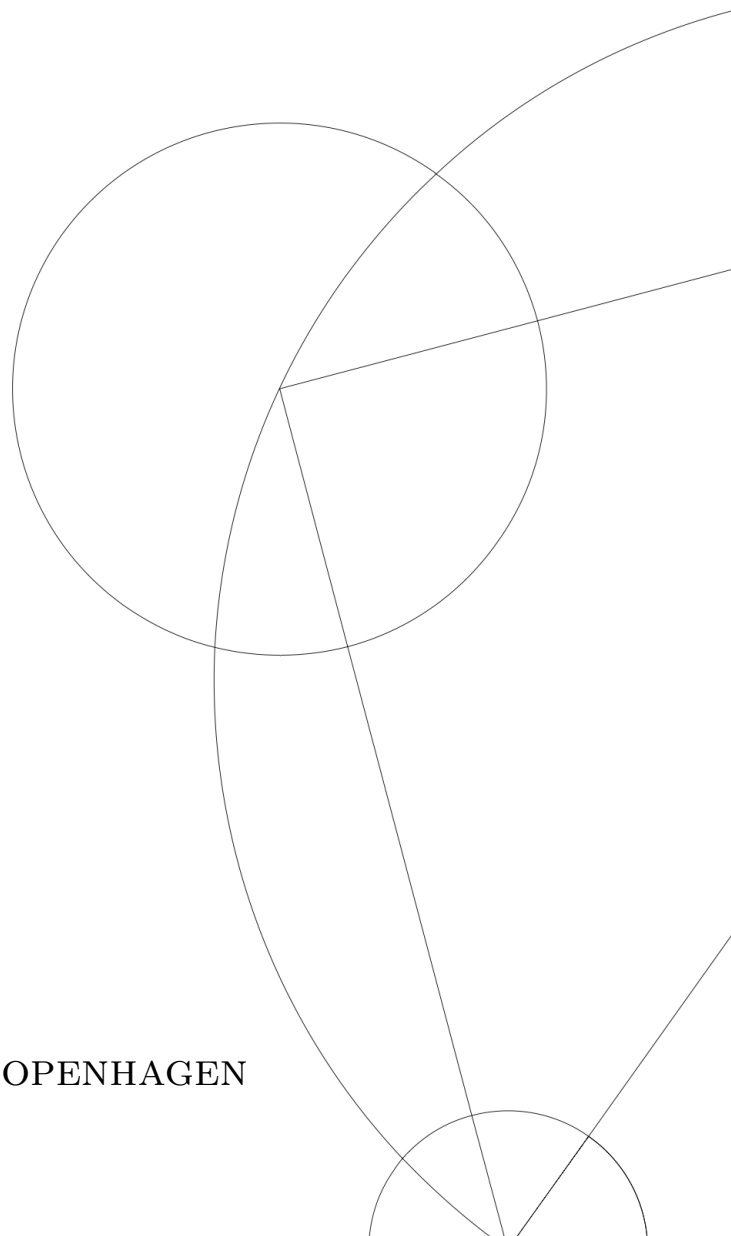
High-Resolution Meshes for Two-Dimensional Molecular Surfaces

M.Sc in Physics

Written by *Nikolai Plambech Nielsen*

03. September 2021

Supervised by
James Emil Avery



University of Copenhagen

UNIVERSITY OF
COPENHAGEN

Faculty:	Faculty of Science
Institute:	The Niels Bohr Institute
Author(s):	Nikolai Plambech Nielsen
Email:	LPK331@alumni.ku.dk
Title and subtitle:	High-Resolution Meshes for Two-Dimensional Molecular Surfaces
Supervisor(s):	James Emil Avery
Handed in:	2021-09-03
Defended:	2021-10-03

Name _____

Signature _____

Date _____

Abstract

Fullerenes are a group of molecules, forming cenvex polyhedral cages purely out of carbon. Few can be routinely synthesized, but those that can have found use in a wide array of fields, from cutting edge engineering, to novel medicinal treatments. Currently, systematic screening of the isomer space for promising candidate specimens is unfeasible, due to the large amounts of time it takes to accurately analyze molecules. The CarMa project is a new computational paradigm focussing on the rapid screening of low-dimensional carbon molecules like fullerenes, along with aiding the effort of synthesizing specimens of interest. More specifically, a DFT framework specialized to non-Euclidean manifolds is used for calculating the electronic structure.

The central problem addressed in this thesis is how to increase the precision of the DFT without compromising its accuracy: Mesh subdivision is used to increate the precision, but comes at the cost of distorting the curvature by introducing a singularity. This has a negative effect on the accuracy of the kinetic energy operator, as it is intrinsically linked to curvature through the presence of the Laplace-Beltrami operator: the generalization of the standard Laplacian operator to non-Euclidean geometry.

The problem is solved by treating the curvature as an independent variable, which can be "smeared" across the surface after subdivision, such that accuracy is maintained. The smearing procedure is done by modelling the curvature as heat spreading across the surface, and is solved in two different ways: one using a FEM framework previously developed for the CarMa project [1], and one using a discretization of the Laplace-Beltrami operator known as the cotan Laplacian.

With an acceptable distribution of curvature, a new manifold is then calculated by setting up a constrained optimization problem of the angles and lengths of the mesh, to make sure the manifold is embeddable, as predicted by Alexandrov's Uniqueness Theorem.

Finally, the result is visualized by explicitly calculating a local 3-dimensional embedding of the area of interest by cleverly leveraging conditions and symmetries inherent to the mesh.

Contents

Chapter 1

Introduction

The field of numerical quantum chemistry is mature but many simulations takes days or weeks to generate accurate results. This makes the systematic analysis of large groups of molecules infeasible. One such group of molecules is fullerenes. These are an allotrope of carbon forming convex polyhedral cages. The carbon atoms bond via sp^2 electrons, creating pentagonal and hexagonal rings. Hexagons are flat in the sense of Gaussian curvature (see section 2.5), as we know from graphene: one atom thick sheets of carbon, in its perfect hexagonal glory. The pentagons are what makes the fullerenes; they induce a Gaussian curvature of $2\pi/6$ per pentagon, causing the surface to bend up, sort of like a shallow bowl. Introduce more, and the molecule starts to close up, more and more, until 12 pentagons are introduced, whereupon the molecule closes completely.

The number of hexagons however, can be arbitrarily large. Indeed, the easiest fullerene to imagine might be $C_{20} - I_h$, made with zero hexagons: a regular dodecahedron, one of the Platonic solids. This fullerene is thus the smallest fullerene possible, but it is also unstable, due to the highly concentrated Gaussian curvature. An empirical rule has been derived, known as the isolated-pentagon rule (IPR) [2], which state that fullerenes with adjacent pentagons tend to be less stable than fullerenes without any adjacent pentagons. The simplest IPR fullerene is then $C_{60} - I_h$, where hexagons are placed in between every pentagon, creating a truncated icosahedron, one of the Archimedean solids, whose shape is also well known for being used in making footballs.

This fullerene is also the first to be discovered, back in 1984. It, and indeed the whole family of molecules, are named in honor of the late Buckminster Fuller, who had passed away the year before. $C_{60} - I_h$ is thusly named buckminsterfullerene, or more informally, a "buckyball". Of course, these are not the only ones: the molecules can be arbitrarily large, with a wealth of different symmetries (or indeed none!). $C_{20}, C_{60} - I_h$ and another, less symmetrical specimen are seen in figure 1.1

The theory of fullerenes tells us that the number of C_n fullerenes grows rapidly. While only a single C_{20} can form, with 60 atoms there are 1,812 possible fullerene molecules. With 100 atoms this grows to 285,194, and with 400 atoms a staggering 132,247,999,328 different fullerenes can form. In general, the number grows as $\mathcal{O}(n^9)$.

However, so far only a small number of fullerene have been synthesized, and the methods for doing so are non-selective: Using laser ablation or vaporization of graphite and then performing chromotography on the resulting soot. As such, only very stable isomers are produced. But those that have been, have found uses in a wide range of applications, from

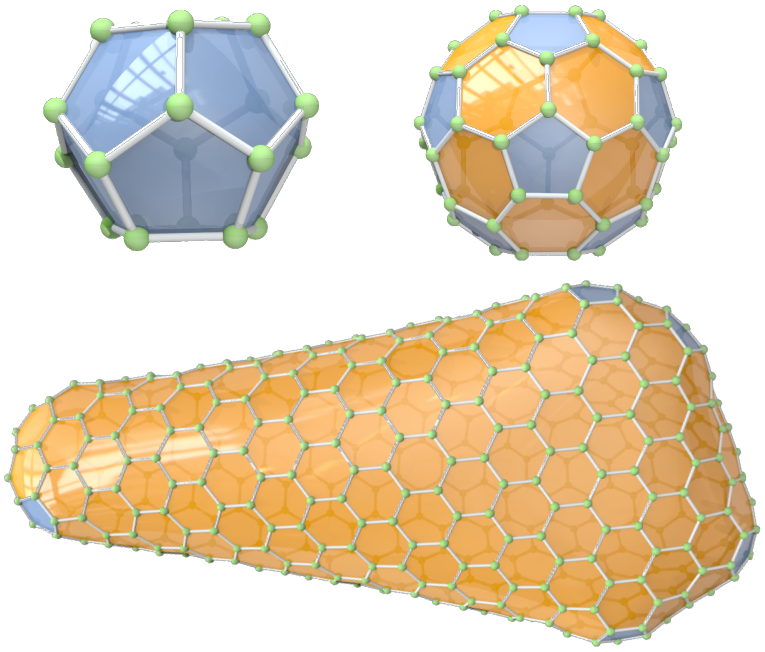


Figure 1.1: Examples of fullerenes. On the top left is the smallest fullerene C_{20}, in the top right is the smalles non-IPR fullerene, $C_{60} - I_h$, and on the bottom is $C_{524} - C_1$. In all fullerenes the hexagons are colored yellow, while the pentagons are colored blue. The atoms are represented by the green balls. From [3] with permission.

the medicinal industry, with asthma and allergy medicines, to promising cancer treatments [4, 5, 6]. In cutting edge engineering, the fullerenes see use in organic solar cells, biosensors and printable electronics [7, 8, 9].

With such a host of applications for even a small number of fullerenes, who knows what might be possible with a systematic knowledge of the full isomer spaces? This knowledge unfortunately comes at the cost of expensive calculations. The gold standard in calculating the quantum mechanics of molecules is the Coupled Cluster method, but this is infeasible for even small fullerenes. We therefore turn to the current best method for large molecules: the ab initio approach known as Density Functional Theory (DFT). But with the method taking weeks to complete for even a single specimen, a systematic search of the full isomer space is infeasible.

And this is only one part of the puzzle - it does not give any insight into how to synthesize the interesting specimens. We therefore need new methods both for calculating the properties and for discovering paths to synthesis. This is where the approach of the CarMa project (CARbon MAnifolds) comes in, spearheaded by the efforts of associate professor at the Niels Bohr Institute, James Emil Avery.

The main idea of the CarMa project is to forego looking at the explicit 3-dimensional structure of the molecule, and instead restrict our view to the surface of the molecule, defined by its hexagonal and pentagonal faces. This surface is defined by strict combinatorial rules of the bond graph of the molecule, and can be calculated at blazing speeds, compared to the optimization of 3-dimensional geometry [3].

We then restrict the movement of the electrons, such that they move exclusively along the surface, and calculate the electronic structure of the molecules using a DFT tailored to working on these non-Euclidean surfaces. This approach promises to be orders of magnitude faster than traditional methods - but at what cost?

On the face of it, it seems an absurd approximation: The wave function exists in our 3-dimensional space, after all! And yes, the accuracy of this method will not be able to compete with traditional methods, such as full DFT, but that is also not our goal. What we want is to be able to quickly and exhaustively screen the isomer space of fullerenes up to a given size, estimate their physical properties and weed out the uninteresting specimens. Once this is done, we select a handful of the most promising molecules and hand these to quantum chemists for a full analysis using traditional DFT. We can then further hand over approximate recipes for rational synthesis of the interesting molecules to synthesis chemists, giving them considerable momentum in the search for stable synthesis paths.

But what foundation do we have to go on with this approximation? For one, the 3-dimensional structure of the fullerenes are predicted by the intrinsic geometry of the bond graph save a slight deformation [3]. Additionally the chemistry of graphene tells us that the area of interest is the surface itself: The π-bonds formed by the sp^2 electrons exhibit an extremely high electron mobility [10], whilst out-of-surface effects are minimal. Further, the probability density for electrons in a molecule drops off exponentially with distance outside the molecule.

There is of course a key difference between graphene and fullerenes, as mentioned before: The former is flat, while the latter certainly is not. For small molecules then, the Coulombic interations between electrons on opposite "sides" of the molecule will be large, since the inverse-square law has yet to fall off appreciably. But as the size of the molecule increases, so too will this contribution decrease, enough so that the "graphene" contribution of high electron mobility dominates.

After calculating the 2-dimensional electron density, we use the ansatz of exponential decay out-of-surface, to calculate the 3-dimensional electron density, in something akin to a holographic principle. This will allow us to directly compare our results to that of traditional DFT methods.

We can thus create a 2-dimensional manifold without any reference to 3-dimensional geometry, and then solve our quantum mechanical problems on this manifold instead. This is exactly the contents of the Simon Krarup Steensens M.Sc. thesis, who created a prototype finite element method framework for solving partial differential equations (including those found in DFT) on the manifolds arising from the surface of fullerenes [1].

In pratice we work with the dual of the fullerene surface, where every face is turned into a vertex, and every vertex into a face. Since each carbon atom binds to 3 others (making the bond graph *cubic*), the dual graph will be a triangulation. We further approximate this triangulation to be one of only equilateral triangles, an approximation grounded in reality: measurement of the distance between centres of adjacent faces show that they form approximately equilateral triangles. This means that a pentagon and its hexagon neighbourhood will constitute a regular pentagonal pyramid in the dual manifold.

Because the manifold describes a non-Euclidian space we can not use the regular Laplace operator in calculating the kinetic energy, but must use the Laplace-Beltrami operator, and the popular cotan Laplacian: a linear discretization to the Laplace-Beltrami operator on triangulated surfaces, which is intimately linked to the curvature of the surface, and determines the surface geodesics.

In the linear finite element method we associate a single floating point number per vertex on the manifold. In the case of the dual manifold, have $n/2 + 2$ floats to work with. This is not nearly enough to capture the intricacies of the quantum mechanics of the system. As such we need to define new points on the manifold. An easy way to achieve this is by subdividing each triangle, but as we shall see, this also has the unfortunate side effect of concentrating the curvature on a smaller and smaller area, and in the limit of infinite subdivisions it creates a singularity of curvature.

An illustration of the problem is seen in figure 1.2. Here we have first unfolded the pyramid and laid it flat on the plane. Then we imagine all the curvature to be smeared out constantly over the area which is closer to the top of the pyramid than any other point (known as the Voronoi area, see section 2.6 and figure 2.8 for further explanation). Below this unfolding we have also plotted the actual pentagon, with all of the subdivided triangles showing. This has been done for the case of no subdivisions (left), a single subdivision (middle) and a double subdivision (right). Note how the area shrinks quadratically with the number of subdivisions. And to keep the volume of the polyhedron constant (equal to the total curvature) the height needs to grow quadratically.

We then stand at an impasse. We either have to find some new, higher-order discretization for the Laplace-Beltrami operator, or develop some method for "smearing" the curvature over a larger area when subdividing. This thesis takes the second approach, which leads us nicely to the problem we aim to address: How do we create high-resolution triangulations, without using explicit 3-dimensional geometry, with an appropriate curvature distribution. These triangulations are then to be used in solving partial differential equations (particularly those found in DFT) on the 2-dimensional surface arising from polyhedral molecules such as fullerenes.

An illustration of the solution to the problem of smearing curvature for the pentagon is seen in figure 1.3. Here we have plotted the pyramid in a black wireframe, and a "rounded" version of it in red.

1.1 Thesis contents

The rest of this thesis is structured in two main parts. The first part is split into two chapters and serves as the prerequisite theory which will be the foundation for the second part. The first chapter, Chapter 2, describe the mathematics of fullerenes, starting with a description of the CarMa project. A primer on graph theory is presented, along with how it relates to and describes fullerenes. We then move on to describing metrics and the consequences of inducing one upon our graph. Next a short introduction to differential geometry and how to solve differential equations on non-Euclidean manifolds is presented. Lastly, we describe how to increase the precision of the solutions to differential equations, and how this will lead to a singularity in curvature.

The second chapter, Chapter 3 deals with how to calculate electronic structure of fullerenes in the CarMa project. In it, we describe the general foundation of many-body quantum mechanics and density functional theory. We end the chapter with a look at precisely what the curvature singularity will do in terms of the accuracy of manifold DFT simulations.

The second part of the thesis describes my work and deals with the central problem of the thesis: How we construct high resolution meshes for polyhedral molecules like fullerenes, without introducing singularities in the curvatures. Chapter 4 states the problem in more

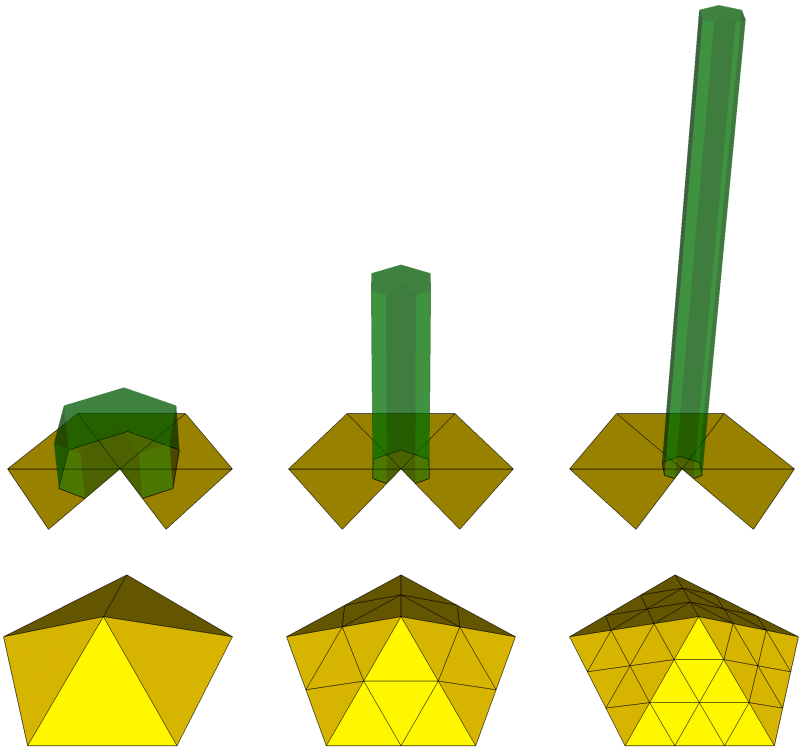


Figure 1.2: **Top**: A pentagonal pyramid cut along one edge and unfolded. On top is drawn a representation of curvature. The green polyhedron has cross sectional area equal to the Voronoi region of the vertex at the top of the pyramid, and volume equal to the total pentagonal curvature $(2\pi/6)$. This is shown for no subdivisions, a single subdivision, and a double subdivision. **Bottom**: The actual pentagonal pyramids, with all subdivided triangles shown. Again with either none, one or two subdivisions.

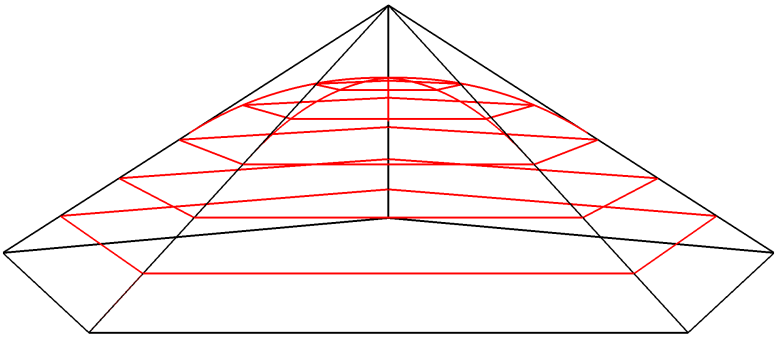


Figure 1.3: A sketch of a pentagonal pyramid before and after smearing. In black wireframe is the original pentagonal pyramid, and in red is the smeared version.

detail, and highlights some essential simplifications resulting from the regularities of the intrinsic fullerene geometry. In Chapters 5 and 6 I introduce two methods for manipulating the curvature of the mesh, weighing their benefits and drawbacks, before heading to Chapters 7 and 8 where I describe how to compute the discrete Riemannian manifold resulting from the new curvature distribution. Chapter 9 deals with a method for visualizing the resulting geometry explicitly, and lastly Chapter 10 is a discussion of the methods introduced, their results and shortcomings, and gives outlook on the future. Lastly, an appendix follows, detailing the main algorithms introduced in the second part of the thesis.

1.2 Software

The software written in the process of this project was implemented in Python, using Numpy for array programming and Scipy to handle linear algebra and optimization. Visualizations were done in Matplotlib and Vedo. The source files can be found on GitHub, under the folding-carbon repository, in my playground subdirectory. Of note is the following files and modules:

- `FEM`: A copy of the Finite Element Method framework developed by Simon Krarup Steensen [1]. Used for solving the heat equation using FEM.
- `refinement.py`: Functions pertaining to subdividing the mesh and constructing the relevant data structures (corresponding to the first part of chapter 7).
- `triangle_masks.py`: Functions for constructing the vertex and face masks used in restricting the mesh for the heat equation (chapter 4).
- `heat_equation.py`: Smearing of the curvature by solving the heat equation, both using FEM and the graph based Laplacian (chapters 5 and 6).
- `geometry_reconstruction.py` and `optimization.py`: Used for calculating the manifold geometry from the curvature distribution (chapters 7 and 8, repectively).
- `embedding.py`: For calculating the local embedding of the manifold (chapter 9).

Prerequisite Theory

Chapter 2

The Mathematics of Fullerenes

When analysing a physical system, we usually start with looking at the physical characteristics of the system: Its constituent parts and their dimensions, the interactions between them, etc. We then create a mathematical model to describe the system. In our case we have a molecule which is a 3-dimensional entity, and we can describe the atoms and the bonds between them as the vertices and edges of a graph.

So precisely what *can* we tell, by looking purely at the bond graph of fullerenes? It turns out that the carbon chemistry that underlies the formation of fullerenes imposes regularities such that the graph is not just a description of the atoms and how they bond to their neighbours. These regularities imply unambiguous faces and (with the inclusion of simple assumptions about distances between neighbours in the dual or cubic graph) even a unique isometric embedding in three dimensions, corresponding to the 3-dimensional geometry of the actual molecule!

The CarMa formalism is explored in this chapter: We will start with the basic object: the graph, and see how we can build the necessary machinery on top of it.

2.1 Planar Graphs and their embeddings

A graph $G = \{V, E\}$ is a set of vertices V connected to each other by a set of edges E. We denote the vertices as non-negative integers, $V = \{0, 1, \ldots, n\}$, and the edges by a set of unordered pairs of vertices: $E = \{e_0, e_1, \ldots, e_m\}$, with $e_i = \{i', j'\}$, where $i' \in V, j' \in V$ and $i' \neq j'$ for all edges. The requirement for unordered pairs makes the graph *undirected* - we make no distinction on the edge, if it goes from vertex i to j, or from j to i. To represent a graph we often use an adjacency matrix, $\mathbf{A}$, which is a defined by $A_{ij} = 1$ if $\{i, j\} \in E$. For undirected graphs, the adjacency matrix is symmetric. In our case, the adjacency matrix is very sparsely populated, and it is more useful to work with the sparse adjacency matrix $\mathbf{A}_S$, where each row A_i consists of the vertices connected to vertex i. Below is the first 10 rows

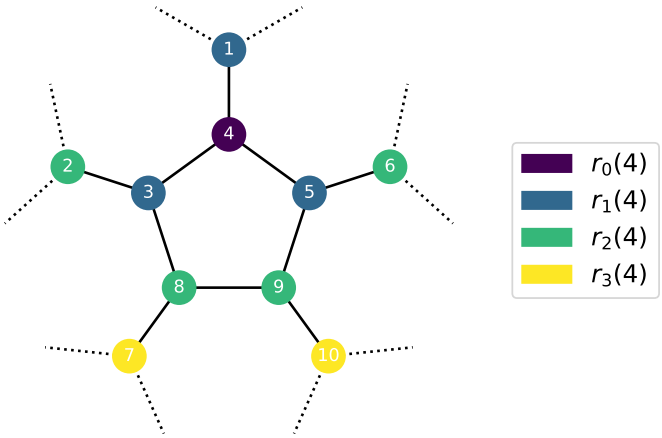


Figure 2.1: Part of a cubic graph. Vertices colour-coded after their distance to vertex 9.

and columns of the adjacency matrix, and its sparse variant, for the graph in figure 2.1.

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & \dots \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \dots \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}, \quad \mathbf{A}_S = \begin{pmatrix} 4 & \cdot & \cdot \\ 3 & \cdot & \cdot \\ 4 & 2 & 8 \\ 5 & 1 & 3 \\ 6 & 4 & 9 \\ 5 & \cdot & \cdot \\ 8 & \cdot & \cdot \\ 9 & 3 & 7 \\ 10 & 5 & 8 \\ 9 & \cdot & \cdot \\ & \vdots & \end{pmatrix},$$

Usually for sparse matrices, the CSR format or others are used, but since fullerene graphs are regular and every vertex has degree 3, the above format works well.

Next we need the *distance* between two vertices on a graph. This is defined as the minimum number of edges one needs to traverse to get from the starting to the ending vertex. This feeds into the n-rings associated with a vertex i, defined as the set of all vertices, which are a distance n from vertex i, denoted as $r_n(i)$. Here the 0-ring of i is defined as the vertex itself: $r_0(i) = \{i\}$. The graphs of fullerenes are all planar graphs, meaning they can be drawn in a plane with straight, non-intersecting edges (except at vertices, where the edges meet). Equivalently, they can be embedded on a sphere, in which case they form a polyhedron. This should not be surprising: if a fullerene is a carbon allotrope forming polyhedral cages, then surely the bond graph that describe them must be embeddable on the sphere without intersections.

Here is the rub, though: We have an infinite number of ways to do this! Think for example of a given embedding. We can perturb the vertices however we want (so long as we do not disturb them so much that any edges start crossing), without changing the topology. That is, there are an infinite number of (convex) polyhedrons with the same topology.

We would like a unique embedding per fullerene, which properly describe the dimensions of the molecule, since this will allow us to work directly on the graph, instead of worrying about the 3-dimensional geometry. And while the above sounds discouraging, we are in luck! While it is true that the graph is, by itself, not quite enough to guarantee a unique embedding, as we shall see, we need to invoke another property of the graphs and induce a metric, letting us arrive at a discrete Riemannian manifold, which will guarantee us a nice, unique embedding. First, a bit more on the graphs:

2.2 Polyhedral graphs

In addition to being planar, the fullerene graphs are also not 3-connected *polyhedral*. The definition is that we cannot split the graph in two connected subgraphs unless we remove at least 3 vertices. The consequence of this is that the topology of the graph is unique, and it will allow us to unambiguously define faces from the graphs: No matter the embedding, faces will be created from the same vertices. It also determines the orientation of the faces (up to a choice of direction: clockwise or counter-clockwise), so the same vertices appear in the same order.

This is a consequence of the unambiguity of orientation. Given a vertex and its neighbours, we can arrange them in a clock-wise or counter-clock-wise sequence, and this sequence will not change with the embedding. The way to create a face is then to start with a vertex v, and choose any of its neighbours v'. Then we simply walk along the edge between the two, and choose the next edge in a given choice of orientation. For example in figure 2.1, we could start with $v = 4$ and $v' = 3$. To get the next vertex in the counter-clockwise direction, we look at vertex v', and see which of its neighbours are *clockwise* from vertex v. In this case it is vertex 8. We can then keep doing this, setting $v \to v'$, and letting the new v' be the newly found neighbour.

This uniqueness of topology has the further application: it allows us to define a unique dual to the graph, where each face f becomes a vertex v' of the dual, each vertex v becomes a face f' of the dual and each edge is flipped to connect up the new dual vertices.

