 1$	-6.18	38.7	43.4
default	-2.54	69.1	66.9

Updating Priorities 6

This chapter discusses the enhancement of the algorithm based on continuously updating the priorities of unprocessed gates. It also describes the implementation and provides experimental results and their analysis.

6.1 Definition

The algorithm benefits from the priority-driven mesh traversal, as described in Section 2.3.2. During the traversal, it forms new gates every time it extends the processed part of the mesh with a triangle. It evaluates their priorities and adds them to the priority queue. The priority of a gate remains unchanged for its existence in the priority queue and is bound to the state of the unprocessed part of the mesh at the moment when it was evaluated. However, the state of the local area, which determines the priority, may change as the traversal proceeds, and so may the priority. Keeping the priorities of the unprocessed gates synchronised with the changes in the unprocessed part provides additional information the algorithm may utilise to improve the mesh traversal, eventually improving the data rate.

The algorithm utilises the approximated gate priorities defined by Eq. (2.6) and discussed in Section 2.3.2. A gate's priority may change only at the moment when its best or second-best candidate or both cease to be candidate vertices. A candidate vertex ceases to be a candidate when the algorithm extends the processed part of the mesh with a triangle, updating the candidate filtering mechanism so that the candidate vertex becomes inadmissible for the particular gate. As discussed in Section 2.3.3 and Chapter 5, a candidate vertex is inadmissible if all the triangles of the fan formed around it belong to the processed part of the mesh, defining it to be a *closed vertex*, or if any of the left or right half-edges of the candidate triangle already exists in the processed part of the mesh.

Suppose a gate g and its best and second-best candidate vertices c_1 and c_2. Extending the processed part of the mesh from the gate g' with a triangle that completes a fan formed around the best or the second-best candidate vertex with all the triangles belonging to the processed part of the mesh, closing the particular vertex, updates the candidate filtering mechanism so that the particular candidate vertex becomes inadmissible for the gate g. Also, extending the processed part with a triangle having a half-edge in common with the best or the second-best candidate triangle results in the particular candidate vertex becoming inadmissible for the gate g. Fig. 6.1 illustrates these scenarios.

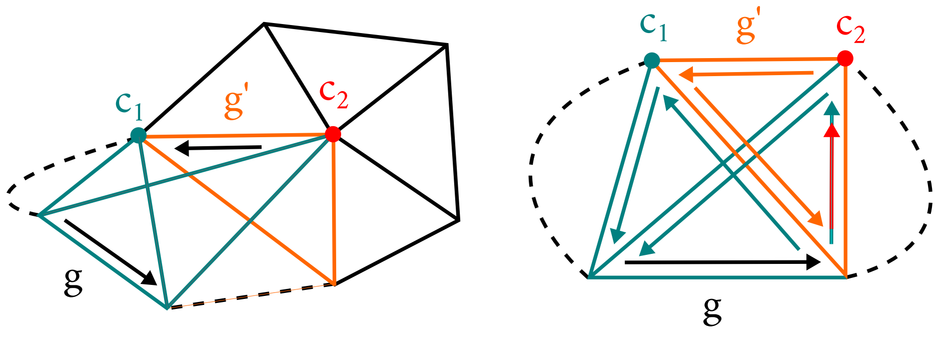

Figure 6.1: Guarding gate's relevance for a priority update

A new mechanism is introduced to ensure the priorities of unprocessed gates reflect the actual state of the unprocessed part of the mesh. It mimics the logic of the candidate filtering mechanism and, based on that, updates the priorities of relevant gates. A gate is relevant for a priority update if any of its best or second-best candidate vertices ceases to be a candidate.

6.2 Implementation

The mechanism may be implemented by storing the gates and associating them with their best and second-best candidate vertices and their best and second-best candidate triangle's left and right half-edges. It cooperates with the candidate filtering mechanism, which stores the vertices closed and the half-edges added to the processed part of the mesh in the last iteration of the algorithm. Then, a gate is relevant for a priority update if it has a stored association with any of the vertices closed or any of the half-edges added to the processed part in the last iteration of the algorithm.

As many gates may be associated with a particular vertex or half-edge, the gates associated with one vertex or half-edge are stored in a hash set. In the case of vertex associations, a set of associated gates to a vertex is stored on the particular vertex index in an array. In the case of half-edge associations, a set of associated gates to a half-edge is stored in a dictionary.

Algorithm 6.1 illustrates the algorithm enhanced by the continuous priority updating mechanism. It introduces two new processes that enable its functionality. First, when the algorithm forms new gates, it sets up a guard that monitors their relevance for a priority update, handled by the procedure `Guard`. Then, when the algorithm extends the processed part of the mesh with a triangle or processes a boundary edge, it queries the gates whose priority it affected, handled by the function `QueryRelevantGates`, and reevaluates their priorities.

Algorithm 6.1: Continuous priority updating of unprocessed gates

```
Input: Mesh, Initial triangle, Weight configuration
Output: List of candidate indices
1  Initialise data structures
2  UpdateFilter(initial triangle)
3  priority queue ⟵ initial triangle's edges
4  Guard(...initial triangle's edges)
5  while priority queue is not empty do
6      priority queue ⟶ gate
7      if gate is completed then continue
8      if gate is boundary then
9          candidate indices ⟵ EncodeBoundary(gate)
10         UpdateFilter(gate)
11     else
12         candidates ⟵ QueryCandidates(gate, Quality(tip vertex))
13         candidates ⟵ Filter(candidates)
14         candidate indices ⟵ EncodeIndex(tip vertex, candidates)
15         UpdateFilter(triangle formed by gate and tip vertex)
16         priority queue ⟵ newly formed left and right gates
17         Guard(...newly formed left and right gates)
18     end
19     relevant gates ⟵ QueryRelevantGates()
20     foreach gate in relevant gates do
21         Unguard(gate)
22         if gate is completed then continue
23         priority queue ⟵ gate
24         Guard(gate)
25     end
26     mark gate as completed
27 end
```

Procedure 6.2 illustrates setting up a guard for a gate's priority update. It associates the gate with its best and second-best candidate vertices and the left and right half-edges of its best and second-best candidate triangle.

Function 6.3 illustrates the process of listing the gates relevant for a priority update. It utilises the vertices closed and the half-edges processed in the last iteration of the algorithm to determine the gates whose priority the last iteration affected.

Procedure 6.2: Setting up a guard for a gate's priority update

```
Input: Gate
1 Associate gate ⟶ best candidate
2 Associate gate ⟶ second-best candidate
3 Associate gate ⟶ HalfEdge(best candidate, V1(gate))
4 Associate gate ⟶ HalfEdge(V2(gate), best candidate)
5 Associate gate ⟶ HalfEdge(second-best candidate, V1(gate))
6 Associate gate ⟶ HalfEdge(V2(gate), second-best candidate)
```

Function 6.3: Querying gates relevant for a priority update

```
Output: List of gates relevant for a priority update
1 foreach vertex in latest closed vertices do
2     relevant gates ⟵ QueryAssociatedGates(vertex)
3 end
4 foreach half-edge in latest processed half-edges do
5     relevant gates ⟵ QueryAssociatedGates(half-edge)
6 end
7 return relevant gates
```

After a triangle extends the processed part of the mesh or a boundary edge is processed, the algorithm queries relevant gates and reevaluates their priorities. The gate's priority update process begins with removing the set-up associations between the gate and its former best and second-best candidate vertices and the best and second-best candidate triangle's left and right half-edges, handled by the `Unguard` procedure, ensuring the mechanism cannot later falsely assess the gate as relevant for a priority update. Then, if the gate is not marked as completed, it evaluates its priority and puts it into the priority queue. Finally, it sets up a guard for its priority update again.

