PI161-2-DECSim
Discrete Exterior Calculus: a Framework for Computer Simulation and Geometric analysis based on advanced mathematical concepts

Camilo Rey Torres
PI161-2-DECSim
Discrete Exterior Calculus: a Framework for Computer Simulation and Geometric Analysis based on Advanced Mathematical Concepts

Autor:
Camilo Rey Torres

MEMORIA DEL TRABAJO DE GRADO REALIZADO PARA CUMPLIR UNO DE LOS REQUISITOS PARA OPTAR AL TÍTULO DE

MAGÍSTER EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Director
Leonardo Florez Valencia

Comité de Evaluación del trabajo de Grado
Germán Combariza
Ing. Daniel Suarez

Página web del Trabajo de Grado
http://pegasus.javeriana.edu.co/~PI161-2-DECSim/

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
MAESTRÍA EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN
BOGOTÁ, D.C.
Junio, 2016
PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE INGENIERÍA
MAESTRÍA EN INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Rector Magnífico
Jorge Humberto Pelaez S.J.

Decano Facultad de Ingeniería
Ingeniero Jorge Luis Sánchez Tellez

Director Maestría en Ingeniería de Sistemas y Computación
Ingeniera Ángela Carrillo Ramos

Director Departamento de Ingeniería de Sistemas
Ingeniero Efraín Ortiz Pabón
Artículo 23 de la Resolución No. 1 de Junio de 1946

“La universidad no se hace responsable de los conceptos emitidos por sus alumnos en sus proyectos de grado. Solo velará porque no se publique nada contrario al dogma y la moral católica y porque no contengan ataques o polémicas puramente personales. Antes bien, que se vean en ellos el anhelo de buscar la verdad y la justicia.”
Acknowledgements

The road to completing this document has been steep, rocky and intricate. Finally, the end is near and I look back and see a lot of people that have supported me to see this research project coming to an end.

First of all, I would like to thank Rafael García Ph.D. for throwing me into Computer Graphics (without my consent): you set me on the path that I had been searching for. To my advisor Leonardo Florez-Valencia Ph.D., whose infinite patience and words of wisdom shaped me into the half-breed of mathematician and computer scientist I am today and helped me give birth to this, my first major result in Computer Sciences. There are not enough words in any language to thank both of you.

To my friends, Ingrid, Angelita, Pilar, long-gone Augusto, Andrea, and Claudia for giving me the emotional support and the much needed tough-love that pushed me to type the final letters of this document. Mom and Alex, my families, you have taken part with me in this seemingly never-ending effort of working, living, programming and writing, thank you for helping me lift burdens that I could not have carried alone.

Thanks to all of you guys I have not crashed and burned among thousands of lines of code, calculations and Goldberg’s Curvature and Homology. None of this would be possible without any of you.
## Contents

**Introduction**

1. **Topology of Triangulated Surfaces**
   1.1 Simplicial Complexes
   1.1.1 Concrete Simplicial Complexes in $\mathbb{R}^n$
   1.1.2 Rings and Neighborhoods of Simplicies
   1.2 Duality in simplicial complexes
   1.2.1 Duality in Concrete Complexes
   1.3 An algebraic structure over simplicial complexes
   1.3.1 Concrete $p$-chains in $\mathbb{R}^n$
   1.3.2 Neighborhood chains
   1.3.3 The Quad-Edge using $p$-chains

2. **DEC: Discrete Exterior Calculus**
   2.1 Continuous Exterior Calculus in a nutshell
      2.1.1 Basic Operators in Exterior Calculus
      2.1.2 Integration of differential forms
   2.2 Discrete Exterior Calculus for Triangulated Surfaces
      2.2.1 Losing the Atlas: The de Rham Map
      2.2.2 Interpolation functions
      2.2.3 The discrete differential operator
      2.2.4 Duality and Discrete Differential Forms
      2.2.5 Maps between Discrete Vector Fields and Differential Forms

3. **Using DEC**
   3.1 The Lean-and-Mean DEC Processing Library
      3.1.1 Features of *Lean-and-Mean*
      3.1.2 Creating simplicial complexes
      3.1.3 Fields and Forms
      3.1.4 Operators
      3.1.5 Rendering Features
   3.2 DEC Based Constructs
      3.2.1 DEC Based Differential Operators
      3.2.2 DEC Based Finite Elements
      3.2.3 DEC Related facts on Curvature
4 Results, Conclusions and Future Work

4.1 Discussion of Results

4.1.1 On the topology of Triangulated Manifolds

4.1.2 On DEC as a theoretical framework

4.1.3 Implementing DEC: The \textit{lean-and-mean} Processing library

4.2 Conclusions

4.3 Future Work
ABSTRACT

The following document summarizes the results of a research project regarding the construction and implementation of Discrete Exterior Calculus (DEC) applied to the context of commercially made triangulated surfaces (.obj meshes) using the Processing IDE. The document presents a full construction of the foundations of DEC theory starting from a simple approach to the topology of triangulated surfaces which allows us to build a theory of calculus that borrow from the theory of exterior calculus on differentiable manifolds to build a fully discrete theory of calculs that manages to relate topology to geometry introducing the notion of duality at several levels. We show both algorithm designs and Java code for Partial Differential Equation solution, mesh analysis and mean discrete curvature calculation as examples of applications of this theory and provide actual tests performed on meshes created with MakeHuman and Blender 2.74b.

RESUMEN

El presente documento resume los resultados de un proyecto de investigación sobre la construcción e implementación del cálculo exterior discreto (DEC) aplicado al contexto de superficies trianguladas (mallas) comerciales utilizando el IDE Processing, que presenta una construcción completa de la teoría DEC a partir de una aproximación simple a la topología de las mallas trianguladas, que permite una construcción natural de un cálculo complementamente discreto sobre superficies trianguladas repitiendo muchos de los resultados de la teoría de la geometría de formas diferenciales sobre variedades diferenciables a partir de elementos simples, computacionalmente amables que aprovecha las relaciones topológicas de las superficies trianguladas para definir operadores diferenciales aprovechando la noción de dualidad a varios niveles. El documento muestra tanto el diseño de algoritmos basados en DEC como código de Processing para la solución de ecuaciones diferenciales parciales, análisis de mallas y cálculo de curvatura media, sobre modelos creados con MakeHuman y Blender 2.74b.
Introduction

The following document summarizes the results of a research project regarding the construction and implementation of Discrete Exterior Calculus (DEC) applied to the context of commercially made, triangulated surfaces (meshes) using a commercially-made programming environment: The Processing IDE.

The document presents a full construction of the foundations of DEC theory starting from a simple approach to the topology of triangulated surfaces (chapter 1) which will allow us to build a theory of discrete calculus that mimics the theory of differential forms on manifolds using an \textit{ex-geometrica} approach (Chapter 4.1.1) that can easily be implemented in any object-oriented programming language as we show in chapter 3.

The foundations for DEC theory go back to the XIX century, specifically to the theory of Finite Elements and to the analysis of polyhedral surfaces; DEC yields algorithms and routines similar to Finite Volumes Schemes using Whitney Forms dating to the middle of the XX century (see Cangiani’s \textit{On low order mimetic Differences} [6] for a full historical background on DEC-like constructions). A. Hirani’s in 2003 coins the term \textit{Discrete Exterior Calculus (DEC)} [18] proposing a unified theoretical framework that allows the treatment of problems in Discrete Differential Geometry, Partial Differential Equations and mesh analysis in a consistent and \textit{in-situ} based fashion.

Since 2003, many known authors as Eitan Grinspun, Peter Schröder, Konrad Polthier, Yiyin Tong and Mathieu Desbrun have used DEC as a framework publishing important results in the field of discrete differential forms even in variational settings (see for example [8] and [19]). There are important endeavours in the field of discrete electromagnetism [23, 5, 37], as well as in CFD (Computational Fluid Dynamics) [36, 10, 33, 20] and on DEC based Lagrangian mechanics [28, 16, 7, 39, 32].

From the point of view of Computer Sciences, Computer Graphics stems from Computational Geometry, while Image Processing can be regarded as an \textit{applied} science. Computer Simulation however, seems to be nothing more than a \textit{term}, defined individually for each discipline: civil engineering, biology, physics, etc. Though fundamental nowadays, Computer Simulation is treated within science like a \textit{craft}, that involves using many techniques derived from numerical analysis, computer graphics, statistics, etc.

In our opinion, Computer Graphics, Image Processing and Computer Simulation are part of a single approach to Computation, that implies a constant tension between the continuous and the discrete that neither numerical analysis nor computational geometry can manage alone. Thus, we wish to work with Discrete Exterior Calculus (DEC) and present it in this document as a framework that allows us to define methods for dealing with differential quantities that surpasses the continuous-vs-discrete issue, that is based solely on discrete objects [10, 18].
One of the promises of DEC as a theory is to give a solid framework to develop concepts and tools for digital geometry processing involving elements of algebraic topology, that lead to fully discrete versions of algebraic topology topics such as de Rham homology and even the Poincaré conjecture (now theorem). The DEC Framework has been used also in mesh analysis like in *Discrete Hodge Operators* by R. Hiptmair [17] or SIGGRAPH’s DDG course notes of 2012[1] that apply many of the DEC concepts to create alternate solutions to problems in mesh optimization, tetrahedral mesh construction and curvature measuring.

Essentially, DEC relies on treating functions and vector fields over discrete surfaces and volumes as assignments on distinct elements of a mesh (vertices, edges, etc) and define differential operators as weighted sums of these scalar assignments over topologically related elements and using geometric quantities of the surface (e.g. edge length, face area, angles between edges) as weighing factors[10].

Following the theory of continuous exterior calculus, DEC exploits the consequences of Stokes’ Generalized Theorem to redefine differential equations as differential forms (integral principles) that create relations among quantities stored in different elements of a mesh to distribute known and unknown quantities among topologically related elements of a mesh, exploiting the notion of the boundary of a mesh element and the dual elements of a mesh to create sparse linear systems that represent differential equations, leading to hybrid schemes that resemble both FEM and Finite Differences[21, 8, 11].

In a scientific setting, DEC can help rewrite problems in differential equations as differential forms and allows us to connect the topology of a surface to its geometry (in the vector calculus sense) in a completely discrete manner, avoiding the use of limits[19, 17, 4]. Moreover, DEC-based algorithms to solve differential equations and DEC-based definitions of Differential Geometry concepts are friendly to computer programmers: by separating the topology and the geometry of a mesh and defining only two operators ∂ and ⋆, DEC leads to a hybrid scheme of mesh storing that combines many of the features of the Quad-Edge with the polygon-soup, over which the definition of differential operators arises in a natural fashion.
Chapter 1

Topology of Triangulated Surfaces

Part of the beauty of the riemannian geometry is its ability as a theory to combine two mathematical disciplines: geometry and topology that provide us with the elements to understand a variety of geometric objects called manifolds [13, 25, 2].

In a nutshell, manifolds are geometric objects that locally resemble known euclidean spaces onto which we can define differential structures that allow for the construction of differential equations in complex settings like special relativity theory. DEC theory takes elements of riemannian geometry, differential forms and differential geometry to provide us all the goodies of advanced mathematical tools for geometric analysis on a totally discrete setting by constructing a fully discrete theory of calculus comprising an advanced approach to differential geometry from a topological perspective, developing a small, yet powerful set of ingredients to represent differential operators that will allow us to solve differential equations in a completely discrete setting.

The purpose with this chapter is to provide the reader with an understanding of the topology of triangulated surfaces from very simple elements. Contrary to the mathematical definition of a topology (Munkres [30]) we will take a graph-theoretical approach, by introducing the topology of a triangulated surface by means of the notions of adjacency and cobordancy - the way vertices are related to each other to construct edges and faces - following the approach of simplicial complexes presented in Goldberg’s Curvature and Homology [13]. We point out that the concepts developed in this chapter are presented in a way that they can be implemented in many programming languages in a straight-forward fashion.

1.1 Simplicial Complexes

A mesh in Computer graphics is basically a collection of points in \( \mathbb{R}^n \) (vertices) that are related to one another to compose the faces of the mesh: triangles, quads, polygons in such a way that can represent assets in videogames, real world objects in scientific computing, etc. [21, 8].

In computer Graphics, the topology of the mesh refers to the way in which the faces of the mesh are put together. In the same sense that analysis make use of topology for defining many of the concepts needed calculus, we wish to derive a theory of discrete exterior calculus that derives from the notion of the topology of discrete surfaces. Our theoretical resource in this case will be the notion of a simplicial complex that will allow us to define an algebraic structure from which Discrete Exterior Calculus will arise in a natural fashion.
Our presentation derives from the notion of Simplicial Complexes following S.I. Goldberg’s *Homology and Curvature* [13] with a hint of inspiration from L.Guibas and J.S. Stolfi’s *Primitives for the Manipulation of General Subdivision and Computation of Voronoi Diagrams* [24].

**Definition 1.1.1 (Abstract Complex (S.I.Goldberg)).** A closure finite abstract complex $K$ is a countable collection of objects $\{S^p_i\}_{i \in \mathbb{N}}$ called simplexes satisfying the following properties:

1. to each simplex $S^p_i$ there is an associated integer $p \geq 0$ called its dimension ($\dim(S^p_i) = p$).
2. To the simplices $S^p_i$ and $S^{p-1}_j$ there is an associated integer denoted by $[S^p_i : S^{p-1}_j]$ called their incidence number.
3. There are only a finite number of simplexes $S^{p-1}_j$ such that $[S^p_i : S^{p-1}_j] \neq 0$
4. For every pair of simplexes $S^{p+1}_i$, $S^{p-1}_j$ whose dimension differs by two, we have

$$\sum_k [S^{p+1}_i : S^p_k][S^p_k : S^{p-1}_j] = 0$$

The dimension of a simplicial complex $K$, $\dim(K)$ is defined in this setting as the highest number $p$ found on the simplexes in $K$. The incidence number between simplices will be related to the adjacency between composing elements of a simplicial complex $K$ and will provide us with paramount features for DEC.

Goldberg’s definition of simplices as abstract objects is also followed by Guibas and Stolfi to study meshes that may be not only piecewise linear [17], but also whose faces The basic idea behind the Simplicial complex is to simplify the analysis of a surface to constituing elements that are simpler to analyze and understand than the surface as a whole [17] [13].

### 1.1.1 Concrete Simplicial Complexes in $\mathbb{R}^n$

Our objective in this section is to formally define a triangulated surface as we use in Computer Graphics. To do so, we start defining the simplest example a geometric object in $\mathbb{R}^n$, the simplex:

**Definition 1.1.2 ($p$-Simplex).** A $k$-simplex $\sigma \subset \mathbb{R}^n$ is the convex hull of $k + 1$ points in $\mathbb{R}^n$. That is, if $v_0, v_1, \ldots, v_p \in \mathbb{R}^n$,

$$\sigma^p = \text{hull}\{v_0, v_1, \ldots, v_p\} = [v_0, v_1, \ldots, v_p] = \left\{ \sum_{i=0}^p \alpha_i v_i : \alpha_i \in \mathbb{R}, \sum_{i=0}^p \alpha_i \leq 1 \right\} \quad (1.1)$$

the $v_i$ are known as the vertices of $\sigma$ and we call $k$ its dimension.

A surface embedded in $\mathbb{R}^3$ is in Goldberg’s sense a simplicial complex of dimension 2. We call the 2-simplices faces, the 1-simplices edges and the 0-simplices vertices. A concrete simplicial complex is a collection of well-behaved simplices in the sense that they do not overlap, there are no isolated vertices and no edges can be found that are not part of the boundary of faces in the complex.