In the context of embeddings, be they planar or spherical, the dual vertices is placed in the middle of the primal (original) faces, while the dual faces are placed such that their centres coincide with their respective primal vertices. See for example figure 4.1, where a "bowl" consisting of a pentagon and five neighbouring hexagons is shown, along with its dual, which constitutes a hexagonal pyramid.

Euler Characteristics and The 12 Pentagon Rule

Since we can now rigorously define faces, we can show that a fullerene must contain exactly 12 pentagons: Fullerenes are convex polyhedra, so their Euler characteristic is $\chi = 2$. Both the primal and dual graph then obey the Eulers theorem with

$$|V| - |E| + |F| = 2,$$

where $|V|$ is the number of vertices in the graph, $|E|$ the number of edges and $|F|$ the number of faces. For the primal graph we denote these quantities as V_p, E_p, F_p and for the dual V_d, E_d, F_d, where $E_p = E_d, V_p = F_d, F_p = V_d$. The number of vertices in the primal graph is $V_p = N_C$, the number of carbon atoms in the molecule. The faces are made up of N_H

hexagons and N_P pentagons, so $F_p = N_H + N_P$. The number of edges can be related to the number of vertices, and the number of faces. Each vertex connects 3 edges, and each edge is connected to two vertices, so $E_p = 3V_p/2$. Likewise, each edge is a part of 2 faces, and there are 5 edges to a pentagon or 6 to a hexagon: $E_p = (5N_P + 6N_H)/2$. This then gives us

$$F_p - \frac{E_p}{3} = N_P + N_H - \frac{5N_P}{6} - N_H = 2, \quad N_P = 12.$$

2.3 Metrics, Embeddability and Alexandrov's Uniqueness Theorem

This is as far as the pure graphs will take us in our current context of unique embeddings. But if we induce a metric on our graph, making it a (discrete) Riemannian manifold, we can get further. In simplistic terms, a metric is a way of calculating distances along the manifold. The one we are most used to is the Euclidean metric of our 3-dimensional space. Here the distance between two points is our regular old Euclidean distance, given by the pythagorean theorem.

There are however an infinite ways of doing this. In the CarMa formalism we choose a metric such that the distance between centres of neighbouring faces are always unit length. Since the primal graph is cubic, then by definition its dual is a triangulation. The resulting dual manifold then becomes a set of connected equilateral triangles. To compute distances along the manifold, we cut along its edges, and unfold it onto the Euclidean plane, and use the length of the straight line path between two points. This is a *geodesic*. However, there are many such geodesics, since we make a choice on how to cut and unfold. As such we are not guaranteed that the resulting geodesic is the globally shortest path. It will of course be locally shortest, since any perturbation from a straight line path in a (locally) Euclidean space will produce a longer path.

Our choice of metric is grounded in measurements: while the bond lengths between the carbon atoms is dependent upon whether or not they are a part of hexagons or pentagons, the difference is small. And measurements of the distance between centres of neighbouring faces are indeed approximately the same across fullerenes.

With a chosen metric we now invoke Alexandrov's Uniqueness Theorem [11]:

Theorem 1. *Let M be a convex polyhedral metric on the sphere. Then there exists a convex polyhedron $P \subset \mathbb{R}^3$ such that the boundary of P is isometric to M. Moreover, P is unique up to a rigid motion.*

Let us unpack this. A polyhedral metric is a metric induced on a polyhedral graph. For it to be on the sphere, means we have not unfolded our manifold onto the plane, but embedded it onto a sphere. The theorem then tells us, that there is only one 3-dimensional convex polyhedron which preserves the metric. That is, the distances are the same in the abstract manifold and along the surface as it sits embedded in 3-dimensional Euclidean space. And this is exactly what we were looking for! We are now guaranteed a unique (isometric) embedding, which corresponds to the 3-dimensional configuration of the atoms.

The proof is unfortunately non-constructive, so we do not get a nice way to generate the isometric embedding. But the recent thesis of David Dedenbach, also supervised by James Emil Avery at the eScience group has made great strides towards building this unique embedding for the dual manifold of fullerenes [12].

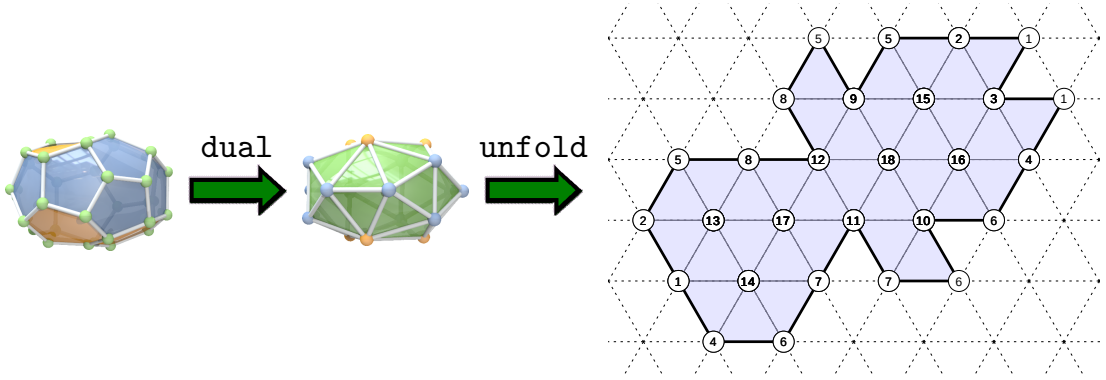


Figure 2.2: The primal (left) and dual (middle) mesh of $C_{32} - D_{3h}$. The colors of the dual mesh mirror those of the primal: The green, triangular faces correspond to the green vertices of the primal. The blue vertices correspond to the blue, pentagonal faces of the primal, and the yellow vertices are of course the hexagonal faces of the primal. On the right is a possible unfolding of the $C_{32} - D_{3h}$ dual onto the Eisenstein plane. From [3] with permission.

2.4 Eisenstein Unfoldings

We mentioned before that the dual manifold of our chosen metric is a connected set of equilateral triangles. We can then unfold this manifold onto what is called the Eisenstein plane: a Euclidean plane tiled by equilateral triangles. This is incredibly useful for visualizing functions on the manifold, since it allows us to view the whole molecule from a single angle, at the cost of some abstraction.

This cutting means that a given vertex may be placed at more than one point on the plane (if one of its connecting edges were part of a cut). Each vertex is a part of either 5 or 6 triangles, and upon unfolding the fullerene all pentagon vertices must be on the edge of the resulting polygon. If a pentagon vertex was a part of the interior of the polygon, the polygon must have a hole, which cannot be done when cutting along a single path of edges. With each unfolded vertex we associate a label (such that a single vertex on the fullerene can correspond to several unfolded vertices), and two integer coordinates (a, b), the "Eisenstein coordinates". The set of coordinates and their connections then defines a polygon in a right angled coordinate system, where every lattice point inside the polygon corresponds to an unfolded vertex. For them to overlay correctly with the unfolded fullerene, we transform the coordinates with the following matrix, such that triangles become equilateral instead of right angled isoscoles triangles:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 & \frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}$$

An example fullerene with its dual and one possible unfolding is seen in 2.2. A vertex with coordinates (a, b) has either 5 or 6 neighbours, with coordinates (in positive orientation)

$$(a+1, b),\ (a, b+1),\ (a-1, b+1),\ (a-1, b),\ (a, b-1),\ (a+1, b-1).$$

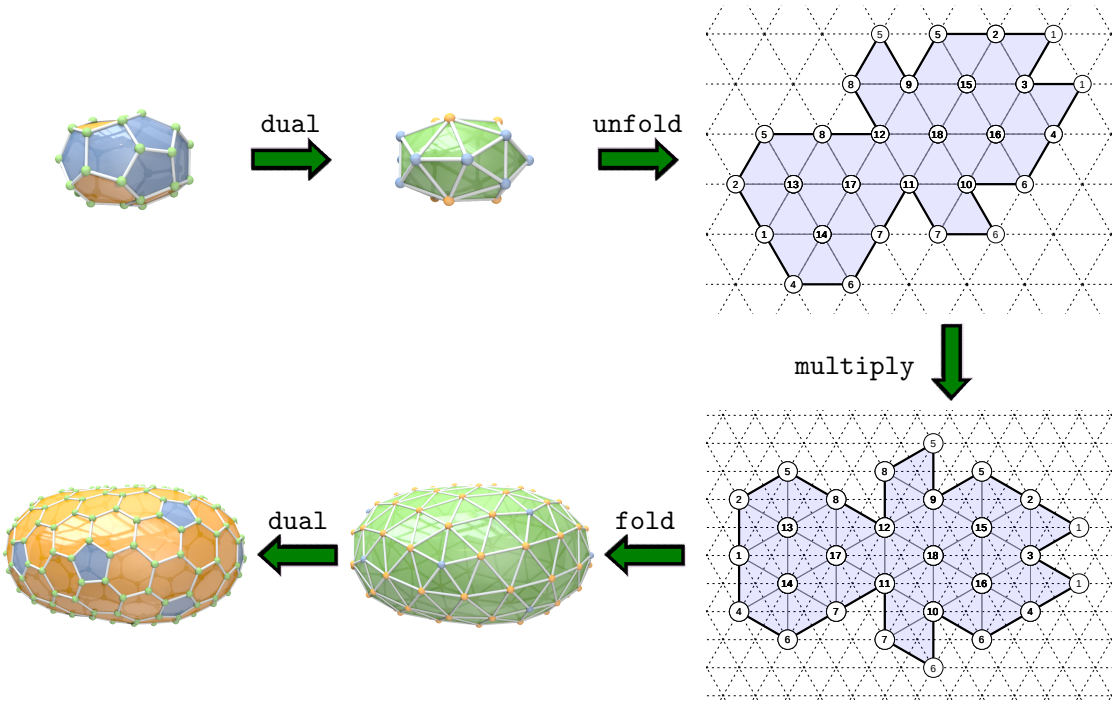


Figure 2.3: Top: The primal and dual manifold of $C_{32} - D_{3h}$ along with its unfolding. Bottom: The unfolding is then subjected to a transformation, $\mathrm{GC}_{2,1}$, generating an unfolding of $C_{224} - D_{3h}$. This can then be folded to get the dual manifold, and dualized to get back the primal manifold. From [3] with permission.

Generating New Fullerenes: Halma Transformations

This plane also suggest a set of transformations for generating new fullerenes from existing ones, called Goldberg-Coxeter transforms. The simplest of these are called Halma transforms, denoted $\mathrm{GC}_{H,0}$, and are characterized by the Halma index $H \in \mathbb{N}$.

A Halma transform consists of scaling up every dual face by H, sending each point (a, b) to (Ha, Hb). The result is that each dual face now contains within it H^2 equilateral triangles. Each of these triangles are then interpreted as a carbon atom. For example, performing a Halma transformation of $C_{20} - I_h$ with $H = 2$ results in each of the 20 dual faces becoming 80 dual faces, creating the dual unfolding of $C_{80} - I_h$.

The transformation $\mathrm{GC}_{2,1}$ is shown used on $C_{32} - D_{3h}$ in figure 2.3. Here all the Eisenstein coordinates (a, b) of the vertices are not only scaled, but also rotated.

2.5 Differential Geometry

Much as the metric determines distances along the manifold, it also determines the curvature of the surface. In fact, the two are equivalent! This curvature is the realms of differential geometry. For each point on a manifold we associate two unit vectors, the "principal" directions. These are perpendicular to each other, and point in the direction of maximal and minimal curvature. For each of the principal directions we have a principal curvature, denoted κ_1 and

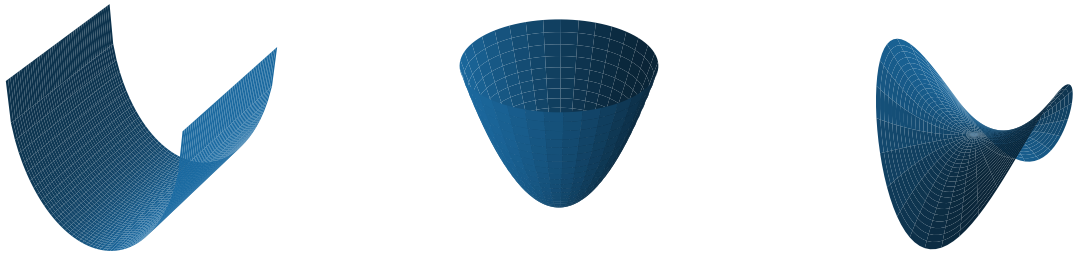


Figure 2.4: Typical examples of surfaces with Gaussian curvature. Left: zero Gaussian curvature (but positive mean curvature). Middle: Positive Gaussian curvature (and positive mean curvature). Right: Negative Gaussian curvature (and zero mean curvature)

κ_2. These are scalars describing just how much the surface curves at that point, in each of the two directions.

From the principal curvatures we define the mean curvature M and the Gaussian curvature K:

$$M = \frac{\kappa_1 + \kappa_2}{2}, \quad K = \kappa_1 \kappa_2.$$

A note: Mean curvature is usually denoted H, but since we reserve this letter for the Halma index, we will be using M to denote the mean curvature. For the case of fullerenes, we especially work with the Gaussian curvature, since this is "intrinsic" to the surface: It depends on the bond graph and our choice of metric, and not on the actual isometric embedding in space.

Prototypical examples of surfaces with positive, negative and zero Gaussian curvatures are seen in figure 2.4. Gaussian curvature can also be defined as an angle defect: 2π minus the angle sum of a circle of infinitesimal size on the surface. For zero curvature we have an angle sum of 2π as expected, and the surface can be produced from a flat piece of paper without tearing. For positive curvatures, the angle sum is smaller than 2π, and the surface can be cut and folded out onto a flat plane, without any overlap. This is not the case for negative curvature however, where the angle sum is larger than 2π, and if unfolded the surface must overlap.

For the case of fullerenes we are dealing with hexagons and pentagons. Hexagons induce no curvature, whilst pentagons induce a Gaussian curvature of $2\pi/6$. The Gauss Bonnet theorem connects the integral of the Gaussian curvature to the Euler characteristic of the surface (or equivalently its genus). In the case of a boundaryless surface S it states:

$$\int_S K \, \mathrm{d}A = 2\pi\chi = 4\pi(1 - g)$$

where S is the surface we integrate over, χ is the Euler characteristic, and g is the surface genus. For spheres and convex polyhedra $\chi = 2$ and the total curvature is 4π. As such 12

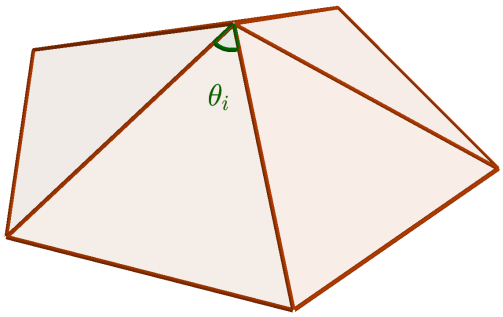


Figure 2.5: Angles θ_i incident on a pentagonal pyramid. In this case $\theta_i = \pi/3$ for all i, giving a total Gaussian curvature of $2\pi/6$.

pentagons are needed to close the fullerenes, precisely as seen from the direct calculation in Eulers theorem.

In the case of discrete Riemannian manifolds, every point on the interior of a face has zero curvature. Points on the edge between faces always have zero principal curvature along the edge, but may have non-zero principal curvature perpendicular to it. However, the surface must still integrate to 4π Gaussian curvature. This means that all Gaussian curvature is concentrated in the only points that are left: the vertices. Here we use the angle defect as the measure of the Gaussian curvature. On the dual manifold, with all triangles equilateral, we then see that angle defects of hexagonal and pentagonal vertices are $2\pi - \sum_i \theta_i = 2\pi - 6\pi/3 = 0$ and $2\pi - 5 \cdot \pi/3 = 2\pi/6$ respectively, see figure 2.5 for incident angles on a pentagonal vertex. This is precisely as expected, since a hexagonal vertex of the dual mesh corresponds to a hexagon with zero curvature on the primal mesh, whilst the pentagonal vertex is of course a pentagon on the primal mesh, inducing $2\pi/6$ curvature. For the rest of the thesis we will focus almost exclusively on the dual manifold, as triangulations are much easier to work with than general polygonal meshes.

2.6 Solving Differential Equations on Discrete Riemannian Manifolds: The Finite Difference Method

We are now at a point where we can adequately describe the manifold arising from fullerene molecules, which is all well and good. But what we really want, is to solve differential equations on the manifolds. Particularly those that arise when computing the electronic structure of molecules. And as is the case for the vast majority of quantum many-body problems, we need to use numerical methods since they afford no analytical solutions.

To build this machinery, we will start with a primer on how to do this in regular Euclidean space. Here we have three main methods for solving the problems, all of which rely on different ways of approximating derivatives: finite differences, finite elements and finite volumes. The first method is very intuitive, working directly from the definitions of differential calculus:

when taking a derivative, what if we do not take a full limit, but instead stop at some finite value? That is, we use a secant of the function, in stead of the full tangent:

$$\frac{df(x)}{dx} \approx \frac{f(x+\delta x) - f(x)}{\delta x}.$$

This approach is especially simple if we only care about rectangular domains and evenly spaced sampling of the function. But in our case the domain is a set of connected polygons, which permit no global coordinate system, so we instead turn to the finite element method where we approximate functions as piecewise polynomials, defined on the polygons of the surface.

This overview follows much like [1]. For a more comprehensive text see for example [13, 14]. A schematic overview of the finite element method is as such:

- State the problem in variational or *weak* form.
- Restrict the solution to a finite-dimensional subspace of functions, capable of satisfying the variational form of the problem (the Ritz-Galerkin approximation).
- Discretize the domain of the problem into a polygonal mesh and equip each polygon with a function from the above mentioned subspace.
- Construct the system of linear equations which govern the discretized weak form of the problem
- Solve the system of equations.

Step one, the variational form, consists of multiplying the differential equation by a test function and integrating over the domain. A prototypical example being the Poisson equation with Dirichlet boundary conditions:

$$-\nabla^2 u(x,y) = \rho(x,y),\ \forall (x,y) \in \Omega. \quad u(x,y) = f(x,y),\ \forall (x,y) \in \partial\Omega,$$

u being the desired solution, ρ and f being known functions and Ω being the computational domain, with boundary $\partial\Omega$. Requirements on the test function v are for it to be sufficiently regular (usually $v \in L^2$ on the domain Ω), and for it to vanish at the boundary: $v(x,y) = 0, \forall (x,y) \in \partial\Omega$. This then gives the variational form (omitting variables for space)

$$-\int_\Omega v\nabla^2 u \ \mathrm{d}\mathbf{x} = \int_\Omega v\rho \ \mathrm{d}\mathbf{x}, \quad \int_{\partial\Omega} vu \ \mathrm{d}\mathbf{x} = \int_{\partial\Omega} vg \ \mathrm{d}\mathbf{x}.$$

Using integration by parts we can move one ∇ to the test function at the cost of a sign and a boundary term, which vanishes due to the boundary condition of the test function:

$$-\int_\Omega v\nabla^2 u \ \mathrm{d}\mathbf{x} = \int_\Omega \nabla v \cdot \nabla u \ \mathrm{d}\mathbf{x} - \int_\Omega \nabla \cdot (v\nabla u) \ \mathrm{d}\mathbf{x},$$
$$\int_\Omega \nabla \cdot (v\nabla u) \ \mathrm{d}\mathbf{x} = \int_{\partial\Omega} (v\nabla u) \cdot \mathbf{n} \ \mathrm{d}\mathbf{x} = 0.$$

As such the variational form is

$$\int_\Omega \nabla v \cdot \nabla u \ \mathrm{d}\mathbf{x} = \int_\Omega v\rho \ \mathrm{d}\mathbf{x},$$

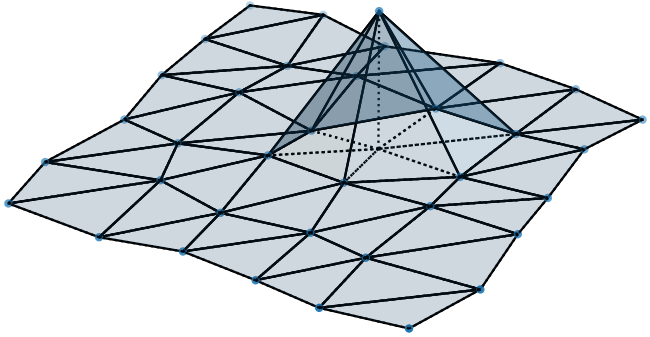


Figure 2.6: Example mesh equipped with linear basis functions. The basis function for a single vertex is highlighted.

or introducing the shorthand:

$$A(f,g) = \int_\Omega \nabla f \cdot \nabla g \, \mathrm{d}\mathbf{x}, \quad F(f) = \int_\Omega \rho f \, \mathrm{d}\mathbf{x}.$$

We now seek a solution u for which

$$A(u,v) = F(v), \quad \forall v \in V = \{A(v,v), v(\partial\Omega) = 0\}$$

where we restrict the u to the same space V. The next step is to substitute V for some finite-dimensional subspace: $\tilde{V} \subset V$. This is the Ritz-Galerkin approximation, where we now seek a solution in this restricted subspace:

$$\tilde{u} \in \tilde{V}, \text{ for which } A(\tilde{u},\tilde{v}) = F(\tilde{v}), \ \forall \tilde{v} \in \tilde{V}.$$

The problem has now been transformed into a continuous problem in a restricted subspace of functions, the actual contents of which we will choose after the next step, where we discretize the domain. Here we divide the domain into a set of non-overlapping polygons, in our case triangles. We choose to have 3 function values (nodes) per polygon, placed on the vertices of the triangle. This allows us to equip each triangle with a linear polynomial, resulting in a linear Lagrange triangle element.

We can now view the full function over the domain as a set of triangular planes, one for each element, with the function values being linearly interpolated on the triangle, from the three vertices. The general form of the plane is $u(x,y) = ax + by + c$, and inputting the function value along with the vertex coordinates gives a 3×3 system of linear equations, from which we can calculate the coefficients. However, this way of looking at the problem turns out to not be as beneficial in the long run, since we want to calculate the function values at the vertices. Another, equivalent picture, is to define a piecewise linear basis function ϕ_i for each vertex i, defined such that $\phi_i(x_j, y_j) = \delta_{ij}$, where (x_j, y_j) are the coordinates for the vertex j, and δ_{ij} is the Kronecker delta. And example mesh, with a single basis function is

shown in figure 2.6. The function is then just the sum of the piecewise linear polynomials for all vertices, scaled by the function values of each vertex:

$$u(x,y) = \sum_i u_i \phi_i(x,y),$$

where we can then "section off" the solution on each triangular element, if we want the planes. We can see that the set of piecewise linear polynomials is an allowed function space $\tilde{V}$, provided that each element is of a finite size. This is because the gradient of a linear polynomial is constant, and a constant integrated over a finite space has a finite value. Indeed, any set of piecewise polynomials, no matter the degree, is an allowed function space, since the integral of any polynomial (and therefore also a product of their gradients) is finite. However, for higher degree basis functions we need a larger set of nodal values per element, since the polynomials will otherwise not be uniquely defined. If we want a full polynomial of degree k to be uniquely defined, we need one point for each of its coefficients. For two dimensions, this is just k choose 2: $C(k,2) = (k+1)(k+2)/2$.

This choice of basis allows us to elegantly construct the integral over the full domain. Returning to the Ritz-Galerkin form of the problem, we let the test-function be a member of the same subspace of functions, so: $\tilde{v} = \sum_j v_j \phi_j$. Inserting this we get

$$A(\tilde{u},\tilde{v}) = F(\tilde{v}), \quad \Leftrightarrow \quad \sum_j v_j A(\tilde{u},\phi_j) = \sum_j v_j F(\phi_j).$$

For this to hold for all $\tilde{v}$, we let $A(\tilde{u},\phi_j) = F(\phi_j)\ \forall j$. Expanding our choice of $\tilde{u}$ then gives a set of linear equations:

$$A(\tilde{u},\phi_j) = F(\phi_j), \quad \Rightarrow \quad \sum_i u_i A(\phi_i,\phi_j) = F(\phi_j),$$

or

$$\mathbf{Au} = \mathbf{F}, \quad A_{ij} = A(\phi_i,\phi_j) = \int_\Omega \nabla\phi_i \cdot \nabla\phi_j \ \mathrm{d}\mathbf{x}, \quad F_i = F(\phi_i) = \int_\Omega \rho\phi_i \ \mathrm{d}\mathbf{x},$$

and with unknowns $\mathbf{u}^T = [u_0, u_1, \ldots, u_{n-1}]$ being the function values for the system. Traditionally the matrices $\mathbf{A}$ and $\mathbf{F}$ are called the stiffness matrix and source vector, respectively.

The Laplace-Beltrami Operator

When accounting for non-Euclidean geometry in problems involving the Laplacian operator, one needs to use its generalization, the Laplace-Beltrami operator. For continuous space in n dimensions, this is

$$\Delta f = \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{1}{\sqrt{|g|}} \frac{\partial}{\partial x^i}(\sqrt{|g|} g^{ij} \frac{\partial f}{\partial x^j})$$

with g being the metric tensor and $|g|$ being the absolute value of its determinant. This expression is not entirely helpful for our discrete Riemannian manifolds since there is no coordinate form of the metric g. Instead we need a discretization of the operator, the most popular of which is the cotan Laplacian $\mathbf{L}^C$, with rows $\mathbf{L}_i^C$ defined as such:

$$\mathbf{L}_i^C \mathbf{u} = \frac{1}{2} \sum_{j \in r_1(i)} (\cot\alpha_j + \cot\beta_j)(u_j - u_i) \tag{2.1}$$

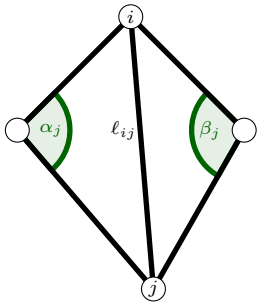


Figure 2.7: Angles and edge lengths used in the cotan Laplacian

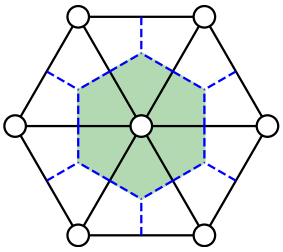


Figure 2.8: The Voronoi cells for 7 vertices, arranged in equilateral triangles. Shaded green is the Voronoi cell for the middle, hexagonal vertex. Note how the hexagonal Voronoi area precisely corresponds to a hexagonal face on the primal manifold.

where the angles α_j, β_j are defined as in figure 2.7. This matrix is then used as a drop-in replacement for the stiffness matrix. Further, we are allowed to scale by some area factor, to minimize the discretization error incurred. As shown in [15], scaling each row i of the matrix by the inverse of the Voronoi cell area for vertex i, is the optimal choice of area.

Voronoi cells

For non-blunt triangles the Voronoi cell for vertex i is constructed by joining the perpendicular bisectors of every edge terminating at vertex i, that is all edges $e_n = \{i, j\}, \forall j \in r_1(i)$. These meet at the circumcentres of the triangular faces, creating convex polygons, which together tile the domain. This also has the consequence that all points in space closer to vertex i than any other vertex, is a part of the Voronoi cell of vertex i. The Voronoi cell for a hexagonal vertex is seen in figure 2.8. Note that this is exactly the hexagons of the primal graph. Likewise the Voronoi region around a pentagonal vertex is itself a regular pentagon, corresponding to the primal graph pentagon! The area of the Voronoi cell is calculated as [15]

$$A_i^v = \frac{1}{8} \sum_{j \in r_1(i)} (\cot \alpha_j + \cot \beta_j) ||\mathbf{x}_i - \mathbf{x}_j||^2, \tag{2.2}$$

from which we see, that in addition to the angles of the mesh, we also need all edge lengths of the system.

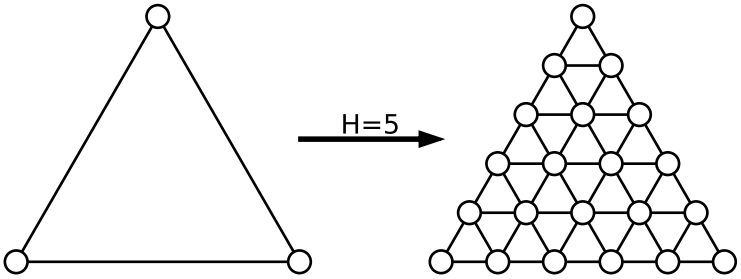


Figure 2.9: Subdivision of a triangle using a Halma subdivision index $H = 5$, yielding 5 rows of equilateral triangles. In the new mesh there are $H^2 = 25$ triangles and $(H+1)(H+2)/2 = 21$ vertices.