6.3 Experimental Results

The impact of continuous priority updates of unprocessed gates was experimentally evaluated by measuring the data rate and encoding and decoding time. The experiment used the same datasets as in Section 5.3, and the algorithm utilised the enhanced candidate filtering mechanism discussed in Chapter 5. Table 6.1 presents the results by comparing the mean per-mesh change in data rate and encoding and decoding time when the algorithm utilises the continuous priority updating mechanism to the results obtained when the algorithm does not utilise this mechanism.

The results show that the experimental datasets do not benefit from utilising

Table 6.1: Continuous priority updating mechanism performance summary

		mean per-mesh change [%]					
		$\mathbf{w}_{\text{default}}$			$\mathbf{w}_{\text{dataset}}$		
dataset	**# meshes**	**bpf**	$\mathbf{t}_{\text{e}}$	$\mathbf{t}_{\text{d}}$	**bpf**	$\mathbf{t}_{\text{e}}$	$\mathbf{t}_{\text{d}}$
abc_regular	1000	1.90	104.9	102.1	1.53	154.4	143.7
abc_irregular	1000	7.84	94.7	80.0	9.18	87.3	68.5
thingi10k	1000	3.32	67.3	61.3	3.53	77.9	66.7
tosca	80	0.11	117.4	95.5	0.15	77.4	59.3
mcgill	458	0.19	112.5	107.0	-0.17	137.5	129.3
casual_man	546	-0.13	124.5	120.1	0.15	253.9	246.9

this mechanism for the default and the dataset-optimised weight configurations. It leads to a significant processing time increase, ranging from 59.3 to 253.9 per cent, and, more importantly, it actually worsens the data rate, as the improvement ranges from negative 9.18 per cent to 0.17 per cent. The mechanism likely performs better, but still rather worse than if it is not utilised, on datasets containing meshes with smoother surfaces and more regular triangulation, e.g., the *tosca*, *mcgill*, or *casual_man* datasets. On the other hand, it performs worse on datasets containing CAD models with sharply curved surfaces and more sliver triangles, e.g., the *abc_regular*, *abc_irregular* and *thingi10k* datasets.

6.4 Validation

A validation experiment was performed on a subset of the experimental datasets comprising ten meshes with at most 5000 triangles to disprove a mistake in the implementation, as this data rate worsening behaviour was not expected. It compared the implemented mechanism to one that updates the priority of every unprocessed gate in every iteration of the algorithm. It checked whether the priorities of unprocessed gates matched the ones determined by the other mechanism in every iteration. This dataset-restricted experiment indicates that they do, and the mechanism works as expected.

Another experiment was performed on the *delaunay_plane* mesh. It is the same experiment as in Section 5.4, but it evaluates the performance of the continuous priority updating mechanism. Table 6.2 summarises the experimental results. Utilising the mechanism improves the data rate by 1.5 percent when the algorithm uses the optimal weight configuration. However, the improvement decreases for less appropriate weight configurations, eventually resulting in a worse data rate by 0.2 percent for the default weight configuration. Assuming that this observation applies in general, then the mechanism improves the data rate for well-suited weight con-

Table 6.2: Continuous priority updating mechanism performance validation

	change [%]		
w	**bpf**	$\mathbf{t_e}$	$\mathbf{t_d}$
optimal	-1.50	170.2	103.2
$w_1 = 1$	-1.25	63.1	55.9
$w_{1,3} = 1$	-0.16	59.2	48.8
default	0.39	80.6	73.0

figurations, and the improvement decreases and eventually becomes negative for less appropriate weight configurations.

Several factors may contribute to the data rate increasing behaviour. The mesh traversal utilises the priority as defined by Eq. (2.6) in Section 2.3.2. However, it only approximates the property the priority represents, resulting in an approximation error. As defined, the priority also depends on the accuracy of the utilised candidate's quality metric function, which introduces additional errors to the utilised approximated priority if it is not well-suited. Another factor that needs to be considered is how Dvořák et al. obtained the default and the dataset-optimised weight configurations, as discussed in Section 2.3.4. They only provide a globally optimised data rate for the small sample of selected meshes for which they were optimised. However, more importantly, they were optimised without considering continuous updating of the priorities of unprocessed gates, which may later introduce additional error, as the weight configurations were optimised to provide only the initial heuristic estimate of the priority and considered less the actual candidate's quality metric function. Then, the weight configuration is not sufficiently well-suited for a particular mesh and does not determine the priority well when later utilising the continuous priority updating mechanism. Therefore, the traversal yields ranks with higher entropy, which the utilised CABAC entropy coder adapts to, eventually resulting in the increased data rate.

Determining Weights 7

This chapter proposes an enhancement based on estimating the optimal weight configuration per mesh, i.e. the mesh-optimal weight configuration, utilising the knowledge of its global surface statistics. It also describes the implementation and provides experimental results and their analysis, comparing it to the default and the dataset-optimised weight configurations.

7.1 Definition

Choosing an appropriate weight configuration for a mesh is a critical step in the compression process, as it defines the utilised candidate's quality metric function, defined by Eq. (2.1) in Section 2.3.1. Also, the utilised approximated priority derives from the best and the second-best candidate qualities, as discussed in Section 2.3.2. The algorithm's resulting data rate benefits from the priority-driven mesh traversal, assuming that the utilised candidate's quality metric function consistently ranks the candidates appropriately and the utilised priority well approximates the probability that the gate can distinguish the actual tip vertex well in the list of candidates.

Assuming that the meshes from the same dataset share common properties, Dvořák et al. determined the optimised weight configuration per dataset, as discussed in Section 2.3.4. However, the dataset-optimised weight configuration only provides a globally optimised data rate for the small subset of the dataset. Thus, it generally cannot suit the individual meshes sufficiently, as every dataset comprises varying meshes, even though they may share some relevant properties. This chapter introduces a new mechanism for estimating the mesh-optimal weight configurations independent of the dataset to which the mesh belongs.

The mechanism may estimate the optimal weight configurations for meshes based on a pre-computed dataset comprising sample complete descriptive statistics of meshes and the related optimal weight configurations, utilising the *well-behaved relation* between them, as discussed in Chapter 3. A well-behaved relation is a function that behaves predictably and smoothly at every local point of its domain and has some convenient properties. It is continuous and differentiable at every point.

Additionally, it is monotonic and, therefore, can be well-utilised to find its optimum unambiguously by iteratively performing gradient descent.