**Definition 1.1.3 (Simplicial Complex).** A collection $\Sigma$ of simplices of up to dimension $n$ is called an $n$-dimensional Simplicial Complex if the following conditions are satisfied:
1. If $\sigma \in \Sigma$ is a $k$-simplex ($k \leq n$), and $\gamma \subset \sigma$ is a $k-1$ simplex (a sub-simplex of $\sigma$), then $\gamma \in \Sigma$

2. If $\sigma_1, \sigma_2 \in \Sigma$, either $\sigma_1 \cap \sigma_2 = \emptyset$ or $\sigma_1 \cap \sigma_2 \in \Sigma$.

**Definition 1.1.4** ($p$-face of an $r$-simplex). We say that $\sigma^p \subset \sigma^r$ ($p \leq r$) if the vertices of $\sigma^p$ are all vertices of $\sigma^r$. In such case, we say that $\sigma^p$ is a $p$-face for $\sigma^r$ [18] and we write $$\sigma^p \leq \sigma^r$$ (1.2)

We have modified the notation used by A.Hirani in *Discrete Exterior Calculus* [18]. We can say that two simplices are equal, via the $\leq$ relation. Essentially, two simplices are the same if they possess the same vertices or if they are faces of one another.

**Definition 1.1.5** (Equality between simplices). For simplices of the same dimension $\sigma_1^p$ and $\sigma_2^p$, we say that $\sigma_1^p$ and $\sigma_2^p$ are equal if and only if $$\sigma_1^p \leq \sigma_2^p \land \sigma_2^p \leq \sigma_1^p$$ (1.3)

These very basic definitions provide us with some fundamental constructs that will be useful later on and that give us hints as to how to implement DEC theory in the final chapter of this thesis.

We will define connectedness taking advantage of the incidence number proposed in [1.30] and following definition [1.1.1]. So far, an $n$-simplicial complex is defined as a collection of simplices following the requirements of definition [1.1.3] for $\Sigma$ an $n$-dimensional complex and an integer $0 \leq p \leq n$ define the $p$-skeleton of $\Sigma$ as follows [18]:

$$S_p(\Sigma) = \{ \sigma \in \Sigma : \text{dim}(\sigma) \leq p \}$$ (1.4)

we can isolate the sets of faces, edges and vertices of simplicial complexes by defining the $p$-subcomplexes of $\Sigma_p \subset \Sigma$ as follows:

$$\Sigma_p = S_p(\Sigma) \setminus S_{p-1}(\Sigma)$$ (1.5)

We can now define connectedness between two simplices using these subcomplexes:

**Definition 1.1.6** (Simplex connectedness). Let $0 \leq p \leq n$. Given $\sigma_1, \sigma_2 \in \Sigma_p$, we say that $\sigma_1$ and $\sigma_2$ are connected if and only if $$\sigma_1 \cap \sigma_2 \in \Sigma_{p-1}$$ (1.6)

We can also name connectedness as cobordancy and we will use these terms exchangeably throughout this document. In the next section we define the notion of neighborhoods and rings of simplices and take advantage of the notion of connectedness defined [1.1.6] and relation $\leq$ defined over simplices.

### 1.1.2 Rings and Neighborhoods of Simplicies

One of the most used tools for topological analysis of spaces is the notion of a basis for a topology. A basis for a topology is a essentially a more manageable way to characterize topological properties not by deciding over open sets (which can be complex enough), but by defining families of smaller open sets, from which we can construct open sets (see [30] for a definition of the basis of a topology in mathematics).

In contrast with mathematical topology, for simplicial complexes in $\mathbb{R}^n$ we are interested in defining topological concepts that can be construed solely on the components of the simplicial complex. Much in the same sense of digital topology over digital images (see for example E. Melin’s *Digital Topology and Khalimsky Spaces* [26]), we have to define open sets in terms of what the simplicial complex offers.
**Definition 1.1.7** (n-Ring of simplices). Given a simplicial complex $\Sigma$, let $0 \leq n \leq \dim(\Sigma)$ and choose $p < n$. Given $\sigma^p \in \Sigma_p$ we define the $n$-ring of $\sigma^p$ as the set

$$R_n(\sigma^p) = \{ \sigma^k \in \Sigma : \sigma^p \leq \sigma^k, p < k \leq n \}$$

(1.7)

In the case of abstract simplicial complexes, we speak of a simplex $\sigma$ to be contained in another simplex $\hat{\sigma}$ if and only if $\dim(\sigma) < \dim(\hat{\sigma})$ and define it as

$$\sigma < \hat{\sigma} \iff \sigma \subset \hat{\sigma}$$

(1.8)
in the case of concrete complexes in $\mathbb{R}^n$, it suffices to check that all the vertices of $\sigma$ belong to $\hat{\sigma}$ since concrete simplices are formed as the convex hull of their vertices. Needless to say, that simplex $\sigma_1 = [v_1, \ldots, v_p]$ is contained in simplex $\sigma_2$ if and only if

$$\sigma_1 \leq \sigma_2 \iff (\forall i| i = 0, \ldots, p : v_i \in \sigma_2)$$

(1.9)

Based on our definition of rings, we can now define well-behavedness in terms of adjacency as follows:

**Definition 1.1.8** (Well-behaved simplicial complex). An $n$-dimensional simplicial complex $\Sigma$ is said to be well behaved if, for every simplex $\sigma \in \Sigma$, such that $\dim(\sigma) < n$,

$$R_n(\sigma) \neq \emptyset$$

(1.10)

We will deal only with well-behaved complexes from now on. Given a $p$-dimensional simplex on a simplicial complex of dimension $n$ we can define the ascending chain of simplices for a given simplex as follows.

**Definition 1.1.9** (Ascending chain of simplices). Given a simplex $\sigma^p \in \Sigma_p$ and $n = \dim(\Sigma)$, we define an ascending chain for a simplex $\sigma^p \in \Sigma_p$ as a collection $\sigma^{p+1}, \sigma^{p+2}, \ldots \sigma^n \in R_n(\sigma^p)$ such that

$$\sigma^p < \sigma^{p+1} < \sigma^{p+2} < \ldots < \sigma^n$$

(1.11)

Ascending chains are subsets of the $n$-ring of a simplex in a simplicial complex. The neighborhood of a simplex is defined as a subset of its ring where all simplices are of the same dimension.

**Definition 1.1.10** (Neighborhood of a $p$-simplex). The neighborhood of a simplex $\sigma \in \Sigma$ is the following subset:

$$N(\sigma) = \{ \hat{\sigma} \in R_n(\sigma) : \dim(\hat{\sigma}) = \dim(\sigma) \}$$

(1.12)

In a discrete setting, we can say that an isolated vertex is a vertex that has empty ring of faces and consequently, empty neighborhood. In the case of well-behaved 2-complexes, the ring of faces around an edge contains at most 2 elements (1 in the case of border edges). For concrete 2-complexes in $\mathbb{R}^3$ we can define neighborhoods as follows:

**Theorem 1.1.11** (Concrete Neighboorhoods in $\mathbb{R}^2$). Let $\Sigma$ be a 2-complex embedded in $\mathbb{R}^3$. We define the neighborhood of a vertex $v \in \Sigma_0$ as the following collection

$$N(v) = \{ w \in \Sigma_0 : [v, w] \in \Sigma_1 \}$$

(1.13)

if $e \in \Sigma_1$ is an edge in $\Sigma$, we define its neighborhood as

$$N(e) = \{ \hat{e} \in \Sigma_1 : e \cap \hat{e} \in \Sigma_0 \}$$

(1.14)

and if $f \in \Sigma_2$ is a face in $\Sigma$, we define its neighborhood as

$$N(f) = \{ f \in \Sigma_2 : f \cap \hat{f} \in \Sigma_1 \}$$

(1.15)
1.2 Duality in simplicial complexes

We have defined connectedness over triangulated surfaces making use of the concept of neighborhoods and rings. Now we are ready to present the other fundamental DEC ingredient: duality. As we will show in upcoming chapters, connectedness is the key to define differential operators in the realm of triangulated surfaces and the notion of circulation of quantities (that for example the Laplacian measures). The notion of flux over a simplicial complex, we require the creation of a new object called the dual complex for which this section is dedicated.

Given an \( n \)-dimensional simplicial complex \( \Sigma \), we can define a graph structure over \( \Sigma_p \) by means of the rings of simplices defined previously (recall 1.1.7). Given \( \sigma_1, \sigma_2 \in \Sigma_p \), we can say that \( \sigma_1 \) and \( \sigma_2 \) are related if they are adjacent to each other:

\[
\sigma_1 \sim \sigma_2 \Leftrightarrow \sigma_2 \in N(\sigma_1)
\] (1.16)

relation \( \sim \) is always reflexive and symmetric, but always not transitive (only in small simplicial complexes). In this sense, we can think of set \( \Sigma_p \) of an \( n \)-dimensional simplicial complex as the graph

\[
G_p = (\Sigma_p, \sim)
\] (1.17)

In the case of concrete simplicial complexes, \( G_p \) composes a planar graph\([21, 1]\). This fact is easily verified for the only possible intersection between two simplices in a well-behaved simplicial complex occurs at simplex of lower dimension\([13, 24]\). Moreover, \( G_p \) is a connected graph if \( \Sigma \) is well-behaved. We denote the dual graph to \( G_p \) as \( G_p^* \) as follows

\[
G_p^* = (\Sigma_p^*, \sim)
\] (1.18)

where \( \Sigma_p^* \) stands for the dual elements to the simplices in \( \Sigma \). Recall from definition 1.1.1 that simplices were merely defined as objects. In the sense of this definition, we define \( \Sigma_p^* \) as follows:

\[
\Sigma_p^* = \{ \star(S^p_i) : S^p_i \in \Sigma_p \}
\] (1.19)

where operator \( \star \) transforms an object into its dual objects. \( \Sigma_p^* \) and \( \Sigma_p \) are isomorphic in a set theoretic way via operator \( \star \).

We can define the dual complex to \( \Sigma \) as the collection of all the duals to the simplices in \( \Sigma \):

\[
\Sigma^* = \bigcup_{p=0}^{n} \Sigma_p^*
\] (1.20)
1.2.1 Duality in Concrete Complexes

In the sense of definition 1.1.1, the dual complex is also an abstract simplicial complex. However, in the sense of the concrete simplicial complexes over \( \mathbb{R}^n \) as we have presented in subsection 1.1.1, the dual of a simplicial concrete is not necessarily composed of concrete simplices, but composed of cells \([18, 11, 10]\).

Given that concrete simplices in \( \mathbb{R}^n \) are defined as the convex hull of given points, we define the dual cells also by means of convex hulls but take into account the connectedness relation necessary for defining the dual graph. Thus, for a concrete \( p \)-dimensional simplex embedded in \( \sigma \subset \mathbb{R}^n \), we define its dual cell \( \star(\sigma) \) -and therefore operator \( \star \)-as follows

**Definition 1.2.1** (Dual cell to \( p \)-simplex (Dualizing star operator\([18]\))). Given \( \sigma^p \in \Sigma_p \) on a concrete \( n \)-dimensional simplicial complex \( \Sigma \), we define its dual cell as the union of the convex hulls of the center of every ascending chain from \( \sigma \)

\[
\star(\sigma) = \bigcup_{\sigma^p < \sigma^{p+1}, \ldots < \sigma^n} [c(\sigma^p), c(\sigma^{p+1}), \ldots, c(\sigma^n)]
\]

(1.21)

as a set, we can restate this solely using the \( n \)-ring of a \( p \)-simplex by taking the convex hull of all the center

\[
\star(\sigma) = \text{hull}\{c(\hat{\sigma}) : \sigma \in R_n(\sigma)\}
\]

(1.22)

Operator \( c \) calculates the center of a given concrete simplex in \( \mathbb{R}^n \) for which we have two options according to the literature:

1. the barycenter of \( \sigma \)

2. the circumcenter of \( \sigma \),

In the case of 2-simplices we could also add the incenter of the face (a weighted sum of its vertices starting using the face perimeter). The DEC literature favors the circumcenter for when the simplicial complex composes a Delaunay Triangulation, its dual complex results in the Voronoi diagram of the vertices\([24]\). Circumcentric subdivision -in the context of Delaunay Triangulations- offers geometric advantages for DEC\([18, 10, 11, 28, 9]\). Figure 1.2 shows a small complex with its distinct dual elements. In the case of concrete 2-complexes in \( \mathbb{R}^n \), we can see the graphical difference between circumcentric and barycentric subdivision. We will name \( \Sigma \) as the primal complex whereas \( \Sigma^* \) as the dual complex from now on. We can write the dual sub-complexes (dual to the ones defined in equation 1.5) as

\[
\Sigma^*_p = \{\star(\sigma) : \sigma \in \Sigma_p\}
\]

(1.23)

The topological concepts and constructions we have defined so far work exactly the same over the dual complex. Moreover, as sets, the primal and the dual complex they have the same cardinality: as algebraic structures, they are isomorphic as we will present in the following section.

1.3 An algebraic structure over simplicial complexes

At the core of DEC as a computational method lies the idea of describing fields and functions over simplicial complexes as quantities attached to faces, edges and vertices of a complex and defining operators to let these quantities flow through rings and simplices. This notion of attaching a vector or a scalar to a simplex is done by the introduction of the notions of free abelian groups and free vector spaces and the
treatment of these quantities allows us to define an algebraic structure over a simplicial complex.

Over an abstract simplicial complex $K$, its free abelian group over a group $G$ is taken as the set of formal sums of simplices that are accompanied by coefficients in group $G$: 

$$\sum_{i \geq 0} c_i S_i^p$$

(1.24)

for $c_i \in G$. When taking groups like $\mathbb{Z}$ we could think that there is a geometric meaning to these sums (e.g. enlarging faces or edges). In practice, this assumption is only partially right. The boundary of a simplex $S_i^p$ of dimension $p$ can be written as the following formal sum 

$$\partial S_i^p = \sum_j [S_i^p : S_j^{p-1}] S_j^{p-1}$$

(1.25)

where the sum is taken over all $p - 1$-dimensional simplices. The space of all these formal sums composes the set of chains over a simplicial complex $K$, $C^k(K, G)$.

**Definition 1.3.1** (The Space of $p$-chains over an Abstract Simplicial Complex (Goldberg [13])). *Given a simplicial complex of dimension $K$ and an integer for $0 \leq p \leq \text{dim}(K)$, we define the space of $p$-chains over $K$ as the abelian group generated by the $p$-dimensional simplices of $K$ with coefficients on $\mathbb{Z}$:*

$$C_p(K, G) = \left\{ \sum_j c_j S_j^p : c_j \in G \right\}$$

(1.26)
As we will show in upcoming chapters, instead of thinking that the coefficients in $p$-chains operate over the simplices geometrically, we can think of these coefficients as global representations of quantities defined over the complex (e.g. temperature, curvature). To the sign of these numbers however, we can attach a geometric meaning as we will show in the following section.

### 1.3.1 Concrete $p$-chains in $\mathbb{R}^n$

Recall, the definition of the boundary as a linear operator between $p$-chains and $p-1$-chains. In our realization of Simplicial Complexes, we will work for the moment with the free abelian group

$$C_p(\Sigma) = C_p(\Sigma, \mathbb{Z})$$

(1.27)

The boundary of a $p$-simplex as the collection of its $(p-1)$-faces and can be written as a $(p-1)$-chain:

$$\partial \sigma^p = \partial [v_0, \ldots, v_p] = \sum_{i=0}^{p} (-1)^i [v_0, \ldots, \hat{v}_i, \ldots, v_p]$$

(1.28)

where $\hat{v}_i$ removes vertex $v_i$ from simplex $\sigma^p$. From definition [1.1.1] we recall the incidence number between simplices and we present it in the concrete setting as:

$$[[v_0, \ldots, v_i, \ldots, v_p] : [v_0, \ldots, \hat{v}_i, \ldots, v_p]] = (-1)^i$$

(1.29)

and we can extend its value between arbitrary faces and edges and between edges and vertices as

$$[\sigma^p : \sigma^{p-1}] = \begin{cases} 0, & \sigma^{p-1} \text{ is not a part of the boundary of } \sigma^p \\ \pm 1, & \text{otherwise} \end{cases}$$

(1.30)

The notion of orientation arises in this point. The ordering of the vertices of a concrete simplex proposes us a way of traversing the boundary of the simplex -according to equation [1.28]. In the case of edges, the orientation of edge $e = [v_0, v_1]$ defines a way of parametrizing edge as a line segment that starts on $v_0$ and ends in $v_1$ when the orientation is 1 and reverses the direction when the orientation is $-1$.