Luckily, for our choice of metric this is easy. Every angle is $\pi/3$ and every edge length is unit length. Every weight becomes $1/\sqrt{3}$ and the Voronoi area becomes $A_i^v = \deg(i)\sqrt{3}/12$.

Increasing Precision With Mesh Refinement

Due to our choice of linear basis functions, we only have a single number per dual vertex to describe the solution to our differential equations. This gives a total of $N_C/2+2$ numbers per fullerene, which will not nearly be enough to capture the intricacies of quantum many-body interactions. We therefore need to refine our mesh, so we have more points to work with.

The procedure we employ is similar to Halma transforms: Given an integer H, called the Halma index, we subdivide every triangle into new equilateral triangles, forming H "rows" of equilateral triangles. See figure 2.9 for an example with $H = 5$. This subdivides the original dual face into H^2 new ones and the total number of triangles in the mesh is then $F_d = H^2 N_C$. This then gives the number of vertices in the dual mesh, after refinement, as:

$$V_d = \frac{F_d}{2} + 2 = \frac{H^2}{2} N_c + 2.$$

Unfolding Refinements

When refining the mesh, we can modify an existing unfolding to accomodate the new subdivisions. The Eisenstein coordinates of new vertices are easily calculated using an affine transformation. We first define a prototypical triangle, with corners (0,0), (1,0) and (1,1). Subdividing this by the index H then yields a set of points, with coordinates $(a, b) = (i/H, j/H)$, where i and j are integers with $0 \leq i \leq H$ and $0 \leq j \leq i$. The total number of vertices in the triangle is just the $(H + 1)$'th triangle number, $(H + 1)(H + 2)/2$.

This triangle, and the vertices inside it, can now be transformed anywhere on the plane by:

$$\begin{pmatrix} a' \\ b' \end{pmatrix} = \mathbf{A} \begin{pmatrix} a \\ b \end{pmatrix} + \mathbf{t}$$

where $\mathbf{A}$ is a matrix transformation, consisting of rotations, scalings and/or permutations, and $\mathbf{t}$ is a translation. These can be encoded nicely using the augmented transformation

matrix $\mathbf{T}$:

$$\mathbf{T} = \begin{pmatrix} A_{11} & A_{12} & t_1 \\ A_{21} & A_{22} & t_2 \\ 0 & 0 & 1 \end{pmatrix}$$

where $\mathbf{T}$ has one more row and column than $\mathbf{A}$. This matrix then acts on the homogeneous coordinate representation, where we just append a 1 to every vector. We then have

$$\begin{pmatrix} a' \\ a' \\ 1 \end{pmatrix} = \mathbf{T} \begin{pmatrix} a \\ b \\ 1 \end{pmatrix}$$

We can then calculate the transformation matrix automatically, if we know the (un)transformed coordinates of 3 vertices, so $\mathbf{X'} = \mathbf{TX}$ or $\mathbf{T} = \mathbf{X'X}^{-1}$, where $\mathbf{X}$ is the 3×3 matrix whose columns are the augmented coordinates for the prototypical triangle, and $\mathbf{X'}$ is the augmented matrix for the corners of the destination triangle. With the prototypical triangle described above, we have

$$\mathbf{X} = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \quad \mathbf{X}^{-1} = \begin{pmatrix} -1 & 0 & 1 \\ 1 & -1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

which is just a constant, and $\mathbf{X'}$ is then easily constructed from the Eisenstein coordinates of a given triangle, and we can calculate the Eisenstein Coordinates of all new vertices in the triangle, from a single matrix multiplication.

2.7 Refinements and Curvature

We note now, that since every new vertex spawned when refining is a hexagon vertex, the curvature is still concentrated in the 12 pentagon vertices (like for the larger fullerene generated by the equivalent Halma transformation). This will become a problem: what we are doing is essentially approximating a smooth surface. We have an implicitly defined geometry (by the fullerene dual graph and our choice of metric). We then place new points on the surface, and join them up with the old points.

On the actual smooth surface however, the curvature will become a singularity, and we will have a kink in the surface (ie, the cone point). We choose instead to view the surface as having its curvature smeared out over a region, namely the Voronoi region, as a curvature density. Referring back to figure 1.2 we see an illustration of the original pentagon vertex and its 1-ring neighbourhood, with the curvature of the pentagon vertex represented as a uniform curvature density on the Voronoi region. When refining the mesh, the same amount of curvature has to fit into a region scaled by $1/H^2$, yielding a much larger density. More specifically, the curvature is a constant $K = 2\pi/6$, whilst the Voronoi area is $A = 5\sqrt{3}/(4H^2)$, so the density ρ_K is

$$\rho_K = \frac{K}{A} = \frac{\pi H^2 4\sqrt{3}}{15},$$

and in the limit of $H \to \infty$ we have $\rho_K \to \infty$: A singularity.

So how do we reconcile this need for higher precision, obtained by mesh refinement, with this ever increasing curvature density? The answer is to still refine the mesh, but change its geometry such that the curvature is still smeared over the entire original Voronoi region. In other words, we need to adopt a new metric, at least within the pentagon Voronoi region.

Mesh Fairing

Mesh fairing algorithms manipulate the geometry of a mesh, and are regularly used in denoising meshes captured using 3D-scanners. Common among them, though, are that they do not work on the manifold level. They all need explicit 3-dimensional embeddings. Take mean curvature flow (MCF) as an example, where the vertex coordinates are updated using the heat equation:

$$\Delta \mathbf{x} = \dot{\mathbf{x}},$$

with Δ being the cotan Laplacian. The cotan Laplacian acting upon the geometry is related to the mean curvature by $\Delta \mathbf{x} = 2M\mathbf{N}$, where $\mathbf{N}$ is the vertex normal, see [16]. We then need to calculate the cotan Laplacian using the formulas 2.1 and 2.2 and use them to solve the heat equation for each coordinate in turn. But for our case we do not have an explicit 3-dimensional geometry! So we will have to come up with a new method for achieving this goal, which is precisely the subject of the second part of this thesis.

Chapter 3

Electronic Structure

We now have all the tools we need to describe the intrinsic geometry of fullerenes, and we see why usual methods for mesh fairing will not work. With this we move on to the main purpose of the CarMa formalism: how do we calculate quantum mechanical quantities of interest for fullerenes?

We start by describing the quantum mechanics of the fullerene system and then move on to how to solve it. Many methods exist for this, like chemical graph methods and tight binding, which also work directly on the fullerene bond graph, but both of which are unsuited for our purposes as they are not nearly accurate enough. At the other end we have ab initio methods like the coupled clusters, which have an enormous complexity. In the sweet spot sits density functional theory (DFT), the simplest ab initio quantum chemical approach.

The ab initio methods, as noted in the introduction, work with explicit 3-dimensional geometry. Usually this scales as $\mathcal{O}(N^3)$, with N being the number of electrons in the system. However, linearly scaling DFTs have been introduced, for example in [17, 18]. The idea for a linearly scaling FEM-DFT on the dual manifold is from [3], with a prototype implementation in [1]. This is the path we follow, as it will allow us to obtain more accurate results than chemical graphs and tight binding, whilst still keeping computational costs low.

3.1 Quantum Mechanics

In quantum mechanics we seek to calculate the properties of a some system of interest. Be this a single particle or a set of particles, such as an ensemble or a molecule. The properties are determined by the wave function Ψ which in turn is governed by the Schrodinger equation:

$$\hat{H}\,|\Psi\rangle = (\hat{T} + \hat{V})\,|\Psi\rangle = i\hbar\frac{\partial}{\partial t}\,|\Psi\rangle\,,$$

where $\hat{T}$ is the kinetic energy operator, and V is the potential energy operator. In our case we are not concerned with time-dependent Hamiltonians, giving us a stationary eigenvalue problem of the form:

$$(\hat{T} + \hat{V})\,|\Psi\rangle = E\,|\Psi\rangle$$

Our first approximation is to regard each nuclei as a single quantum mechanical particle, such that the potential energy can be split into four terms:

$$\hat{V} = \hat{V}_{ee} + \hat{V}_{nn} + \hat{V}_{en} + \hat{V}_{ext},$$

the electron-electron interaction, nuclei-nuclei interaction, nuclei-electron interaction and an external potential. For now we assume that the molecules are free of any external interaction, so $\hat{V}_{ext} = 0$. The remaining terms are

$$\hat{V}_{ee} = \sum_{i=1}^{N_e} \sum_{j=i+1}^{N_e} \frac{e^2}{|\mathbf{x}_i - \mathbf{x}_j|}, \quad \hat{V}_{nn} = \sum_{i=1}^{N_n} \sum_{j=i+1}^{N_n} \frac{Z_i Z_j e^2}{|\mathbf{X}_i - \mathbf{X}_j|}, \quad \hat{V}_{en} = \sum_{i=1}^{N_e} \sum_{j=1}^{N_n} \frac{Z_j e^2}{|\mathbf{x}_i - \mathbf{X}_j|},$$

with N_e and N_n being the number of electrons and nuclei respectively, Z_i and $\mathbf{X}_i$ being the number of protons in nuclei i along with its position, and lastly $\mathbf{x}_i$ being the position of electron i.

The two-body nature of these operators means we cannot exactly separate the wave function into a purely electronic part and a purely nucleic part. But the nuclei are much heavier than the electrons (around 4 orders of magnitude for the case of a carbon nuclei), and as such we will have a separation of time scales. This is the foundation of the Born-Oppenheimer approximation, where we assume that $\mathbf{X}_i$ is fixed for all i, such that $\hat{V}_{nn}$ is a constant. This then eliminates the $3N_n$ nucleic degrees of freedom, leaving us with $3N_e$ electronic degrees of freedom.

3.2 Density Functional Theory

Even with the Born-Oppenheimer approximation, the computational complexity of the problem is much too great, and we turn to density functional theory. Instead of the wave function with its $3N_e$ degrees of freedom, the fundamental quantity in density functional theory is the electronic density $\rho(\mathbf{x})$:

$$\rho(\mathbf{x}) = N_e \prod_{i=2}^{N_e} \int d^3\mathbf{x}_i |\Psi(\mathbf{x}, \mathbf{x}_2, \dots \mathbf{x}_{N_e})|^2,$$

with its measly 3 degrees of freedom. This seems like a major simplification of the problem, maybe even an oversimplification. But it is justified by the two Hohenberg-Kohn theorems. These are

Theorem 2. *The electronic density of a bound system (and hence also its wave function) is uniquely determined by the eternal potential, aside from an additive constant.*

Theorem 3. *The ground state energy can be computed from the electronic density using a variational principle, and the density which minimizes this energy is indeed the unique ground state of the Hamiltonian (provided we are in a non-degenerate case)*

These theorems give us a firm ground to stand on, but does not, by themselves, lead us closer to the solution. This is where Kohn-Sham theory comes in. Here we approximate the full, many-body electronic wave function as a Slater determinant of non-interacting single particle electronic wave functions (orbitals):

$$\Psi(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{N_e}) = \frac{1}{\sqrt{N!}} \begin{vmatrix} \psi_1(\mathbf{x}_1) & \psi_2(\mathbf{x}_1) & \dots & \psi_{N_e}(\mathbf{x}_1) \\ \psi_1(\mathbf{x}_2) & \psi_2(\mathbf{x}_2) & \dots & \psi_{N_e}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_1(\mathbf{x}_{N_e}) & \psi_2(\mathbf{x}_{N_e}) & \dots & \psi_{N_e}(\mathbf{x}_{N_e}) \end{vmatrix}$$

In this formulation, the total particle density is just due to the $N_e/2$ orbitals with lowest energy being occupied:

$$\rho(\mathbf{x}) = \sum_{i=1}^{N_e/2} |\psi_i(\mathbf{x})|^2$$

These orbitals are eigenstates of a, as of yet undefined, single-particle Hamiltonian, with the form

$$\hat{H}_{KS}\psi_i = -\frac{\hbar^2}{2m_e}\nabla^2\psi_i + \hat{V}_{KS}\psi_i = \varepsilon_i\psi_i.$$

The potential $\hat{V}_{KS}$ appearing in the Hamiltonian is known as the Kohn-Sham potential, or reference potential, and is to be defined in such a way that the total energy coincides with the ground state energy of the full problem. Then due to the second Hohenberg Kohn theorem, this electron density will be exactly the ground state electron density! Now, the total energy in Kohn-Sham theory is given by the *functional* (we here omit the independent variable $\mathbf{x}$ of the density ρ to reduce clutter):

$$E_{KS}[\rho] = \langle\Psi[\rho]|\hat{T}_{KS} + \hat{V}_{KS}|\Psi[\rho]\rangle\,,$$

where $\langle\Psi[\rho]|\hat{T}_{KS}|\Psi[\rho]\rangle$ is the total kinetic energy of this Kohn-Sham wave function, which is just the sum of the kinetic energy of each orbital:

$$\langle\Psi[\rho]|\hat{T}_{KS}|\Psi[\rho]\rangle = -\frac{\hbar^2}{2m_e}\sum_{i=1}^{N_e/2}\langle\psi_i|\nabla^2\psi_i\rangle\,.$$

While the Kohn-Sham potential is important in its own right, the exact form of it is not important for this thesis, and will thus be neglected. Suffice to say, it can be split into several terms: an external potential, a classical electrostatic potential (known as the Hartree term) and an exchange/correlation potential, which makes up for the quantum effects neglected in the Hartree term.

3.3 QM, DFT and FEM for Fullerenes

The treatmeant of quantum mechanics and density functional theory has so far been in the case of eudlidean geometry in 3 dimensions, but as we saw in Chapter 2, we treat our system as a discrete Riemannian manifold. The formulas are easily adapted for 2-dimensional manifolds using the linear finite element methods introduced in Section 2.6, if we remember to use the cotan Laplacian, which we saw has an exceedingly simple formulation for our dual manifolds. While regular DFTs scale as $\mathcal{O}(N^3)$ with the number of electrons, this surface DFT scales as $\mathcal{O}(N)$ [19].

We still suffer the problem of low precision due to the resolution of the mesh used in the calculations, which is why we subdivide our mesh. But this has an adverse effect on the curvature, which in turn will have an adverse effect on the accuracy of the kinetic energy operator. We can think of the Laplacian in general as measuring the curvature of the function it is applied to. A function with high curvature then has a high kinetic energy. But in our case the space itself has an intrinsic curvature, and we need to account correctly for this in our calculations. This is why we need the ability to smear the curvature, so as not to unduly penalize solutions around the pentagonal vertices, where all the curvature is concentrated.

A Method for Creating High Resolution Meshes for Convex Polyhedral Metrics

Chapter 4

The Problem Statement

A Set of Congruent Pentagons

We now have all the tools we need to start working on the problem at hand: How do we develop a method for subdividing the discrete Riemannian manifold without reference to 3-dimensional geometry, and without distorting the Gaussian curvature? In this chapter will see how the problem can be simplified by the use of Voronoi regions, and we will see just what makes an acceptable curvature distribution for our purposes.

4.1 The Problem Statement

As we saw in section 2.7, the naive way of refining the mesh by subdividing each triangle leads to all Gaussian curvature being concentrated in ever decreasing areas, with an infinite Gaussian curvature density in the limit of the Halma subdivision index $H \to \infty$. This reduces the accuracy of the kinetic energy operator, whose contribution makes up a substantial part of the total energy of the molecule. Further, common mesh fairing techniques are not applicable: They depend explicitly on the 3-dimensional geometry of the mesh, and they are not guaranteed a "nice" distribution of Gaussian curvature.

This then leads us to the central problem statement of this thesis: How do we develop a method for constructing high resolution meshes for polyhedral molecules, which works directly with the combinatorial geometry of the intrinsic, non-Euclidean 2-dimensional manifold level, and gives us a "nice" distribution of curvature?

Here we propose to turn the mesh fairing on its head: instead of manipulating the geometry in the hopes that it will generate a favourable distribution of curvature, we will instead start with the curvature, "smear" it over the manifold and work our way backwards to calculate the resulting manifold. In effect, we treat the curvature of each vertex as a parameter which we are free to adjust as we see fit (provided of course, that the result works).

Conceivably, this procedure could be done on the whole molecule. But all curvature is concentrated around the pentagons in the bond graph, and every pentagon in the original mesh is congruent with each other. We can therefore cut all 12 pentagons out of the mesh, smear the curvature out on just one of them, and then stitch this new smeared pentagon back into the mesh, replacing the 12 original pentagons.

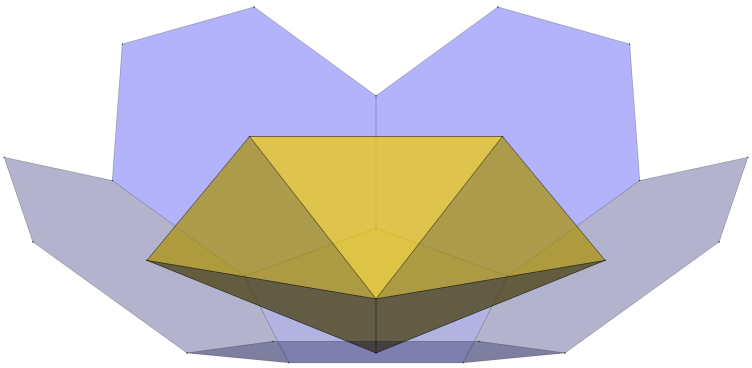


Figure 4.1: A pentagon surrounded by 5 hexagons on the primal mesh (blue), and the regular pentagonal pyramid resulting from connecting their centres (yellow).

There is however a caveat to this: if the two or more pentagons are adjacent, then we cannot necessarily just cut out the 12 individual pentagons, and the computations will therefore be more involved. For this reason we will restrict ourselves to IPR fullerenes, the most chemically stable class. All 12 pentagons of the manifold are therefore always separated, and fused pentagons are relegated to future work.

Since we are working with the dual representation of the fullerene, the pentagon of the primal representation, along with its 5 neighbouring hexagons, will constitute a regular 5-sided pyramid in the dual representation (see figure 4.1). The area of the regular pentagonal pyramid which corresponds to the original pentagon will then be the Voronoi cell for the vertex at the top of the pyramid (see figure 4.2. The Voronoi region is plotted in green, and corresponds to the original pentagonal face). It is thus this area on which we want to smear the curvature. A further simplification comes from the symmetry inherent to the pentagonal pyramid: Each triangle of the pyramid is congruent with each other. So in theory, we could even confine ourselves to smearing the curvature on a single side of the pyramid (given appropriate boundary conditions, of course). The full process is illustrated in figure 4.3, with an isometric embedding of the dual manifold of $C_{320} - I_h$. Here the pentagonal pyramids are first identified and cut out of the mesh (shown as an exploded view on top right). Then, since each pentagon is congruent with the others, we smear the curvature over just a single pentagon (represented in middle right as a red pentagon). The 12 pyramids that have been cut out are then switched for the smeared variant, and stitched back together, to form a manifold where only the pentagonal pyramids have been smeared. We note, that while the figure is showing a full isometric embedding for the sake of intuition, this is not how the algorithm will work: We keep everything on the manifold level to avoid any costly calculations of 3-dimensional geometry. Because each side is congruent with each other, we will henceforth denote each side of the pentagonal pyramid (along with any subdivided triangles and vertices on it) as a *sector* of the pyramid. Each pyramid then consists of 5 congruent sectors.

A problem arises in this view, however, since the subdivided triangles resulting from the mesh refinement do not necessarily fit neatly into one Voronoi cell or the other. See for example figure 4.4. Here some vertices lie on the boundary of the Voronoi cell, whilst others

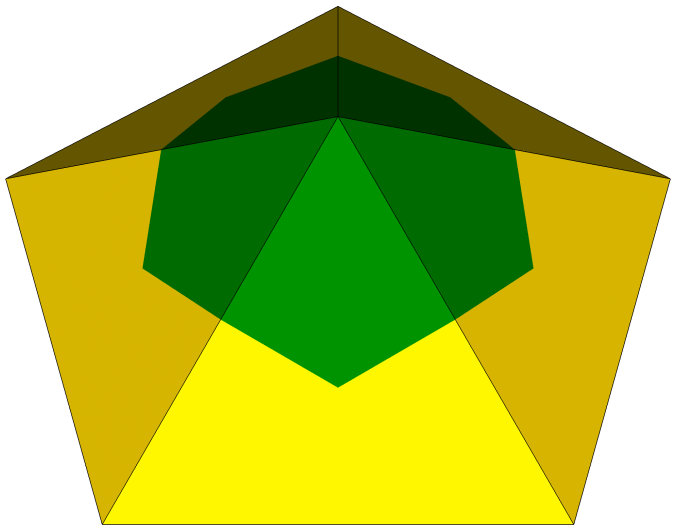


Figure 4.2: A pentagonal pyramid with the Voronoi region of the pentagon vertex superimposed in green.

do not. For us to be able to treat the pentagons separately from the hexagons, we need to make sure to only smear curvature over the area of the dual manifold corresponding to the pentagons: that is, only the vertices inside the original Voronoi area of the pentagonal vertex can be non-zero. Any vertex outside of this must have zero curvature, since it is a part of a hexagonal Voronoi region.

To achieve this, we need to make sure that all neighbours of vertices in hexagonal Voronoi regions (the grey vertices of figure 4.4) must lie on the original, unsmeared manifold. Some of these neighbours will be in hexagonal Voronoi regions (other grey vertices) while some will be in pentagonal Voronoi regions. This is the case for red vertices that are a part of triangles that have been shaded yellow in figure 4.4. These are either within the pentagonal Voronoi region, or on its boundary, but have a neighbour *outside* of it.

The easiest way to make sure these vertices stay on the unsmeared manifold is to demand that every triangle with at least one vertex outside the Voronoi cell must stay equilateral, with a side length of $1/H$ (remember, the original side length of our triangles is unity, and a Halma subdivision of H splits each edge into H new ones). The consequence is thus that any vertex inside the Voronoi cell, with at least one neighbouring vertex outside, must stay fixed. This does not necessarily mean that the same vertex cannot have a non-zero curvature. Remember, the curvature of a given vertex depends on the relative positions of the vertex itself and its neighbours.

To stitch the mesh back together (provided that we have calculated a satisfactory curvature distribution, and computed the resulting manifold) we need to update the edge lengths and angles of every triangle in the mesh which is not fixed. That is, every triangle corresponding to the green-shaded triangles in figure 4.4: replacing the default value of $1/H$ and $\pi/3$ respectively, with what has been calculated.

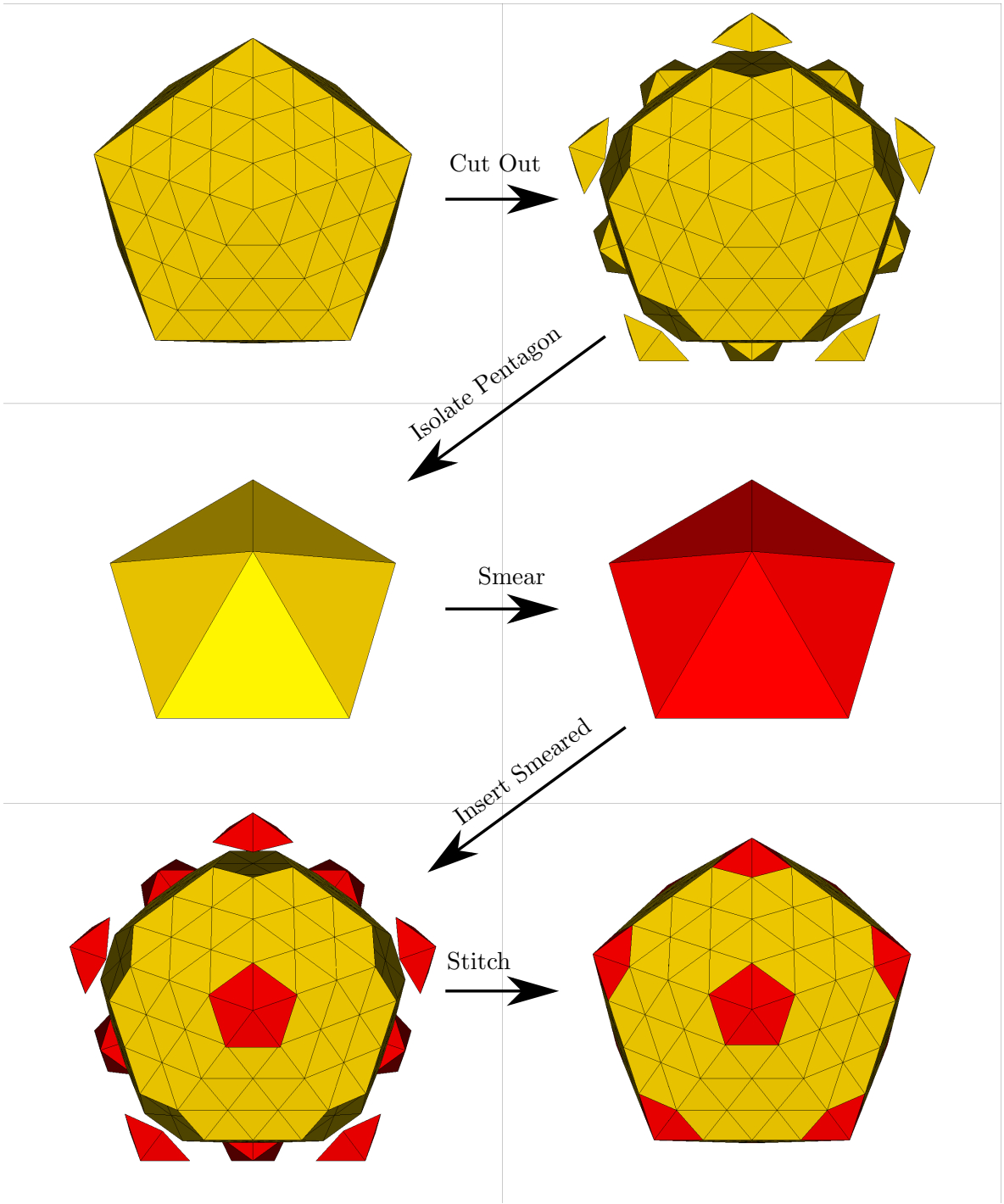


Figure 4.3: Illustration of the full smearing procedure, on the isometric embedding of the $C_{320} - I_h$ dual manifold. First the pentagonal pyramids are "cut out" of the manifold. Then a single pentagon is smeared (represented as a red pentagonal pyramid). The cut out pentagons are then replaced by the smeared version, and the whole mesh is stitched back together. Note, that algorithmically, this is all done on the manifold level, and not on the isometric 3-dimensional embedding.

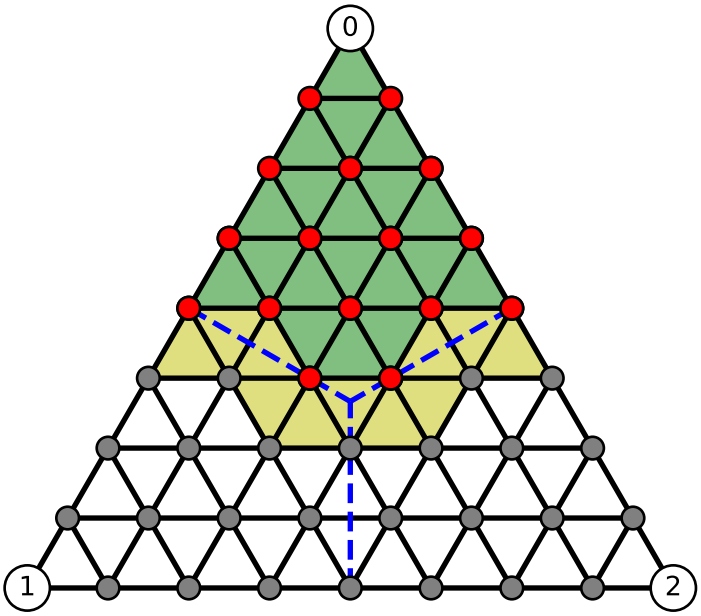


Figure 4.4: A single sector of the computational domain. Vertices in red (and vertex 0) are a part of the computational domain: the pentagonal Voronoi region. Faces shaded in green are part of the domain. Faces shaded in yellow include parts of the basis functions for vertices on the border, and must be included when calculating total curvature.