Since obtaining the complete descriptive statistics of a mesh is generally impossible, the mechanism utilises only a subset of selected descriptive properties of a mesh. Utilising incomplete descriptive mesh statistics introduces an approximation error and violates the assumption of a well-behaved relation between the mesh statistics and the related optimal weight configurations. The relation cannot be injective since many meshes may possess the same incomplete descriptive statistics but differ in the optimal weight configurations.

The implemented mechanism approaches the problem more comprehensively than the one mentioned in Chapter 3, computing the global surface statistics of a mesh and determining the respective approximant of the data rate function instead of only considering the optimal weight configuration. It compresses the mesh using various weight configurations to obtain the samples for approximation. It builds a dataset of many sample pairs of global surface statistics and respective data rate approximants and utilises an artificial neural network to model the relation. It trains the model to predict weight configurations that minimise the data rate, utilising the approximate data rate provided by the approximants.

7.2 Implementation

Assuming the relation between the incomplete descriptive mesh statistics and the optimal weight configurations is complex and difficult to represent, the mechanism uses an artificial neural network to model it, as discussed in Section 7.2.4. The algorithm's data rate for a mesh is a function of three parameters defining the candidate's quality metric function. Considering the diversity of meshes, the corresponding functions of the data rate are also diverse since different weight configurations suit different meshes optimally, and different changes in particular weights influence the data rate differently. Considering this diverse data rate behaviour and the insufficient description of a mesh with incomplete global surface statistics, the goal is not to predict the optimal weight configuration, focusing on minimising the error of the predicted weights, as it is too constraining and, more importantly, impossible to achieve, since the modelled relation is not well-behaved. The goal is to predict the weight configuration, focusing on minimising the data rate, which relaxes the modelled relation and makes the problem tractable.

The mechanism samples and approximates the data rate function, as discussed in Section 7.2.2, respectively Section 7.2.3, to enable the model to utilise the knowledge of the data rate for various weight configurations. The following sections describe the process of building a dataset of sample pairs of global surface statistics of meshes and the respective approximated functions of the data rate, as discussed in Section 7.2.3, which is then used to train the model to estimate the mesh-optimal weight configurations, which is discussed in Section 7.2.4. The dataset comprises meshes from the datasets described in Section 2.3.4, utilised in [Dvo+22] to assess the performance of the reference implementation.

7.2.1 Surface Statistics Computation

The mechanism considers seven mesh properties to determine its global surface statistics. These are:

1. triangle inner angles,
2. distances from the parallelogram prediction,
3. dihedral angles between adjacent triangles,
4. similarity of adjacent triangles,
5. triangle equilaterality,
6. edge lengths and
7. vertex degrees.

The first four properties directly relate to the components of the candidate's quality metric function. The other three may provide additional information about the mesh.

The implemented mechanism determines the inner angles of triangles and uses them to compute the absolute value of its deviation from sixty degrees, which is the inner angle of an equilateral triangle. It also determines the distance of a triangle's tip vertex from the parallelogram prediction constructed from every half-edge as if it were a gate. Also, it determines the dihedral angle between every pair of adjacent triangles. It also determines the similarity of every pair of adjacent triangles. Also, it determines the measure of equilaterality of triangles, defined as the distance of the tip vertex of a unit equilateral triangle to the tip vertex of a triangle scaled so that its base has a unit length and is aligned with the base of the unit equilateral triangle, considering each triangle's edge as a base. Finally, it determines the lengths of edges and degrees of vertices.

The algorithm traverses the mesh and measures the data while marking the processed vertices, edges and triangles, ensuring that data is collected only once. It utilises the *corner table* [Ros01] data structure to detect a boundary, query adjacent triangles and move around the mesh. Measuring all the data, it normalises the distances by the average edge length to ensure mesh scale invariance and computes the twelve resulting statistics. These are the mean values and the standard deviations of the first five values and the standard deviations of the other two. It does not consider the mean values of the lengths of edges or the vertex degrees since they do not provide any information.

7.2.2 Weight Configuration Optimisation

The data rate is a function $f_M : \mathbb{R}^3_{\geq 0} \rightarrow \mathbb{R}_+$ of a mesh M and a weight configuration $\mathbf{w} \in \mathbb{R}^3_{\geq 0}$. Since this function cannot be expressed analytically and its behaviour is generally unknown, the optimal weight configuration can be found by searching for it exhaustively over the range of feasible values. Considering the computational cost of the compression and the unlimited range of feasible weight configurations, limiting the search range and the number of measured samples is necessary.

The mechanism proceeds by defining an initial grid of feasible weight configurations and measuring the data rate for each of them. Iteratively, the neighbourhood of the weight configuration resulting in minimal data rate defines a local grid that the mechanism evaluates again. The mechanism converges to finding the optimal weight configuration, assuming the data rate function is well-behaved, or the grid is fine enough not to get stuck at a local minimum.

The mechanism utilises a limited range of feasible weight configurations derived from the dataset-optimised weight configurations, assuming the mesh-optimal weight configurations do not differ significantly from the dataset-optimised ones. The utilised space of feasible weight configurations is defined as

$$\bigtimes_{i=1,2,3} \left\langle 2/3 \operatorname*{argmin}_{D \in \text{datasets}} w_i^D, 3/2 \operatorname*{argmax}_{D \in \text{datasets}} w_i^D \right\rangle,$$

bounded by the constant multiples of the minimal and the maximal weights of the dataset-optimised weight configurations. The symbol w_i^D represents the i-th weight of the dataset-optimised weight configuration for the dataset D.

The mechanism samples the data rate using local equidistant grids for simplicity but in a *transformed weight configuration space*. A *weight configuration transformation function* transforms the points of this space to the respective weight configurations. The utilised weight configuration transformation function is derived from the dataset-optimised weight configurations and is set to map the space $\langle 0, 1 \rangle^3$ to the utilised space of feasible weight configurations, $\langle 0, 1 \rangle^3$ representing the transformed utilised space of feasible weight configurations. It is specifically tuned to approximately match the expected distribution of the mesh-optimal weight configurations, assuming the mesh-optimal weight configurations lie close to the dataset-optimised weight configurations. Satisfying this assumption leads to sampling the data rate function for the weight configurations lying close to the mesh-optimal ones, requiring fewer samples in total and speeding up the optimisation process.

Algorithm 7.1 summarises the process of searching for the optimal weight configuration for a mesh alongside sampling the data rate function. The mechanism

Algorithm 7.1: Weight configuration optimisation

```
Input: Mesh
Output: Optimal weight configuration, Data rate function samples
1  initial grid ← Partition(space of feasible weight configurations)
2  samples ← Evaluate(initial grid)
3  do
4      if improvement search depth is exceeded then break
5      neighbourhood ← Neighbourhood(best sample)
6      grid ← Partition(neighbourhood)
7      samples ← Evaluate(grid)
8  while best sample improves
9  while best sample lies on the border do
10     if extended search depth is exceeded then fail
11     neighbourhood ← Neighbourhood(best sample)
12     grid ← Partition(neighbourhood)
13     samples ← Evaluate(grid)
14 end
```

additionally requires a pre-defined maximum search depth, grid partitioning and weight configuration transformation function. It starts with partitioning the transformed space $\langle 0, 1 \rangle^3$ of feasible weight configurations, forming the initial grid. Then, it evaluates the data rates for the points of the initial grid by transforming them to the respective weight configurations, utilising the weight configuration transformation function and compressing the mesh with these weight configurations. Afterwards, it iteratively refines the local grid formed in the neighbourhood of the best-performing point of the grid. It stops this local refinement process when the best data rate stops improving, or the algorithm exceeds the maximum search depth to improve the data rate. Then, if the best-performing point of the grid lies on its border, it is likely that a better-performing point exists outside the already sampled grid. Therefore, it iteratively further attempts to search for it, repeating the refinement process. The optimisation process ends successfully if the best-performing point of the grid lies inside the grid before exceeding the maximum search depth of the extended grid search, and the best-performing weight configuration is declared to be the mesh-optimal weight configuration. Otherwise, it ends with a failure since the process failed to find the best-performing weight configuration even with the extended grid search. Fig. 7.1 visualises the sampled data rate function of an example mesh.