In the case of faces, we can define the normal vector to face $f = [v_0, v_1, v_2]$ as the vector that defines the direction outwards from face $f$. Changing the ordering of the vertices within face $f$ will alter the
To complete the requirements of definition 1.1.1, we have to notice that in a 2-complex in $\mathbb{R}^2$, every edge belongs to at most two faces and every vertex is surrounded by a finite number of edges (condition 3 on the definition) [18, 25, 8]. These facts can be used to see that $\partial^2 \equiv 0$. Indeed, let $v$ be a vertex of a face $f = [v_0, v_1, v_2]$. Without loss of generality assume that $v = v_0$, notice that

$$v_0 \leq [v_0, v_1] \leq [v_0, v_1, v_2]$$

$$v_0 \leq [v_0, v_2] \leq [v_0, v_1, v_2]$$

(1.31)

Define $e_0 = [v_1, v_2]$, $e_1 = [v_0, v_2]$ and $e_2 = [v_0, v_1]$. Notice that so that for $v_0$ and $f = [v_0, v_1, v_2]$ we have that

$$\sum_{k} [e_k : v_0][e_k : f] =$$

$$= [e_0 : v_0][e_0 : f] + [e_1 : v_0][e_1 : f] + [e_2 : v_0][e_2 : f]$$

$$= (0)(1) + (1)(-1) + (1)(1)$$

$$= 0$$

(1.32)

In Computer Graphics we are always working with finite simplicial complexes. Thus, we have finite sets of vertices $\Sigma_0$, edges $\Sigma_1$ and $\Sigma_2$ and we can think of the $p$-chain spaces as (1) finitely generated, and (2) as having a representation as vectors in the sense of linear algebra. DEC takes advantage of this vector representation to treat-for example-PDE problems as sparse linear systems in the same sense that FEM and Finite Difference Methods do.

Essentially, for triangulated surfaces, we can write their $p$-chain spaces as follows:

$$C_0(\Sigma) = \left\{ \sum_{i=0}^{N_v} c_i v_i : c_i \in \mathbb{Z} \right\}$$

$$C_1(\Sigma) = \left\{ \sum_{i=0}^{N_e} c_i e_i : c_i \in \mathbb{Z} \right\}$$

$$C_2(\Sigma) = \left\{ \sum_{i=0}^{N_f} c_i f_i : c_i \in \mathbb{Z} \right\}$$

(1.33)

where $N_v$, $N_e$ and $N_f$ stand for the number of vertices, edges and faces respectively. In the case of 2-complexes, there are two boundary operators $\partial_2$ and $\partial_1$ working from faces to edges and from edges to vertices. Condition 4 of definition 1.1.1 guarantees that the chain of operators $\partial_2$ and $\partial_1$ conforms an exact sequence [2, 13]:

$$0 \xleftarrow{\partial} C_0(\Sigma) \xrightarrow{\partial_1} C_1(\Sigma) \xrightarrow{\partial_2} C_2(\Sigma) \xrightarrow{\partial} 0$$

(1.34)

Notice that in terms of $p$-chains, we can also define connectedness between simplices by means of

$$\partial \sigma_2 - \partial \sigma_1 \neq 0$$

(1.35)

Recall equation 1.21 defining the dual to a $p$-simplex $\sigma^p$. In terms of $p$-chains, we can restate it using $p$-chains as

$$*(\sigma) = \sum_{\sigma_p \subset \sigma^{p+1} \ldots \subset \sigma^n} c_{\sigma_p \sigma^{p+1} \ldots \sigma^n} [c(\sigma^p), c(\sigma^{p+1}), \ldots, c(\sigma^n)]$$

(1.36)
for $c_{\sigma_{p+1}} \in 0, \pm 1$ that are carefully chosen to respect orientation of the dual cell complex. In the context of $p$-chains as algebraic structures ($\mathbb{Z}$-modules), we say that $\partial$ acts as a linear operator

$$\partial : C_p(\Sigma) \rightarrow C_{p-1}(\Sigma)$$

(1.37)

while operator $\star$ acts as a linear operator between $\Sigma$ and $\Sigma^*$:

$$\star : C_p(\Sigma) \rightarrow C_{n-p}(\Sigma^*)$$

(1.38)

where $n$ is the dimension of the simplicial complex. We will abuse the notation provided by $p$-chains. Given a $p$-chain $c = \sum_{i=0}^{N_p} c_i \sigma_i$ we say that $p$-simplex $\sigma$ belongs to $c$ if and only if the coefficient associated to $\sigma$ is non-zero.

### 1.3.2 Neighborhood chains

We can restate the neighborhoods of simplices using the operators defined so far. This result will condition the presentation of many DEC concepts we will explore in the next chapter over surfaces and constitutes one of the major contributions of this work to DEC theory. Through these neighborhood chains we will be able to state differential operators using compact notation that though strange is very computationally friendly, specially as we introduce the *iterators* over triangulated surfaces in chapter 3.

![Figure 1.4: Primal-Dual face neighborhood relations](image)

**Theorem 1.3.2 (Neighborhood chains).** Let $\Sigma$ be a 2-complex embedded in $\mathbb{R}^n$. We can write the neighborhoods of elements of $\Sigma$ and $\Sigma^*$ using only operators $\star$ and $\partial$ as $p$-chains as follows:

1. Given a primal vertex $v \in \Sigma_0$ the 0-chain associated with its neighborhood is given by

$$N_v = \partial(\star(\partial(\star(v))))$$

(1.39)

2. Given a primal edge $e \in \Sigma_1$, the 1-chain associated with its neighborhood is given by

$$N_e = \star(\partial(\star(\partial e)))$$

(1.40)

3. Given a dual vertex $w \in \Sigma^*_2$, the 0-chain associated with its neighborhood is given by

$$N_w = \partial(\star(\partial(\star(w))))$$

(1.41)
4. Given a dual edge $g \in \Sigma^*$, the 1-chain associated with its neighborhood is given by

$$N_g = \star(\partial(\star(g)))$$

(1.42)

In similar fashions, we can define neighborhoods of primal and dual faces in a very simple fashion. Given $f$ a primal face of $\Sigma$

$$N(f) = \{\star(w) : w \in N(\star(f))\}$$

(1.43)

and given a dual face $h \in \Sigma^*$ we have

$$N(h) = \{\star v : v \in N(\star(h))\}$$

(1.44)

(a) (b) (c)

Figure 1.5: Neighborhoods of Dual cells. (a) Dual vertex neighborhood, (b) dual edge neighborhood, and (c) dual face neighborhood.

1.3.3 The Quad-Edge using $p$-chains

L. Guibas and J. Stolfis’ paper on the primitives for the calculation of Voronoi Diagrams proposes a topological data structure known as the Quad-Edge as an alternative for the construction of the Voronoi Diagram and the Delaunay Triangulation[24] that was defining in constructing our approach to DEC. To honor them, we present this small subsection about an algebraic construction of the Quad-Edge structure using only operators $\partial$ and $\star$. In our terms, a Quad-Edge is a structure based on edges to store both the topology and the geometry of an object that enables traversing both the primal and dual complex by storing the primal and dual edges in the same object[24 22 12]. The Quad-Edge proposes an algebraic structure of operators to traverse the complex using an operator called $\text{rot()}$ as a dualizing operator allowing us to traverse from the primal to the dual complex, exactly like operator $\star$. Figure [1.7] shows the contents of a Quad-Edge over an edge $e$:

1. the edge $e$
2. its dual edge $\text{rot}(e)$
3. the vertices composing the boundary of $e$, $\text{org}(e)$, $\text{dest}(e)$.
4. the faces onto which $e$ is a border: $\text{fLeft}(e)$, $\text{fRight}(e)$

Quad-Edge takes into account the orientation of the simplicial complex and introduces the notions of left and right for faces from the point of view of the edges. Our approach does not concern itself with orientation as much as the Quad-Edge, however, we see strong points of connection between the Quad-Edge and our operators $\star$ and $\partial$ as follows:
Figure 1.6: Quad Edge Construction using DEC topological Operations. (a) initial edge, (b) dual edge, (c) dual edge neighborhood and (d) Quad-Edge edges obtained from dual face boundaries.

Figure 1.7: The contents of a Quad Edge. Primal Edge (red). Dual Edge (blue), left and right faces (grey), and Quad-Edge edges (black).

1. \( \text{rot}(e) = \ast(e) \)
2. \( \ast(\partial(e)) \) stores both \( f_{\text{Left}}(e), f_{\text{Right}}(e) \)
3. the vertices \( \text{dest}(e) \) and \( \text{org}(e) \) compose the boundary of \( e \). In terms of \( p \)-chains \( \text{dest}(e) - \text{org}(e) = \partial e \)

Figure 1.6 refers to an algorithmic construction of the Quad-Edge based on our definitions. Following algebraic trend we have followed throughout this chapter, we can write the Quad-Edge not as a single \( p \)-chain, but as an extended \( p \)-chain belonging to the direct sum of the \( p \)-chain spaces over a 2-complex:

\[
\Sigma_0 \oplus \Sigma_1 \oplus \Sigma_2 \oplus \Sigma_1^* \tag{1.45}
\]

The Quad-Edge defines several operators to traverse the mesh that define what the previous and next edges should be:

1. \( \text{oNext}(e) \): the next edge with the same origin.
2. \text{sym}(e)\): the edge with the same vertices but exchanging their order (switching destination and origin)

by using these two operators, the Quad-Edge manages to calculate all adjacent edges to \(e\) that have a common face (the left face or the right face):

1. \(\text{RNext}(e) = \text{sym}(\text{oPrev}(e))\):
2. \(\text{LPrev}(e) = \text{sym}(\text{oNext}(e))\):
3. \(\text{RPrev}(e) = \text{sym}(\text{dPrev}(e))\):
4. \(\text{LNext}(e) = \text{sym}(\text{dNext}(e))\)

We interpret operator \text{sym}() in terms of \(p\)-chains as multiplying by \(-1\). Edges \(e_1 = \text{dNext}(e), e_2 = \text{oNext}(e), e_3 = \text{dPrev}(e),\) and \(e_4 = \text{oPrev}(e)\) are defined as the adjacent edges to \(e\) that share either the common left face or right face. In terms of adjacency, we must have that

\[
e_1, e_2, e_3, e_4 \in \partial f_1 + \partial f_2
\]

Notice that

\[
\ast(e) = c_1[c(e), c(f_1)] + c_2[c(e), c(f_2)]
\]

thus we must have that \(\text{invRot}(e)\) and \(\text{rot}(e)\) are in terms of \(p\)-chains multiples of \(\ast(e)\) as figure 1.6 part (b) shows. edges \(e_1, e_2, e_3, e_4\) compose the boundary of \(f_1\) and \(f_2\) and are reachable by means of a \(p\)-chain. Namely:

\[
\ast(e_1), \ast(e_2), \ast(e_3), \ast(e_4) \in \partial \ast(f_1) + \partial \ast(f_2)
\]

notice that these edges are all part of the neighborhood of \(\ast(e)\). So that the edges mentioned in the quad-edge can be found easily as

\[
\text{quadEdge}(e) = \{\ast(\hat{e}) : \hat{e} \in N(\ast(e))\}
\]

as figure 1.6 part (c) and (d) show. Thus, we can write a Quad-Edge as a \(p\)-chain over the contents of the dual edge neighborhood by choosing appropriate coefficients:

\[
\text{quadEdge}(e) = \sum_{\hat{e} \in N(\ast(e))} c_{\hat{e}} \ast(\hat{e})
\]
Chapter 2

DEC: Discrete Exterior Calculus

We have decided to show first show the necessary topological elements for stating DEC theory. Now, we present this chapter that details the construction of DEC theory starting from Continuous Exterior Calculus to develop Discrete Exterior Calculus over simplicial complexes. We strive to give the reader a natural approach to the construction of DEC theory.

Last chapter we defined a handful of elements useful to study the topology of triangulated surfaces. Upon careful reading, the reader can notice that the elements we propose apply not only to triangulated surfaces, to collections of tessellated objects. We deliberately avoided speaking about coordinates. Now, we must deal with the geometric aspects of triangulated surfaces and by geometric we mean differential geometric and link the algebraic structure defined in section 1.3 to the idea of differential quantities in triangulated meshes.

In order for us to present Discrete Exterior Calculus, we must first take a look at Continuous Exterior Calculus and a little bit about the theory of differential forms. Though we are constructing a fully discrete theory, certain aspects of the continuous version of exterior calculus is needed for us to be able to present DEC in a coherent way. Indeed, the notion of interpolation in Discrete Exterior Calculus relates to the topic of partitions of the unit in the case of Continuous Exterior Calculus, while the notion of a differential form in the discrete sense can only be understood in depth by analyzing the continuous definition.

2.1 Continuous Exterior Calculus in a nutshell

Exterior Calculus deals with geometric objects outside of the familiar $\mathbb{R}^n$ (hence the name exterior). Instead of thinking about objects as being defined ultimately in terms of one coordinate system, we can think of geometric objects (manifolds) as topological spaces onto which we can define a patch-like collection of coordinate systems that will allow us to bring vector calculus in a local fashion[2, 27]. This local resemblance of an object to the familiar $\mathbb{R}^n$ is achieved by introducing the notion of a chart into the geometric object:

**Definition 2.1.1** (Chart). Let $M$ be a topological space satisfying the Hausdorff axiom. Given a point $p \in M$ a chart for $M$ around $p$ is a pair $(U, \phi)$ such that

1. $p \in U$ and $U$ is an open set of $M$.
2. $\phi : U \rightarrow \mathbb{R}^n$ is an homeomorphism over $\phi(U)$. 

23
An informative collection \( \{ (\phi_a, U_a) \}_{a \in A} \) of charts for a manifold \( M \) comprises an Atlas for \( M \) and defines \( M \) as an \( n \)-dimensional differentiable manifold\[27\]. The trick in exterior geometry is to think that we can borrow the idea of differentiation and integration from \( \mathbb{R}^n \) and import it into the setting of manifolds by using the Atlas to define a differential structure using the charts of the Atlas, the Chain Rule and the Inverse Function Theorem[27, 13, 29].

In this sense, we can start thinking about differentiation in the realm of manifolds from the idea of directional derivatives, by means of the idea of a tangent vector to a manifold \( M \). Given \( f : M \to \mathbb{R} \) a continuous function and a point \( p \in M \), let \((U, \phi)\) be a chart around \( p \). Then, by the chain rule[2], we can define the partial derivative of \( f \) as

\[
\frac{\partial f}{\partial x^i}(p) = \left. \frac{\partial}{\partial \alpha_i} (f \circ \phi^{-1}) \right|_p
\]  \hspace{1cm} (2.1)

A tangent vector \( X \) in the sense of manifolds is a directional derivative acting over scalar functions on the manifold[27, 13]:

\[
X = a_1 \frac{\partial}{\partial x^1} + a_2 \frac{\partial}{\partial x^2} + \ldots + a_n \frac{\partial}{\partial x^n}
\]  \hspace{1cm} (2.2)

for \( a_1, a_2, \ldots, a_n \in \mathbb{R} \). In the usual vector calculus notation, we can write this as

\[
X(f)|_p = \mathbf{a} \cdot \nabla (f \circ \phi^{-1})
\]  \hspace{1cm} (2.3)

The idea of \( X \) as a vector arises from regarding the \( a_i \) as components of a vector \( \mathbf{a} = (a_1, a_2, \ldots, a_n) \). The value of \( X(f) \) depends on the choice of coordinates in each chart. However, the idea of vector \( \mathbf{a} \) is unique for each choice of charts around a point[2, 13, 27] and leads us to the definition of the tangent space to point \( p \in M \) as:

\[
T_p M = \text{span} \left\{ \frac{\partial}{\partial x^i} \bigg|_p : i = 1, \ldots, n \right\}
\]  \hspace{1cm} (2.4)

Notice that \( T_p M \) begs for functions in order to compose an actual vector space[27]. However, we can regard \( T_p M \) as vector space over \( \mathbb{R} \) with the usual notions of addition and scalar product. Furthermore, we can regard \( T_p M \) as an inner product space[27], so that we can begin to define the notion of the dual space to \( T_p M \), denoted by \( T^*_p M \), with regards to this inner product and we can introduce the notion of a covector[13].