4.2 The Computational Domain

We have shown that it is possible to cut out the 12 identical pentagonal pyramids, and we now wish to smear the curvature on them. To do this, we need to know for exactly which triangles and vertices on the pyramids 5 sectors we are allowed to smear the curvature. This is the computational domain. It consists of all vertices within or on the pentagonal Voronoi region (red shaded vertices in figure 4.4), and all dual faces which include only red vertices (the green shaded triangles).

We now need a way to determine programmatically which vertices are in the Voronoi region. Due to the IPR-requirement, we can use a single mask to select vertices on all the sectors. To create this mask, we consider the Eisenstein coordinates of a prototypical sector, and calculate the conditions under which a given vertex is within the pentagonal Voronoi region. We assume a Halma subdivision index H, and assign the coordinates $(0,0), (1,0), (0,1)$ to the corner vertices. We then need to figure out which part of the Eisenstein plane is within the Voronoi region of the first vertex $(0,0)$. We do this by using barycentric coordinates, since these relate positions to the corners of a triangle, instead of some external set of coordinates. And crucially, an affine transformation of one triangle to another does not alter barycentric coordinates.

So we can affinely transform the equilateral triangle of figure 4.4 to one where the barycen-

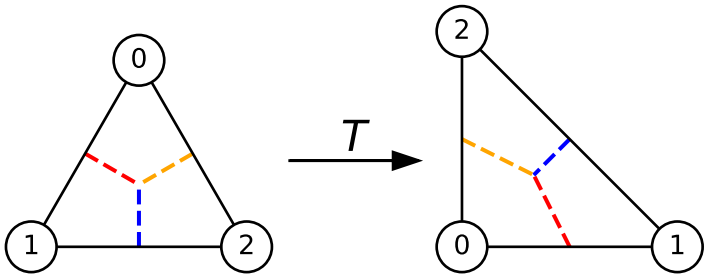


Figure 4.5: An equilateral triangle and its Voronoi cells affinely transformed to the prototypical isoceles triangle. The orange boundary is described by $f(x)$ and the red by $g(x)$ in equation 4.1

tric coordinates align with the Eisenstein coordinates of the points, which is the triangle defined by the points $(0,0),(1,0),(0,1)$. (the setup is seen in figure 4.5). We can then relate these to the ragged indices of our data structure (to be defined in section 7.1), allowing us to determine whether a given vertex, in a given sector, is part of the computational domain.

On an equilateral triangle the circumcentre and barycentre coincide, so the Voronoi regions for the three corners go from the middle of each side to the barycenter. On the transformed triangle this defines the Voronoi region as bounded by the points:

$$(0,0) \to \left(\frac{1}{2},0\right) \to \left(\frac{1}{3},\frac{1}{3}\right) \to \left(0,\frac{1}{2}\right) \to (0,0).$$

The first and last boundary is accounted for in the Eisenstein coordinates of the vertices already (as in, the Eisenstein coordinates of vertices in this sector have $x > 0$ and $y > 0$), so only the two remaining boundaries need to be checked. The conditions for being in the Voronoi region is then

$$y \leq f(x) = \frac{-x+1}{2} \quad \wedge \quad y \leq g(x) = -2x+1 \tag{4.1}$$

Or in the ragged coordinates of sector 7.1:

$$i + j \leq H, \quad 2i - j \leq H.$$

We can then perform these checks for all ragged indices of a sector to create a mask, and apply this mask to all sectors in the mesh. The simplest way to get the resulting triangulation is then to take the full triangulation of the pentagonal pyramid, and simply check which vertices are in the mask. That is, if they obey the conditions of equation 4.1. We then only keep the triangles whose vertices are in the mask.

The method of identifying vertices, as defined above, has a drawback. It leaves "gaps" in the list of vertex indices. Say the original triangulation consists of 5 triangles, arranged in a pentagonal pyramid, with vertex 0 being the pentagon vertex. Then vertices 1 through 5 will all be outside the Voronoi region of vertex 0. This will cause an error in the FEM calculation, since it assumes that the vertices are labelled from 0 through $N-1$, where N is the number

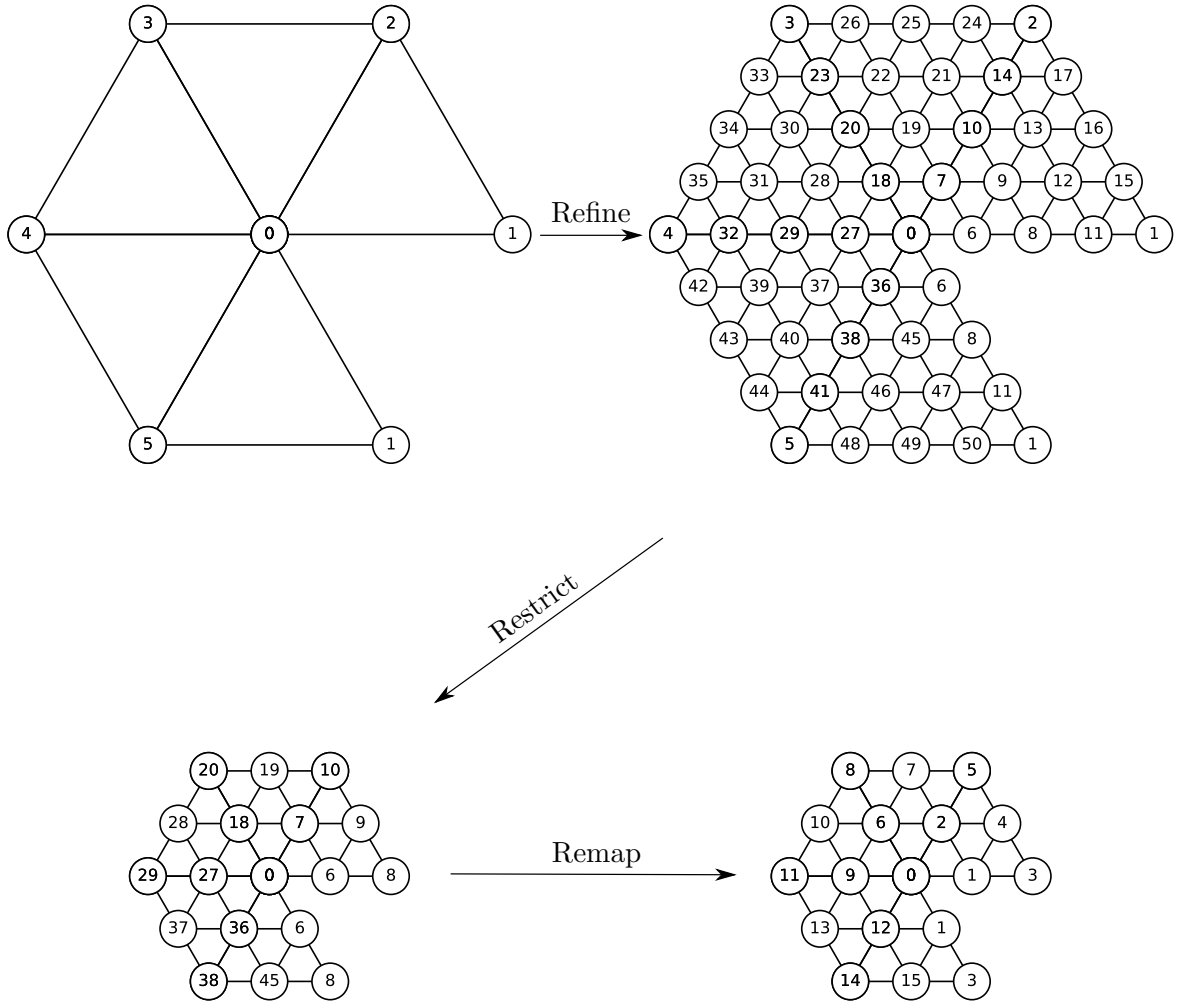


Figure 4.6: The process of restricting the triangulation of the pentagonal pyramid to the computational domain. First the mesh is refined (top left to top right), in this case with $H = 4$. Then the mesh is restricted by leaving out any vertices whose Eisenstein coordinates do not fulfil equation 4.1 (top right to bottom left). This leaves gaps in the indices, so we therefore remap the remaining indices from 0 to n (bottom left to bottom right)

of vertices in the triangulation. To rectify this, we create a mapping from the vertex indices of the full triangulation to the restricted triangulation, by noting the index of every vertex in the Voronoi region, when applying the vertex mask.

We can then check if a given dual face is in the pentagonal Voronoi region by looking at the ragged indices of its vertices. When we have the global indices of every vertex in the mask, we can assign them new indices local to the computational domain. The full process of obtaining a triangulation of the computational domain, from the unrefined pentagonal pyramid is seen

in figure 4.6 and an implementation is given in algorithm 8. With this triangulation of the computational domain, we can then construct the matrices needed using the FEM software, and solve the heat equation.

4.3 Desirable Curvature Distributions And Where To Find Them

We now know precisely how to generate a triangulation of the computational domain, the area of our manifold where we can smear the curvature. We are then left with two sub-problems to deal with:

1. How do we smear the curvature on this computational domain? That is, how do we compute a "nice" curvature distribution?

2. With a given curvature distribution, how can we compute the resulting manifold?

We will tackle these in the order they are presented. For the curvature distribution problem, we first need to define what a "nice" curvature distribution entails. Primarily, we have two criteria which the original distribution and manifold obey, and we therefore want the smeared manifold to obey them as well:

1. The total curvature in the Voronoi cell must equal $2\pi/6$, the curvature induced by the original pentagon.

2. The curvature must obey the same symmetry as the pentagonal pyramid, namely the D_5 symmetry group.

The first criterion should be self evident - we do not want to add or subtract from the overall curvature. The second is a statement of the fact that we are trying to refine an already existing surface, and as such should not change the symmetries inherent to the mesh.

With these rules we need a method for generating an acceptable distribution. In the continuous and flat space case, the heat equation obeys all three rules: it is a conservative equation, is invariant under orthogonal transformations (therefore including all symmetry operations of the D_5 group, and a function with initial conditions obeying D_5 will then also obey D_5), and for a delta function initial condition, the solution is a Gaussian, which decreases monotonically with the distance from the centre.

The proof of the second criterion for the flat space Laplacian is quite simple: given an orthogonal coordinate transform $\mathbf{Q}$ such that $\mathbf{x}' = \mathbf{Q}\mathbf{x}$, then the del operator transforms likewise: $\nabla' = \mathbf{Q}\nabla$, where the prime indicates differentiation with respect to primed coordinates. The Laplacian is $\nabla \cdot \nabla = \nabla' \cdot (\mathbf{Q}^T\mathbf{Q}\nabla') = \nabla' \cdot \nabla'$. A sufficient condition for criterion two in the discrete case is that the computational mesh and initial conditions both obey the D_5 symmetry.

We can then use the finite element method to solve the heat equation on the computational domain, terminating when the curvature has been smeared out to a desirable degree.

Chapter 5

Smearing Curvature, part 1

A Heat Based Curvature Flow

In this chapter we introduce the first method for smearing the curvature about the computational domain: solving the heat equation using the finite element method. We will define the heat equation and see how to solve it within the framework, and then we will analyze the results to see just how well it performs at the objectives outlined in chapter 4.

5.1 The Heat Equation In The Finite Element Framework

The heat equation with appropriate boundary conditions, such that it is able to represent a valid curvature distribution, can be written as:

$$\dot{u} = c\Delta u,\ \forall u \in \Omega. \quad u = 0,\ \forall u \notin \Omega. \quad \int_\Omega u\, d\mathbf{x} = 2\pi/6$$

with $c > 0$ being a constant, Ω being the chosen domain. The first step is representing the heat equation in weak form. We use a forward finite difference discretization in time:

$$\dot{u}(t) \approx \frac{u(t+\delta) - u(t)}{\delta}$$

On the RHS of the heat equation we use $u(t+\delta)$ to make for an implicit integration scheme. With this we can properly recast the equation in weak form. We start with the flat space version. First we separate the two iterates $u(t)$ and $u(t+\delta)$, multiply by the test function and integrate:

$$\int_\Omega vu(t)\ \mathrm{d}\mathbf{x} = \int_\Omega vu(t+\delta)\ \mathrm{d}\mathbf{x} - c\delta \int_\Omega v\Delta u(t+\delta)\ \mathrm{d}\mathbf{x}.$$

Next, we use the Ritz-Galerkin approximation to get

$$\sum_j v_j \int_\Omega \phi_j u(t)\ \mathrm{d}\mathbf{x} = \sum_j v_j \left[\int_\Omega \phi_j u(t+\delta)\ \mathrm{d}\mathbf{x} - c\delta \int_\Omega v\Delta u(t+\delta)\ \mathrm{d}\mathbf{x}\right],$$

resulting in the set of linear equations:

$$(\mathbf{M} + c\delta\mathbf{L})\mathbf{u}(t+\delta) = \mathbf{M}\mathbf{u}(t)$$

where

$$\mathbf{M}_{i,j} = \int_\Omega \phi_i(\mathbf{x})\phi_j(\mathbf{x})\, d\mathbf{x}, \quad \mathbf{L}_{i,j} = \int_\Omega \nabla\phi_i(\mathbf{x}) \cdot \nabla\phi_j(\mathbf{x})\, d\mathbf{x}$$

are the mass and stiffness matrix for flat space, respectively. For our purposes, we do not need to know the specific time at which a given solution occurs, and so without loss of generality we set $c = 1$, to give a single control parameter.

For the initial conditions we need to use the starting curvature distribution. In the continuum case (corresponding to infinite refinements), we have a singularity of curvature at the centre vertex - a delta function with scaling such that the distribution is properly normalized. In our case of piecewise polynomial functions, this is not representable. The closest is to set $\mathbf{u}(0) = 0$ everywhere other than the centre vertex.

Boundary Conditions

This just leaves the boundary conditions. Here we need to keep $\mathbf{u}(t) = 0$ for all t. A naive way would be to clamp the right hand side. Say vertex i' is on the boundary, then we set row i' of the LHS matrix $\mathbf{A} = (\mathbf{M} + \delta\mathbf{L})$ to

$$\mathbf{A}_{i',j} = \delta_{i',j},$$

and on the RHS we set $\mathbf{y}_{i'} = (\mathbf{Mu})_{i'} = 0$ at every time step. However, this will break the first rule we put in place, and decrease the total curvature, once it has propagated to the boundary.

We are in luck however, since the boundary condition is $u = 0$. We can then just exclude any fixed vertices from the domain (grey vertices in figure 4.4), since this implicitly sets $u = 0$ for these vertices. Then rule 1 is still obeyed, if the integration scheme is conserving.

5.2 Calibrating the Heat Based Flow

In figure 5.1 a set of key frames from a run of the heat equation simulation using the FEM software is shown, using the flat space Laplacian. We see a set of negative curvatures appear, even after a single iteration (plotted as red vertices. Non-negative curvatures plotted in blue). These form "waves" that propagate outward from the centre vertex, until they reach the edge of the mesh and disappear.

To investigate whether this is an artefact of the discretization, we perform the same simulation with different time steps, spanning several orders of magnitude, and plot the minimum curvature for each time step. The result is seen in figure 5.2. We see that the waves all appear at the same time (barring the first simulation, where the time step is so large that the first iteration is visibly later than for the other time steps), and more crucially, leave at the same time.

The problem is exacerbated with increasing mesh resolution, as can be seen in figure 5.3, where the Halma index was halved, from 16 to 8. The same trend appears (waves appearing and disappearing at the same time), but on a shorter time scale.

With this we see that on long enough time scales the non-negativity condition is upheld. We next turn to the condition of curvature conservation. We now run the simulation in two different cases. One simulation using the flat space Laplacian and one using the cotan Laplacian. In both cases we use 5 sectors. For both simulations we plot the total Gaussian

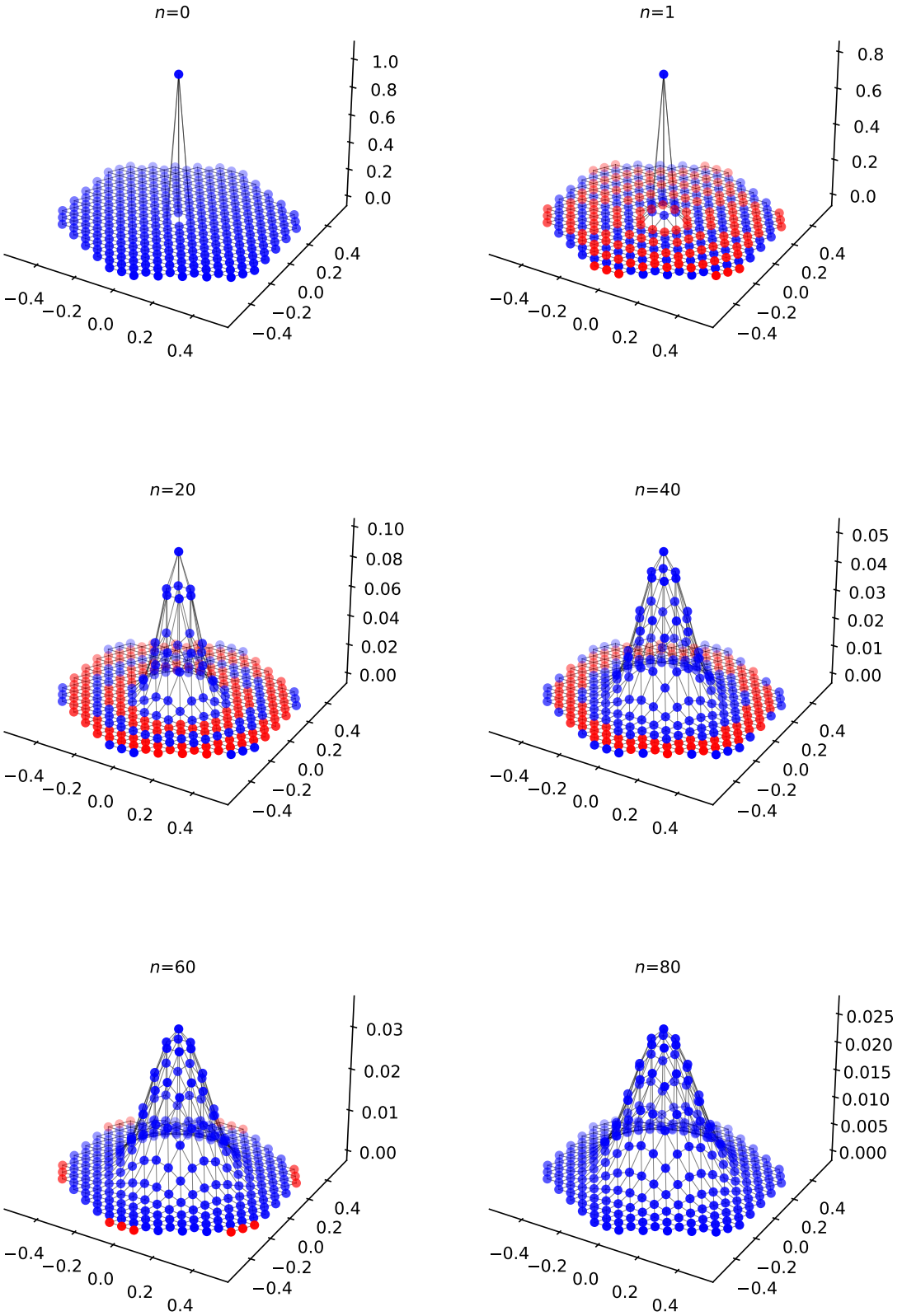


Figure 5.1: Key frames for a typical FEM heat equation simulation. Here $H = 16$, $\delta = 0.1$ and the simulation was run for 80 iterations. Vertices shown in blue have a non-negative curvature, while those in red have negative curvature. Note how the negative curvature appears as radial waves after the first iteration, and it takes quite a large number of iterations for them to disappear, and as a result, the final curvature distribution is quite flat.

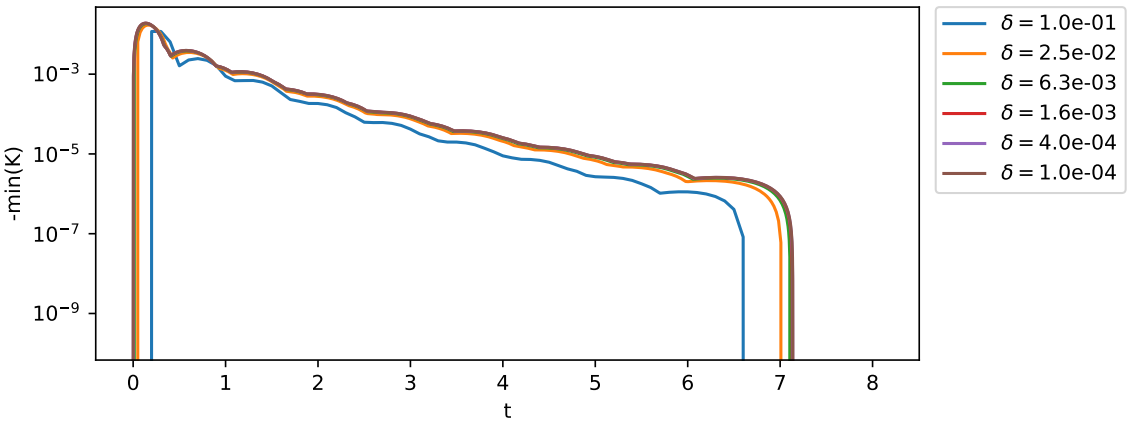


Figure 5.2: The minimum curvature plotted for each time step, for several values of δ. For all simulations $H = 16$. Note that the time it takes for the negative curvatures to leave the mesh is roughly constant.

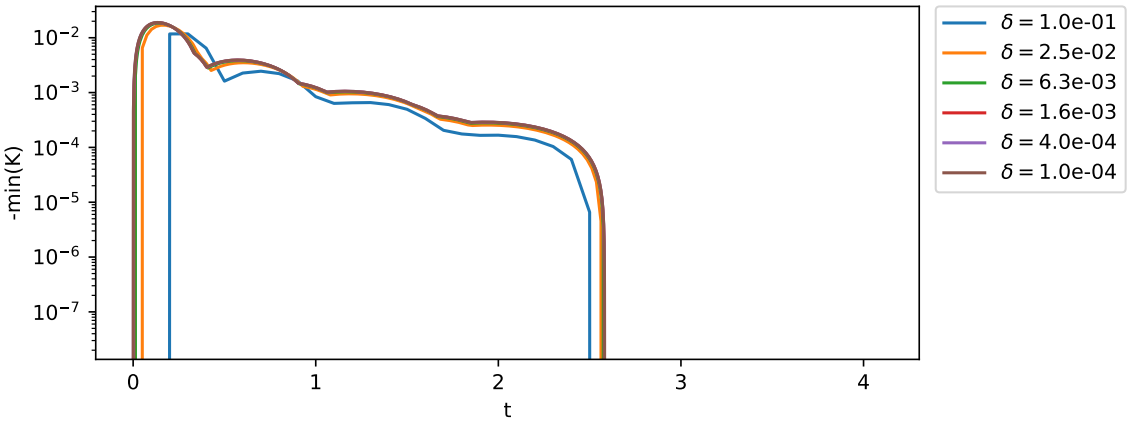


Figure 5.3: The same experiment as in 5.2 but with $H = 8$. Note again that the time it takes for the negative curvatures to leave the mesh is roughly constant.

curvature per sector and subtract the expected curvature, such that any curvature excess or deficit is highlighted directly. At the same time, we plot the difference in minimum and maximum curvature, to show how close the simulation is to the asymptotic solution of a constant distribution. The result of this is seen in figure 5.4. As expected, the flat space Laplacian is perfectly conserving, even on the non-embeddable geometry of 5 sectors.

The cotan Laplacian however, is not conserving. We see a steady increase in total curvature, even as the simulation approaches convergence. A curious thing happens on larger time scales - the method is unstable. This is seen as the sharp increase in difference between maximum and minimum value, along with a steep increase in total curvature. If we are to use the cotan Laplacian then, we will need to address these two problems. The fixes, however, are easy: non-conservation of curvature is fixed by normalizing the distribution after each time step, while the instability just sets a maximum simulation time.

In summary, the FEM heat method works, but there are limitations: The negative cur-

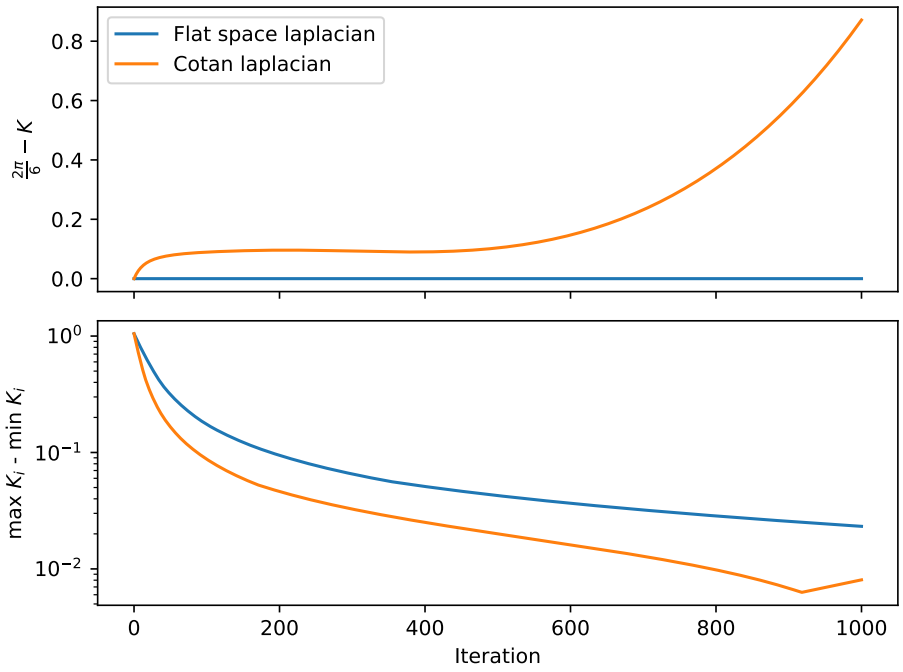


Figure 5.4: The expected total curvature and the span of values for the FEM heat equation simulation, plotted as a function of iterations, for two cases. In blue is shown the flat space Laplacian, and in orange the cotan Laplacian. Note the flat space Laplacian is conserving, and the span of values decays slower than for the cotan. In contrast, the cotan Laplacian is non-conserving, and appears to grow unbounded. Even more peculiar, the span seems to start to grow after a time!

vatures could be alleviated using a constant curvature offset and then renormalizing the distribution:

$$u_i \to u_i + \min_i u_i \to \frac{2\pi}{6} \frac{u_i}{\sum_j u_j}$$

But this does not get rid of the radial oscillations in curvature. As such there is a limit on how steep a curvature distribution we can attain, since a large time has to pass before the curvature is both strictly non-negative and monotonically decreasing with the distance from the pentagonal vertex.

Chapter 6

Smearing Curvature, part 2

A Graph Based Laplacian

In this chapter we will introduce the second method for smearing curvature, by defining a Laplacian directly on the graph of the refined mesh. By imposing conditions on the operator we are able to assure that it obeys the symmetries of the mesh and does not produce negative curvatures.