The mechanism is implemented to perform at most three grid refinements to improve the data rate and at most five local grid searches in total. It partitions

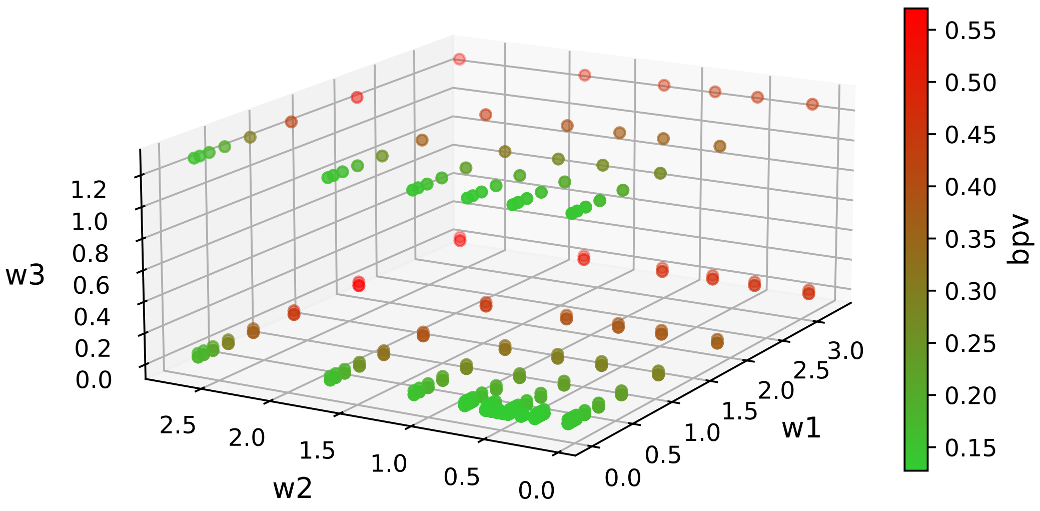

Figure 7.1: Sampled data rate function

the initial space into eight, six, and four partitions in the axes corresponding to the transformed weights w_1, w_2, and w_3. The partitions are defined in this ratio based on the assumed impact of a particular weight on the data rate, trying not to introduce an error leading to the mechanism getting stuck at a local minimum since the global minimum cannot be recognised due to a too coarse grid. It defines the neighbourhood to be centred around the best-performing grid point and to be one local grid partition wide per axis. It partitions this local space into five partitions per axis. In case there are many best-performing grid points, it prefers selecting the one lying on the border, if there are any, and selecting the one positioned in their centre. The mechanism stores the processed points of the grid in order not to evaluate the data rate for the already processed points of the refined grids when they overlap. The mechanism evaluates the data rate for the grid points in parallel, utilising maximum computational power.

7.2.3 Data Rate Approximation

As obtained by Algorithm 7.1 in Section 7.2.2, the sampled data rate function is subject to approximation since it enables estimating the data rate for the weight configurations for which the function was not evaluated, assuming a well-behaved relation between the weight configurations and the resulting data rate. Also, it is computationally effective since it enables using a small number of costly evaluations of the data rate by compressing the mesh and a much larger number of much less costly approximations of the data rate by evaluating the approximant.

The RBF approximation, described in Section 4.2, is used to approximate the data rate function. Before determining the approximant, the utilised RBF kernel must be considered, and the RBF centres must be selected. The approximant's accuracy depends on this selection and the choice of explicit RBF shape parameters, if there are any.

As discussed in Section 7.2.2 and visualised in Fig. 7.1, the weight configuration optimisation process partitions the local grids of the transformed weight configuration space equidistantly, respectively the local grids of the weight configuration space with a different distribution for every axis, determined by the weight configuration transformation function. Therefore, the approximant operates in the transformed weight configuration space to capture the behaviour of the data rate function well and not to depend on the scale of the axes.

The implemented approximant comprises 50 radial basis functions and respective centres, which are the selected weight configurations for which the data rate is known, assuming that a sum of this many functions can sufficiently well capture the behaviour of the approximated data rate function. The centre selection mechanism selects the optimal weight configuration as the first centre. Then, it iteratively selects the remaining centres. In every iteration, it selects the weight configuration that is the farthest, in terms of the Euclidean distance in the transformed weight configuration space, from all the already selected weight configurations, leading to a convenient distribution of the centres across the space. Fig. 7.2 visualises the selected weight configurations for the centres of the RBF approximant of the same sampled data rate function visualised in Fig. 7.1.

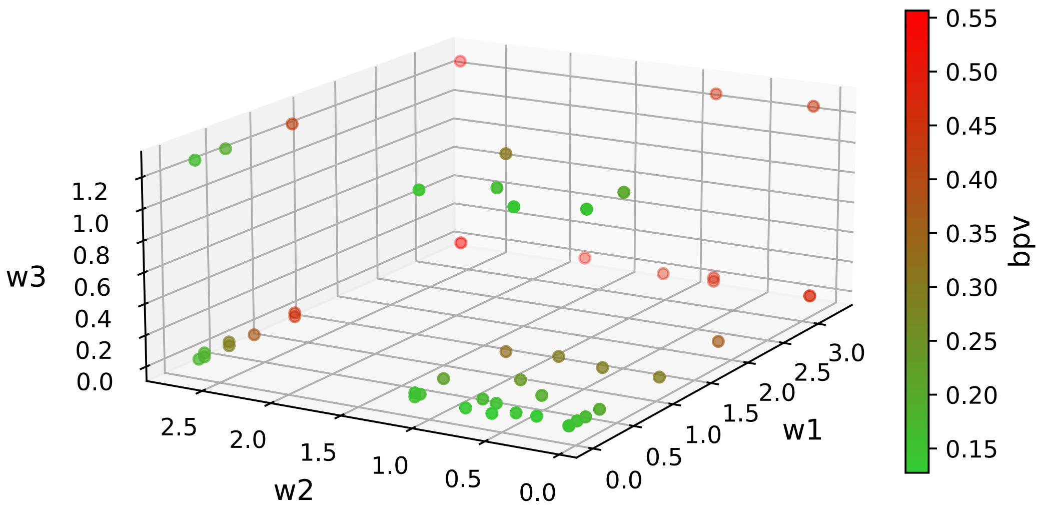

Figure 7.2: Selected RBF centres

The implementation considered the thin plate spline RBF kernel for its simple application, as it does not require tuning any explicit shape parameter. However, it diverges with an increasing radius. Therefore, the approximant diverges outside the space defined by the selected centres, extrapolating the data rate. However, extrapolating the data rate may help train the model since some mesh-optimal weight configurations can lie outside the initial grid of feasible weight configurations and can be found during the extended grid search, as discussed in Section 7.2.2. Therefore, the approximant utilises the Gaussian RBF kernel, which converges to zero as the radius increases. It utilises this property to define a limit value the approximant converges to, distancing from the space defined by the selected centres and extrapolating the data rate. Fig. 7.3 illustrates this behaviour on a simplified one-dimensional example. It shows that the value estimated by the approximant is the weighted sum of the Gaussian RBF kernels subtracted from the limit value. The kernels are visualised so that their sum models the estimated values into the level specified by the limit value.