**Definition 2.1.2** (Covector over a Manifold (Goldberg[13])). A covector over a manifold \( M \) around a point \( p \in M \) is a linear functional

\[
\omega : T_p M \to \mathbb{R}
\]  \hspace{1cm} (2.5)

the collection of all covectors around point \( p \in M \) composes the cotangent space to \( M \) at \( p \) denoted by \( T^*_p M \).

In the same sense that partial differentials compose the tangent space to a manifold (an \( n \)-dimensional vector space), the cotangent space can be thought of as a vector space in itself, \( T^*_p M \) as the vector spaces spanned by a set of basis functions:

\[
dx^j : T_p M \to \mathbb{R}
\]  \hspace{1cm} (2.6)

defined by

\[
dx^j \left( \frac{\partial}{\partial x_i} \right) = \begin{cases} 1 & , \ i = j \\ 0 & , \ otherwise \end{cases}
\]  \hspace{1cm} (2.7)
The choice of notation using partial derivatives and integration symbols is not gratuitous. Indeed, tangent vectors will encode information about differential quantities defined over the manifold. Meanwhile, covectors will help us define the fundamental ingredient of exterior geometry: differential forms. Differential forms allow us to perform calculations on differential quantities and allow us to encompass things like the curl of a vector field, advection, etc.\[10, 21\]. Moreover, differential forms can be also differentiated, integrated, glued together, etc by means of a small set of operators that generalize the operations of vector calculus to the realm of Manifolds.

2.1.1 Basic Operators in Exterior Calculus

One of the advantages of working in exterior calculus is the possibility to define a relative small set of operators that will accomodate not only the usual differential operators of vector calculus that describe the rate of change of both scalar and vector functions (the circulation of tensors), but introduces operators that allow us to speak about their flux when moving through surfaces\[25 18 17\].

We present a small subset of the operators over manifolds that we can call fundamental: \(\sharp\) and \(\flat\) that define mappings between vectors and covectors, the differential \(d\) that will generalize the usual differential operators of vector calculus (gradient, curl, divergence, etc.) , and the Hodge Dual \(\ast\) that defines the notion of flux in the context of Manifolds. A full presentation of this operators can be found in Montesdeoca’s Introducción a las variedades diferenciables\[27\] or Marsden’s Manifolds, Tensor Analysis and Applications\[2\].

The vector-covector relation in the sense of duality is more easily understood by the introduction of an inner product among them. Formally speaking, we can think of an inner product among vectors (or covectors) as a special kind of tensor (a generalization of the dot product)\[2\]. Indeed, we can construct the usual dot product between vectors and between covectors that defines the duality between and allows us to construct the \(\sharp\) and \(\flat\) isomorphisms between forms and vectors. For each \(X \in T_pM\) we can define a covector \(X^\sharp : T_pM \to \mathbb{R}\)

\[
X^\sharp(Y) = X \cdot Y
\]  
(2.8)

since inner products are linear, \(X^\flat\) can be thought of as a covector. Thus, operator \(\flat\) defines a way of transforming vectors into covectors, and defines an isomorphism between \(T_pM\) and \(T^*_pM\):

\[
X = a_1 \frac{\partial}{\partial x_1} + \ldots + a_n \frac{\partial}{\partial x_n}
\]  
(2.9)

\[
X^\flat = a_1 dx^1 + \ldots + a_n dx^n
\]

In the same sense, for each covector \(\omega\), we can define an inner product over \(T^*_pM\) that we denote using operator \(\sharp\):

\[
\omega^\sharp(\alpha) = \omega \cdot \alpha
\]  
(2.10)

so that operator \(\sharp\) defines an isomorphism from \(T^*_pM\) into \(TM\) as

\[
\omega = b_1 dx^1 + \ldots + b_n dx^n
\]

\[
\omega^\sharp = b_1 \frac{\partial}{\partial x_1} + \ldots + b_n \frac{\partial}{\partial x_n}
\]  
(2.11)

While the inner product always yields scalar quantities (as the dot product does), the exterior product yields vectors and covectors generalizing the cross product in \(\mathbb{R}^n\) and is defined in terms of the tensor product as follows\[13 27 2\].
Definition 2.1.3 (Exterior Product over a Vector Space (Goldberg[13])). Let \( E \) be an \( n \)-dimensional vector space over \( \mathbb{R} \) with basis \( \beta = \{ e_1, e_2, \ldots, e_n \} \), define the exterior product among \( e_i \) and \( e_j \) as

\[
e_i \wedge e_j = e_i \otimes e_j - e_j \otimes e_i
\] (2.12)

the exterior product holds the following properties:

1. \( v \wedge v = 0 \) for all \( v \in E \)
2. \( v \wedge w = -(w \wedge v) \) for all \( v, w \in E \).
3. \( (a_1 v_1 + a_2 v_2) \wedge w = a_1 (v_1 \wedge w) + a_2 (v_2 \wedge w) \) for all \( a_1, a_2 \in \mathbb{R} \) and \( v_1, v_2, w \in E \)

In the same way that linear algebra uses the determinant and the cross product to calculate area and volume, the wedge product allows us to define area and volume in a general setting.

As we have mentioned before, the choice of using the symbols \( dx \) for the covectors is not gratuitous. While the idea of tangent vectors is to define operators like the gradient and the divergence, covectors comprise differential forms that will define things to integrate[21]. Formally speaking a differential form is an antisymmetric tensor constructed only using covectors and operator \( \wedge \) [13, 27, 2]

Definition 2.1.4 (\( k \)-dimensional basic differential forms (Montesdeoca [27])). Let \( M \) be an \( n \)-dimensional differentiable manifold and let \( \{ dx_i : i = 0, \ldots, n \} \) be the basis for the cotangent space at point \( p \in M \). Let \( 0 \leq k \leq n \) and choose \( i_1, i_2, \ldots, i_k \) such that

\[
0 \leq i_1 < i_2 < \ldots < i_k \leq n
\] (2.13)

the basic differential form associated to the multi-index \( I = (i_1, i_2, \ldots, i_k) \) is given by

\[
dx_{i_1} \wedge dx_{i_2} \wedge \ldots \wedge dx_{i_k}
\] (2.14)

We can construct the space of \( k \)-differential forms over a manifold \( M \) as the \( C^\infty(M) \)-free vector space induced by set of basis differential forms[13]:

\[
\Omega^k(M) = \left\{ \sum_{i_1 < i_2 < \ldots < i_k} \alpha_{i_1 i_2 \ldots i_k} dx_{i_1} \wedge dx_{i_2} \wedge \ldots \wedge dx_{i_k} : 0 \leq i_1 < i_2 < \ldots < i_k \leq n, \alpha_{i_1 i_2 \ldots i_k} \in C^\infty(M) \right\}
\] (2.15)

In a similar fashion, we can define the space of Vector Fields over a manifold \( M \) as the \( C^\infty(M) \)-free vector space

\[
\mathcal{X}(M) = \left\{ \sum_{i=1}^{n} \alpha_i \frac{\partial}{\partial x^i} : \alpha_i \in C^\infty(M) \right\}
\] (2.16)

Notice that if \( M \) is \( n \)-dimensional, there can only exist one \( n \)-dimensional differential form called the volume form:

\[
dV = dx_1 \wedge dx_2 \wedge dx_3 \wedge \ldots \wedge dx_n
\] (2.17)

this volume form allows us introduce yet another notion of duality:

Definition 2.1.5 (Hodge Dual (Montesdeoca [27])). given a basic differential form \( \omega = dx_{i_1} \wedge dx_{i_2} \wedge \ldots \wedge dx_{i_k} \in \Omega^k(M) \), we can define a form \( \ast \omega \in \Omega^{n-k}(M) \) such that

\[
\omega \wedge \ast \omega = dV
\] (2.18)
the last operator we will define here is the differential operator among differential forms, $d$ that produces a $k + 1$ form out of a $k$-form by differentiating the component functions and using the wedge product. Given a differential form

$$\omega = \sum_{i_1 < i_2 < \ldots < i_k} \alpha_{i_1i_2\ldots i_k}(x_1, x_2, \ldots, x_n)dx_{i_1} \wedge dx_{i_2} \wedge \ldots \wedge dx_{i_k}$$ (2.19)

we can define a $k + 1$ form $d\omega$ as

$$d\omega = \sum_{i_1 < i_2 < \ldots < i_k} \frac{\partial \alpha_{i_1i_2\ldots i_k}}{\partial x^i} dx_i \wedge dx_{i_1} \wedge dx_{i_2} \wedge \ldots \wedge dx_{i_k}$$ (2.20)

By means of operator $d$ and using Hodge Duality we can construct every differential operator of vector calculus: the gradient, the curl, the divergence and the laplacian$^2$. Indeed, by regarding scalar functions in $\mathbb{R}^3$ as real valued 0-forms, the gradient of a function $f$ can be written as a vector field defined by

$$\nabla f = (df)^\sharp$$ (2.21)

Similarly, given a vector field $F$ in $\mathbb{R}^3$, we can think of its curl as

$$\text{rot}(F) = \nabla \times F = (dF^\flat)^\sharp$$ (2.22)

while the divergence of a vector field can be written as

$$\text{div}(F) = *d *F^\flat$$ (2.23)

Operators $\sharp$ and $\flat$ allow us to change back and forth between differential forms and vector fields, so that this small set of operators allows us to write all known operators used in differential geometry and partial differential equations in the setting of manifolds$^{21, 25}$.

### 2.1.2 Integration of differential forms

Differential forms and vector fields over manifolds attempt to generalize the concepts of vector calculus to the realm of manifolds. In Exterior Calculus integration plays a fundamental role. Integration acts as a link between the geometric components of a manifold and the tensor based constructs we have defined so far and we can relate the differentials of forms with distinct sets of a manifold, realizing that the relation between integration and differentiation not as a mere cancellation but as a link between structuring elements of a manifold$^{13, 18, 11}$.

A $k$-differential form is something to be integrated over a $k$-dimensional subset $D$ of an $n$-dimensional manifold $M$ and we can speak of the integral of a form

$$\int_D \omega = \int_D \sum_{i_1 < i_2 < \ldots < i_k} \alpha_{i_1i_2\ldots i_k}(x_1, x_2, \ldots, x_n)dx_{i_1} \wedge dx_{i_2} \wedge \ldots \wedge dx_{i_k}$$ (2.24)

these integrals allow us to define length, area, volume and physical quantities like flux or vorticity for Manifolds. As we will show later on, following the same direction of Vector Calculus that takes advantage of various theorems (e.g. Green, the fundamental Theorem of Calculus, Gauss) to relate the value of an integral to the value that the integrated quantity (i.e. the differential form) takes over the boundary of its domain$^{13}$, we will show that operator $\partial$ and operator $d$ are fundamentally related via Stokes’ Generalized
For manifolds, the actual process of integration relies on \textit{Partitions of Unity} to account for the fact that the differential structure is only defined locally by the choice of an Atlas to weigh the values obtained from the integrals at a local scale. Partitions of Unity are common in many Computer Simulation methods as Finite Elements and Finite Volumes\cite{17,21}.

\textbf{Definition 2.1.6 (Partition of unity over a Manifold (Montesdeoca \cite{27}))}. Given an \textit{n}-dimensional manifold \( M \). A collection of functions \( \{ \phi_i \}_{i \in I} \) of functions \( \phi_i : M \to \mathbb{R} \) defines a partition of unity over \( M \) if the following conditions are satisfied

1. \( \phi_i(p) \geq 0 \) for all \( p \in M \) and \( i \in I \).
2. the support of the \( \phi_i \), \( \{ \text{supp}(\phi_i) \}_{i \in I} \) conforms a locally finite covering of \( M \)
3. For all \( p \in M \)
   \[
   \sum_{i \in I} \phi_i(p) = 1 \tag{2.25}
   \]

A locally finite covering of manifold \( M \) is defined as an open covering of \( M \) such that for every \( p \in M \) there exist a \textit{finite} number of indices \( i_1, i_2, \ldots, i_k \) such that \( p \in \text{supp}(\phi_{i_j}) \). In order for a partition of unity to work well for integration within a Manifold, there needs to be a relation between the charts of the atlas and the support of the functions in the partition\cite{27}. This behavior is defined in exterior Calculus as is known as \textit{subordination}.

So far we have defined three notions of duality: (1) the topological dual presented in section 1.2, (2) the duality between vectors and covectors given by equations 2.10, 2.8 and (3) Hodge duality given in definition 2.1.5. Now, we can introduce a new kind of duality that underlies a major result in exterior calculus known as \textit{Stokes’ Generalized Theorem}\cite{2,27,29}. Essentially, Stokes’ theorem states that the value that the integral of the differential of a form on a \( k + 1 \)-domain takes, is related to the value that a \( k \)-form takes over the \textit{boundary} of this domain.

In Vector Calculus we see a progression of this result in Green’s, Stokes’ and Gauss’ Theorem. The underlying duality relation states that we can basically relate the boundary operator defined in chapter 1 to the differential of a differential form.

\textbf{Theorem 2.1.7 (Stokes’ Generalized Theorem for Differential Forms (Goldberg \cite{13}))}. given a differential form \( \alpha \in \Omega^k(M) \) and its differential \( d\alpha \in \Omega^{k+1}(M) \),

\[
\int_M d\alpha = \int_{\partial M} \alpha \tag{2.26}
\]

integration is not just an operation that calculates an area or the work done by a force. As Stokes’ Generalized Theorem suggests, integration relates the values of differential forms among a manifold \( M \) with the values they take on its boundary. This fact states a position regarding the role of Topology as we have defined it, that we will exploit and is one of DEC’s key principles that will help us develop discrete counterparts to many concepts found in continuous exterior calculus.
2.2 Discrete Exterior Calculus for Triangulated Surfaces

Exterior calculus allows us to treat differentiable and integrable quantities under a single object (differential forms)\[17\], independent of a choice of coordinates and Stokes’ Generalized theorem replaces the problem of integration from mere antidifferentiation to allocating values within the manifold making use of various notions of duality\[17\] \[13\].

Discrete Exterior Calculus exploits the underlying topological considerations that Stokes’ Theorem suggests to restate many of the constructs of the continuous case as problems in terms of $p$-chains and define differential forms under one notion of duality, while taking advantage of operator $\star$ to redefine Hodge Duality over triangulated surfaces\[19\].

In a nutshell, DEC manages to relate the differential operator $d$ with the boundary operator $\partial$ and the dualizing star $\star$ with the Hodge dual $\ast$ and reduces problems stated in terms of differential forms to analyzing the effect making differential operators act as circulation rules for quantities among different elements of a simplicial complex\[18\] \[8\] \[25\].