6.1 What is in A Laplacian?

If we forego the finite element method for discretizing the Laplacian, and focus instead only on using the graph structure and a couple of rules and heuristics, we can define a Laplacian which makes the problem quite a bit simpler. Say we want an explicit updating method, and still use a finite difference for the time discretization:

$$\mathbf{u}(t+\delta) = \mathbf{u}(t) + \delta\mathbf{L}\mathbf{u}$$

We can now define a Laplacian matrix $\mathbf{L}$ acting upon the vector of curvatures, defined on the vertices through the angle defect. Again we want the three main rules to hold: symmetry, non-negativity and conservation. Conservation is easy to see: we want the sum of curvature on each side of the equation to be equal, resulting in:

$$\sum_i (\mathbf{L}\mathbf{u})_i = \sum_j u_j \sum_i L_{ij} = 0$$

To achieve this for all $\mathbf{u}$, we demand that all columns in the Laplacian matrix sum to zero:

$$\sum_i L_{ij} = 0, \ \forall j.$$

Next we use the observation that the Laplacian of a point is the average of those around it [20], which together with the conservation-criterium leads to an obvious candidate matrix:

$$L_{i,j} = \begin{cases} 1 & \text{if } j \in r_1(i), \\ -\deg(i) & \text{if } j = i, \\ 0 & \text{else.} \end{cases}$$

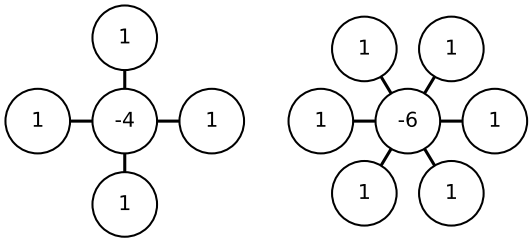


Figure 6.1: Stencils used for the Laplacian. On the left is the stencil used for the Laplacian in the finite difference method on a uniform grid. On the right is the stencil for the graph-based Laplacian for a hexagonal vertex.

where $\deg(i)$ here is the degree of the vertex on the "restricted" mesh, and so is the total number of neighbours, minus the number of fixed neighbours. This form is also reminiscent of the finite difference method stencil used for the flat space Laplacian in two dimensions (scaled, and assuming a uniform grid). The stencil is shown for the cases of $\deg(i) = 4$ (the FDM case) and $\deg(i) = 6$ (the hexagonal case) in figure 6.1

This is incidentally the exact same form as the cotan Laplacian defined on the unsmeared fullerene graph! We can further show that this matrix also has the required symmetry: if $\mathbf{u}$ possesses D_5 symmetry, then any operation from that symmetry group just amounts to a certain permutation of the vertices. For rotations this permutation is such that the ragged vertex indices of a given vector is the same before and after the operation (though of course, the sector which a given vertex is placed in has changed). For reflections we have $j \to -j$, while i remains fixed. With this definition of the Laplacian, the transformation matrices $\mathbf{P}$, corresponding to the group operations, leave the Laplacian fixed:

$$\mathbf{P}(\mathbf{Lu}) = \mathbf{L}(\mathbf{Pu}) = \mathbf{Lu}$$

where the last equation follows from the requirement that $\mathbf{u}$ possesses D_5 symmetry.

Making sure the Laplacian only gives positive values for a vector with non-negative values amounts to a scaling of the Laplacian, or equivalently a restriction on the time step. The condition is:

$$u_i(t + \delta) = u_i + \delta(\mathbf{Lu})_i \geq 0, \forall i$$

The "worst case" happens if all curvature is concentrated in a single vertex i', whereupon

$$u_{i'} - \delta \deg(i')\, u_{i'} \geq 0 \quad \Leftrightarrow \quad \delta \leq \deg(i')^{-1}.$$

So as long as we set $\delta < (\max_i \deg(i))^{-1}$ we fulfil the non-negativity condition.

6.2 Analysis

We first show the results of a typical simulation in figure 6.2. We here choose a mesh with 6 sectors, but note that the results would be exactly the same with a different number of sectors, provided the initial condition was scaled properly, as described in section 4.2. Qualitatively

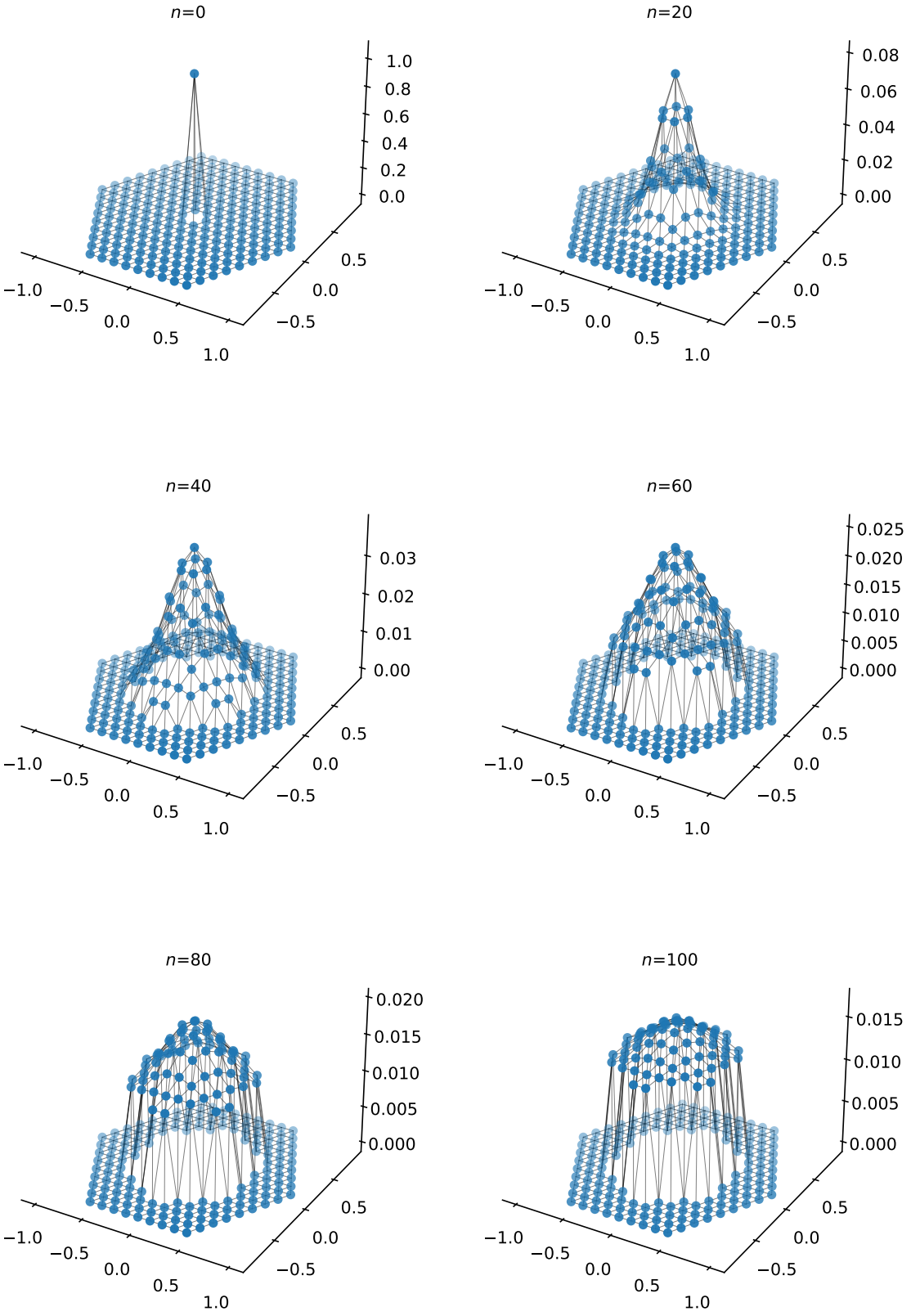


Figure 6.2: Typical simulation of heat equation with the graph based Laplacian. Here $H = 8$ and $\delta = 30^{-1} = (5 \max_i \deg(i))^{-1}$. Note the scale of the z-axis changes with the iteration number to highlight the relative differences in values for each iteration.

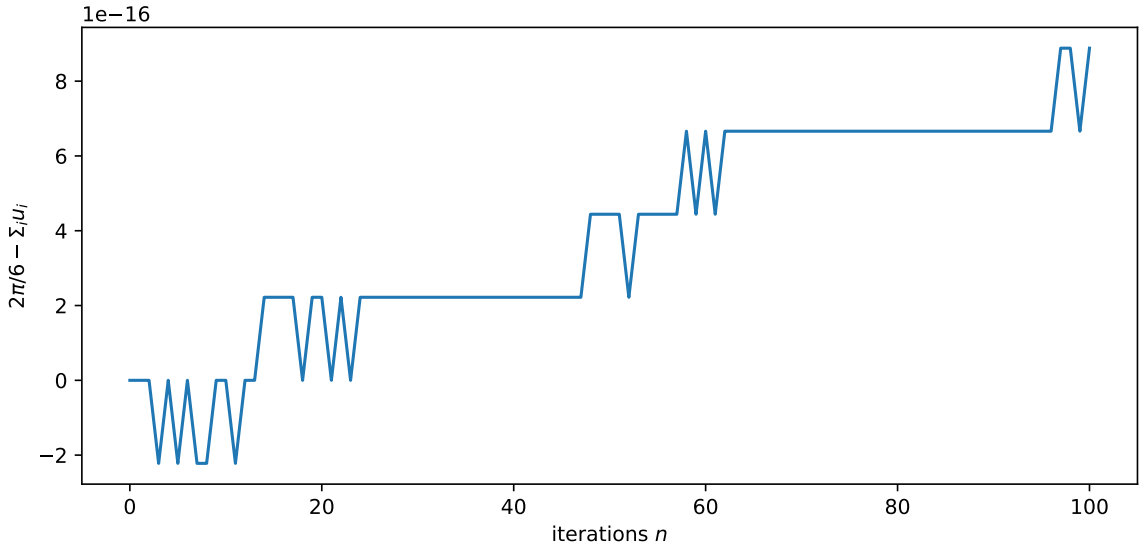


Figure 6.3: Difference between expected total curvature and actual total curvature plotted for each iteration of the simulation. Parameters are the same as in 6.2.

the results look as expected: every non-fixed vertex has a positive curvature and the solution has the right symmetry.

Next we check the total curvature is constant. The result is seen in figure 6.3, and indeed, the total curvature is constant, up to rounding errors.

For good measure, we check the monotonicity of the solutions by plotting the value of each vertex as a function of their Euclidean distance from the centre vertex. The result is shown in 6.4, and we see that sometime between 60 iterations and 80 iterations, the monotonicity is broken, if only slightly. This is in contrast to what we might expect in the flat, continuous two dimensional case, where starting with a delta function initial condition yields a Gaussian:

$$u(\mathbf{x}, t) = \frac{1}{4\pi t} \exp\left(-\frac{\mathbf{x} \cdot \mathbf{x}}{4t}\right).$$

Of course, our setup is different from the case of a flat Euclidean plane, so we might expect some deviation in behaviour.

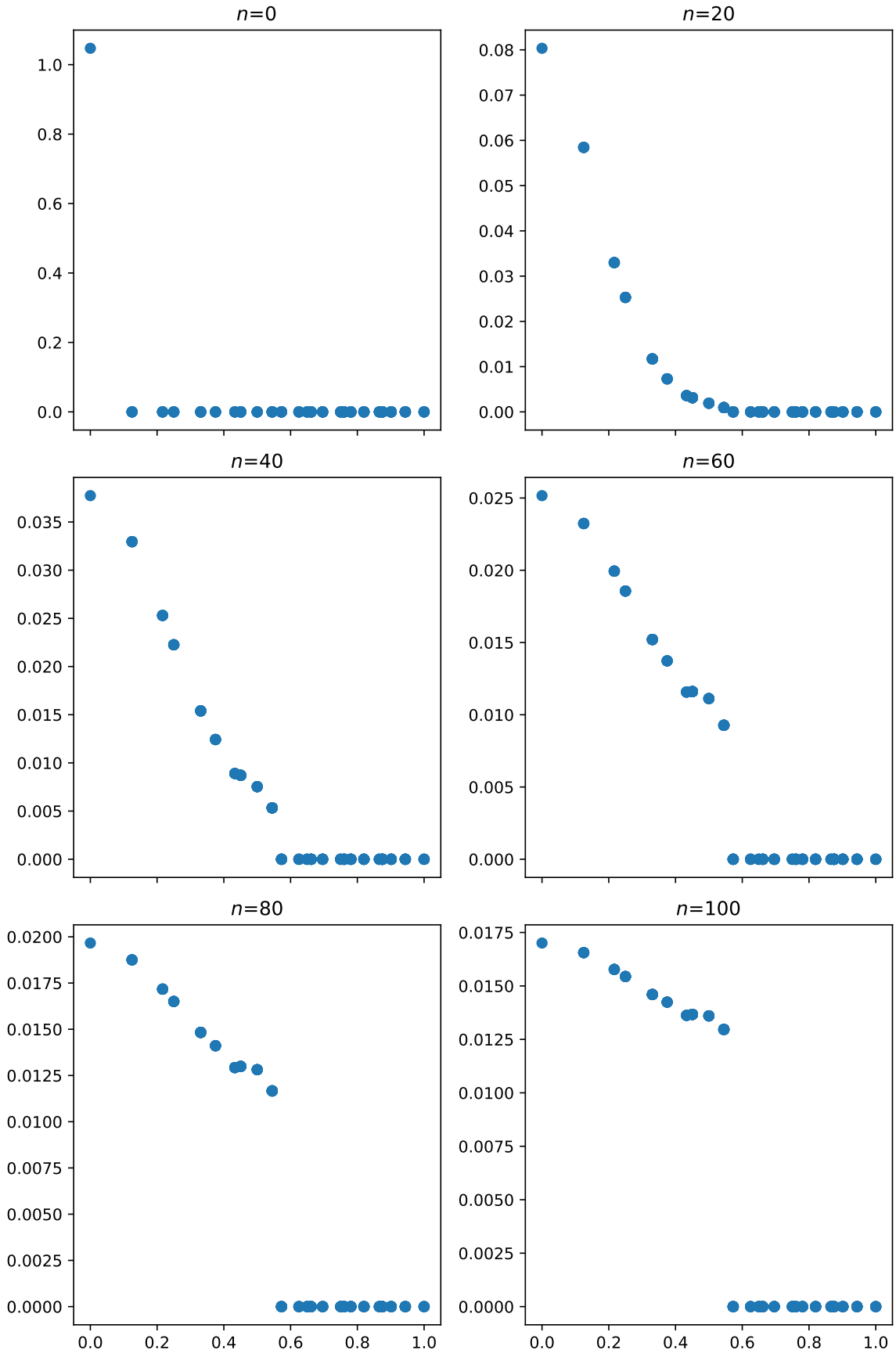


Figure 6.4: Curvature plotted as a function of Euclidean distance from the centre. Parameters are the same as in 6.2

Chapter 7

From Curvature to Manifold

A Journey to the Angles and the Lengths

With a suitable distribution of Gaussian curvature over the pentagon Voronoi region, we need to reconstruct the manifold geometry. The first step is to calculate the new angles. The discrete Gaussian curvature is linear in each angle, and so is the angular sum of the triangular faces. These define the "vertex" and "triangle" equations. Given the reflection symmetry of the sector (the 5 identical sides that make up a pentagonal pyramid), we can also set up a number of "reflection" equations. We can then setup a system of linear equations to solve for the angles in the Voronoi regions. We focus on just a single sector in the region to simplify calculations.

After calculating the angles, we just need a single known edge length on the manifold to set the scale of the mesh. From there we can use the law of sines to calculate the rest of the edge lengths in the system, thereby reconstructing the full manifold.

So far, however, we have played fast and loose with our data, and just assumed that we had a subdivided manifold upon which we could solve the heat equation and smear the curvature. Let us therefore take a detour into the realm of data structures, so we can define one which suits our needs.

7.1 Data Structures

For the implementation of these algorithms, we want data structures that facilitate easy calculation. The original data structures, in particular the sparse adjacency matrix and the triangulation, are great for navigating the mesh and computing quantities using the finite element method. Since these are still important, we need to construct these same structures after refinement. Further, for the algorithms described above to work (particularly the geometry reconstruction), we want a data structure that, in some way, encodes the reflection symmetry of the subdivided mesh.

For this we store the graph positions of the vertices, relative to the original triangles, in a 3-tuple, called the vertex sector tuple. The first index of the tuple indicates which original triangle (ie sector), the vertices are a part of. The second index specifies the ring number, relative to the first vertex of the sector. The last index specifies the graph distance along the ring, relative to the edge of the original triangle. The length of the third index is variable,

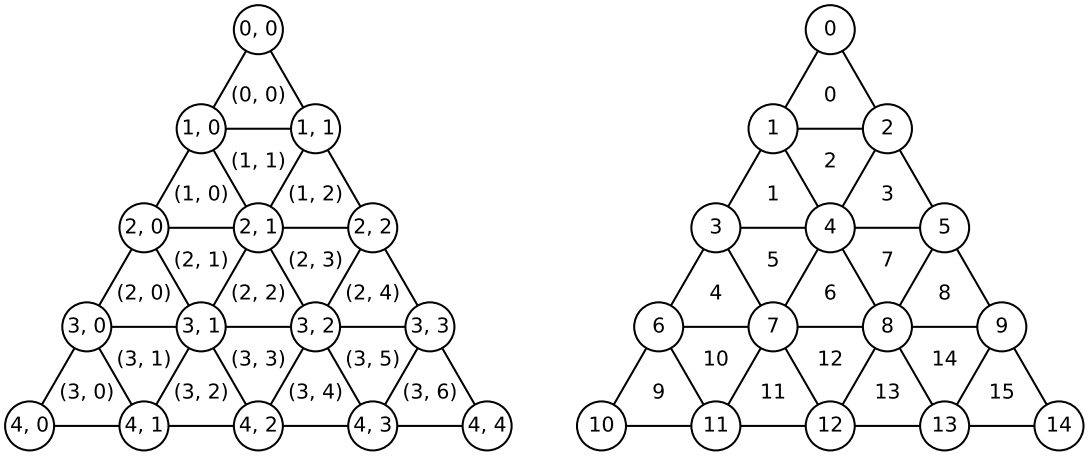


Figure 7.1: A mesh sector, after a refinement with $H = 4$. On the left is the ragged indices for both vertices and triangles. On the right is the corresponding flat indices.

since the number of vertices in a ring is dependent on the ring number. This can be remedied by combining the second and third index from a pair of "ragged" indices, into a running "flat" index, similar to how one would flatten a matrix into a vector. We keep the 3-index structure for illustrative purposes, however. The resulting structure for a given sector, after a subdivision with $H = 4$, can be seen in figure 7.1, with both the "ragged" and "flat" indices. Note that only the second and third index (i, j) is shown, as these are indices local to a given sector. In figure 7.2 several features are highlighted: We see how the first index, i, runs *down* the sector, denoting the graph distance from vertex 0 to a given vertex, while the second index, j, runs *across* a given ring, denoting the graph distance from the left edge. As an example, vertices 3-5 (shaded red) all have $i = 2$, since their graph distance to vertex 0 is 2. Vertex 3 is the leftmost vertex, and therefore has $j = 0$, while vertex 5 has a graph distance of 2 from vertex 3, giving it $j = 2$.

Likewise, we store the index of the triangles in another 3-tuple. The concept is the same, except the "ring number" for the triangles, is not the same as the ring number of the primal bond graph. Instead, triangle ring i is "row" i of the triangles, consisting of vertices from vertex ring i and $i+1$. For example, in figure 7.2, triangles 4-8 (in blue) are a part of triangle ring 2, since their vertices are in vertex ring 2 (vertices 3-5, in red) or ring 3 (vertices 6-9).

Indeed, a given triangle in ring i has two neighbouring triangles in the same ring, and another in an adjacent ring. Triangle 6 is in ring 2, and has two neighbouring triangles in the same ring: triangles 5 and 7. Its third neighbouring triangle is triangle 12, which is in triangle ring 3. The third index for the triangle sector tuple is again similar to that of the vertex sector tuple - it describes the (dual graph) distance from the first triangle of the ring.

These structures allows us to utilize the reflection symmetry across a given sector. As seen from figure 7.1, there are $i + 1$ vertices in vertex ring i, and $2i + 1$ triangles in triangle ring i. For a given ring i (vertex or triangle) we can obtain the reflection of vertex (triangle) j across the sector. The reflected vertex has third index $i - j$, while the reflected triangle has

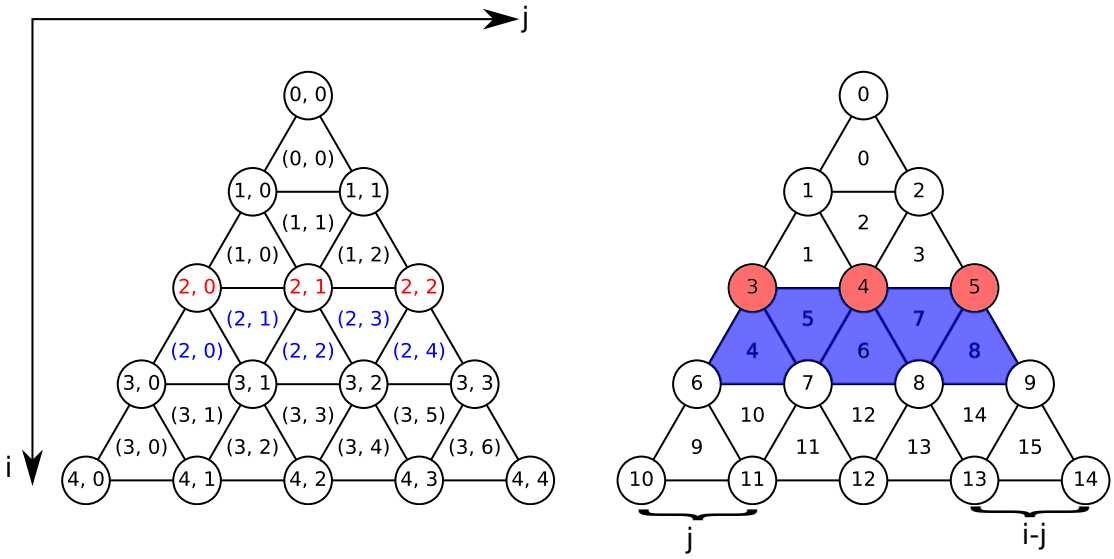


Figure 7.2: Figure 7.1 with added features: The "directions" of the ragged indices i and j, a shaded vertex ring (red) and shaded triangle ring (blue), both with $i = 2$. How to get the reflected vertex is also shown.

third index $2i - j$ - we just step the same graph distance in, from the other edge. Vertex 11 on figure 7.2 has indices $i = 4$ and $j = 1$. Its reflection then has the same second index, but its third index is $i - j$ In this case it has indices $(4, 4 - 1) = (4, 3)$, and is vertex 13.

Constructing the Vertex Sector Tuple

To construct the vertex sector tuple, we go through each original triangle in the triangulation. For each of these triangles, we first place the original vertices. Say that the row n of the original triangulation is (v_0, v_1, v_2), then the indices for v_0 are $(n, 0, 0)$, for v_1 they are $(n, H, 0)$ and lastly, for v_2 they are (n, H, H). Next we need to check if an adjacent sector has been filled in, since the vertices on the edge of a sector is shared with its neighbour.

To facilitate this, we store a dictionary with keys being the original (ordered) edges, and values being all the vertices along the this original edge. When filling in a given sector, with corners (v_0, v_1, v_2), we check the ordered edges $(v_0, v_2), (v_2, v_1)$ and (v_1, v_2), to see if these have been filled in. If they have, we reverse the order of the vertices and fill them into the current sector. With this, each sector has been fully connected to its neighbour (provided that the neighbours have been filled in). This then leaves any vertices not yet updated. Here we run through each ring of the sector, assigning an index to any remaining vertices. The algorithm for a single sector is sketched out further in algorithm 1, and for the full data structure in algorithm 2.

The Triangulation and Triangle Sector Tuple

The triangulation can be constructed by looking at each sector in isolation, since every subdivided triangle is wholly within a single sector. We note again that the triangles of ring i consists of vertices from rings i and $i + 1$. For the first i vertices in ring i, we create 2

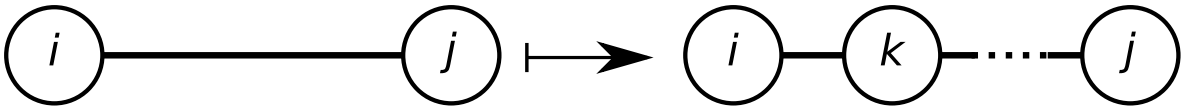


Figure 7.3: Schematic overview of how to fill in the sparse adjacency matrix for original vertices. To the left is a set of connected vertices, before refinement. After refinement, this maps to a series of connected vertices. The arc between i and j has been populated by $H-1$ new vertices. The new neighbour of i in the direction of j is then k.

triangles, and for the last vertex, we create just a single triangle. Appendix A, Algorithm 3 shows an implementation of this. This procedure creates the triangles in the same order as the flat index of the triangle rings. As such, the triangle sector tuple can be easily created along with the triangulation.

The Sparse Adjacency Matrix

When creating the sparse adjacency matrix, we have 3 different cases, depending on where the vertex (i, j) is placed in a given sector. If it is in the inner part, all of its neighbours are in the same sector and have ragged indices as specified in 2.4. We can thus grab them directly from the vertex sector tuple. If the vertex is on the edge of two sectors we have three subcases, depending on which edge the vertex is on. The first time the vertex is encountered, say in sector n, the first three entries of the sparse adjacency matrix is filled in with vertices from n. The second time it is encountered, in sector n' say, the last three entries are inserted with vertices from n'.

If the vertex is on the corner of a sector, we look at the original sparse adjacency matrix, to find its original neighbours, and the corresponding edges. We then take the second entry of each relevant value in the edge dictionary from the construction of the vertex sector tuple. See for example figure 7.3. For the full implementation details see algorithm 4.

The Angle- and Edgelength Arrays

With the above data structues we can now refine the mesh to our hearts content. In these lie the implicit assumptions of equilateral triangles. To smear the curvature we therefore need to specify the angles and edgelengths. For this we use two arrays, of size $N \times 3$, to store these values, where $N = H^2$ is the number of dual faces in a sector. They coincide with the triangulation, such that the angle stored at index i, j, is the angle of triangle i, incident upon the j'th vertex (in local indices). Likewise, the length at index i, j is the length of the opposing edge to the vertex.

These arrays are great for setting up the angular equations and propagating the edge lengths through the mesh, but not great for calculating the weights in the cotan Laplacian. For this, arrays with the same structure as the sparse adjacency matrix are preferable. The algorithm for generating these are seen in algorithm 7.

Retrieving the Old Datastructures

In the vertex sector tuple and new sparse adjacency matrix lies all information needed to retrieve the old data structrures. The old triangulation is easily obtained by going through

each sector and grabbing the corner vertices. Likewise, for the sparse adjacency matrix we can go through the original vertices of the new sparse adjacency matrix (which are the first N_C vertices, where N_C is the number of carbon atoms in the fullerene), and step forward H times, or equivalently, until an original vertex is encountered. To see why this is true, we note that original vertices are always placed at the corners of a sector, and the edges of a sector correspond to an original arc. To get from one corner of the sector to another, we must travel the length of its edge. This is a graph distance of exactly H. See for example figure 7.1, where $H = 4$. To get from vertex 0 to any other corner (vertices 10 and 14), we need to traverse 4 edges.

Summary

A summary of the data structures introduced, along with their sized are seen in table 7.1

Data Structure	Description	Size
Triangulation	Which vertices are parts of which dual face	$N_C H^2 \times 3$.
Sparse Adjacency Matrix	Denotes neighbouring vertices	$((N_C H^2)/2 + 2) \times 6$
Vertex Sector Tuple	Holds vertex graph positions relative to sector	$N_C \times (H+1) \times j$. j dependent on second index.
Triangle Sector Tuple	Holds dual face graph positions relative to sector	$N_C \times H \times j$. j dependent on second index.
Angle array	Stores angles of each triangle in the manifold/sector	$N_C H^2 \times 3$ (full manifold). $H^2 \times 3$ (single sector).
Edge length array	Stores edge lengths of each triangle in manifold/sector	$N_C H^2 \times 3$ (full manifold). $H^2 \times 3$ (single sector).

Table 7.1: Summary of the data structures used

7.2 Angular Equations

We now have a set of data structures which allows us to efficiently construct the set of linear equations governing the distribution of angles in the mesh. The general way we implement them is by setting up the recipe (ie, the formula), which refers to entries in the angle array, so for each equation we define a $(H^2, 3)$ array of the coefficients for the equation at hand. This array is then flattened and appended to a list, and when every equation has been set up, we create a $(N, 3H^2)$ matrix from the list of vectors, where N is the total number of equations. A schematic overview is seen in figure 7.4

Triangle Equations The "triangle"-equations is just stating that we are working with Euclidean triangles:

$$\sum_{j=0}^{2} \theta_j = \pi, \tag{7.1}$$

with one equation for each triangle in the sector.