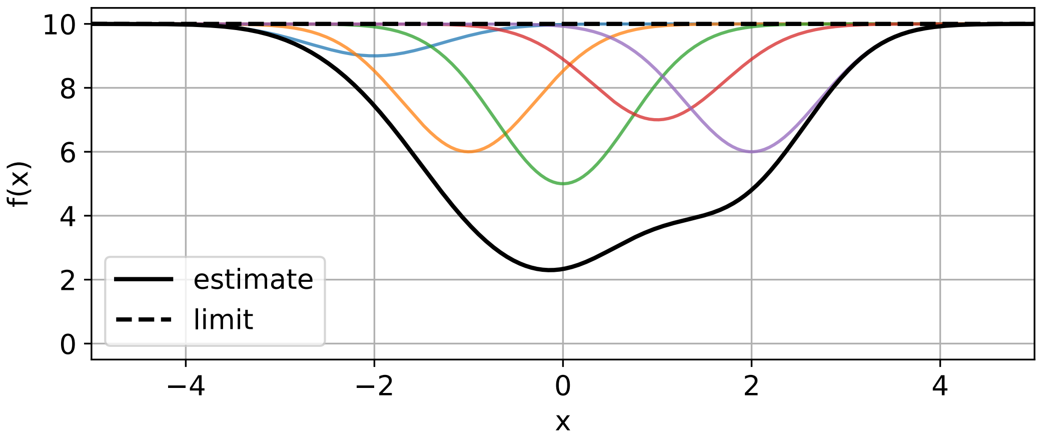

Figure 7.3: Convergence to the limit value

The approximant is obtained by solving the overdetermined system $\mathbf{A}\boldsymbol{\lambda} = \mathbf{y}$ of linear equations as $\mathbf{A}^{\mathrm{T}}\mathbf{A}\boldsymbol{\lambda} = \mathbf{A}^{\mathrm{T}}\mathbf{y}$, using the least squares method, as discussed in Section 4.2. The matrix $\mathbf{A}$ is of size $M \times N$, where M and N are the number of the selected centres and the number of all the data rate function samples, and is defined as $\mathbf{A}_{ij} = \phi_\epsilon(\|\mathbf{c}_j - \mathbf{w}_i\|^{\mathrm{t}})$, where ϕ_ϵ is the Gaussian RBF kernel with the shape parameter ϵ, $\mathbf{c}_j$ is the j-th centre, $\mathbf{w}_i$ is the i-th weight configuration and $\|.\|^{\mathrm{t}}$ is the Euclidean distance in the space of transformed weight configurations. The vector $\mathbf{y} = (y_1, y_2, ..., y_N)^{\mathrm{T}}$ is a vector of the transformed data rate for the respective weight configuration, defined as $y_i = d_{\mathrm{lim}} - d_{\mathbf{w}_i}$, where $d_{\mathrm{lim}} = 4d_{\mathrm{max}}$ is the data rate limit value, $d_{\mathbf{w}_i}$ is the data rate for the i-th weight configuration and d_{max} is the maximum sampled data rate. The vector $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_M)^{\mathrm{T}}$ is a vector of the weight of the respective RBF in the linear combination defining the approximant. Then, the approximated data rate $d'_{\mathbf{w}}$ for a weight configuration $\mathbf{w}$ is evaluated as $d'_{\mathbf{w}} = d_{\mathrm{lim}} - \sum_{i=1}^{M} \lambda_i \phi(\|\mathbf{c}_i - \mathbf{x}\|^{\mathrm{t}})$. Transforming the data rate when defining the approximant and then when evaluating the approximant ensures that the approximant yields the expected data rate inside the space defined by the selected centres and that the extrapolated data rate converges to the data rate limit value, as illustrated in Fig. 7.3.

Since the implemented approximant utilises the Gaussian RBF kernel, it is required to set its shape parameter ϵ. It is tuned to optimise the sum of squares of the differences between the actual and the approximated data rate for the weight configurations appearing in the system of linear equations, which is solved to define the approximant. This implementation determines it by solving this system for a range of feasible values and selecting the best-performing one. Fig. 7.4 visualises the relation between the shape parameter ϵ and the sum of squares of the residuals, i.e. the approximation error for three randomly selected meshes of the utilised dataset.

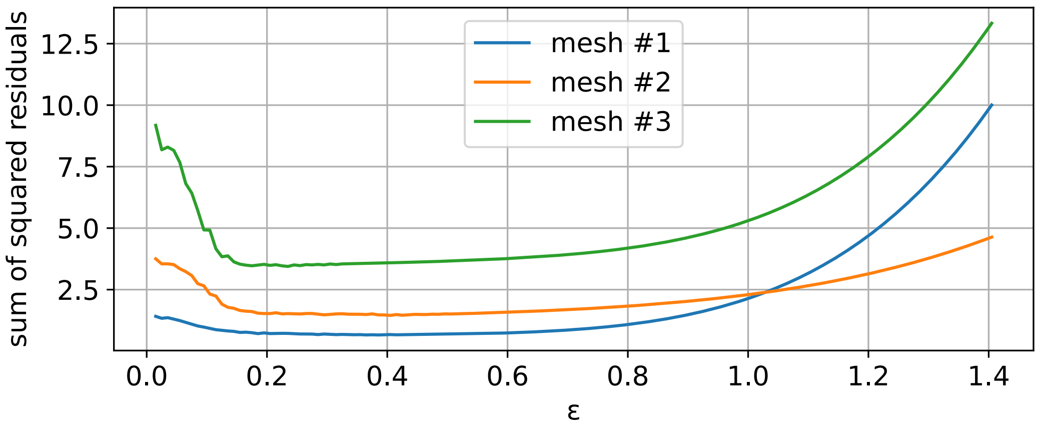

Figure 7.4: Relation between the Gaussian shape parameter and error

The data rate function may be too complex to be sufficiently well approximated by the approximant mentioned earlier, impairing the overall accuracy, for example, when the data rate function contains an outlier to which the approximant cannot adapt sufficiently or the data rate function behaviour is too volatile to be captured well by the limited number of utilised radial basis functions. Therefore, the approximant implements a measure to mitigate this problem. It does so by weighting the squared residuals of the least squares solution. It sets the weights so that it prefers minimising the residuals for weight configurations that result in a better data rate over the residuals for weight configurations that result in a worse data rate, and it does not consider the residuals for the weight configurations resulting in an outlier data rate at all. This process leads to smoothing the approximation curve of the data rate function. However, it sufficiently preserves the data rate function behaviour around the optimal weight configuration. The implemented approximant is then obtained by solving the overdetermined system $\mathbf{A}\boldsymbol{\lambda} = \mathbf{y}$ of linear equations as $(\mathbf{WA})^{\mathrm{T}}\mathbf{WA}\boldsymbol{\lambda} = (\mathbf{WA})^{\mathrm{T}}\mathbf{Wy}$, using the weighted least squares method, where $\mathbf{W}$ is a diagonal matrix of the square roots of the weights of respective squared residuals.