Figure 2.1 represents the topological relations that the discrete versions of the differential operator $d$ and the Hodge dual $\ast$ pose in the context of DEC and pretty much summarizes the idea behind DEC: flux and circulation of quantities among triangulated surfaces. While operator $d$ measures the circulation of quantities by measuring the output of a quantity from its measure on the boundary of simplices, topological duality arises in the DEC sense as the support volume over which quantities flux through simplices\[18\].

As we will see in this chapter, both operator $d$ and $\ast$ are intrinsically related to operators $\partial$ and $\ast$, yielding yet another notion of duality: geometric vs. topological. Indeed, since we are dealing with quantities that flow and circulate through simplices in a complex, and are basically related to one another via Stokes’ theorem. The combination of both ideas and constructing operators in the DEC sense implies basically traversing the complex (both primal and dual), adding quantities that are stored in different parts of the mesh following the arrows drawn in Figure 2.1, weighing their values according to rules related to basic
geometric quantities: area of a face, length of an edge, etc.

### 2.2.1 Losing the Atlas: The de Rham Map

In classical manifold geometry a *triangulation* for a continuous *differentiable* manifold $M$ is defined as a function of a simplicial complex $\Sigma$ onto the manifold $M$:

$$ h : \Sigma \rightarrow M $$

(2.27)

so that for every vertex in $\Sigma$ its *open-neighborhoods* can be regarded as piecewise-linear submanifolds. In the sense of chapter 1 this matches the definitions of rings and neighborhoods as defined in 1.1.7 and 1.12. However, the idea of *triangulating* a manifold differs largely from *inducing* a manifold structure over a given triangulation. Not to say that we will ignore the concepts developed in the continuous case, but in light of one of the fundamental premises of DEC (no coordinates) the atlas can be replaced in the context of Concrete Simplicial Complexes by the idea of *Piecewise-Linear Manifolds*: Polytopes in $\mathbb{R}^n$.

From the idea of piece-wise linear manifolds we can now begin to delve into the geometric aspects complementing the concepts presented on the previous chapter using an approximation to Manifolds that is compliant with definition 1.1.1. While in the case of continuous manifolds many different Atlases can be defined for the same manifold, concrete simplicial Complexes as piecewise linear Manifolds possess already an obvious structure of coordinate systems over its simplices: their barycentric coordinates (recall definition 1.1.2). These barycentric coordinates are calculated as ratios of the *volume* induced by point $v$ to the total *volume* of the simplex:

$$ \alpha_i = \frac{Vol([v_0, \ldots, \hat{v}_i, \ldots, v_p] \cup \{v\})}{Vol([v_0, \ldots, v_i, \ldots, v_p])} $$

(2.28)

*Volume* here is understood in a wider sense: *length* in terms of an edge, *area* in terms of a face and actual volume for a tet.

**Definition 2.2.1 (Volume of a Simplex).** Given a simplicial complex $\Sigma$ in $\mathbb{R}^n$, we can define a function

$$ Vol : \Sigma \rightarrow \mathbb{R} $$

(2.29)

such that for a $p$-simplex $\sigma^p$ in the sense of definition 1.1.2 $Vol(\sigma^p)$ assigns $\sigma^p$ a real number. If $\Sigma$ is well-behaved $Vol(\cdot)$ satisfies the following properties:

1. $Vol(\sigma) \geq 0$
2. $Vol(\sigma) = 0 \iff \sigma = \emptyset$
3. $Vol(\sigma^p_1 \cup \sigma^p_2) = Vol(\sigma^p_1) + Vol(\sigma^p_2)$

Let $\sigma^p$ be a $p$-simplex belonging to a simplicial complex $\Sigma$. Given a $p$-form $\omega$ defined throughout $\Sigma$, we can define a *map* that takes $\sigma$ and $\omega$ and *blends* them into a scalar quantity by means of the integral

$$ \langle \sigma^p, \omega \rangle = \int_{\sigma^p} \omega $$

(2.30)
that can be calculated using the Change of Coordinates Theorem to barycentric coordinates in $\mathbb{R}^n$ (see [29] for a full explanation on this theorem). We can speak of the integral of a differential form throughout a concrete simplicial complex $\Sigma$ as

$$\langle \Sigma_p, \omega \rangle = \int_{\Sigma_p} \omega = \int_{\bigcup_{p\in\Sigma_p} \sigma_p} \omega = \sum_{p\in\Sigma_p} \int_{\sigma_p} \omega$$  \hspace{1cm} (2.31)$$

The value that the differential form $\omega$ takes over $\Sigma_p$ is merely a linear combination of the values it takes on each $p$-simplex: a $p$-chain. We can even think about the integral of a differential form over a $p$-chain $c = c_1\sigma^p_1 + \ldots + c_n\sigma^p_n$ (recall definition 1.3.1) as

$$\int_{c} \omega = \int_{c_1\sigma^p_1 + \ldots + c_n\sigma^p_n} \omega = c_1\int_{\sigma^p_1} \omega + \ldots + c_n\int_{\sigma^p_n} \omega$$  \hspace{1cm} (2.32)$$

where $c_i \in \mathbb{Z}$. Thus, we can define the de Rham Map of a simplicial complex as a function that maps continuous $p$-differential forms into discrete real valued $p$-chains.

**Definition 2.2.2 (The de Rham Map).** Let $\Sigma$ be an $n$-dimensional, simplicial complex and let $\omega$ be a $p$-differential form over $\Sigma$. Define the de Rham map

$$R : \Omega^p(\Sigma) \to C(\Sigma_p, \mathbb{R})$$  \hspace{1cm} (2.33)$$

by

$$R(\omega) = \sum_{p\in\Sigma_p} \langle \sigma^p, \omega \rangle \sigma^p$$  \hspace{1cm} (2.34)$$

this map transforms a $p$-differential form into a $p$-chain with real coefficients and its called the de Rham map over the simplicial complex $\Sigma$.

From a numerical analysis point of view, the literature suggests that these integrals can be calculated by *sampling* the differential form on different parts of its domain (see Crane, [21], Desbrun et al. [10] for example) which suggests that we use a *quadrature* approach to the calculation of this de Rham Map over a given simplicial complex. The suggestion of the literature also is to treat the $p$-chains as column vectors.

Under the DEC scope, we define discrete differential forms as *cochains* that select distinct components of the $p$-chains generated by the de Rham Map[8]. Indeed, since we are assigning essentially a scalar quantity to each $p$-simplex, it makes sense to define the differential forms as linear functionals acting on the different simplices of the compound respecting the notion of duality. In the DEC context, we can think of the result of the de Rham map as producing column vectors on the free vector space induced by the $p$-subcomplexes, while $p$-cochains (i.e. differential forms) can be written as row vectors on the same free vector spaces[21 11].

**2.2.2 Interpolation functions**

In the same sense that FEM and FVM methods attempt to construct piecewise continuous functions as solutions to differential equations starting from a triangulation of its domain, DEC theory conceives also a way to construct continuous functions and fields from discrete assignments over simplices of different dimension, putting Partitions of Unity to work[11 34]. While a discrete differential form is essentially in a
scalar assignment to simplices throughout a given complex $\Sigma$, a discrete vector field will be regarded as an assignment (possibly arbitrary) of vectors to distinct simplices [21], the role of these interpolation functions is (1) to allow us to reconstruct a continuous quantity from a discrete one, providing a sort of inverse for the de Rham Map, and (2) give us a discrete version of operators $\sharp$ and $\flat$ as we will later see.

While in FEM methods these Partitions of Unity are called support functions, DEC literature prefers to call them interpolation functions. These functions can be constructed with various degrees of continuity following the technique used in FEM methods [21, 10, 18]. In the DEC realm however, we must take into account both the primal and dual complex, that give rise to a variety of interpolation functions.

**Definition 2.2.3** (Primal-Primal Interpolation Functions (Hirani [18])). Let $\Sigma$ be a concrete simplicial complex in $\mathbb{R}^n$. Given $v_i \in \Sigma_0$ a primal vertex, define the primal-primal interpolation function about vertex $v_i$ as the piecewise linear function $\phi_i : \Sigma \rightarrow \mathbb{R}$ such that

$$\phi_i(v_j) = \begin{cases} 1, & i = j \\ 0, & \text{otherwise} \end{cases}$$

(2.35)

for all other primal vertices $v_j$. The support of this function is the convex hull of the neighborhood of vertex $v_i$:

$$\text{supp}(\phi_i) = \text{hull}\{v_j : v_j \in N(v_i)\}$$

(2.36)

Taking into account the paramount role of topological duality in DEC, we need then to define also interpolation functions for vertices that take into account the dual complex.

**Definition 2.2.4** (Primal-Dual Interpolating Functions (Hirani [18])). Let $\Sigma$ be a concrete simplicial complex embedded in $\mathbb{R}^n$. Given $v_i \in \Sigma_0$ a primal vertex, define the primal-dual interpolation function about vertex $v_i$ as the piecewise linear function $\psi_i : \Sigma \rightarrow \mathbb{R}$ such that

$$\psi_i(v_j) = \begin{cases} 1, & i = j \\ 0, & \text{otherwise} \end{cases}$$

(2.37)

where the support of this function is the convex hull the dual cell to vertex $v_i$:

$$\text{supp}(\psi_i) = \star v_i$$

(2.38)

We can define the dual interpolating functions that take support over different sets and will be vital to define discrete vector fields throughout a simplicial complex.

**Definition 2.2.5** (Dual-Dual Interpolating Functions (Hirani [18])). Let $\Sigma$ be a concrete simplicial complex embedded in $\mathbb{R}^n$. Given $w_j \in \Sigma_0^\ast$ a dual vertex, define the dual-dual interpolation function about vertex $w_j$ as the piecewise linear function $\phi_j^\ast : \Sigma \rightarrow \mathbb{R}$ such that

$$\phi_j^\ast(w_k) = \begin{cases} 1, & j = k \\ 0, & \text{otherwise} \end{cases}$$

(2.39)

for all other dual vertices. The support of this function is the face whose center makes up vertex $w_j$:

$$\text{supp}(\phi_j^\ast) = \star w_j$$

(2.40)
Finally, we can define the dual-dual interpolation functions for vertices that take both a dual and a primal vertex (related topologically) to create an interpolation function that depends on both:

**Definition 2.2.6 (Dual-Primal Interpolating Functions (Hirani,[18])).** Let $\Sigma$ be a concrete simplicial complex embedded in $\mathbb{R}^n$. Given $w_j \in \Sigma_0^*$ a dual vertex and $v_i$ a vertex such that

$$w_j \in *v_i$$

(2.41)

define the dual-primal interpolation function about the dual vertex $w_j$ with regards to the primal vertex $v_i$ as the piecewise linear function $\Phi^i_j : \Sigma \rightarrow \mathbb{R}$ such that

$$\Phi^i_j(w_k) = \begin{cases} 1, & j = k \\ 0, & \text{otherwise} \end{cases}$$

(2.42)

for all other dual vertices $w_k$. The support of this function is the dual cell to vertex $v_i$:

$$\text{supp}(\Phi^i_j) = *v_i$$

(2.43)

### 2.2.3 The discrete differential operator

Stokes’ Generalized theorem relates the value that the differential for a form takes over a domain with the value that the form takes on the boundary of that domain. For simplicial complexes we can write this as

$$d : \Omega^k(\Sigma) \rightarrow \Omega^{k+1}(\Sigma)$$

(2.44)

Stokes’ theorem can be linked to operator $\partial$ (recall equation 1.28) by means of the following consideration

$$\langle \sigma^{p+1}, d\omega \rangle = \int_{\sigma^{p+1}} d\omega = \int_{\partial\sigma^{p+1}} \omega = \langle \partial\sigma^{p+1}, \omega \rangle$$

(2.45)

in this sense we say that operator $\partial$ and operator $d$ are *adjoint* [21][18][10]. Thus, by using the De Rham Map, we can think of the discrete version of $d$ as the linear operator over $p$-chains:

$$d_p : C(\Sigma_p, \mathbb{R}) \rightarrow C(\Sigma_{p+1}, \mathbb{R})$$

(2.46)

Since $\partial^2 = 0$ it follows that $d^2 = 0$

### 2.2.4 Duality and Discrete Differential Forms

Given a $p$-differential form $\omega$, we know that its Hodge dual $*\omega$ is an $n - p$ differential form. Thus, if a differential form $\omega$ integrable over a $p$-simplex, its Hodge Dual $*\omega$ should be integrable over an $n - p$-dimensional domain. We can bring operator $*$ to do the work by defining operator $*$ in the context of DEC as:

$$\langle \omega, \sigma^p \rangle = \int_{\sigma^p} \omega$$

(2.47)

$$\langle *\omega, *\sigma^p \rangle = \int_{*\sigma^p} *\omega$$

From the physical point of view, both differential forms have different meanings: $\omega$ describes the circulation of quantity within $\sigma^p$ while $*\omega$ describes the flux passing through $*\sigma$ (see Figure 2.2). The DEC literature
suggests that the average of both these integrals should be equal\[^{18,10}\]. This allows us to define the discrete Hodge dual \(*\) from the following consideration:

\[
\frac{1}{Vol(\sigma^p)} \langle \sigma^p, \omega \rangle = \frac{1}{Vol(\star \sigma^p)} \langle \star \sigma^p, \star \omega \rangle
\]  

(2.48)

In the sense of a linear transformation between primal \(p\)-chains and dual \(n-p\)-chains:

\[
*_{p} : C(\Sigma_{p}, \mathbb{R}) \rightarrow C(\Sigma^{*}_{n-p}, \mathbb{R})
\]  

(2.49)

### 2.2.5 Maps between Discrete Vector Fields and Differential Forms

In terms of Manifolds, a Vector Field \(X\) is an assignment of a tangent vector for every point in the manifold\[^{27}\]. In the discrete context, Vector Fields are assignments of vectors to elements of the \(p\)-subcomplexes that in the context of DEC arise in two flavors: primal and dual. Indeed, just as differential forms can be defined on the primal and the dual complex, Vector Fields are defined either over primal and dual 0-simplices (vertices).

**Definition 2.2.7** (Primal Discrete Vector Field (Marsden et al. \[^{11}\])). Let \(\Sigma\) be an \(n\)-dimensional simplicial complex embedded in \(\mathbb{R}^n\). A primal discrete vector field \(X\) over \(\Sigma\) is a map

\[
X : \Sigma_0 \rightarrow \mathbb{R}^n
\]  

(2.50)

that assigns a vector to each primal vertex of \(\Sigma\). The value of \(X\) is piecewise constant on the dual \(n\)-cells of \(\Sigma\).

We denote the set of primal discrete vector fields in \(\Sigma\) as \(\mathcal{X}(\Sigma)\). In a similar fashion, a Dual Discrete Vector Field is defined as follows.

**Definition 2.2.8** (Dual Discrete Vector Field (Marsden et al. \[^{11}\])). Let \(\Sigma\) be a simplicial complex. A dual discrete vector field \(X\) over \(\Sigma\) is a map

\[
X : \Sigma^*_{0} \rightarrow \mathbb{R}^n
\]  

(2.51)

such that its value on each dual vertex is tangential to its corresponding primal simplex.
Discrete primal vector fields are only defined for flat complexes. The intrinsic definition of a Vector Field in Continuous Exterior Calculus requires that the Vector Field belongs to the Tangent Space. In Discrete Exterior Calculus, tangentiality to the Complex is also paramount to the definition of a Vector Field, and tangentiality is not uniquely defined at boundary vertices for example. However, dual vector fields can be made tangential for every type of complex via interpolation functions [18].