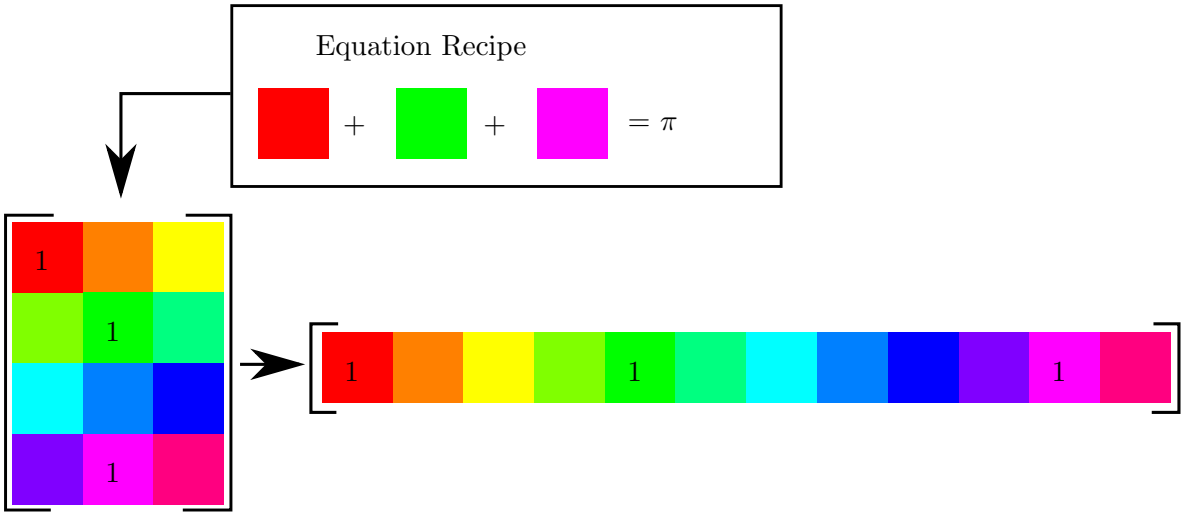


Figure 7.4: Schematic overview of the creation of angle equations: Each colored square corresponds to a single angle in the sector. The rows of the matrix corresponds to the flat index of the triangle, while the columns denote where in the triangle, a given angle is. The "recipe" is in this one where 3 angles must add up to π. For each equation we create a $H^2 \times 3$ matrix storing the coefficients for the equation. In this case, three ones (red: (0,0), green (1,1) and pink (3,1)). This matrix is then flattened out into a $3H^2$ row vector, and makes a single row of the angle equation matrix.

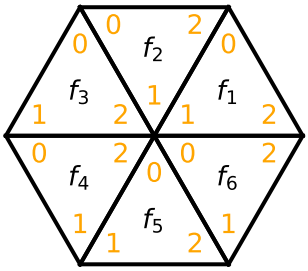


Figure 7.5: Indices used for inner vertex equations. The 6 triangles are shown with their flat triangle indices f_i, and the local ordering of the indices in orange. The row index for the angles is f_i, and the column indices are the integers in orange towards the centre vertex.

Vertex Equations The "vertex"-equations is the Gaussian curvature by way of the angle-defect: $\sum_i \theta_i = 2\pi - K_n$,. Here we have three cases: The first is an inner vertex, where all neighbours of the vertex n is in the same sector. The typical setup is seen in figure 7.5. If the vertex has ragged indices (i, j), then the incident angles will be in the neighbouring triangles, with flat indices $f(n)$, and the angles will be local index $f'(n)$. As such the values of $f(n)$ and $f'(n)$ correspond to a position in the angle array: $f(n)$ corresponds to the *row*, while $f'(n)$ correspond to the *column*. Or equivalently, $f(n)$ denotes which neighbouring triangle the angle is in, while $f'(n)$ denotes which angle of the triangle it is. For any (i, j), the two sets are:

$$f = \{(i-1)^2 + 2j, (i-1)^2 + 2j - 1, (i-1)^2 + 2j - 2, i^2 + 2j - 1, i^2 + 2j, i^2 + 2j + 1\}, \tag{7.2}$$

$$f' = \{1, 1, 2, 2, 0, 0\}, \tag{7.3}$$

and the angle equation is

$$\sum_{n=1}^{6} \theta[f(n), f'(n)] = 2\pi - K[i(i+1)/2 + j] \tag{7.4}$$

The second case is the pentagon vertex. Here only a single angle is in each of the 5 sectors, and by symmetry these must all be equal. We thus have

$$\theta_i = \frac{2\pi - K_{n'}}{5} \tag{7.5}$$

The third case is when a vertex is on the boundary between two sectors. This setup is seen in figure 7.6. Here we leverage the reflection symmetry of the system: it must have half of its angles in each sector. As such the sum of angles in this sector must be half the curvature:

$$\theta[(i-1)^2, 1] + \theta[i^2, 0] + \theta[(i+1)^2, 0] = \pi - \frac{K[i(i+1)/2]}{2}. \tag{7.6}$$

Note that we only need to do this on the left hand side of the sector (where $j = 0$), since the opposite side will be taken care of by the reflection equations. Lastly, if a vertex is a part of a number of fixed triangles m, then we can just subtract $m\pi/3$ from the RHS of the equation, since every fixed triangle is required to be equilateral, with angles $\pi/3$.

Reflection Equations The "reflection"-equations record the reflection symmetry of the sector. Every vertex has a reflected partner (possibly itself). See for example figure 7.2, where vertices 11 and 13 are reflected partners. All angles incident on these two vertices must be equal. For example the angle incident upon vertex 11 in triangle 9 must be equal to the angle incident upon vertex 13 in triangle 15. As such, these equations just set two angles equal to each other.

The reflection equations are easily built by looping over the first $i+1$ triangles of triangle ring i. Here we have two cases, illustrated by figure 7.7. If the triangle has an odd value of j it points "downward", as is the case for triangles 1 and 3. Its reflected counterpart triangle has $j' = 2i - j$, and the three reflection equations are:

$$\theta[i^2+j, 0] = \theta[i^2+2i-j, 2], \quad \theta[i^2+j, 1] = \theta[i^2+2i-j, 1], \quad \theta[i^2+j, 2] = \theta[i^2+2i-j, 0] \tag{7.7}$$

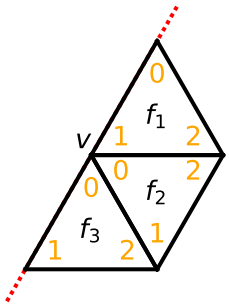


Figure 7.6: Triangles f_i used in the vertex equation for a side vertex v with indices (i, j). The axis of symmetry between two sectors is shown in red. The orange integers denote the local ordering of vertices in the triangles.

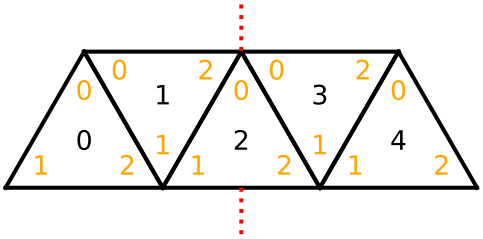


Figure 7.7: The setup for the reflection equations. In this case, a row with $i = 2$ is shown, with the axis of symmetry through the middle of triangle 2. The local ordering of vertices in a given triangle is shown by the orange integers. Triangles 0 and 4 are reflected partners, as are 1 and 3. 2 is reflected in itself. The indices of triangles with even indices j and triangles with odd index j are ordered differently in reflection. Triangle 0 angle 0 is equal to triangle 4 angle 0, while triangle 1 angle 0 is equal to triangle 3 angle 2.

. If j is even, then the triangle points upwards and the equations are

$$\theta[i^2+j,0] = \theta[i^2+2i-j,0], \quad \theta[i^2+j,1] = \theta[i^2+2i-j,2], \quad \theta[i^2+j,2] = \theta[i^2+2i-j,1]. \quad (7.8)$$

If the triangle is in the middle of the ring, then $j = i$, and the line of reflection runs through the triangle itself (triangle 2, in the case of figure 7.7). We set angles 1 and 2 equal, if j is even and angles 0 and 2 if j is odd.

Fixing Equations Lastly, if we have a triangle that is fixed (ie, shaded yellow in the computational domain, see figure 4.4), then we just set all angles equal to $\pi/3$, making it equilateral.

The Discrete Gauss-Bonnet Theorem as a Condition of Consistency

In general, this set of linear equations will be underdetermined, but we still need to ascertain whether or not it is consistent. For example, say we have a triangle n, whose angles have been determined from some subset of equations. Then its reflected counterpart n' only needs 2 reflection equations to set three angles, since the triangle equation for n' and the last reflection equation will be linearly dependent. Another set of linearly dependent equation can be seen from the triangle and vertex equations. If we were to extend the system of equations to the full mesh, we see that every single angle is present once in each set of equations, and we can get a zero-row in our matrix by summing up all the triangle equations and subtracting all the vertex equations. We then get an equation which must be satisfied for the set of equations to be consistent. There are F triangle equations and V vertex equations. The sum of each is:

$$\sum_i \theta_i = F\pi, \quad \sum_i \theta_i = 2V\pi - \sum_n K_n \tag{7.9}$$

Subtracting the other from the one gives:

$$0 = (F - 2V)\pi + \sum_n K_n. \tag{7.10}$$

We can relate the number of triangles to the number of vertices by Eulers formula, stating:

$$V - E + F = \chi, \tag{7.11}$$

where E is the number of edges and χ is the Euler characteristic of the mesh. Each triangle has 3 edges, but each edge is shared by 2 triangles, so $E = 3F/2$. Inserting this into the equation and multiplying by 2 we get:

$$2V - 3E + 2F = 2V - F = 2\chi, \tag{7.12}$$

which, upon insertion in to the sum of triangle and vertex equations gives the discrete Gauss-Bonnet theorem (for a closed mesh):

$$\sum_i K_i = 2\pi\chi. \tag{7.13}$$

As such the validity of the Gauss-Bonnet theorm is a necessary condition for the angular equations to be consistent. This condition has a corollary in the case of a single sector. Summing the equations gives:

$$\frac{K_0}{5} + \frac{1}{2}\sum_i K_i + \sum_j K_j = \frac{\pi}{15} \tag{7.14}$$

where i runs over all vertices on the boundary between two sectors (such that they have half the curvature in each sector), and j runs over all inner vertices. The first term is of course the curvature due to the pentagon vertex. This is then a statement that the total curvature in a given sector (scaled appropriately, to account for the fact that the Voronoi region of each vertex may be in different sectors) must equal a fifth of the total curvature of the pentagon.

Choosing a solution

For underdetermined, consistent systems $\mathbf{Ax} = \mathbf{b}$, there exists an infinite set of solutions. These can be generated using the Moore-Penrose pseudoinverse $\mathbf{A}^g$ (henceforth called the generalized inverse):[21]

$$\mathbf{x} = \mathbf{A}^g\mathbf{b} + (\mathbf{I} - \mathbf{A}^g\mathbf{A})\mathbf{w} \tag{7.15}$$

for any vector $\mathbf{w}$ of the same size as $\mathbf{x}$. Choosing $\mathbf{w} = \mathbf{0}$ gives the minimum norm solution $\mathbf{x}^*$, such that $||\mathbf{x}^*||_2 \leq ||\mathbf{x}||_2$ for all other choices of $\mathbf{w}$ [22]. This solution will prove advantageous in our case: Given a single, unconstrained triangle with angles x_i, the minimum norm solution has $x_i = \pi/3$, as can be seen by a simple symmetry argument: The solution must favor no direction, and must be the same under any permutation of the angles. It can of course also be seen by direct computation. Expressing the 2-norm of the angles in terms of x_1 and x_2 gives

$$||x||_2 = \pi^2 - 2\pi(x_1 + x_2) + (x_1 + x_2)^2, \tag{7.16}$$

This is a paraboloid with positive coefficient, giving a single (global) minimum. The minimum is the solution to the stationary point equations:

$$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \pi \\ \pi \end{pmatrix} \tag{7.17}$$

with solution $x_1 = x_2 = \pi/3$. Inserting into the angle sum then gives the last angle $x_3 = \pi/3$. In general, for a set of unconstrained triangles (like those defined by the triangle equations of our sector), the minimum norm solution will then be a set of equilateral triangles. Including the vertex- and reflection-equations then means the minimum norm solution is the closest (norm-wise) solution to equilateral triangles, which still satisfies the constraints imposed by the vertex- and reflection-equations.

Angles to Manifold

With all angles of the sector calculated, we need to calculate the edge lengths of the system. For this we use the law of sines and the connectivity of the triangles. Given a triangle with a known edge length: any unknown edge lengths is calculated using the law of sines, and then propagated to its neighbouring triangles, resulting in a breadth first updating scheme on the primal graph (see figure 7.8). The fixing of vertices with neighbours outside the sectors gives a known edge length of the system, from which the rest are populated. In figure 4.4 all edges of unshaded and yellow shaded triangles are known.

The cotangent Laplacian (with area weighting) needs all angles in the mesh for the cotangent weights, and the edge lengths for calculating the Voronoi region areas. The areas of each triangular face (with edge lengths a, b and c) is needed for calculating integrals, and can be calculated using Heron's formula:

$$A = \sqrt{s(s-a)(s-b)(s-c)}, s = \frac{a+b+c}{2} \tag{7.18}$$

Cotan Laplacian and Voronoi Areas

With the data structure from algorithm 7, constructing the cotan Laplacian is quite easy. The sparse adjacency matrix contains the all the indices of non-zero value in the cotan Laplacian,

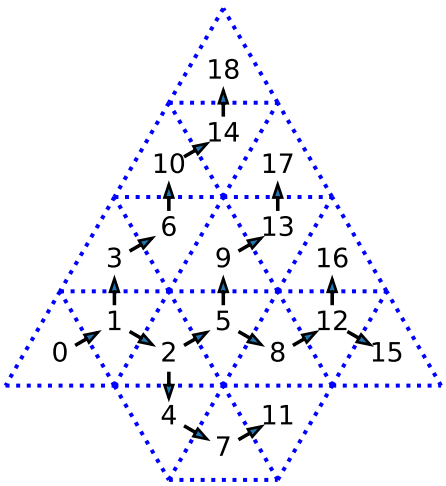


Figure 7.8: Propagation of lengths using a breadth first scheme. The integers denote the order in which the lengths are propagated. The case is shown for all unfixed triangles with H. Starting at a triangle (0) with a known side length, since it is a neighbour to a fixed triangle, the rest of the lengths in the triangle is calculated. These are then propagated to the neighbours of the triangle, which has not already been populated.

and the sparse weights matrix contain the values needed. The diagonal values of the Laplacian is just the row-wise sum of values in the sparse weights matrix. Construction, then, is as follows:

$$L^C_{i,i} = \sum_{j=0}^{5} w_{i,j}, \tag{7.19}$$

$$L^C_{i,j} = -w_{i,j'}, \quad j = s_{i,j'}, \tag{7.20}$$

with w being the sparse weight matrix and s being the sparse adjacency matrix. Note that even for pentagon vertices, we can sum over the whole row of weights, since the weight of the non-existant neighbour is defined to be 0. The voronoi area scaling is just the row-wise sum of the element-wise product between the sparse weights and lengths matrices:

$$A_i = \frac{1}{4} \sum_{j=0}^{5} w_{i,j} \ell_{i,j}. \tag{7.21}$$

Chapter 8

Non-embeddable Results

A Tale of Missing Constraints

Ideally we would now have a discrete Riemannian manifold on our hands, with a nice curvature distribution. But alas, the mathematics proves itself uncooperative! We begin this chapter by showing just where a problem occurs in our line of thinking, and then how we might go about circumventing this, so we can still arive at a proper manifold.

8.1 Problems in edge length propagation

With a given set of angles we can now calculate all the edge lengths. However upon doing so, a discrepancy occurs: Some of the triangles do not fit together! Say for example triangle i is being filled in, since its neighbour triangle j was filled in previously. Triangle i will then, by construction fit with triangle j. Say then, that another neighbour k of triangle i was in the queue as triangle j was being filled in, such that k becomes filled in before i, but does not propagate its lengths to i. Then we are not guaranteed that the edge shared by k and i will be the same length - the triangles may not fit together. The offending edge lengths for $H = 8$ have been highlighted in figure 8.1, where they have been overlain on the computational domain figure 4.4

This tells us that there is a missing set of constraints on our method. In calculating the angles, we see that since the matrix of angle equations $\mathbf{A}$ is not full rank (yet still consistent with the RHS $\mathbf{b}$), there is a subspace of vectors $\mathbf{x}$, that all solve the problem $\mathbf{Ax} = \mathbf{b}$. Our hope was then that every solution $\mathbf{x}$ would satisfy all constraint and produce a proper discrete Riemannian manifold.

We thus chose the vector $\mathbf{A}^g\mathbf{b}$, as the "best" solution $\mathbf{x}^*$ since it is easy to calculate and produces near-equilateral triangles, not realizing this did not satisfy the missing constraints.

How then, do we make sure that we produce a set of triangles that fit together? Well, a single triangle is characterized by 6 numbers: 3 angles and 3 edge lengths. These are of course not independent, but subject to a set of constraints: The angles must be positive and their sum must equal π, and together with the lengths, they must satisfy the law of sines. The law of sines in turn imply the triangle inequality (the triangle inequality can be shown for example using the law of cosines, which can be derived from the law of sines). For a collection of n

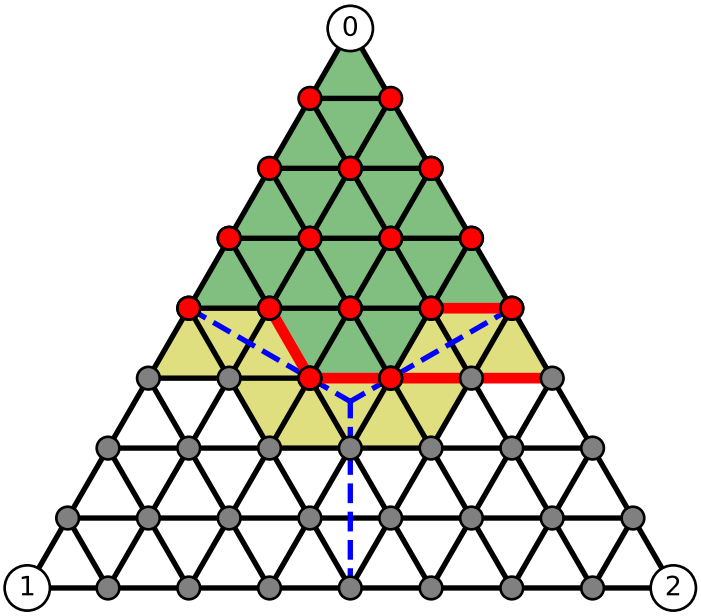


Figure 8.1: The computational domain, from 4.4, with offending edge lengths overlaid. In propagating the edge lengths up the triangle, a discrepancy occured several places. These edges have been highlighted in red.

triangles (or, equivalently, $6n$ numbers), we can leverage Alexandrov's Uniqueness Theorem to ensure embeddability by demanding the following:

- The collection of triangles have the same connectivity as a sector of our pentagonal pyramid.
- The set of angles obey all the angle equations from section 7.2.
- The lengths and angles of each triangle obey the law of sines.
- The edges between two adjacent triangles are equal.

These conditions ensure a unique 3-dimensional embedding exists, and therefore the triangles must fit together. The fault, then, lay with us treating the lengths separately from the angles. And we must therefore make sure to find a set of angles and edge lengths that satisfy all of the above conditions at the same time.

The quest is then: for a given Halma index H, find $6H^2$ variables, divided into angles and lengths, such that they obey all the equations implied by the above conditions at the same time. The equations can be divided into three sets. A set of linear equations relating the angles to each other (precisely the angle equations from section 7.2), a set of linear equations relating the lengths to each other (similar to the angle equations), and a set of nonlinear equations relating the angles of a given triangle to its lengths (the law of sines). The Venn

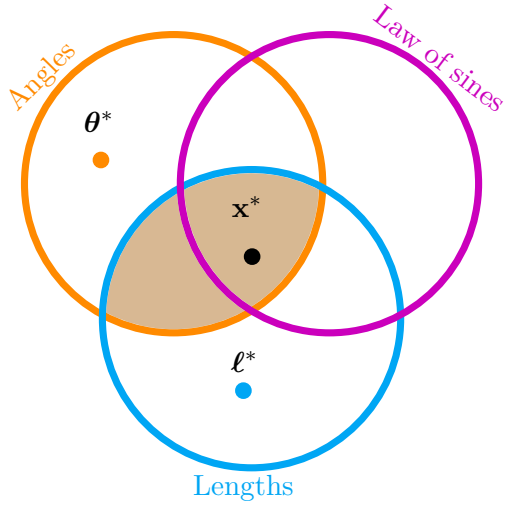


Figure 8.2: Venn diagram of the feasible sets for each set of constraints. In orange is the feasible set for the linear angle equations, and its optimal solution $\boldsymbol{\theta}^*$. In blue is the feasible set for the linear length equations, and its optimal solution $\boldsymbol{\ell}^*$. In purple is the feasible set for the non-linear constraints. The shaded area (the union of all the linear equations) is the set of possible values of $\mathbf{x}$. The optimal solution $\mathbf{x}^*$ however, must lie in the union of all 3 sets, and is not just the combination of $\boldsymbol{\theta}^*$ and $\boldsymbol{\ell}^*$, as this would not satisfy the non-linear constraints.

diagram for the set of solutions to these is seen in figure 8.2. In orange is the solutions to the angle equations, in light blue the length equations and in pink the law of sines. Our previous method landed us in the orange set, and by propagating edge lengths, it found a solution to the law of sines as well. We need to find a solution in the intersection between all three sets. To do this, we first need to set up the last equations: the linear lengths. These can be split into three parts.

1. All adjacent lengths are equal.
2. All lengths reflected across the sector are equal.
3. All lengths of fixed triangles must all equal $1/H$ (fixed triangles being shaded yellow in figure 8.1, corresponding to triangles not entirely in the Voronoi region).

The second part is a consequence of the reflection symmetry of the mesh - like in the case of the angles (and in fact, due to the way the lengths are stored, the procedure for creating the reflection equations for the angles carries directly over to the lengths). The third part is what allows us to stitch the Voronoi-region of the original pentagon vertex together with the rest of the fullerene manifold. Then for each triangle we need two instances of the law of sines, such that all three lengths and angles are related to each other.

With these equations we could in principle set the computer running, trying to find the root of this high-dimensional function of $6H^2$ variables. This would be ill-advised since it yields a high-dimensional non-linear optimization problem. However, we can use the linear equations to easily eliminate a majority of the variables. Conceivably we could totally forego all variables pertaining to fixed triangles, removing just over two thirds of the variables, but

this would make setting up the rest of the linear equations harder. Instead we choose to remove the triangle rings which only contains fixed triangles. Say this number of rings is N_r, then only $6(H - N_r)^2$ variables remain.

Next we notice that the set of solutions for the full problem must be the intersection of the sets of solutions for each of the three subproblems (the angle equations, the length equations, and the law of sines equations). All of the linear equations are easily solved: we draw inspiration from the generalized matrix inverse. All solutions to the (consistent) underdetermined system of linear equations $\mathbf{x} = \mathbf{Ab}$ can be determined from $\mathbf{x} = \mathbf{A}^g\mathbf{b} + (\mathbf{I} - \mathbf{A}^g\mathbf{A})\mathbf{w}$, where $\mathbf{w}$ is an arbitrary vector of same size as $\mathbf{x}$. The matrix $\mathbf{I} - \mathbf{A}^g\mathbf{A}$ is the projection operator $\mathbf{P}_{\ker \mathbf{A}}$ onto the kernel of A, such that $\mathbf{P}_{\ker \mathbf{A}}\mathbf{w}$ for all $\mathbf{w}$ span the kernel of $\mathbf{A}$. This does not decrease the number of parameters in the problem, but we can then construct a set of basis vectors for the kernel of $\mathbf{A}$, and use linear combinations of these to generate any solution $\mathbf{x}$ we want. If $\mathbf{A}$ is of size $m \times n$, with $m > n$, and rank $k < n$, then the basis for the kernel will span a $n - k$ dimensional subspace of $\mathbb{R}^n$. Organizing these vectors into an $n \times (n - k)$ matrix $\mathbf{N}$ gives us the final expression for the solution: $\mathbf{x} = \mathbf{A}^g\mathbf{b} + \mathbf{Nz}$, with $\mathbf{z}$ being an arbitrary vector with $n - k$ entries.

In the context of our mesh, we organize the full set of angles and lengths as a $6(H - N_r)^2$-length vector x, formed from two subvectors $\boldsymbol{\theta}$ and $\boldsymbol{\ell}$ (each of length $3(H - N_r)^2$) with $H - N_r$ the number of rows in the sector which have at least one, non-fixed triangle (ie, the top 5 rows in figure 8.1, where a green shaded triangle denotes a non-fixed triangle):

$$\mathbf{x} = \begin{bmatrix} \boldsymbol{\theta} \\ \boldsymbol{\ell} \end{bmatrix}$$

from which we can express the linear and nonlinear equations. Next we find all solutions to the linear equations by the method above, such that

$$\boldsymbol{\theta} = \mathbf{A}_\theta^g\mathbf{b}_\theta + \mathbf{N}_\theta\boldsymbol{\phi}, \quad \boldsymbol{\ell} = \mathbf{A}_\ell^g\mathbf{b}_\ell + \mathbf{N}_\ell\boldsymbol{\lambda},$$

where $\boldsymbol{\phi}$ and $\boldsymbol{\lambda}$ are vectors carrying the linear coefficients for the null space of angles and lengths respectively. We now have a space of variables $\boldsymbol{\phi}$ and $\boldsymbol{\lambda}$ (organized in a vector $\boldsymbol{\chi} = \begin{bmatrix}\boldsymbol{\phi} & \boldsymbol{\lambda}\end{bmatrix}^T$) which we can use to generate all solutions to the linear angle and length equations. All possible values of $\boldsymbol{\chi}$ then trace out the shaded area of figure 8.2. We can then express the nonlinear equations in terms of the new set of linearly independent variables $\boldsymbol{\chi}$, and solve for them. For $H = 8$, for example, $N_r = 3$, giving 150 linearly dependent variables, which can be reduced to 10 free angle parameters and 16 free length parameters!

But how many equations do we have left to solve? We can immediately discard all the law of sines for fixed triangles (those that may still be left in the reduced sector mesh), but since we have reflection symmetry in both lengths and angles, if we solve for a given triangle with ragged indices (n, m), then the solution also holds for its reflected partner, the triangle $(n, 2n - m)$. Likewise, triangles in the middle of the ring are by symmetry isosceles and so only a single law of sines is needed (provided that we relate two sides of differing lengths).

In total, we have $(H - N_r)^2$ triangles in the reduced sector mesh. Then we have $H - N_r$ middle ring triangles, which only need a single equation. If N_f is the number of fixed triangles remaining in the reduced sector mesh, this then leaves us with $(H - N_r)^2 - H - N_r - N_f$ triangles, where half of them do not need any nonlinear equations, since the variables in these triangles are already fixed by the reflection equations. This means each triangle only needs a single equation (or equivalently, half of them need two). Adding all these up gives a total of

$(H - N_r)^2 - N_f$ equations. As an example, for $H = 8$, we have $N_f = 6$, leaving us with 19 nonlinear equations to solve for.