The implementation sets

$$\mathbf{W}_{ii} = \sqrt{1 - \frac{d_{\mathbf{w}_i} - d_{\text{opt}}}{d_{\text{t}} - d_{\text{opt}}}}$$

for the respective data rate $d_{\mathbf{w}_i} \in \langle d_{\text{opt}}, d_{\text{t}} \rangle$, where d_{opt} is the mesh-optimal data rate and $d_{\text{t}} = 10 d_{\text{opt}}$ is the outlier-threshold data rate. Otherwise, the respective data rate is considered an outlier and $\mathbf{W}_{ii} = 0$. Thus, the weight of a squared sum of residuals decreases from one to zero linearly from the optimal to the outlier-threshold data rate and is zero for the outlier data rate. Fig. 7.5 visualises the resulting approximation of the sampled data rate function, as visualised in Fig. 7.1, with centres as visualised in Fig. 7.2, by the implemented approximant.

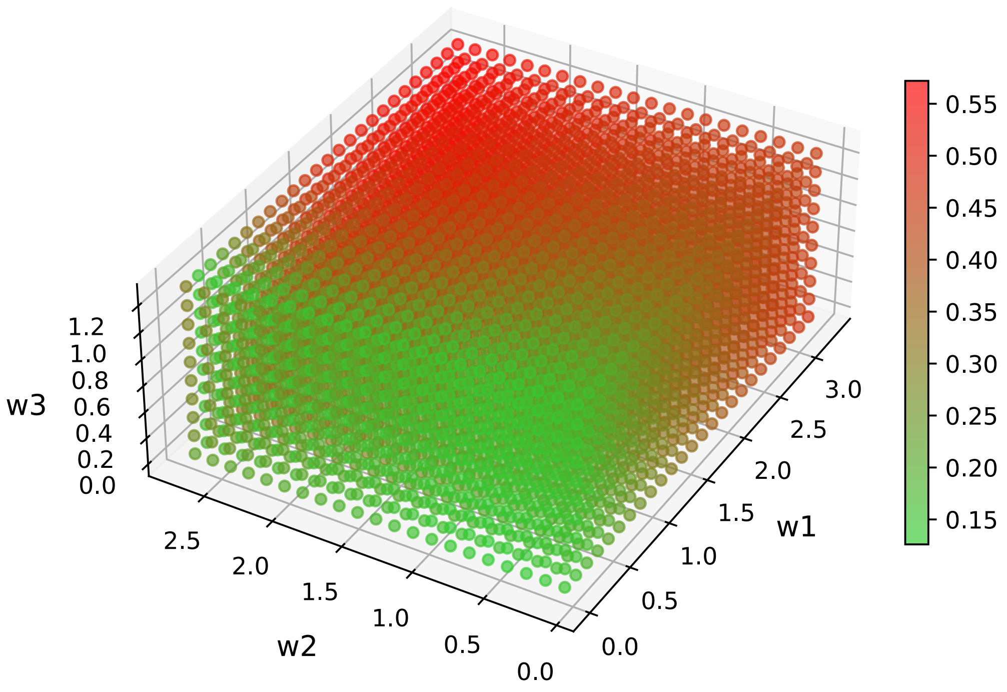

Figure 7.5: Data rate function approximant

7.2.4 Model Training

The dataset utilised to train the model comprises sample pairs of global surface statistics of meshes, obtained as in Section 7.2.1, and the respective approximated functions of the data rate, obtained as in Section 7.2.2, respectively Section 7.2.3.

The training mechanism partitions the dataset into the *training subset* and the *validation subset*. It uses the training subset to train the model, i.e. predict the optimal weight configuration based on the mesh's global surface statistics and adjust the neuron weights accordingly based on a computed loss. It utilises the validation subset to control the learning process and stops it when the model performs the best to prevent overfitting using the *early stopping mechanism* [Pre98]. This mechanism works by computing the loss on the separate validation subset and automatically stopping the learning process at the moment when the loss on this subset stops decreasing. If it did not stop, the loss on the training subset would continue decreasing. However, the loss on the validation subset would start increasing as the model learns the specific patterns of the training subset but loses the ability to generalise the patterns of the whole dataset, worsening the overall performance.

Generally, the artificial neural network can learn better [SHH96] if the input data has a suitable range and variance. Since the global surface statics of meshes have wide and unsuitable ranges of possible values, which may vary substantially, the training mechanism applies statistical standardisation to them, as described in Section 4.1, before passing them as input data to the artificial neural network.

The model optimiser adjusts the neuron weights to minimise the loss, defining the modelled relation between the global surface statistics of meshes and weight configurations. Therefore, choosing the loss function is critical to achieving the desired results. This work trains two models with different objectives. The first model utilises the loss function l_{sum} defined as

$$l_{\text{sum}}(M, \mathbf{w}) = d'_M(\mathbf{w}), \tag{7.1}$$

where d'_M is the data rate function approximant for a mesh M, and $\mathbf{w}$ is the predicted weight configuration. This loss function minimises the total sum of the approximated data rate for the meshes of the dataset. The other model utilises the loss function l_{ratio} defined as

$$l_{\text{ratio}}(M, \mathbf{w}) = \frac{d'_M(\mathbf{w})}{d'_M(\mathbf{w}^M_{\text{opt}})}, \tag{7.2}$$

where $\mathbf{w}^M_{\text{opt}}$ is the optimal weight configuration for a mesh M. This loss function minimises the total sum of the ratio of the approximated data rate to the approximated optimal data rate for the meshes of the dataset. It utilises the approximated optimal data rate to improve the accuracy of the computed ratio. If the approximant fits the approximated function well, it is equivalent to using the actual optimal data rate. However, if it does not, it may help to preserve the ratio accuracy in the neighbourhood of the optimal weight configuration since the implemented approximant may scale the resulting data rate rather than provide entirely erroneous results as it smooths the approximation curve for the data rate functions that are difficult to approximate.

7.2.5 Integration

The models are trained using the *PyTorch* [SAV20] framework and saved in the *ONNX* [SAV20] format. This format enables the model to be independent of the training and inferring environments, requiring a runtime to load the model and perform inference. A model represents an end-to-end transformation of a mesh's global surface statistics to the respective estimated mesh-optimal weight configuration. As implemented, it also allows loading various models with an arbitrary internal structure, trained on arbitrary datasets, and using it to estimate the mesh-optimal weight configurations.