The multiplicity of options regarding interpolation functions calls for multiple definitions of operators \( \sharp \) and \( \flat \) (A.Hirani calls it a proliferation of operators). Indeed, the different sharp operators using the interpolation functions as weights from formulas of the form

\[
X^\sharp(v) = \sum_{v_i \in \Sigma_0} X(v_i)\phi_i(v) \tag{2.52}
\]

for primal vertices. In the case of dual vertices we also have

\[
X^\flat(w) = \sum_{w_j \in \Sigma_0^*} X(w_j)\phi_i^*(w) \tag{2.53}
\]

The reader can find a complete discussion on the proliferation of \( \sharp \) and \( \flat \) in A.Hirani’s Discrete Exterior Calculus [18]. Since our interest is to treat triangulated surfaces we will only present the following definition of the discrete flat that applies to non-flat simplicial complexes, using the notation and concepts presented on the previous chapter.

**Definition 2.2.9** (Discrete Flat Operator for Dual Vector Fields (Desbrun et al.[11])). Given a 2-Simplicial Complex \( \Sigma \) embedded in \( \mathbb{R}^3 \), the discrete flat operator on a dual vector field acting on a Dual Discrete Vector Field \( X \) is defined by its evaluation on a primal edge \( e = [v_0, v_1] \) as

\[
\langle X^\flat, e \rangle = \sum_{f \in R(e)} \frac{Vol(*e \cap f)}{Vol(*e)} X(*f) \cdot (v_1 - v_0) \tag{2.54}
\]

In the discrete sense, we have to turn a quantity that resides on an edge (a 1-simplex) into a vector attached to a vertex. To do this, we have to use a rotation element in the simplicial complex at the vertices yielding a dual vector field to preserve tangentiality.

**Definition 2.2.10** (Discrete Sharp operator for a primal 1-form (Desbrun et al.[11])). Let \( \Sigma \) be a 2-simplicial complex embedded in \( \mathbb{R}^n \). The discrete sharp operator on a primal 1-form is a function \( \sharp : \Omega^1(\Sigma) \to \mathfrak{X}(\Sigma^*) \) defined by its evaluation on a dual vertex \( w \in \Sigma_2^* \) as follows:

\[
\alpha^\sharp(w) = \sum_{v \in \mathcal{N}(w)} \langle \alpha, *[w, v] \rangle \left[ \sum_{f \in R([w, v])} \frac{Vol(*v \cap f)}{Vol(f)} (w - v)^\perp \right] \tag{2.55}
\]

where \((w - v)^\perp\) is the vector generated by rotating edge \([w, v]\) towards the center of cell \(f\).
Chapter 3

Using DEC

This chapter is dedicated to the computational aspect of the implementation of DEC constructs and details our experience using DEC as a framework. While previous chapters were dedicated to the theory of DEC, now present now a functioning computational framework that testifies to many of the advantages of DEC and showcases the power of thinking about topology and geometry as duals to one another in the DEC context.

In the first part of the chapter we present the structure and features of our implementation: the Lean-and-Mean DEC Processing library for creating simplicial complexes from triangulated surfaces encoded in .obj files. We have chosen to work directly with complex geometry meshes to showcase DEC in a realistic Computer Graphics work setting. Our models were created using MakeHuman\footnote{Freely available at \url{http://www.makehuman.org/}} Blender 2.74b\footnote{Freely available at \url{https://www.blender.org/}}.

The second part of this chapter is dedicated to present a collection DEC-based examples, presenting both the DEC solution as well as the lean-and-mean code used. The examples cover the following topics (1) Discrete Vector Calculus, (2) the solution to the Diffusion Equation from both the DEC and the FEM perspective, and (3) a DEC based approach to calculating Mean Curvature.

3.1 The Lean-and-Mean DEC Processing Library

In order to showcase DEC and to understand the practical applications of DEC in Computer Graphics we constructed a fully functioning Java library available for download in the project website. We wanted something that could be used easily and fast and that would be fully compliant with the .obj file format.

We chose to use the Processing IDE since it allows us to use many of the Swing and AWT functionalities for graphical development and it provides full OpenGL support in an easy and straight-forward fashion\footnote{https://processing.org/reference/environment/}. Additionally, the Processing environment is rich with libraries that range from Computer Vision to web-services that will allow the non-scienfic user to have the experience of DEC.

Lean-and-Mean is ready to be used from within the Processing environment by invoking the appropriate packages summarized in table \ref{table:packages}. In the Processing philosophy, all Processing assets are contained within one single PApplet that works as the main class of the application and is transparent to the user.
3.1.1 Features of Lean-and-Mean

The structure of Lean-and-Mean is presented in Figure 3.1, while Table 3.1 summarizes the components. When designing it, we had the following requirements in mind:

- Create Complexes from OBJ files automatically.
- Traverse complexes topologically and as p-chains.
- Create continuous Differential Forms and Vector Fields over the complexes.
- Discretize differential forms and vector fields as chains and cochains.
- Create and perform DEC based operations.
- Create and solve DEC based sparse linear systems for the solution to common PDE based problems.
- Visualize the complex and the result of DEC Based operations.

![General Package Diagram for Lean-and-Mean](https://example.com/lean-and-mean-diagram.png)

**Figure 3.1:** General Package Diagram for Lean-and-Mean (Created with Star UML).

3.1.2 Creating simplicial complexes

In order to create a simplicial complex using Lean-and-Mean, we must use in collaboration the packages `dec_complex`, `dec_readers`, and `dec_containers` that provide a simple interface for loading and manipulating simplicial complexes as Figure 3.2 shows. Following the example of ITK and VTK, the philosophy behind the lean-and-mean library is to use a pipe-line strategy to create the Simplicial Complex from the .obj file. To open and process the lines of the .obj file, we have used the OBJLoader library developed by SAITO and Matt Ditton[^4] that is one of the standard libraries provided with the Processing IDE. The following block shows how to load an .obj file using Lean-and-Mean:

[^4]: Available at [https://code.google.com/archive/p/saitoobjloader/](https://code.google.com/archive/p/saitoobjloader/)
<table>
<thead>
<tr>
<th>Package Name</th>
<th>Package Description</th>
<th>Package classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>dec_complex</td>
<td>creates and handles simplices and simplicial complex</td>
<td>DEC_Complex, DEO_Object, DEC_PrimalObject, DEC_DualObject, DEC_Iterator</td>
</tr>
<tr>
<td>dec_readers</td>
<td>Reads data files, processes information from the file</td>
<td>OBJMeshReader</td>
</tr>
<tr>
<td>dec_containers</td>
<td>Contains and handles geometric information about the complex</td>
<td>DEC_GeometricContainer</td>
</tr>
<tr>
<td>dec_renderers</td>
<td>Renders objects using the Processing Environment</td>
<td>SimplexViewer, ComplexViewer, ScalarAssigmentViewer, VectorAssignmentViewer</td>
</tr>
<tr>
<td>dec_operators</td>
<td>Creates DEC operators acting on forms fields.</td>
<td>DEC_MatrixOperator, BoundaryOperator, HodgeStar, DifferentialOperator, Laplacian, DeRhamOperator, SharpOperator, FlatOperator</td>
</tr>
<tr>
<td>dec_functions</td>
<td>Creates differential forms and vector fields both continuous and discrete</td>
<td>VectorAssignment, ScalarAssignment, ContinuousVectorField, DiscreteVectorField, ContinuousDifferentialForm, DiscreteDifferentialForm, ScalarFunction, ScalarFunctionMatrix</td>
</tr>
<tr>
<td>dec_exceptions</td>
<td>Handles errors</td>
<td>DEC_Exception</td>
</tr>
<tr>
<td>dec_utils</td>
<td>Useful algorithms and data structures for complex management, geometric quantities, numerical integration routines and linear system solutions</td>
<td>IndexSet, CalculationUtils, GeometricUtils, SparseVector, SparseMatrix</td>
</tr>
</tbody>
</table>

Table 3.1: DEC Lean-and-Mean Package Summary.

myReader = new OBJMeshReader(fileName,this);
myContainer = new DEC_GeometricContainer();
myComplex = new DEC_Complex();
myViewer = new ComplexViewer(this);
myReader.loadModel(centerType,false); //load model without texture
myViewer.setModelScale(50,myReader.getModelBoundingBox());
myContainer.setContent(myReader);
myContainer.printContainerInfo();
try{
    myComplex.setComplex(myContainer,myReader);
    myComplex.printComplexInformation();
}catch(DEC_Exception ex){
    println("something went wrong trying to create complex");
    ex.printStackTrace();
}

We leave the possibility to construct the dual complex using the barycenter, the incenter or the circumcenter by setting the value of variable centerType to one of the following values

BARYCENTRIC, CIRCUMCENTRIC, INCENTRIC

An example of the graphical difference between choosing a circumcentric and a barycentric subdivision scheme is found in Figure 1.3. In order to reduce the dependency on the DEC_GeometricContainer object and to simplify the process of rendering scalar and vector functions, we have included a feature inside the DEC_Object class, its vectorContent

protected HashMap<String, PVector> vectorContent;

to rapidly access commonly needed information from each simplex. The idea behind using a HashMap is to avoid the confusion of handling indices and allow the user to have instant access to various features of a simplex using their names:

1. "CENTER" provides access to the geometric center of a simplex (depending on the subdivision scheme defined for the complex)

2. "NORMAL" provides access to the normal vector defined at the simplex (only for vertices and faces).

3. "UV" provides access to the texture indices defined by this simplex (vertices and faces).

Since the size of OBJ objects is not standard, we take the liberty of normalizing the model so that the user can then fit it to the PApplet by using the setModelScale() method.

The .obj file format encodes not only the geometry of the faces, but also encodes the vertex normals and in some cases the texture coordinates. In the scope of the Lean-and-Mean library, the normals are required for the models to work since we depend on them to define the orientation of the simplices. Texture
coordinates are optional and its use can be introduced by modifying the value of variable `withTexture`. The `OBJLoader` library offers additional features such as organizing the face normals and face centers (barycentric) that are taken advantage of by our library. Internally, the `OBJMeshReader` object acts as a manager for the `OBJLoader` library that allowed us to create both the primal and the dual complex in a fully automatic fashion.

The `DEC_Complex` class is in charge of storing the adjacency information of the whole complex. This information is basically a group of lists of `DEC_Object` objects that comprise the various elements of the complex following the ideas of PyDEC and Kahler[4]. Though the idea of a primal and dual simplex in the context of DEC is conceptually different, in terms of computational representation and behavior a lot of their behavior remains the same. We used the `DEC_Object` class to encapsulate the common behavior of both primal and dual objects, reserving special features for classes `DEC_PrimalObject` and `DEC_DualObject`. Essentially a simplex is a container for indices over the vertices of the simplicial complex. To model this approach to simplices we introduced in the `DEC_Object` class the attribute

```java
protected IndexSet vertices;
```

the `IndexSet` object belongs to the `dec_utils` package simplifies the process of establishing relations among simplices as cobordancy, being contained in and being equal performed by matching indices within an `IndexSet`:

```java
public boolean contains(DEC_Object object) throws DEC_Exception;
public boolean isContainedIn(DEC_Object object) throws DEC_Exception;
public boolean isEqual(DEC_Object object) throws DEC_Exception;
```

It is important to notice that the sets of indices are separate for the primal and the dual complex, though interdependent. The indices that are stored in the `DEC_PrimalObject` are derived straight from the obj file while the indices in the dual complex are filled when creating the dual vertices and the dual complex. Within the `DEC_Complex` object, the process of creating the dual complex follows definition [1.2.1] starting from a primal complex completely built taking advantage of the definition of the ring of simplices (recall [1.1.7]). To create the primal complex, and reserving the same index for both the primal and the dual complex (its attribute `protected int index`) so that operator `⋆` can be implemented on only $O(1)$ operations. To create the primal complex we follow three steps:

```java
createPrimalVertices();
createPrimalFaces();
createPrimalEdges();
```
Figure 3.4: Various ways of rendering the same complex (detail) constructed using MakeHuman. (a) Vertices over the primal faces. (b) Edges over primal faces. (c) Primal faces (negatively oriented faces are translucent). (d) Primal Vertices and dual edges, (e) Primal and Dual Edges (using barycentric subdivision), and (f) Dual Vertices and primal edges.

While the primal vertices and the primal faces can be created using the information provided by the OBJMeshReader and the DEC_GeometricContainer, creating the primal edges makes use of the DEC_Iterator over the faces stored and the boundary operator producing a code of the following form

```java
DEC_Iterator faceIterator = createComplexIterator(2, 'p');
while(faceIterator.hasNext()){
    DEC_PrimalObject face = (DEC_PrimalObject) faceIterator.next();
    ArrayList<DEC_Object> faceBoundary = face.boundary();
    for(int i=0;i<faceBoundary.size();i++){
        int edgeIndex = objectIndexSearch(faceBoundary.get(i));
        if(edgeIndex == -1){
            addToComplex(new DEC_PrimalObject(faceBoundary.get(i));
        }
    }
}
```

in order to construct the dual complex, we take advantage of the ring of simplices translating into code definition [1.2.1]. For example, to create the dual edges we have the following block of code:

```java
DEC_Iterator edgeIterator = createComplexIterator(1, 'p');
while(edgeIterator.hasNext()){
    DEC_PrimalObject edge = (DEC_PrimalObject) edgeIterator.next();
    ArrayList<DEC_PrimalObject> containingFaces = facesContainingSimplex(edge);
```
if(containingFaces.size() == 2){
    int i0 = containingFaces.get(0).getIndex();
    int i1 = containingFaces.get(1).getIndex();
    DEC_DualObject dualEdge = new DEC_DualObject(new IndexSet(i0,i1));
    ...
} else{
    //edge is border in complex
    ...
}

Figure 3.5: Two examples of 2-manifolds with a boundary. Inner edges are shown in black, boundary edges are shown as thick red lines.

though most DEC based constructs are presented in the literature are done on closed manifolds (without boundary), we decided to leave the possibility to work on any type of model. With this in mind introduce the notion of a Boundary Edge to resolve issues in for example dual edge creation.

**Definition 3.1.1** (Boundary Edge in a 2-Complex). Given a 2-complex \( \Sigma \). We say that edge \( e \) is a boundary edge if and only if its ring is composed of only one face:

\[
|R(e)| = |\{f_e\}| = 1
\]  \hspace{1cm} (3.1)

in this case, we have

\[
\epsilon e = \epsilon_e [c(e), c(f_e)]
\]  \hspace{1cm} (3.2)

where \( \epsilon_e = \pm 1 \) and has the same sign as edge \( e \).

By extension, we have that boundary vertices belong to boundary edges and boundary faces have edges that are boundary. One of the tests we did on Lean-and-Mean to assert that we can work with open surfaces was to detect boundary edges and marking them as such. Figure 3.5 shows an example of a 2-complex with boundary. The following code shows the boundary edges as red thicker lines over a Complex.
DEC_Iterator edgeIterator = myComplex.createIterator(1,'p');
try{
    while(edgeIterator.hasNext()){
        DEC_PrimalObject edge = (DEC_PrimalObject) edgeIterator.next();
        if(edge.isBorder()){
            SimplexViewer edgeViewer = new SimplexViewer(this,edge);
            edgeViewer.getGeometry(myContainer);
            edgeViewer.setScale(myViewer.getModelWHD());
            strokeWeight(5);
            stroke(255,255,255);
            edgeViewer.plotPrimalEdge();
        }
    }
} catch(DEC_Exception ex){
    println("something went wrong trying to plot boundary edges");
    ex.printStackTrace();
}

3.1.3 Fields and Forms

Creating fields and forms in Lean-and-Mean requires the collaboration of multiple packages that interact with one another (see 3.1). Though creating scalar and vector assignments is a very straightforward task when working with polygon soups, we focused a lot on implementing the de Rham Operator, which required us to include features like numerical integration (essentially Gaussian Quadrature following indications in the literature) and allow now to work with both continuous differential forms and vector fields.