We are unfortunately not quite ready to set the ball rolling yet, since we have an underdetermined system of equations. As such, if we just use simple root finding, we are not guaranteed to get a good solution. Instead we will use constrained optimization. The problem is stated as this:

$$\min_{\boldsymbol{\chi}} f(\boldsymbol{\chi}), \quad \text{subject to } \mathbf{c}(\boldsymbol{\chi}) = \mathbf{0}$$

where f denotes the objective function and $\mathbf{c}$ is the set of constraints, organized as a vector function. In our case the set of constraints is then the law of sines, expressed in terms of the coordinates $\boldsymbol{\chi}$, since the linear constraints have been taken care of by our choice of variables $\boldsymbol{\chi}$. The set of vectors $\mathbf{x}$ that satisfy the constraints is called the *feasible set*. Our objective function will be the squared norm of the angle vector: $f = \sum_i \theta_i^2$, as was the case for the implicit optimization done when using the generalized inverse. We do not need to explicitly recast f or $\mathbf{c}$ in the new variables, but to speed up computation we do recast their Jacobians using the chain rule:

$$\begin{aligned}
\nabla f &= 2\mathbf{N}_\theta^T(\boldsymbol{\theta}^* + \mathbf{N}_\theta \boldsymbol{\phi}), \\
\frac{\partial c_i}{\partial \phi_j} &= \ell_n (N_\theta)_{m,j} \cos\theta_m - \ell_m (N_\theta)_{n,j} \cos\theta_n, \\
\frac{\partial c_i}{\partial \lambda_j} &= (N_\ell)_{n,j} \sin\theta_m - (N_\ell)_{m,j} \sin\theta_n.
\end{aligned}$$

Next we need a good initial guess. But what would this be? If we naively use $\left[\boldsymbol{\theta}^*, \boldsymbol{\ell}^*\right]^T$ as our initial guess (corresponding to $\boldsymbol{\phi} = 0, \boldsymbol{\lambda} = 0$), we would be quite far away from a solution, since the vast majority of lengths will be 0 initially (as this gives the optimal norm for the length vector). Ideally for constrained optimization, one would start with a vector from the feasible set, but since this set is unknown to us, we have to make do without. An obvious starting point would be $\boldsymbol{\phi} = 0$ and $\boldsymbol{\lambda}$ such that $\boldsymbol{\ell} = 1/H$, but even this might be too far away from the feasible set, and the algorithms employed might not make any headway.

For our purposes we use the constraint violations as our measure of optimality, since if all constraint violations are 0, we have a solution perfectly in the feasible set. We test the method by running the heat equation simulation for 200 iterations, with $\delta = 1/60$, for both $H = 4$ and $H = 8$, and plot the magnitude of the maximum constraint violation, along with the mean constraint violation. The results are seen in figure 8.3. For $H = 4$ we see that both the maximum constraint violation and the mean constraint violation are very small: the mean constraint violation ends at $2.671 \cdot 10^{-13}$, while the mean violation ends at $8.583 \cdot 10^{-15}$. For $H = 8$ the results are not quite as good, the maximum violation ends at $9.256 \cdot 10^{-4}$ and the mean at $-9.253 \cdot 10^{-5}$.

A Better Initial Condition

We might improve our results with a better initial condition: one that is closer to the feasible set. The problem being, that we do not know one, but we might be able to calculate one! For example we definitely know the optimal solution to the problem of all curvature centred at the pentagon vertex: $\boldsymbol{\theta} = \pi/3, \boldsymbol{\ell} = 1/H$, and the curvature distribution after a single iteration is not far away from the original. Indeed, the curvature distribution from any single iteration is

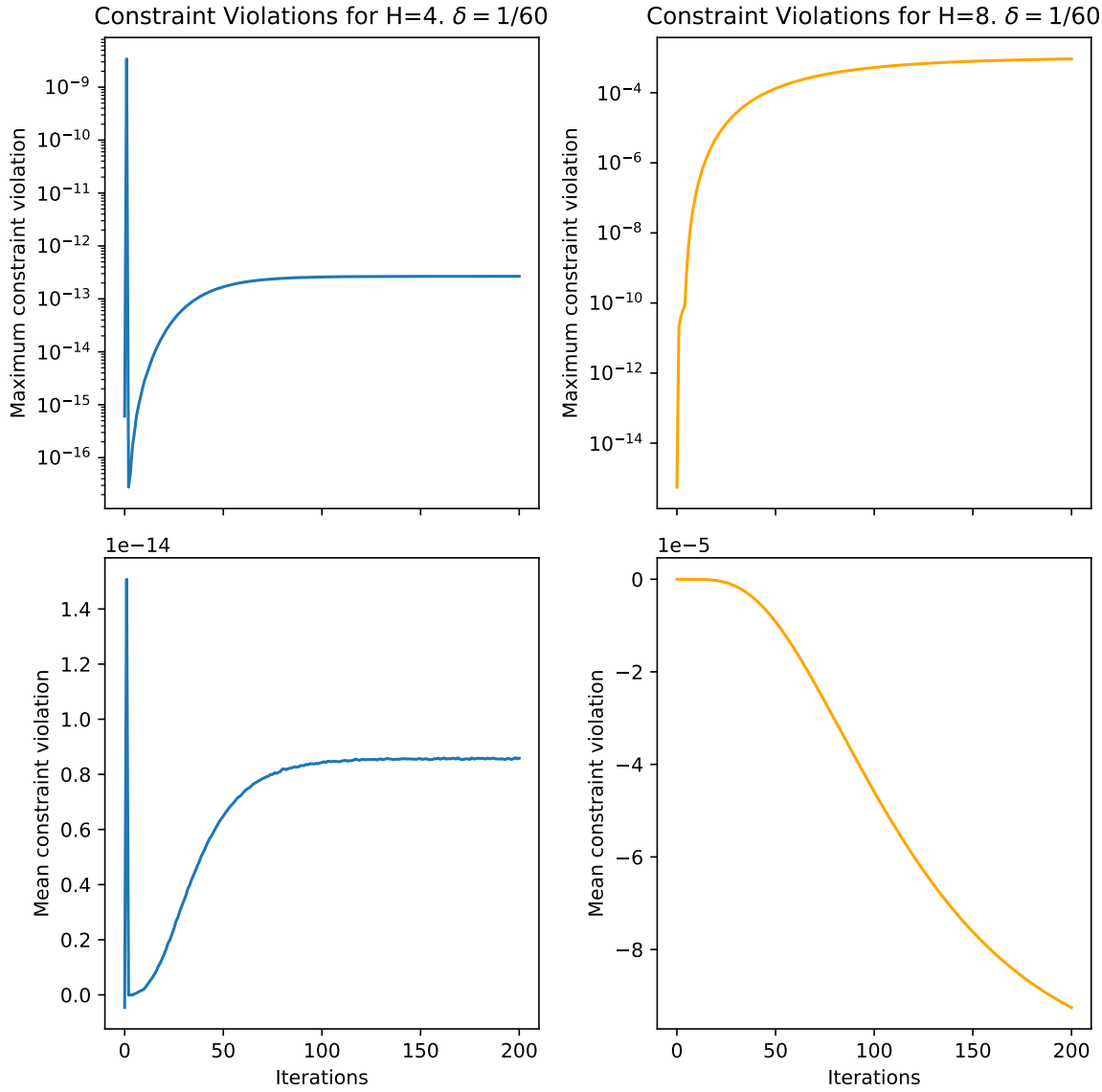


Figure 8.3: Plots of the magnitude of the maximum constraint violation and the mean constraint violation, for simulations with $H = 4$ (left) and $H = 8$ (right). In both cases the heat equation is evolved for 200 iterations with $\delta = 1/60$.

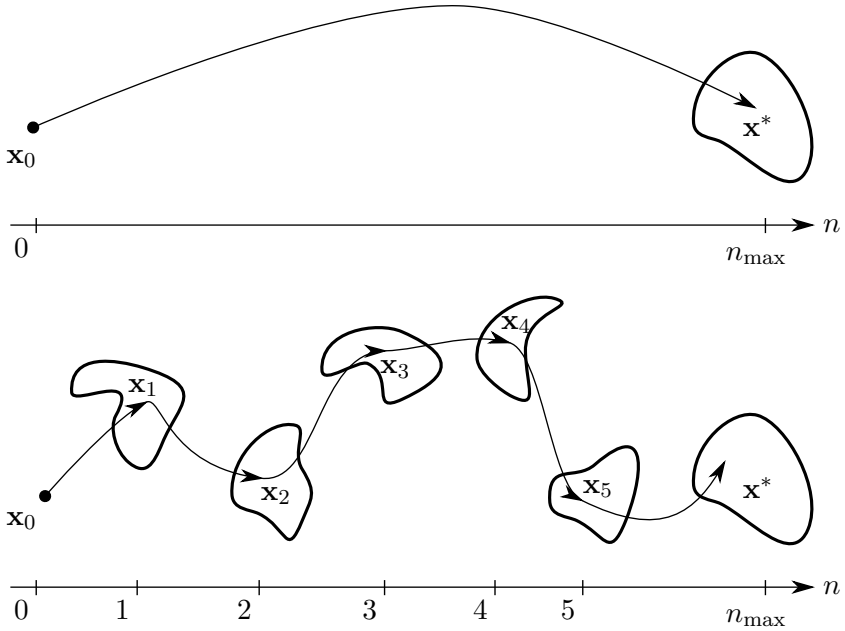


Figure 8.4: Schematic overview of the normal and "Born Oppenheimer approximation" or "adiabatic approximation" approach used in our optimization. In the normal (top) we only perform optimization on the last time step, leading to difficulties in finding the feasible set. In the adiabatic approximation (bottom) we perform optimization in each time step, which might make finding the feasible set easier.

quite close to its predecessor. This motivates the following iterative optimization scheme: For a given iteration in the heat equation, use the optimal solution from the previous iteration as the initial guess, and for the first iteration, use $\boldsymbol{\theta} = \pi/3, \boldsymbol{\ell} = 1/H$.

This scheme has an obvious parallel in the Born-Oppenheimer approximation we used to make the quantum mechanical problem tractable: In the approximation we see a separation of time scales t and τ, with t being the time scale of the electrons, and τ being that of the nuclei, with $\tau \gg t$. Any change in the nuclei will certainly induce transient behaviour in the electronic structure, but on a much shorter time scale. In our case, the long time scale would be the heat equation simulation, and the short time scale would be the optimization: Every time the curvature changes (and it does so "slowly", given a sufficiently small δ), it "induces" a transient behaviour in the lengths and angles of the system. This transient behaviour is then captured by the optimization step. A schematic overview of this approach is seen in figure 8.4.

This approach will undoubtedly be slow, since it relies on optimization on each time step of the heat equation, but it might be offset by the ease of having a better initial guess for the final curvature distribution.

We test this with the same simulation parameters as in figure 8.3, but only $H = 8$. Again we plot the max and mean violation, along with the squared difference in solutions: $\sum_i (\mathbf{x}_{\text{default}} - \mathbf{x}_{\text{AA}})_i^2$. The result is seen in figure 8.5. And we see that this method unfortunately does not improve our results. Indeed for most of the iterations, the two produce almost the same result, only differing towards the end (around iteration 180 and forward), where the adiabatic approximation appears to "oscillate".

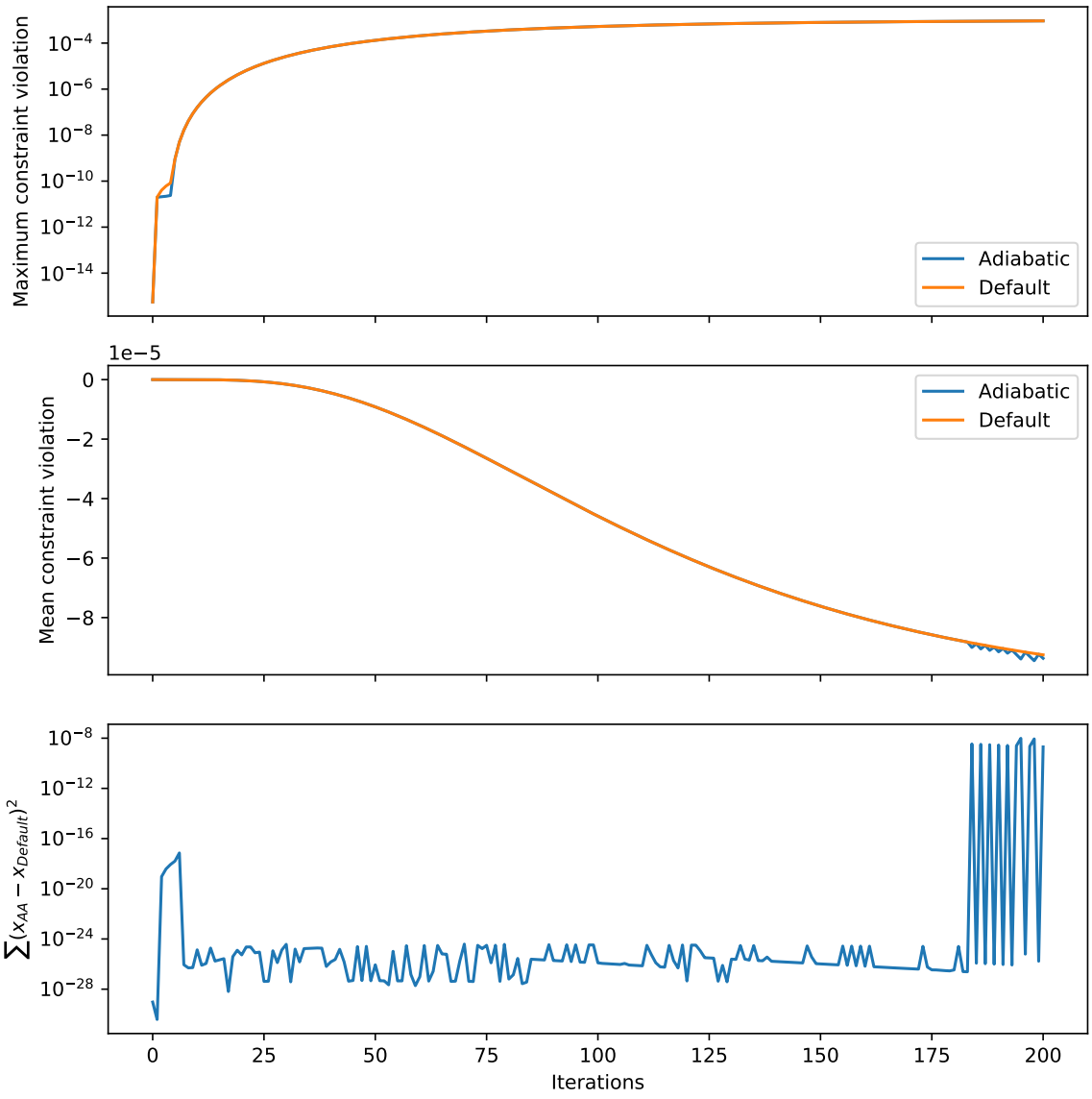


Figure 8.5: Comparison of the default initial condition and the "adiabatic approximation". Setup is as in figure 8.3, but with only $H = 8$. Again the magnitude of the maximum violation along with the mean violation is graphed. On the bottom is shown the discrepancy between the solutions, as the sum of squared differences in the solutions.

As such, we conclude that the default method works better, since we attain no better results, and at much lower speeds since the "adiabatic approximation" approach needs to "bootstrap" itself, by optimizing on every iteration. With this, our method is currently limited to relatively low resolution, below $H = 8$.

Chapter 9

Visualization

A Figure Is Worth a Thousand Floats

The output of the algorithms outlined above is just a set of numbers defining the resulting manifold. These numbers are important, since they give us the ability to calculate the quantities of interest, but they do not give us a nice visual representation of the manifold. This is rectified in this chapter. We introduce a way of calculating the local embedding of the smeared pentagonal pyramid by leveraging the position of fixed vertices, and with the use of triple-sphere intersections.

9.1 Manifold to Isometric Embedding

With the manifold populated, we can actually also calculate the full isometric embedding of the smeared pentagonal pyramid, since we have the coordinates for all fixed vertices. In particular, the bottom five corners are placed in the xy plane with coordinates

$$\mathbf{p}_i = (\ell \cos\theta_i, \ell \sin\theta_i, 0)\,, \quad \theta_i = \frac{\pi}{2} + \frac{2i\pi}{5}, \quad \ell = \frac{1}{2\sin(\pi/5)}. \tag{9.1}$$

The top (pentagon) vertex of the pyramid is then placed at $(0, 0, h)$, with $h = \sqrt{1-\ell^2}$ ensuring all side lengths equal to unity. The coordinates of the unsmeared, refined vertices can then be easily calculated using an affine transformation, given the original triangle corners.

Lastly only a subset of vertices are to be moved from their original position - namely those, whose edge lengths have been changed. In general this will be the case for the fixed vertices in each sector. See section 4.2 and figure 4.4 on the computational domain. We only need to calculate around a tenth of the remaining vertex positions due to the D_5 symmetry of the mesh. Just calculate the coordinates of a single side, and rotate by $2\pi/5$ radians about the z-axis to get the next side. But due to the reflection symmetry, we can calculate just half the vertices on each side, and get the rest through a simple reflection.

To compute the position $\mathbf{r}$ of a given vertex, we need to know the position of 3 of its neighbours $\mathbf{r}_i$, along with the edge lengths between the vertex and its neighbours ℓ_i. This gives us 3 equations of the type $\ell_i^2 = ||\mathbf{r}_i - \mathbf{r}||^2$, with 3 unknowns: $\mathbf{r}^T = [x, y, z]$.

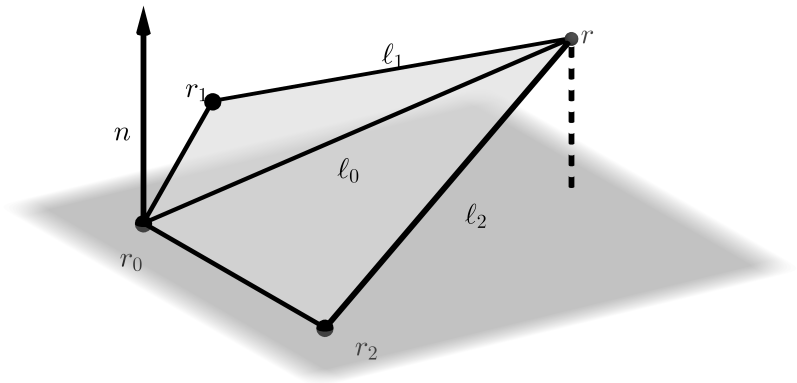


Figure 9.1: Setup for single point solver. $\mathbf{r}$ is the unknown position, with $\mathbf{r}_i$ being neighbours with known positions and distances ℓ_i to $\mathbf{r}$. The correct solution has $\mathbf{r} \times \mathbf{n} > 0$

Calculating the vertex position

We are given a vertex with (unknown) position $\mathbf{r}^T = [x, y, z]$, and known neighbours $\mathbf{r}_0, \mathbf{r}_1, \mathbf{r}_2$, arranged as on figure 9.1, with $\ell_i = ||\mathbf{r} - \mathbf{r}_i||_2$. This gives three quadratic equations with three unknowns $\ell_i^2 = ||\mathbf{r}_i - \mathbf{r}||^2$. First we precondition the solution by translating the system such that $\mathbf{r}_0 = \mathbf{0}$. The three equations are then

$$\ell_0^2 = x^2 + y^2 + z^2, \quad \ell_1^2 = (x_1 - x)^2 + (y_1 - y)^2 + (z_1 - z)^2, \quad \ell_2^2 = (x_2 - x)^2 + (y_2 - y)^2 + (z_2 - z)^2$$

Subtracting the first equation from the second and expanding gives

$$\ell_1^2 - \ell_0^2 = x_1^2 - 2x_1 x + y_1^2 - 2y_1 y + z_1^2 - 2z_1 z$$

which is now linear in the unknown coordinates. Likewise can be done for the third equation, giving us 2 linear equations with 3 unknowns:

$$x_i x + y_i y + z_i z = c_i, \quad c_i = \frac{1}{2}(x_i^2 + y_i^2 + z_i^2 + \ell_0^2 - \ell_i^2)$$

The matrix system is

$$\begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{pmatrix} \mathbf{r} = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$$

Gaussian elimination gives

$$\begin{pmatrix} 1 & 0 & z_1' \\ 0 & 1 & z_2' \end{pmatrix} \mathbf{r} = \begin{pmatrix} c_1' \\ c_2' \end{pmatrix}, \quad z_2' = \frac{\begin{vmatrix} x_1 & z_1 \\ x_2 & z_2 \end{vmatrix}}{\begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix}}, \quad c_2' = \frac{\begin{vmatrix} x_1 & c_1 \\ x_2 & c_2 \end{vmatrix}}{\begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix}}, \quad z_1' = \frac{z_1 - y_1 z_2'}{x_1}, \quad c_1' = \frac{c_1 - y_1 c_2'}{x_1}$$

This fixes x and y for a given z. We now plug these into the first quadratic, and solve:

$$x = c_1' + z_1' z, \quad y = c_2' + z_2' z, \quad \Rightarrow \quad \ell_0^2 = (c_1' + z_1' z)^2 + (c_2' + z_2' z)^2 + z^2$$

Expanding and collecting terms

$$\begin{aligned} 0 &= \alpha z^2 + \beta z + \gamma, \\ \alpha &= (z_1')^2 + (z_2')^2 + 1, \\ \beta &= -2(z_1' c_1' + z_2' c_2'), \\ \gamma &= (c_1')^2 + (c_2')^2 - \ell_0^2 \end{aligned}$$

and of course

$$z_\pm = \frac{-\beta \pm \sqrt{\beta^2 - 4\alpha\gamma}}{2\alpha}$$

There being two solution is a reflection of the fact that the system is symmetric across the plane spanned by the vectors $\mathbf{r}_1 - \mathbf{r}_0$ and $\mathbf{r}_2 - \mathbf{r}_0$. The solution we want is the one "above" the plane, where

$$\mathbf{r} \cdot \mathbf{n} > 0, \quad \mathbf{n} = (\mathbf{r}_1 - \mathbf{r}_0) \times (\mathbf{r}_2 - \mathbf{r}_0),$$

since this will ensure the convexivity of the resulting embedding. As a last step the system is translated back, such that $\mathbf{x}_0$ is placed at its original position.

This method, however, has a couple of caveats. Firstly, if all three points $\mathbf{r}_i$ are colinear, the system degenerates to give an infinite set of solutions, all on a circle. In this case there will be a division by zero in the definition of z_2' and c_2'. If this case were to appear, we would need either another known point, not colinear with the other three, or some other piece of information, such as a dihedral angle, which could fix the position.

Another instance of division by zero occurs if the projections of $\mathbf{r}_1 - \mathbf{r}_0$ and $\mathbf{r}_2 - \mathbf{r}_0$ upon the xy-plane are colinear. This results in the same division by zero as with fully colinear points, but the system still has only two solutions (since the two vectors are still linearly independent, due to their z-coordinates). In this case, we will have to parametrize another variable, instead of z. This can be done by performing the same row operations, but first permuting the third column of the matrix with some other column: a pivot. The inverse permutation must then be applied in the end, to get the coordinates in the correct positions.

Lastly, if $x_1 = x_0$ before translation (or $z_1 = z_0$, in the case of parallel projections), then another division by zero occurs. If we have checked the above cases however, we can just switch the points $\mathbf{r}_1$ and $\mathbf{r}_2$ (permuting the rows of the matrix), with an appropriate redefinition of $\mathbf{n}$, so we still choose the correct orientation. This is because if the vector projections upon the xy-plane are *not* parallel, then the determinant $x_1 y_2 - y_1 x_2 = -x_2 y_1 \neq 0$ implies $x_2 \neq 0$ (after translation of $\mathbf{r}_0$).

We can then implement these three edge cases in this order, to make sure no degeneration occurs as well as no division by zero.

Propagating coordinates up the pyramid

We then start with the last ring with vertices within the Voronoi region, going from left to right, from the first unknown vertex position. Figure 9.2 shows the first 4 vertices to calculate in the case of $H = 8$. For vertex 0 we already know the coordinates of 3 neighbours. The position of vertex 1 can be found using a reflection across the sector (the red dashed line). This fills the first ring. On the next ring up, vertex 2 has two vertices with known coordinates in the current sector. The third known vertex is in the adjacent sector, and can be found by a reflection across the border between sectors (the orange dashed line)

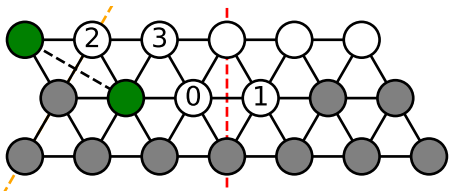


Figure 9.2: The order of vertices to calculate positions for, in a subset of a sector. Vertex 0 has three fixed neighbours. Vertex 1 is a reflection of vertex 0 (across the red dashed line). Vertex 2 has 2 neighbours with explicitly known positions. Its third neighbour is found by reflection across the boundary between sectors (green vertices, reflected across the orange dashed line). Vertex 3 has, after calculation of vertex 0 and 2, three fixed neighbours.

In general, the two reflections needed are reflections in the xy-plane, across some line with angle θ. Choosing the side defined by the pentagon vertex, $\mathbf{p}_1$ and $\mathbf{p}_2$ (from equation 9.1), the reflection across the plane is just a reflection across the y-axis (coresponding to θ_0 of equation 9.1), and the reflection across the side has $\theta = \theta_1$. The reflection matrix can be formed by first rotating the xy-plane by $-\theta$, so the axis of reflection coincides with the x-axis. Then we reflect across the x-axis, and rotate back with θ. This then gives

$$\text{refl}(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} = \begin{pmatrix} \cos 2\theta & \sin 2\theta \\ \sin 2\theta & -\cos 2\theta \end{pmatrix}$$

We can then calculate half the vertices on this ring, reflect across the plane to get the rest, proceed to the next ring, reflect across the sector to get the coordinates of the third known neighbour, rinse and repeat, until the whole side has been calculated. Then we just rotate the side by $2\pi/5$ radians, 4 times, to get the rest of the vertex coordinates in the pyramid.

Isometrically Embedding The Full Pentagonal Pyramid

The vertex mask from sections 4.2 is the starting point for figuring out which vertices we need to calculate new positions for. In the full algorithm (algorithm 9) we also implement the optimizations due to the symmetries of the mesh. Since some of the vertices in the mask have a fixed geometry, we can either opt to modify the vertex mask, or check on the fly, whether the geometry should be fixed or not. Since this modified vertex mask would only be used in this specific algorithm we forego an explicit modification of the mask, and instead check on the fly. The check itself can be reduced to checking whether a specific neighbouring vertex is in the mask or not. This is because, for vertices with fixed geometry, and second ragged index $j \leq \lceil i/2 \rceil$, we are guaranteed that the vertex $(i+1, j)$ is both a neighbour and not in the vertex mask.

The embedding algorithm described in section 9.1 creates a local isometric embedding of the pentagonal cusp, that is, relative to the base centre of the pyramid. We can then use another affine transformation to move all points of the cusp to the correct places in a global isometric embedding. For this we need 4 points with known positions, constituting a non-degenerate tetrahedron. One easy solution is to use the pentagon vertex and three hexagon vertices in a row, since the pentagonal pyramid is rotationally symmetric. We then just pick

the pentagon vertex and the first 3 neighbours in the sparse adjacency matrix, and perform the affine transform.

Results

We now show the resulting embedding for a pentagonal pyramid subdivided with $H = 4$. The heat equation was simulated for 200 iterations, with $\delta = 1/420$. The default embedding is shown, along with embeddings for every 40th iteration, in figure 9.3. Note that only the top two rows of dual faces are smeared, as these are the only faces fully in the pentagonal Voronoi region.