The end-to-end compression process is integrated into the project and performed automatically. First, it computes the mesh's global surface statistics, as described in Section 7.2.1. Then, it passes them to the model that standardises this input, as described in Section 7.2.4 and infers the respective weight configuration. Finally, it compresses the mesh with the collected output weight configuration.

7.3 Experimental Results

This work builds an experimental dataset to evaluate the performance of the proposed and implemented mechanism estimating the mesh-optimal weight configuration based on the knowledge of the global surface statistics of a mesh. It comprises subsets of the datasets used in [Dvo+22], described in Section 2.3.4. It builds the dataset by pairing the global surface statistics of meshes, obtained as in Section 7.2.1, and the respective data rate function approximants, obtained as in Section 7.2.2, respectively Section 7.2.3. To preserve dataset quality, it filters out the meshes with outlying global surface statistics, primarily the highly irregularly triangulated CAD models. Table 7.1 presents the composition of the dataset, partitioned into the part

Table 7.1: Experimental dataset

	train	*test*
dataset	**# meshes**	
abc_regular	16570	1819
abc_irregular	8130	903
thingi10k	1313	153
tosca	68	11
mcgill	392	54
casual_man	490	55

used for training, respectively validating the models and the part utilised to evaluate their performance, i.e. the *test subset* of the dataset.

The Gaussian RBF kernel shape parameter ϵ is determined by experimentally testing feasible values and selecting the best-performing one, as discussed in Section 7.2.3. A common pattern is observed, as visualised in Fig. 7.4, repeating the process for many meshes, based on which the shape parameter is selected and applied to all the data rate function approximants.

The experimental dataset comprises 29958 meshes and is partitioned into the training, validation and test subsets in the ratio 7:2:1. The number of layers and neurons is tuned manually, training the models and minimising the loss on the validation subset. Both the models have 3 layers with 96 neurons per each. The models use the hyperbolic tangent function as the activation function between layers.

Determining the approximant, as discussed in Section 7.2.3, for the sampled data rate function, as discussed in Section 7.2.2, does not always, on the utilised dataset, lead to the desired property of it steadily converging to the data rate limit value while distancing from the sampled space of weight configurations and instead providing erroneous approximations since the approximant, as defined, does not fit the approximated function well. Therefore, an additional measure is implemented to train the model, limiting the range of predicted weight configurations. A custom layer is implemented that maps the output of the last layer of the artificial neural network to the feasible range of predicted weight configurations. It is a conveniently scaled and transformed hyperbolic tangent function. Doing this limits the potential of improving the data rate. However, it is necessary to train the model correctly. The results are not expected to worsen significantly since this problem does not arise frequently, and if it does, a significantly lower data rate is not expected to be located outside the feasible range of predicted weight configurations. The implementation experimentally sets the feasible range of predicted weights w_1, w_2 and w_3 to be $w_1 \in \langle 0.0657, 4.6847 \rangle$, $w_2 \in \langle 0.0097, 3.3888 \rangle$ and $w_3 \in \langle 0.000003, 4.1043 \rangle$ by observing the model's ability to learn with various tested ranges.

The impact of the mechanism was experimentally assessed by measuring the data rate for the meshes of the test subset of the experimental dataset, using the estimated mesh-optimal, default, dataset-optimised and optimal weight configurations. The experiment utilises the enhanced candidate filtering mechanism discussed in Chapter 5 and does not use the continuous priority updating mechanism discussed in Chapter 6. Table 7.2 presents the experimental results, comparing the results obtained for both the trained models to the default, dataset-optimised and optimal weight configurations.

The results show that there exists a relation between the measured global surface statistics of meshes and the corresponding well-performing weight configurations for the compression, and the model can learn it. The results differ based on the loss function utilised in the learning process.

The model that utilises the loss function defined by Eq. (7.1) decreases the total

Table 7.2: Mesh-optimal weight configuration estimation performance summary

bpv *change [%]*	*total sum* **loss**$_{\text{sum}}$			*mean per-mesh* **loss**$_{\text{ratio}}$		
dataset	$\mathbf{w}_{\text{default}}$	$\mathbf{w}_{\text{dataset}}$	$\mathbf{w}_{\text{optimal}}$	$\mathbf{w}_{\text{default}}$	$\mathbf{w}_{\text{dataset}}$	$\mathbf{w}_{\text{optimal}}$
abc_regular	-19.00	-14.50	18.04	-20.63	-12.23	15.14
abc_irregular	-17.31	-20.02	25.90	-15.68	-18.50	27.51
thingi10k	-12.73	-3.63	15.09	-11.73	-4.03	17.36
tosca	-5.83	-4.80	2.14	-5.59	-3.64	2.87
mcgill	-1.03	8.52	20.44	1.45	18.85	23.70
casual_man	-6.07	2.19	2.96	-6.90	1.36	2.12

data rate by 1.03 to 19.00 per cent per dataset, comparing it to the default weight configuration and up to 20.02 per cent per dataset, comparing it to the dataset-optimised weight configurations, resulting in 16.69, respectively 16.51 per cent improvement of the total data rate across the dataset over the default and the dataset-optimised weight configurations. However, it performs worse by 2.19, respectively, 8.52 per cent for the *casual_man* and the *mcgill* datasets. The default and the dataset-optimised weight configurations perform 46.85, respectively, 46.55 per cent worse for the meshes of the dataset than the optimal weight configurations. This mechanism improves it to 22.34 per cent of the optimal performance.

The other model, which utilises the loss function defined by Eq. (7.2), estimates the weight configurations, improving the resulting data rate up to 20.63, respectively 18.50 per cent per mesh of respective datasets over the default and the dataset-optimised weight configurations. It performs 17.98, respectively, 12.86 per cent better per mesh than the default and the dataset-optimised weight configurations across the dataset. However, it performs up to 18.85 per cent worse per mesh for the meshes of the *mcgill* dataset. The default and the dataset-optimised weight configurations perform 53.56, respectively, 46.56 per cent worse per mesh for the meshes of the dataset than the optimal weight configurations. This mechanism improves it to 18.85 per cent per mesh of the optimal performance. It shows that the model knows the individual mesh-specific properties and exploits them to estimate the weight configuration that performs well for the specific mesh.

The data rate worsening behaviour may be addressed to the inaccuracy of the data rate approximants for the meshes of the respective datasets since there is a correlation between them. The more inaccurate the approximations they provide, the worse the weight configuration estimation mechanism performs. Since the other loss function requires the approximants to fit the data rate function curve well, as it needs to compute the ratio of the data rate for any weight configuration to the optimal weight configuration, the worse is the performance if it does not satisfy the assumption and does not fit the data rate function well enough.

Future Work 8

The reference implementation of the priority-based connectivity coding algorithm leaves room for improvement, and so does the implementation described in this work. Based on the experimental results obtained in this work, some weak areas of the reference implementation and this implementation were identified.

The experimental results in Section 5.4 and Section 6.4 show that utilising a gate's priority as defined by Eq. (2.6) may result in a significant inaccuracy since it considers only the best and second-best candidate vertices. The priority-based traversal would benefit from incorporating more complete information into the priority, approaching its meaning and purpose.