In DEC theory, we can think of both differential forms and vector fields as assignments of scalars and vectors to different elements of the complex. Classes ScalarAssignment and VectorAssignment are responsible for maintaining the assignment given a simplicial complex, a dimension and a type of assignment. See the constructors for both of these classes:

```java
public ScalarAssignment(int dimension,char type);
public VectorAssignment(int dimension, char type);
```

where dimension attains to the dimension of the objects that will contain the assignment (0 for vertices, 1 for edges, etc.), while the type variable will determine whether the assignment is 'p' for primal and 'd' for dual. Internally, both classes store HashMap structures that will encode the actual assignment:

```java
protected HashMap<DEC_Object,Double> values; //ScalarAssignment
protected HashMap<DEC_Object,PVector> directions; //VectorAssignment
protected HashMap<DEC_Object,Double> magnitudes; //VectorAssignment
```

that allow us to link the polygon soup structure of Lean-and-Mean with both rings and neighborhoods. In class VectorAssignment we have decided to split the assignment into direction and magnitude so that the process of rendering is simplified. Rendering both scalar and vector assignments requires creating look-up tables to translate the values contained in the assignment to colors. Additionally, we have left the possibility for the user to choose his or her own arrow length to better present a Vector Field on screen.

The assignment of scalars and vectors to the different elements of the complex is done using methods
public void assignScalar(DEC_Object object, double value);
public void assignVector(DEC_Object object, Double[] vector);

Though Lean-and-Mean has a strong dependency on class PVector of the Processing environment, we have chosen to leave Vector Fields as collections of double for better numerical precision on the calculation side of the library, while using the float-based PVector object for rendering. Also, since the lean-and-mean library works on normalized models, the domain onto which we are defining scalar and vector assignments (in the continuous case) is restricted to the $[-1, 1]$. The user can design his or her own vector and scalar forms adapting the domain onto which these take values by extending classes

ScalarFunction
ContinousVectorField

In the case of continuous 0,1 and 2-forms we followed the continuous definition of differential forms (see for example Munkres’ Analysis on Manifolds [29]), and made the consideration that differential forms are things that eat vector fields and spit scalars [21, 8, 9]. In this sense, we took into account the action of a 1-form $\omega$ of the form

$$\omega(x, y, z) = a_1(x, y, z)dx + a_2(x, y, z)dy + a_3(x, y, z)dz \quad (3.3)$$

a Vector Field $\mathbf{F}(x, y, z) = F_1(x, y, z)i + F_2(x, y, z)j + F_3(x, y, z)k$ as a dot product:

$$\omega(\mathbf{F})(x, y, z) = (a_1, a_2, a_3) \begin{pmatrix} F_1 \\ F_2 \\ F_3 \end{pmatrix} = \sum_{i=1}^{3} a_i(x, y, z)F_i(x, y, z) \quad (3.4)$$

while a 2-form $\alpha$ takes two vector fields $\mathbf{F}_1, \mathbf{F}_2$ and treats them as a bilinear form of the sort

$$\alpha(\mathbf{F}_1, \mathbf{F}_2) = (F_{11}, F_{12}, F_{13}) \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} F_{21} \\ F_{22} \\ F_{23} \end{pmatrix} = \sum_{i=1}^{3} \sum_{j=1}^{3} F_{1i}(x, y, z)a_{ij}(x, y, z)F_{2j}(x, y, z) \quad (3.5)$$

To describe a continuous 1-form we are in need of 3 scalar functions that make up the components $a_i$ while for a 2-form we requires a $3 \times 3$ scalar function matrix. To summarize both types of forms we introduced the ScalarFunctionMatrix that allows the user to define specific scalar functions for creating customized continuous Differential Forms by inheritance on class ScalarFunction. Class ScalarFieldMatrix serves a double purpose: on the one hand it allows the user to define the component functions of continuous differential forms; On the other, it allows the user also to create continous vector fields by merely changing the significance of the component functions, allowing the user to implement continuous versions of the sharp operator as defined in [2.11].

3.1.4 Operators

DEC theory conceives operators of two kinds: matrix-based and iterator-based. Matrix-based operators are basically $d$ and $\ast$ while iterator based operators are essentially $\sharp$ and $\flat$. While matrix-based operators arise from considering operators acting on $p$-chains (recall equations (1.37), (2.47)). Iterator-based operators depend on local quantities that can be found on the ring or the neighborhood of a simplex.

Though we can think of a matrix-based approximation to all operators, we devised the DEC_Iterator class that made the implementation of operators $\sharp$ and $\flat$ a simple translation of formulas into code. Figure
3.6 shows the operators conceived in the lean-and-mean library. Since DEC constructions lead to the creation of large sparse systems, the `DEC_MatrixOperator` contains a special type called `SparseMatrix` that can hold the boundary matrix, the differential matrix and the Hodge Star operator matrices minimizing the amount of space to represent large sparse systems following the example of PyDEC.

The construction of the matrix-based operator is condensed in the method

```java
public void calculateOperator(DEC_Complex complex, int dimension, char type);
```

that each operator must implement first before operating over a complex. To use a matrix-based operator over a complex we use the method

```java
public SparseVector apply(SparseVector vector);
```

over which a `SparseVector` must be introduced. In order for operators to act on chains, the `ScalarAssignment` class possesses a method

```java
public SparseVector toSparseVector();
```

so that to operate on a discrete differential form we need only one line of code:

```java
SparseVector result = operator.apply(scalarAssignment.toSparseVector());
```

reinterpreting the result as a discrete differential form (a $p$-chain) and attaching the result to simplices in the complex would be the responsibility of the user. For operators $d$ and $*$ we have included methods

```java
public DiscreteDifferentialForm apply(DEC_Complex complex, DiscreteDifferentialForm dF);
```

that handle the creation of the resulting differential forms and avoids having to translate them back and forth from `SparseVector`. Locally based operators are defined individually and the possibility for the user to define his or her own operators is open, without inheritance of any kind. The final operator included in `Lean-and-Mean` is the `de Rham Operator` that follows exactly definition 2.2.2 and makes use of the class `CalculationUtils` that contains static methods devised to perform numerical integration using Gauss-Legendre Quadrature of degrees 2, 3 and 4 (with 4 being the default value).

### 3.1.5 Rendering Features

One of the reasons for choosing the Processing IDE was the simplicity of Processing for producing 2- and 3-dimensional Graphics using JavaFX or OpenGL, alongside with installation and usage features that the Processing environment promotes. Indeed, Processing requires almost no complicated installation procedures and its minimal interface provides the programmer the bare minimum to start working with computer graphics. For all rendering features we decided to work with the `PVector` class since, the `OBJLoader` library used for opening and processing `.obj` files encodes mesh vertices as `PVector` objects. The rendering features provided in `Lean-and-Mean` cover the basic constructs DEC theory calls for: visualizing the complex, visualizing scalar forms, and visualizing Vector Fields. See Figure 3.1.2 for an example showcasing `Lean-and-Mean`'s complex display features. The model used for that Figure has with 4904 vertices and 9479 faces.

Rendering features for 0-forms work at two levels: at the vertices and as an interpolated surface and over the faces of the mesh creating a interpolated surface to show differential forms. See Figure 3.9 where the 0-form $f(x, y, z) = \sin(x) \sin(2y) \sin(3z)$ is rendered over a 1687 vertex and 3351 face model of a hand.
To render forms, *Lean-and-Mean* library makes use of the `ScalarAssignmentViewer` class that has various customizable features as vertex size and color scheme using a quite compact code:

```java
zeroFormCoefficients = new ScalarFieldMatrix(1,1);
zeroFormCoefficients.setComponents(new ScalarFunction[]{new sineScalarField()});
myZeroForm = new ContinuousDifferentialForm(zeroFormCoefficients,'p');
try{
    myZeroForm.calculateForm(null,myComplex);
    zeroFormViewer = new ScalarAssignmentViewer(this,myZeroForm,true);
```
Figure 3.8: Class Diagram for components involved during the rendering process

```java
zeroFormViewer.setScale(myViewer.getModelWHD());
zeroFormViewer.createLookUpTable(myComplex);
zeroFormViewer.setColors(color(100,255,255),color(200,255,255));
zeroFormViewer.setVertexSize(5);
}catch(DEC_Exception ex){
    println("something went wrong calculating 0-form");
    ex.printStackTrace();
}

Since we normalized our .obj models, accessing and coordinating the size of different objects (complex, scalar assignments and vector assignments) must be done individually by setting the scale property of all rendering-capable objects. To render scalar assignments we only have to make use of one instruction:

```java
zeroFormViewer.plotAssignment(myComplex,myContainer);
```

In a similar fashion, rendering Vector Field is done with the following block of code:

```java
vectorFieldViewer.plot(myComplex,myContainer);
```

We have left the possibility for the user to define his or her own assignments using the VectorAssignment object, that has to be associated with the complex elements (primal or dual). For example:

```java
solenoid = new SolenoidVectorField();
```
Figure 3.9: Rendering a 0-form (scalar field) \( f(x, y, z) = \sin(x) \sin(2y) \sin(3z) \). (a) Form values at the vertices. (b) Interpolated plot over faces.

```java
vectorAssignment = new VectorAssignment(0, 'd');
try{
    DEC_Iterator dualVertIterator = myComplex.createIterator(0,'d');
    while(dualVertIterator.hasNext()){
        DEC_DualObject dualVert = (DEC_DualObject) dualVertIterator.next();
PVector center = dualVert.getVectorContent("CENTER");
        Double[] fieldValue = solenoid.fieldFunction(center.x,center.y,center.z);
        vectorAssignment.assignVector(dualVert,fieldValue);
    }
    vectorFieldViewer = new VectorAssignmentViewer(this,vectorAssignment,true);
    vectorFieldViewer.setScale(myViewer.getModelWHD());
    vectorFieldViewer.setColors(color(150,255,255),color(220,255,255));
    vectorFieldViewer.setArrowLength(15);
    vectorFieldViewer.setStrokeWeight(1);
    vectorFieldViewer.createLookUpTable(myComplex);
}catch(DEC_Exception ex){
    println("something went wrong trying to create a vector field");
    ex.printStackTrace();
}
```

creates a solenoid vector field, \( \mathbf{F}(x, y, z) = y\mathbf{x} - x\mathbf{j} \) in \( \mathbb{R}^3 \). The assignment is done in this case to the dual verts of the complex. The `VectorAssignmentViewer` class is fully customizable in terms of the length of the resulting arrows, the color scheme related to the magnitude of the Vector Field and the `thickness` of the arrows to draw (see figure 3.10).
3.2 DEC Based Constructs

We will present three examples of DEC in the realm of Discrete Differential Geometry to showcase both thinking in DEC terms and how to program these ideas using Lean-and-Mean. The code for these examples is available with the library for download at the project page (see title page).

3.2.1 DEC Based Differential Operators

A differential operator under the DEC scope is a way to relate the value of a quantity over the topology of a simplicial complex[21, 15]. The rate at which this transference happens is -in the DEC sense- related to the geometry of the simplicial complex (edge length, face area, etc).

Consider a scalar function $u : \Sigma \to \mathbb{R}$. As a differential form we say that $u$ is a 0-form that is stored in the vertices of the complex. The de Rham Map of the complex transforms $u$ into a 0-chain as

$$u = R(u) = \int_{\Sigma_0} u = \sum_{v \in \Sigma_0} u(v)v$$  \hspace{1cm} (3.6)

We write $u$ as the column vector derived from the de Rham Map using [22,2]. By Stokes’ Theorem we know that for an edge $e_{ij} = [v_i, v_j]$

$$\int_{e_{ij}} \nabla u \cdot dr = \int_{\partial e_{ij}} u = u_j - u_i$$  \hspace{1cm} (3.7)

thus, we can construct the differential of 0-chain $u$ as the 1-chain given by

$$du = d_0u = \sum_{e \in \Sigma_1} \langle u, \partial e \rangle e$$  \hspace{1cm} (3.8)

In terms of the lean-and-mean library we can create this discrete 1-form (1-chain) with the following block of code
DEC_Iterator edgeIterator = myComplex.createIterator(1,'p');
HashMap<DEC_Object,Double> functionValues = discreteZeroForm.getValues();
gradientForm = new DiscreteDifferentialForm(1,'p');
while(edgeIterator.hasNext()){
    DEC_PrimalObject edge = (DEC_PrimalObject) edgeIterator.next();
    ArrayList<DEC_Object> edgeBounds = edge.boundary();
    int i0 = myComplex.objectIndexSearch(new DEC_PrimalObject(edgeBounds.get(0)));
    int i1 = myComplex.objectIndexSearch(new DEC_PrimalObject(edgeBounds.get(1)));
    double value = 0;
    if(i0 != -1 && i1 != -1){
        DEC_PrimalObject v0 = myComplex.getPrimalObject(0,i0);
        DEC_PrimalObject v1 = myComplex.getPrimalObject(0,i1);
        double f0 = functionValues.get(v0);
        double f1 = functionValues.get(v1);
        value = edge.getOrientation()*(f1-f0);
        println(value);
    }
    gradientForm.assignScalar(edge,value);
}

Now we can reconstruct $\nabla u$ as a vector field by using

$$\nabla u = (du)^\sharp$$

(3.9)

Now, if we wish to have $\nabla u$ as a vector field we can use the definition of the discrete sharp $\nabla u$ to construct a discrete dual vector field, that emulates the gradient of function $u$ at dual vertex $w$ following

$$\nabla u(w) = \sum_{v \in N(w)} \langle du, *[w,v] \rangle \left[ \sum_{f \in R([w,v])} \frac{Vol(*w \cap f)}{Vol(f)} (w-v)^\perp \right]$$

(3.10)

we can simplify this formula by noticing that we are asked to calculate form $du$ over the edges of the boundary of $*w$. A little bit of algebra and observation allows us to simplify this formula to obtain

$$\nabla u(w) = \sum_{e \in \partial *w} \langle du, e \rangle \left[ \sum_{f \in R(e)} \frac{Vol(*w \cap f)}{Vol(f)} e^\perp \right]$$

(3.11)

in a more general sense, we can regard calculating $\nabla u$ at dual vertex $w$ as a weighted sum of the values of $du$ on the boundary of $*w$:

$$\nabla u(w) = \sum_{e \in \partial *w} A_w(e) \langle du, e \rangle e^\perp$$

(3.12)

using the Quad-Edge notation[24] we can write the weights $A(e)$ as

$$A_w(e) = \frac{Vol(*w \cap *org(e))}{Vol(*org(e))} + \frac{Vol(*w \cap *dest(e))}{Vol(*dest(e))}$$

(3.13)

denotes the vector composed by edge $e$ rotated $\pi/2$ towards the center of the face. We can calculate the volumes in the inner sum by noticing that

$$Vol(*w \cap *org(e)) = Vol([w, c(e), org(e), c(e_o)])$$
$$Vol(*w \cap *dest(e)) = Vol([w, c(e), dest(e), c(e_f)])$$

(3.14)
using the center of edge $e \in \partial \star w$ and denoting by $e_0$ and $e_f$ the edges that conform face $\star w$ attached to $\text{org}(e)$ and to $\text{dest}(e)$. Thus, we can write the following example for $\text{Lean-and-Mean}$ to calculate the gradient of a function as