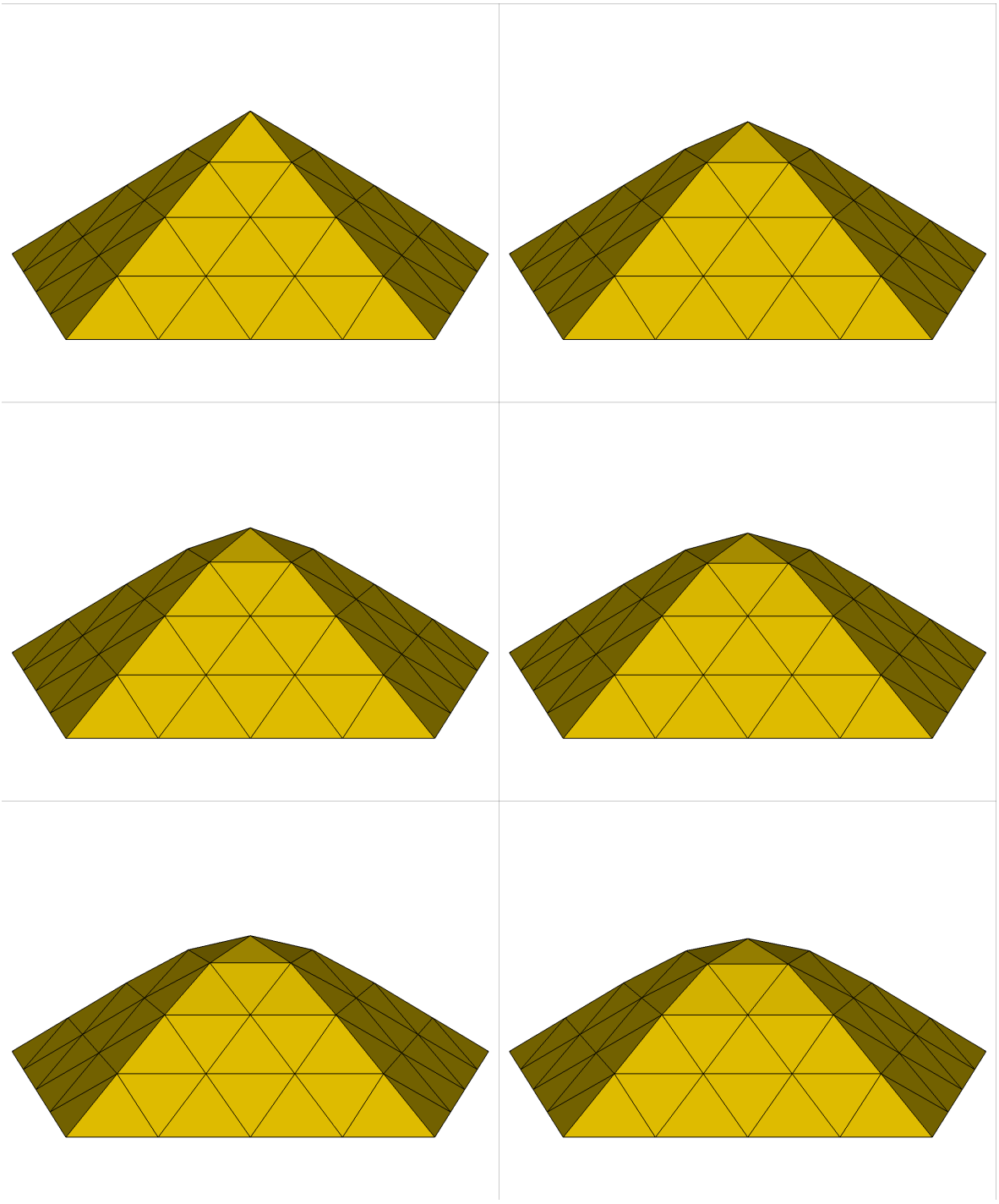


Figure 9.3: Embeddings for a pyramid with $H = 4$, shown for every 40th iteration in a simulation with $\delta = 1/420$.

Chapter 10

Discussion

10.1 Offline and Online Calculations

All of the calculations presented in this thesis could in principle be done for every fullerene, but this would without a doubt be unnecessary. Many calculations are either identical or at least similar for many fullerenes. The ideal case is that a single curvature distribution and a single number of refinements works the best for all fullerenes, in which case we only need to calculate this once, store the data and load it when preparing the DFT calculations. This is an example of offline and online calculations. Those which need to be calculated for every fullerene (like the sparse adjacency matrix for the refined fullerene) are online calculations, while calculations that can be done independent of the fullerene graphs, and are the same for all fullerenes, can be done prior to the DFT calculations, and are therefore offline calculations. In table 10.1 the different calculations mentioned in this thesis are categorized in this way.

10.2 Calibration

So far in this thesis we have focused solely on developing the method for smearing curvature and calculating the resulting manifold, whilst keeping quiet about what this manifold really represents, or how to actually calibrate this method such that the resulting manifold DFT will produce more accurate results. Calibration would be a trivial matter if we had some physical surface intrinsically linked to the kinetic energy, which we could then compare the manifold to. The problem is, we do not have a well defined, boundaryless surface, except for the manifold arising from the bond graph - the very same one that we have smeared ourselves away from! Isosurfaces of density for example, are not necessarily continuous or convex.

Our best bet is to compare the results of our DFT to that of an established 3-dimensional DFT. For example the work of Rebecca Sure et al (including Peter Schwerdtfeger, co-author of the Fullerene program [23]), who has analyzed all 1,812 isomers of C_{60} using DFT and used semi-empirical methods for analyzing the 31,924 isomers of C_{80} [24]. However, this would require that the CarMa DFT account for the same effects and also produce 3-dimensional densities.

Most notably, the attraction due to atomic nuclei is missing from the DFT framework. A promising option is the use of pseudopotentials for capturing the exponential cusps due to the nuclei. To then generate 3-dimensional densities, we need to properly implement the ansatz of exponential decay in density. Then with these 3 different tools, we can calculate the

Offline	Online
1. Construct restricted pentagon mesh and vertex mask	1. Refine mesh given fullerene graph
2. Calculate heat equation solution	2. Identify sectors with pentagonal vertices
3. Calculate particular solution to linear equations, and find basis for null-space	3. Load weights and Voronoi areas for sectors of interest
4. Setup non-linear constraints and objective function	4. Construct default cotan laplacian
5. For each iteration in the heat equation time series, calculate manifold using constrained optimization	5. Replace cotan laplacian weights for vertices in sectors of interest.
6. From null-space coefficients calculate edge lengths and angles	
7. Calculate the Voronoi area and weights for every vertex in the sector	
8. Calculate local pentagon embedding	

Table 10.1: Schematic comparison between calculations that can be done offline and those that need to be done online.

densities of the C_{60} isomers and compare these to the above-mentioned results, and calibrate all 3 tools simultaneously, for example using an optimization of the relative energy levels between isomers.

10.3 Future Work

While calibration is one of the more pressing concerns with regards to producing useable results, other work also still needs to be done. Among these are the ever present issues of program optimization. The current implementation uses Python, and while this is great for prototyping, it is not expedient in its execution. A proper implementation would for example use C++, leveraging the speed inherent to compiled languages over interpreted ones. This will of course be useful in the calibration phase, but even more so in deployment, when we want to actually screen the isomer spaces.

Another missing ingredient is the inclusion of isolated-pentagon rule fullerenes. The issue here lay with our treatment of the computational domain. By excluding IPR fullerenes we were able to use a single computational domain, since we did not have to account for the different ways pentagons can be arranged. Their inclusion can be implemented using at least two different methods: extending the computational domain, or keeping the computational domain, but changing the boundary conditions.

The first is the most intuitive: if two (or more) pentagons are adjacent, well then we just let the heat flow on the union of their computational domains. However, for this to work as an offline calculation, we will need to systematize the different combinations possible

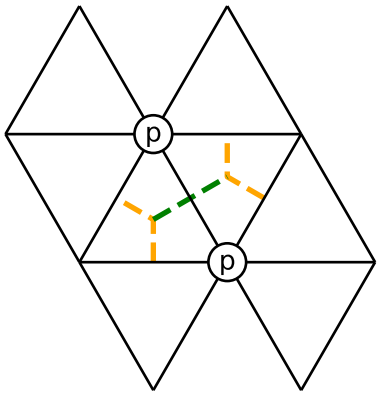


Figure 10.1: Two adjacent pentagon nodes (labelled with "p") and the union of their 1-ring neighbourhoods. Simulating the heat equation on their shared domain can be done by simulating on the full union of their domains, or by just simulating on a single sector, but with different boundary conditions over the shared Voronoi edge (the green dashed line).

and apply them correctly when encountering non-IPR fullerenes. The second approach relies on the symmetry of the adjacent pentagonal vertices. A setup can be seen in figure 10.1, where we see 8 triangles unfolded on the eisenstein plane, with two pentagon vertices. These vertices share two triangles, and two edges of their respective Voronoi regions (one in each triangle), highlighted as the green dashed line. Since the pentagons have the same initial curvature, we can use their symmetry to infer the slope of the curvature across their shared Voronoi boundary: it will always be zero. We can therefore treat them as separate, but with a different boundary condition. This has the added benefit in that we need to consider a smaller number of variations: We just look at each triangle around a pentagon vertex, and load in the corresponding result.

Likewise, we would like to extend this method beyond fullerenes to other low-dimensional systems, such as Fulleroids or Schwarzites. Fulleroids is another carbon allotrope forming polyhedral cages, but where the requirement of only hexagons and pentagons for a convex structure has been relaxed. We can therefore introduce heptagons and octagons, for example, which induce a negative curvature. This leads to exciting new possibilites in the shapes we can produce: from tori to peanuts and basically any embeddable, bounded shape we can imagine. Schwarzites on the other hand are nano-porous, space-filling structures with surfaces of negative curvature.

The methods introduced work directly on the manifold level, and applying them to heptagonal or octagonal meshes should therefore not prove a substantial challenge, especially if we use the symmetry-method for implementing non-IPR fullerenes, because instead of the solution being symmetric about the shared Voronoi boundary, the solution will now just be antisymmetric! This is because the curvature per sector of a heptagon or octagon is equal in magnitude to that of a pentagon, but with an opposing sign.

We also need a more robust optimization scheme for calculating a manifold from curvature. Currently we rely on Scipy to generate an orthonormal basis for the null-space of the angle and length equations, but we are not guaranteed that these vectors are parallel to the basis

vectors of our system. In other words, the free variables are linear combinations of angles and lengths respectively. This has the complication of making further, non-linear variable elimination harder. An algorithmic approach to identifying a set of angles and lengths that can be taken as free variables would alleviate this, and hopefully bring dividends when it comes to the optimization.

10.4 Conclusion

In this thesis we set out to develop a method for producing high resolution meshes for describing the surface of polyhedral molecules, without distorting the curvature per area. Traditional mesh fairing techniques like mean curvature flow proved unsuited for our purposes for two main reasons: First, it directly manipulats 3-dimensional geometry (while we want to keep our method on the manifold level). And second, it does not produce a strictly non-negative curvature.

However, the key insight for the method developed was to turn the method on its head, so to speak: By manipulating curvature directly, as variable whose evolution was governed by the heat equation, we are able to produce a satisfactory curvature distribution. This was achieved by two methods. First, an implementation where the spatial discretization was based on the finite element method, and another where we took a more heuristically driven approach to defining a Laplacian operator directly on the dual graph. The first method proved unfit for our purposes since it also produced negative curvatures. The second method however, produced satisfactory distributions.

We then tackled the problem of constructing a manifold from the given curvature distribution. The first attempt took the form of considering the angles of the triangular mesh separately from the lengths, where we solved a set of linear equations governing the angles. With the angles calculated, the lengths could be infered by a single known edge length, setting the global scaling of the metric and having this propagate out through the manifold.

The method proved insufficient, as some adjacent triangles disagreed on the length of their shared edge. A new approach was taken, where the angles and lengths were taken as a single system of unknowns. This led to three sets of equations: The same set of linear equations as before, relating the angles of the mesh to one another. Another set of linear equations sets the scale of the edge lengths, and makes sure the shared edges have the same lengths. Finally, a set of non-linear equations relating the lenghts and angles of a single triangle to one another: the law of sines.

A straight-forward root-finding on this set would involve a parameter space far larger than what is actually needed. Instead, a particular solution to the two sets of linear equations were found, and combined with an orthonormal basis for their null-spaces, this allowed us to generate all solutions to the equations, but with a much reduced set of independent variables. The non-linear equations were then recast in these new variables to allow for a more effective search for solutions. However, as the set of non-linear equations was underdetermined, we need to use constrained optimization. This yielded poor results for $H = 8$ and above, and while an attempt at improving the method was implemented, it did not produce better results.

For manifolds with a medium resolution, where correct results are produced, a method for calculating a local 3-dimensional isometric embedding was developed and optimized using the inherent symmetries of the manifold.

Appendix A

Algorithms

Algorithm 1: Vertex Sector

input : Original Triangulation T,
Original Triangle Index n,
Halma Index m,
Edge Dictionary A,
Vertex Tally N

output: Vertex Sector V,
Edge Dictionary A,
Vertex Tally N

begin

Create ragged array V with dummy indices
$V[0,0] \leftarrow T[n,0]$ `// Set the corner vertices`
$V[n,0] \leftarrow T[n,1]$
$V[n,n] \leftarrow T[n,2]$
if $(T[n,1], T[n,0]) \in A$ **then** `// First Edge`
  $a \leftarrow A[(T[n,1], T[n,0])]$ `// Grab value of edge dictionary`
  for $i \leftarrow 0$ **to** m **do**
    $V[i,0] \leftarrow a[m-i]$ `// And assign it to the sector`
  end
end
if $(T[n,2], T[n,1]) \in A$ **then** `// Second edge`
  $a \leftarrow A[(T[n,2], T[n,1])]$
  for $i \leftarrow 0$ **to** m **do**
    $V[m,i] \leftarrow a[m-i]$
  end
end
if $(T[n,0], T[n,2]) \in A$ **then** `// Third edge`
  $a \leftarrow A[(T[n,0], T[n,2])]$
  for $i \leftarrow 0$ **to** m **do**
    $V[i,i] \leftarrow a[i]$
  end
end
for $i \leftarrow 0$ **to** k **do** `// Replace remaining dummy indices`
  for $j \leftarrow 0$ **to** i **do**
    if $V[i,j] = -1$ **then**
      $V[i,j] \leftarrow N$
      $N \leftarrow N + 1$
    end
  end
end
$A[(V[0,0], V[m,0])] \leftarrow [V[0,i],\ \forall i \in \{0,\ldots,m\}]$ `// Populate edge dictionary`
$A[(V[m,0], V[m,m])] \leftarrow [V[m,i],\ \forall i \in \{0,\ldots,m\}]$
$A[(V[m,m], V[0,0])] \leftarrow [V[m-i,m-i],\ \forall i \in \{0,\ldots,m\}]$

end

Algorithm 2: Vertex Sector Tuple

input : Triangulation T,
Number of atoms N_C
output: Vertex Sector Tuple V
begin
  Initialize empty tuple V
  Initialize empty dictionary A
  $N \leftarrow \max(T) + 1$ `// Global vertex tally`
  for $n \leftarrow 0$ **to** $N_C - 1$ **do**
    Calculate vertex sector v for triangle $T[n]$
    $V[n] \leftarrow v$.
  end
end

Algorithm 3: Triangulation and Triangle Sector Tuple

input : Vertex Sector Tuple V,
Number of atoms N_C,
Harma Index m
output: Triangulation T,
Triangle Sector Tuple F

begin
  Initialize empty tuple F
  Initialize empty matrix T
  for $n \leftarrow 0$ **to** $N_C - 1$ **do** // Original triangle
    for $i \leftarrow 0$ **to** $m - 1$ **do** // Current vertex ring
      for $j \leftarrow 0$ **to** i **do** // Current vertex along ring
        $o \leftarrow n \cdot m^2$ // Offset due to original triangle
        $k \leftarrow o + i^2 + j$ // Current triangle index: offset + ragged index
        $g_1 \leftarrow V[i, j]$ // Index of current vertex
        $g_2 \leftarrow V[i + 1, j]$ // Index on next ring
        $g_3 \leftarrow V[i + 1, j + 1]$ // Index on next ring + 1
        $g_4 \leftarrow V[i, j + 1]$ // Current vertex + 1
        $T[k] \leftarrow [g_1, g_2, g_3]$ // First triangle
        $T[k + 1] \leftarrow [g_1, g_3, g_4]$ // Second triangle
        $F[n, i, 2j] \leftarrow k$ // First triangle index
        $F[n, i, 2j + 1] \leftarrow k + 1$ // Second triangle index
      end
      $g_1 \leftarrow V[i, i]$
      $g_2 \leftarrow V[i + 1, i]$
      $g_3 \leftarrow V[i + 1, i + 1]$
      $T[k + 2] \leftarrow [g_1, g_2, g_3]$ // Last triangle in ring.
      $F[n, i, 2i + 1] \leftarrow k + 2$
    end
  end
end

Algorithm 4: Sparse Adjacency Matrix

input : Vertex Sector Tuple V,
Number of atoms N_C,
Harma Index m,
Edge Dictionary A
Old Sparse Adjacency Matrix S_{old}.

output: Sparse Adjacency Matrix S.

begin

Initialize Sparse adjacency matrix S with dummy indices (-1)

for $n \leftarrow 0$ **to** N_C **do** `// Run over each sector`

for $i \leftarrow 1$ **to** $m-1$ **do** `// Inner vertices`

for $j \leftarrow 1$ **to** $i-1$ **do**

$g \leftarrow V[n, i, j]$ `// Global index of current vertex`

$n_1 \leftarrow V[n, i+1, j]$ `// Global index of neighbour 1`

$n_2 \leftarrow V[n, i+1, j+1]$

$n_3 \leftarrow V[n, i, j+1]$

$n_4 \leftarrow V[n, i-1, j]$

$n_5 \leftarrow V[n, i-1, j-1]$

$n_6 \leftarrow V[n, i, j-1]$

$S[g] \leftarrow [n_1, n_2, n_3, n_4, n_5, n_6]$ `// Insert neighbors in array`

end

end

for $i \leftarrow 1$ **to** $m-1$ **do** `// First edge`

$g \leftarrow V[n, i, 0]$

$n_1 \leftarrow V[n, i+1, 0]$

$n_2 \leftarrow V[n, i+1, 1]$

$n_3 \leftarrow V[n, i, 1]$

`AssignEdgeNeighbours` (g, n_1, n_2, n_3) `// Algorithm 6`

end

for $i \leftarrow 1$ **to** $m-1$ **do** `// Second edge`

$g \leftarrow V[n, m, i]$

$n_1 \leftarrow V[n, m, i+1]$

$n_2 \leftarrow V[n, m-1, i]$

$n_3 \leftarrow V[n, m-1, i-1]$

`AssignEdgeNeighbours` (g, n_1, n_2, n_3)

end

for $i \leftarrow 1$ **to** $m-1$ **do** `// Third edge, in reverse`

$g \leftarrow V[n, i, i]$

$n_1 \leftarrow V[n, i-1, i-1]$

$n_2 \leftarrow V[n, i, i-1]$

$n_3 \leftarrow V[n, i+1, i]$

`AssignEdgeNeighbours` (g, n_1, n_2, n_3)

end

end

`AssignCornerNeighbours` $(S_{\text{old}}, S, N_C, A)$ `// Algorithm 5`

end

Algorithm 5: AssignCornerNeighbours

input : Old Sparse Adjacency Matrix S_{old},
New Sparse Adjacency Matrix S,
Number of atoms N_C,
Edge Dictionary A.
output: Sparse Adjacency Matrix S.
begin
  $N_v \leftarrow N_C/2 + 2$
  for $i \leftarrow 0$ **to** $N_v - 1$ **do** `// Run over original vertices (corners)`
    for $j \leftarrow 0$ **to** 5 **do** `// Grab original neighbours`
      $n \leftarrow S_{\text{old}}[i, j]$
      if $n = -1$ **then** `// Check if the neighbour exists`
        continue
      end
      $a \leftarrow A[i, n]$ `// Grab vertices along original edge`
      $S[i, j] \leftarrow a[1]$ `// New neighbour is second vertex on edge`
    end
  end
end

Algorithm 6: AssignEdgeNeighbours

input : Sparse Adjacency Matrix S,
Global index of current vertex g,
Global index of neighbours n_1, n_2, n_3.
output: Sparse Adjacency Matrix S.
begin
  if $S[g, 0] = -1$ **then** `// First three neighbours are set`
    $S[g, 3] \leftarrow n_1$
    $S[g, 4] \leftarrow n_2$
    $S[g, 5] \leftarrow n_3$
  else
    $S[g, 0] \leftarrow n_1$
    $S[g, 1] \leftarrow n_2$
    $S[g, 2] \leftarrow n_3$
  end
end

Algorithm 7: Sparse Lengths and Weights matrices

input : Sparse Adjacency Matrix S,
Triangulation T,
Number of triangles N_F,
Angles θ,
Edge Lengths ℓ.

output: Sparse Adjacency Lengths, ℓ_S,
Sparse Adjacency Weights, w.

begin
  Initialize Sparse Length matrix ℓ_S with zeros
  Initialize Weights Matrix w with zeros
  for $n \leftarrow 0$ **to** $N_F - 1$ **do**
    for $m \leftarrow 0$ **to** 2 **do**
      $i \leftarrow T[n, m]$ `// Global index of current vertex`
      $j \leftarrow T[n, (m+1) \mod 3]$ `// Global index of next vertex`
      find j' such that $S[i, j'] = j$
      $k' \leftarrow (m+2) \mod 3$ `// Local index of previous vertex`
      $\ell_S[i, j'] \leftarrow \ell[n, k']$
      $w[i, j'] \leftarrow w[i, j'] + \cot(\theta[n, k'])/2$
    end
  end
end

Algorithm 8: Construct Heat Equation Mesh

input : Halma Index H,
Number of triangles N,
Input Triangulation T.
output: New Triangulation T_{new},
Vertex mapping P.

```
begin
    Refine input triangulation T, to get vertex sector tuple V and intermediate
     triangulation T_int (algorithms 2 and 3)
    Initialize empty list of vertex indices P
    for i ← 0 to H do            // Go through ragged indices and apply mask
        for j ← 0 to i do
            f ← 2i − j
            g ← i + j
            if f ≤ H ∧ g ≤ H then
                // Vertices with these ragged indices are in the mask.  Add
                   all vertices with these indices to the mapping.
                for n ← 0 to N do
                    if V[n, i, j] ∉ P then
                        Append V[n, i, j] to P
                    end
                end
            end
        end
    end
    Sort P
    N_F ← H^2 N
    Initialize empty triangulation T_new
    for i ← 0 to N_F do
        if T_int[i, j] ∈ P ∀i = 0, 1, 2 then
            // The triangle is in the Voronoi area.  Apply vertex mapping
            find a such that P[a] = T_int[i, 0]
            find b such that P[b] = T_int[i, 1]
            find c such that P[c] = T_int[i, 2]
            Append [a, b, c] to T_new
        end
    end
end
```

Algorithm 9: Local Pentagon Embedding

input : Halma Index H,
Vertex sector tuple V,
Vertex mask M.
output: Local Embedding P.
begin
  Construct unsmeared positions P
  $R_1 \leftarrow \text{Refl}(\pi/2 + 4\pi/5)$ `// Reflection across left border`
  $R_2 \leftarrow \text{Refl}(\pi/2)$ `// Reflection across midline of triangle`
  for $n \leftarrow 0$ **to** 4 **do**
    for $i \leftarrow H$ **to** 0 **do**
      for $j \leftarrow 0$ **to** $\lceil i/2 \rceil$ **do**
        if $M[i,j] = \mathit{False}$ **then** `// Make sure vertex is in mask`
          continue
        end
        `// Grab positions of neighbours`
        $v_0 \leftarrow V[n, i-1, j]$
        $p_0 \leftarrow P[v_0]$
        $v_1 \leftarrow V[n, i-1, j+1]$
        $p_1 \leftarrow P[v_1]$
        if $j = 0$ **then** `// If vertex has no left neighbour, make one`
          $p_2 \leftarrow R_1 P[v_1]$
        else
          $v_2 \leftarrow V[n, i, j-1]$
          $p_2 \leftarrow P[v_2]$
        end
        Calculate p using single point solver
        `// Grab vertex index and insert into position array`
        $v \leftarrow V[n, i, j]$
        $P[v] \leftarrow p$
        if $j \leq \lfloor i/2 \rfloor$ **then** `// Reflect vertex across midline of triangle`
          $v \leftarrow V[n, i, i-j]$
          $P[v] \leftarrow R_2 p$
        end
      end
    end
  end
end

Bibliography

[1] Simon Krarup Steen. Wave equations without coordinates. Master's thesis, University of Copenhagen, 2020.

[2] Hao Li and Heping Zhang. The isolated-pentagon rule and nice substructures in fullerenes. *Ars mathematica contemporanea*, 15(2):487–497, 2018.

[3] Peter Schwerdtfeger, Lukas N Wirz, and James Avery. The topology of fullerenes. *WIREs Computational Molecular Science*, 5(1):96–145, 2015.

[4] Sarah K. Norton, Dayanjan S. Wijesinghe, Anthony Dellinger, Jamie Sturgill, Zhiguo Zhou, Suzanne Barbour, Charles Chalfant, Daniel H. Conrad, and Christopher L. Kepley. Epoxyeicosatrienoic acids are involved in the c70 fullerene derivative–induced control of allergic asthma. *Journal of Allergy and Clinical Immunology*, 130(3):761–769.e2, 2012.

[5] John J. Ryan, Henry R. Bateman, Alex Stover, Greg Gomez, Sarah K. Norton, Wei Zhao, Lawrence B. Schwartz, Robert Lenk, and Christopher L. Kepley. Fullerene nanomaterials inhibit the allergic response. *The Journal of Immunology*, 179(1):665–672, 2007.

[6] Pawel Mroz, Anna Pawlak, Minahil Satti, Haeryeon Lee, Tim Wharton, Hariprasad Gali, Tadeusz Sarna, and Michael R. Hamblin. Functionalized fullerenes mediate photodynamic killing of cancer cells: Type i versus type ii photochemical mechanism. *Free Radical Biology and Medicine*, 43(5):711–719, 2007.

[7] Robert D. Kennedy, Alexander L. Ayzner, Darcy D. Wanger, Christopher T Day, Merissa Halim, Saeed I. Khan, Sarah H. Tolbert, Benjamin J. Schwartz, and Yves Rubin. Self-assembling fullerenes for improved bulk-heterojunction photovoltaic devices. *Journal of the American Chemical Society*, 130(51):17290–17292, 2008. PMID: 19053441.

[8] Sanaz Pilehvar and Karolien De Wael. Recent advances in electrochemical biosensors based on fullerene-c60 nano-structured platforms. *Biosensors*, 5(4):712–735, 2015.

[9] M. Neophytou, W. Cambarau, F. Hermerschmidt, C. Waldauf, C. Christodoulou, R. Pacios, and S.A. Choulis. Inkjet-printed polymer–fullerene blends for organic electronic applications. *Microelectronic Engineering*, 95:102–106, 2012.

[10] K.I. Bolotin, K.J. Sikes, Z. Jiang, M. Klima, G. Fudenberg, J. Hone, P. Kim, and H.L. Stormer. Ultrahigh electron mobility in suspended graphene. *Solid State Communications*, 146(9):351–355, 2008.

[11] A. D. Alexandrov. *Convex polyhedra.* Springer monographs in mathematics. Springer, Berlin, 2005.

[12] David Dedenbach. Combinatorial geometry of fullerenes: Towards a mixed-dimensional carbon manifold dft. Master's thesis, University of Copenhagen, 2021.

[13] L. Ridgway Scott Susanne C. Brenner. *The Mathematical Theory of Finite Element Methods*. Springer, 3 edition, 2008.

[14] Alan J. Davies. *The finite element method an introduction with partial differential equations*. Oxford University Press, Oxford ;, 2nd ed. edition, 2011.

[15] P. Schröder A. H. Barr M. Meyer, M. Desbrun. Discrete differential-geometry operators for triangulated 2-manifolds. 2003.

[16] Mathieu Desbrun Peter Schröder Keenan Crane, Fernando de Goes. Digital geometry processing with discrete exterior calculus. In *ACM SIGGRAPH 2013 courses*, SIGGRAPH '13, New York, NY, USA, 2013. ACM.

[17] José M Soler, Emilio Artacho, Julian D Gale, Alberto García, Javier Junquera, Pablo Ordejón, and Daniel Sánchez-Portal. The SIESTA method for ab initio order-n materials simulation. *Journal of Physics: Condensed Matter*, 14(11):2745–2779, mar 2002.

[18] James Emil Avery. *New Computational Methods in the Quantum Theory of Nano-Structures*. PhD thesis, 2011.

[19] James Emil Avery. Wave equations without coordinates I: Fullerenes. *Rendiconti Lincei. Scienze Fisiche e Naturali*, June 2018. Accademia Nazionale dei Lincei, Italian Academy of Sciences.

[20] David J Griffiths. *Introduction to Electrodynamics*. Prentice-Hall, 3 edition, 1999.

[21] M James. The generalised inverse. *Mathematical gazette*, 62(420):109–, 1978.

[22] Åke Björck. *Numerical Methods for Least Squares Problems*. SIAM, 1996.

[23] Peter Schwerdtfeger, Lukas Wirz, and James Avery. Program fullerene: A software package for constructing and analyzing structures of regular fullerenes. *Journal of Computational Chemistry*, 34(17):1508–1526, 2013.

[24] Rebecca Sure, Andreas Hansen, Peter Schwerdtfeger, and Stefan Grimme. Comprehensive theoretical study of all 1812 c60 isomers. *Phys. Chem. Chem. Phys.*, 19:14296–14305, 2017.