As discussed in Section 7.3, the implemented approximant does not always capture the behaviour of the respective data rate function well. However, the approximated data rate is essential for the artificial neural network to learn the model effectively. Despite the quality of approximants, the experimental results in Section 7.3 show that the data rate improves considerably. Therefore, it is promising to improve the data rate function approximation, respectively, the sampling of the data rate function.

This work only briefly discusses the descriptive mesh statistics, as in Section 7.2.1, used as the input to the mechanism estimating the mesh-optimal weight configurations. However, which mesh properties contribute to improving the mesh-optimal weight configuration estimation was not thoroughly assessed. Exploring this problem may result in reducing the number of descriptive mesh statistics, respectively, the computational cost while preserving reasonable performance or developing more descriptive mesh statistics, improving the compression performance.

Conclusion 9

This work explored possibilities for improving the compression ratio of the priority-based connectivity coding algorithm [Dvo+22] developed by Dvořák et al. at the Department of Computer Science and Engineering at the University of West Bohemia.

It introduced the algorithm and put it into the context of mesh compression and a few other existing algorithms. Then, it introduced the possible enhancements and provided the background theory necessary to address these problems. Next, it discussed all the enhancements in detail, describing the implementation, presenting the experimental results, and validating and analysing them.

The first enhancement filters out the additional candidate vertices, which, if considered to extend the processed part of the mesh, would form a non-manifold edge or an unorientable surface in the extended processed part, violating the assumption of an orientable manifold mesh on input. The second enhancement handles continuous priority updates of unprocessed gates, synchronising the priorities with changes in the unprocessed part, exploiting additional recent, more accurate information. The last enhancement estimates the optimal weight configuration for a mesh based on the knowledge of its global surface statistics, defining the candidate's quality metric function and influencing the priority, eventually improving the traversal.

The experiments evaluated the performance of the enhancements, comparing the resulting data rate and computational cost to the reference implementation. Most of the experiments used the same datasets and weight configurations as in [Dvo+22], making the results directly comparable. The experimental results identified weak areas that were suggested for further research.

The enhanced candidate filtering mechanism consistently improved the data rate by 0.41 to 1.13 per cent, costing an additional 35.7 to 52.4 per cent computation time. The continuous priority updating mechanism improved the data rate by 0.17 per cent at best, generally worsening the performance by up to 9.18 per cent and requiring an additional 59.3 to 253.9 per cent computation time. The last enhancement, estimating the mesh-optimal weight configurations, improved the total data rate by 16.69 per cent for the whole experimental dataset, respectively, by 17.98 per cent per mesh of the dataset.

Bibliography

[Bot+10] BOTSCH, Mario; KOBBELT, Leif; PAULY, Mark; ALLIEZ, Pierre; LÉVY, Bruno. *Polygon mesh processing*. CRC press, 2010.

[BBK08] BRONSTEIN, Alexander; BRONSTEIN, Michael; KIMMEL, Ron. *Numerical Geometry of Non-Rigid Shapes*. 1st ed. Springer Publishing Company, Incorporated, 2008. ISBN 0387733000.

[Dre05] DREYFUS, G. *Neural Networks: Methodology and Applications*. Springer Berlin Heidelberg, 2005. ISBN 9783540229803.

[Dvo+22] DVOŘÁK, Jan; KÁČEREKOVÁ, Zuzana; VANĚČEK, Petr; VÁŠA, Libor. Priority-based encoding of triangle mesh connectivity for a known geometry. *Computer Graphics Forum*. 2022, vol. 42, no. 1, pp. 60–71. Available from DOI: `10.1111/cgf.14719`.

[Koc+19] KOCH, Sebastian et al. ABC: A Big CAD Model Dataset for Geometric Deep Learning. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 9593–9603. Available from DOI: `10.1109/CVPR.2019.00983`.

[MS17] MAJDISOVA, Zuzana; SKALA, Vaclav. Big geo data surface approximation using radial basis functions: A comparative study. *Computers & Geosciences*. 2017, vol. 109, pp. 51–58. ISSN 0098-3004. Available from DOI: `https://doi.org/10.1016/j.cageo.2017.08.007`.

[MGS07] MARAIS, P.; GAIN, J.; SHREINER, D. Distance-Ranked Connectivity Compression of Triangle Meshes. *Computer Graphics Forum*. 2007, vol. 26, no. 4, pp. 813–823. Available from DOI: `https://doi.org/10.1111/j.1467-8659.2007.01026.x`.

[MSW03] MARPE, D.; SCHWARZ, H.; WIEGAND, T. Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard. *IEEE Transactions on Circuits and Systems for Video Technology*. 2003, vol. 13, no. 7, pp. 620–636. Available from DOI: `10.1109/TCSVT.2003.815173`.

[PKJ05] PENG, Jingliang; KIM, Chang-Su; JAY KUO, C.-C. Technologies for 3D mesh compression: A survey. *Journal of Visual Communication and Image Representation*. 2005, vol. 16, no. 6, pp. 688–733. ISSN 1047-3203. Available from DOI: `https://doi.org/10.1016/j.jvcir.2005.03.001`.

[Pre98] PRECHELT, Lutz. Early Stopping - But When? In: *Neural Networks: Tricks of the Trade*. Ed. by ORR, Genevieve B.; MÜLLER, Klaus-Robert. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 55–69. ISBN 978-3-540-49430-0. Available from DOI: `10.1007/3-540-49430-8_3`.

[Ros99] ROSSIGNAC, J. Edgebreaker: connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*. 1999, vol. 5, no. 1, pp. 47–61. Available from DOI: `10.1109/2945.764870`.

[Ros01] ROSSIGNAC, J. 3D compression made simple: Edgebreaker with ZipandWrap on a corner-table. In: *Proceedings International Conference on Shape Modeling and Applications*. 2001, pp. 278–283. Available from DOI: `10.1109/SMA.2001.923399`.

[SHH96] SHANKER, M.; HU, M.Y.; HUNG, M.S. Effect of data standardization on neural network training. *Omega*. 1996, vol. 24, no. 4, pp. 385–397. ISSN 0305-0483. Available from DOI: `https://doi.org/10.1016/0305-0483(96)00010-2`.

[SAV20] STEVENS, Eli; ANTIGA, Luca; VIEHMANN, Thomas. *Deep learning with PyTorch*. Manning Publications, 2020.

[TL94] TURK, Greg; LEVOY, Marc. Zippered polygon meshes from range images. In: *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: Association for Computing Machinery, 1994, pp. 311–318. SIGGRAPH '94. ISBN 0897916670. Available from DOI: `10.1145/192161.192241`.

[Zha+05] ZHANG, J.; KAPLOW, R.; CHEN, R.; SIDDIQI, K. *McGill 3D Shape Benchmark*. 2005. Available also from: `https://www.cim.mcgill.ca/~shape/benchMark`.

[ZJ16] ZHOU, Qingnan; JACOBSON, Alec. *Thingi10K: A Dataset of 10,000 3D-Printing Models*. 2016. Available from arXiv: `1605.04797 [cs.GR]`.

List of Figures

List of Tables

List of Algorithms