```java
void createGradientVectorField() throws DEC_Exception{
    DEC_Iterator dualVertIterator = myComplex.createIterator(0,'d');
    HashMap<DEC_Object,Double> functionValues = discreteZeroForm.getValues();
    discreteGradient = new VectorAssignment(0,'d');
    while(dualVertIterator.hasNext()){
        DEC_DualObject dualVert = (DEC_DualObject) dualVertIterator.next();
        DEC_PrimalObject primalFace = myComplex.dual(dualVert);
        PVector faceCenter = primalFace.getVectorContent("CENTER");
        PVector faceNormal = primalFace.getVectorContent("NORMAL_0");
        PVector resultingGradient = new PVector();
        //construct star(partial(dualVert))
        ArrayList<DEC_Object> primalFaceBounds = primalFace.boundary();
        ArrayList<DEC_PrimalObject> primalEdges = new ArrayList<DEC_PrimalObject>();
        ArrayList<PVector> edgeCenters = new ArrayList<PVector>();
        for(int i=0;i<primalFaceBounds.size();i++){
            DEC_PrimalObject edge = primalFaceBounds.get(i);
            int edgeIndex = myComplex.objectIndexSearch(new DEC_PrimalObject(edge));
            if(edgeIndex!=-1){
                DEC_PrimalObject primalEdge = myComplex.getPrimalObject(1,edgeIndex);
                primalEdges.add(primalEdge);
                edgeCenters.add(primalEdge.getVectorContent("CENTER"));
            }
        }
        //construct inner terms in sharp sum
        for(int i=0;i<primalEdges.size();i++){
            double gradientFormValue = gradientForm.getValues().get(primalEdges.get(i));
            int org = primalEdges.get(i).getVertices().getIndex(0);
            int dest = primalEdges.get(i).getVertices().getIndex(1);
            ArrayList<PVector> edgeVertices = primalEdges.get(i).getGeometry(myContainer);
            ArrayList<PVector> supportVolume0 = new ArrayList<PVector>();supportVolume1 = new ArrayList<PVector>();
            supportVolume0.add(faceCenter);supportVolume1.add(faceCenter);
            supportVolume0.add(edgeCenters.get(i));supportVolume1.add(edgeCenters.get(i));
            supportVolume0.add(edgeVertices.get(0));supportVolume1.add(edgeVertices.get(1));
            for(int j=0;j<primalEdges.size();i++){
                if(j!=i){
                    if(primalEdges.get(j).getVertices().containsIndex(org)){
                        supportVolume0.add(primalEdges.get(j).getVectorContent("CENTER"));
                    }
                    if(primalEdges.get(j).getVertices().containsIndex(dest)){
                        supportVolume1.add(primalEdges.get(j).getVectorContent("CENTER"));
                    }
                }
            }
        }
    }
}
```
DEC_DualObject dualFace0 = myComplex.getDualObject(2, primalEdges.get(i).getVertices().getIndex(0));
DEC_DualObject dualFace1 = myComplex.getDualObject(2, primalEdges.get(i).getVertices().getIndex(1));
float dualFaceVol_0 = dualFace0.volume(myContainer);
float dualFaceVol_1 = dualFace1.volume(myContainer);
float volume_0 = GeometricUtils.surfaceArea(supportVolume0);
float volume_1 = GeometricUtils.surfaceArea(supportVolume1);
PVector edgeAsVector = PVector.sub(edgeVertices.get(1), edgeVertices.get(0));
edgeAsVector.mult(primalEdges.get(i).getOrientation());
PVector rotatedEdge = rodriguesRotation(edgeAsVector, faceNormal, HALF_PI);
double edgeWeight = volume_0/dualFaceVol_0+volume_1/dualFaceVol_1;
rotatedEdge.mult((float) gradientFormValue*edgeWeight);
resultingGradient.add(rotatedEdge);
}
Double[] gradient = new Double[]{(double) resultingGradient.x, (double) resultingGradient.y, (double) resultingGradient.z};
discreteGradient.assignVector(dualVert, gradient);
}

Figure 3.11: Calculating the gradient of a function using the DEC Approach for 0-form (scalar function) \( f(x, y, z) = x^2 + z^2 - y \). (a) Original 0-form as interpolated surface, (b) Resulting gradient form, and (c) resulting gradient vector field.

Whose results can be seen in Figure 3.11. Following the example given by Crane in Chapter 8 of his *Discrete Differential Geometry: An Applied Introduction* [21], we can calculate the Laplacian of scalar function \( u \) as

\[
\nabla^2 u = \text{div} (\nabla u)
\]

(3.15)
the Laplacian of a function is also a scalar function which in the context of DEC translates to a 0-form. Equation 2.23 and the fact that $\#$ and $\flat$ are dual operators allows to write

$$\nabla^2 u = \text{div}(\nabla u)$$

$$= \star d \star (\nabla u)$$

$$= \star d \star (\text{du})$$

Thus, by equation 3.8 and the definition of the Hodge Star we must have

$$\star \text{du} = \sum_{e \in \Sigma_1} \frac{\text{Vol}(\star e)}{\text{Vol}(e)} \langle u, \partial e \rangle \star e$$

Now consider a dual cell $C$, by Stokes theorem we have that

$$\int_C d(\star \text{du}) = \int_{\partial C} \star (\text{du}) = \sum_{e \in \partial C} \frac{\text{Vol}(\star e)}{\text{Vol}(e)} \langle u, \partial e \rangle$$

Thus for vertex $v$ we can use its ring of edges to construct the Laplacian as

$$\nabla^2 u(v) = \frac{1}{\text{Vol}(\star v)} \sum_{e \in \mathcal{R}(v)} \frac{\text{Vol}(e)}{\text{Vol}(\star e)} \langle \text{du}, e \rangle$$

that we can translate directly into Lean-and-MeAN code as:

```java
void createLaplacianForm() throws DEC_Exception{
    DEC_Iterator vertexIterator = myComplex.createIterator(0,'p');
    discreteLaplacian = new DiscreteDifferentialForm(0,'p');
    HashMap<DEC_Object,Double> functionValues = discreteZeroForm.getValues();
    while(vertexIterator.hasNext()){*
        DEC_PrimalObject vert = (DEC_PrimalObject) vertexIterator.next();
        int vertIndex = vert.getVertices().getIndex(0);
        double functionValue = functionValues.get(vert);
        DEC_DualObject dualFace = myComplex.dual(vert);
        ArrayList<DEC_PrimalObject> contEdges = myComplex.primalEdgesContaining(vert);
        double discreteLaplacianValue = 0;
        for(int i=0;i<contEdges.size();i++){
            ArrayList<DEC_Object> edgeBoundary = contEdges.get(i).boundary();
            DEC_DualObject dualEdge = myComplex.dual(contEdges.get(i));
            float primalEdgeVol = contEdges.get(i).volume(myContainer);
            float dualEdgeVol = dualEdge.volume(myContainer);
            int otherVertIndex = -1;
            if(edgeBoundary.get(0).getVertices().getIndex(0) != vertIndex){
                otherVertIndex = edgeBoundary.get(0).getVertices().getIndex(0);
            }else{
                otherVertIndex = edgeBoundary.get(1).getVertices().getIndex(0);
            }
            DEC_PrimalObject otherVert = myComplex.getPrimalObject(0,otherVertIndex);
            discreteLaplacianValue += primalEdgeVol * dualEdgeVol *
            discreteLaplacianValue += functionValue *
        }
    }
    discreteLaplacian.setValue(0,discreteLaplacianValue);
}
```
Figure 3.12: Testing the Laplacian. (a) Original form $F(x, y, z) = x^2 + z^2 - y$. (b) Results calculating the Laplacian $\nabla^2 F(x, y, z) = 4$. (c) Results obtained using the Laplacian approximation in 3.19. (d) Cotan-laplace approximation formula 3.38.

```java
double otherVertFunctionValue = functionValues.get(otherVert);
discreteLaplacianValue += contEdges.get(i).getOrientation() 
  *(dualEdgeVol/primalEdgeVol) 
  *(otherVertFunctionValue-functionValue);
}
float dualFaceVolume = dualFace.volume(myContainer);
discreteLaplacianValue /= dualFaceVolume;
discreteLaplacian.assignScalar(vert,discreteLaplacianValue);
```

### 3.2.2 DEC Based Finite Elements

Finite Elements Methods assume that the solution function to a differential equation can be written as a linear combination of support functions $\phi_i$ defined over the vertices of a triangulated domain (the elements of the problem). The differential equation can be used to define a bilinear form (a sort of norm) from which we can construct a sparse linear system to calculate the coefficients of linear combination of support functions by minimizing the error with regards to the norm induced by the differential equation[^21][^3][^38].

This process depends on the choice of a subdivision of the domain, the type of hat functions and the nature of the differential equation under solution and assumes a functional analysis approach to the PDE solving. We can choose the support functions using the primal-primal interpolation functions defined in 2.2.3:

$$
\phi_i(v_j) = \begin{cases} 
1 & , i = j \\
0 & , \text{otherwise}
\end{cases}
$$

[^21]: Reference 21
[^3]: Reference 3
[^38]: Reference 38
for vertex $v_i$ and whose support is the set $\text{supp}(\phi_i) = S_i = \text{hull}(N(v_i))$. We will see the advantages of working in a setting devoid of coordinates while constructing the solution of Poisson’s Equation
\begin{equation}
\nabla^2 u = f
\tag{3.21}
\end{equation}
over a 2-complex $\Sigma$. The FEM technique begins by defining the residual to the equation as
\begin{equation}
R(u) = \nabla^2 u - f
\tag{3.22}
\end{equation}
In terms of DEC related elements, we wish to find a linear combination of the primal-primal interpolation functions (recall 2.2.3) of the form
\begin{equation}
\bar{u} = \sum_{i \geq 0} u_i \phi_i
\tag{3.23}
\end{equation}
such that the error determined by the residual function is minimized\[21]. We will define the error for the residual in terms of an inner product for functions $f, g : \mathbb{R}^3 \to \mathbb{R}$ over $\Sigma_2$ as
\begin{equation}
\langle f, g \rangle = \iint_{\Sigma_2} f(x)g(x) dS
\tag{3.24}
\end{equation}
classic FEM method asks us to find $u_i$ such that the error associated with approximation $\bar{u}$ with regards to every primal-primal interpolation function is zero:
\begin{equation}
\langle R(\bar{u}), \phi_i \rangle = \iint_{\Sigma_2} \bar{u}(x)\phi_i(x) dS = 0, i = 0, \ldots, n
\tag{3.25}
\end{equation}
by the definition of $\bar{u}$ and taking into account that the primal-primal interpolation functions have compact support we can reduce this equation to
\begin{equation}
\sum_{v_j \in N(v_i)} u_j \iint_{S_i \cap S_j} (\nabla^2 \phi_j(x) - f(x))\phi_i(x) dS = 0, i = 0, \ldots, n
\tag{3.26}
\end{equation}
that we can further reorganize to obtain the following equation system
\begin{equation}
\sum_{v_j \in N(v_i)} u_j \iint_{S_i \cap S_j} \nabla^2 \phi_j(x)\phi_i(x) dS = \iint_{S_i} \phi_i(x)f(x) dS, i = 0, \ldots, n
\tag{3.27}
\end{equation}
By Green’s first identity to level the degrees of the derivatives on the left-hand-side of \ref{eq:3.27}. Denote by $A_{ij} = S_i \cap S_j$, then
\begin{equation}
\iint_{A_{ij}} \nabla^2 \phi_j(x)\phi_i(x) dS = \oint_{\partial A_{ij}} \phi_i(x)\nabla \phi_j(x) \cdot dr + \iint_{A_{ij}} \nabla \phi_j(x) \cdot \nabla \phi_i(x) dS
\tag{3.28}
\end{equation}
Notice that in $\partial A_{ij}$ either $\phi_i$ vanishes or $\phi_j$ vanishes, so the line integral vanishes and we are left to calculate the surface integral
\begin{equation}
\iint_{A_{ij}} \nabla \phi_j(x) \cdot \nabla \phi_i(x) dS
\tag{3.29}
\end{equation}
Furthermore, $A_{i,j}$ can be thought of as just the union of two faces $F_{i,j}^1$ and $F_{i,j}^2$ that belong to the ring of $v_i$ and compose the ring of edge $e_{ij} = [v_i, v_j]$, thus, we can write

$$\int \int_{F_{i,j}^1} \nabla \phi_j(x) \cdot \nabla \phi_i(x) dS + \int \int_{F_{i,j}^2} \nabla \phi_j(x) \cdot \nabla \phi_i(x) dS$$ (3.30)

since the $F_{i,j}^k$ are cobordant (they share edge $e_{ij}$) we can write

$$F_{i,j}^1 = [v_i, v_j, v_{ij}]$$
$$F_{i,j}^2 = [v_i, v_j, v_{ij}]$$ (3.31)

and the gradient of $\phi_i$ and $\phi_j$ over face $F_{i,j}^k$ can be written as

$$\nabla \phi_i = \frac{1}{2Vol(F_{i,j}^k)}(v_{ij}^k - v_i)$$
$$\nabla \phi_j = \frac{1}{2Vol(F_{i,j}^k)}(v_{ij}^k - v_j)$$ (3.32)

thus for face $F_{i,j}^k$ we have

$$\nabla \phi_i \cdot \nabla \phi_j = \frac{1}{4Vol(F_{i,j}^k)^2}(v_{ij}^k - v_i) \cdot (v_{ij}^k - v_j)$$ (3.33)

if we denote by $\theta_{i,j}^k$ the angle opposite to edge $e_{ij}$ on face $F_{i,j}^k$, and applying some linear algebra to 3.2.2 we can reduce to

$$\nabla \phi_i(x) \cdot \nabla \phi_j(x) = \frac{1}{2Vol(F_{i,j}^k)} \cot(\theta_{i,j}^k)$$ (3.34)

and we can write 3.29 as

$$\int \int_{F_{i,j}^1 \cup F_{i,j}^2} \nabla \phi_j(x) \cdot \nabla \phi_i(x) dS = \frac{1}{2}(\cot(\theta_{i,j}^1) + \cot(\theta_{i,j}^2))$$ (3.35)

And the linear system representing Poisson’s Equation in DEC terms is given by

$$\sum_{v_j \in N(v_i)} \frac{1}{2}(\cot(\theta_{i,j}^1) + \cot(\theta_{i,j}^2)) (u_j - u_i) = F_i, i = 0, \ldots, n$$ (3.36)

where

$$F_i = \langle f, S_i \rangle = \int \int_{S_i} \phi_i(x)f(x) dS$$ (3.37)

this approximation to the Laplacian is called the cotan-Laplace formula [21, 31, 11, 25, 31]:

$$\nabla^2 u(v_i) = \sum_{v_j \in N(v_i)} L_{ij}(u_j - u_i)$$ (3.38)

where $u_i = u(v_i), u_j = u(v_j)$ and the weights of the Laplacian are given by

$$L_{ij} = \frac{1}{2}(\cot(\theta_{i,j}^1) + \cot(\theta_{i,j}^2))$$ (3.39)
Consider now the possibilities that the cotan-laplace approximation gives us in terms of solutions to differential equations. We will show the solution to the diffusion equation

$$\alpha \nabla^2 u = \frac{\partial u}{\partial t}$$  \hspace{1cm} (3.40)

where $\alpha$ is the diffusion coefficient and $u(x, y, z, t)$ describes a quantity (e.g. temperature, electrostatic potential, light).

We can discretize this quantity at vertex $v_i = (x_i, y_i, z_i)$ and at time instant $t_k = k\Delta t$ as

$$u^k_i = u(v_i, t_k)$$  \hspace{1cm} (3.41)

a regular Finite Differences scheme suggests that we can discretize the time derivative as

$$\frac{\partial u}{\partial t} \approx \frac{u^{k+1}_i - u^k_i}{\Delta t}$$  \hspace{1cm} (3.42)

by substituting $3.42$ and $3.38$ into $3.40$ and reorganizing we can construct approximation schemes for the diffusion equation. An explicit scheme yields a progressive algorithm to approximate quantity $u$ based on the values on the $k$-th iteration as

$$u^{k+1}_i = u^k_i + \alpha \Delta t \sum_{v_j \in N(v_i)} L_{ij}(u^k_j - u^k_i)$$  \hspace{1cm} (3.43)

which is only conditionally stable$^{[23]}$. In order to avoid numerical instabilities, we can develop an implicit scheme by exchanging the time index on the cotan-laplace approximation to get

$$u^k_i = \sum_{v_j \in N(v_i)} -\alpha \Delta t L_{ij}u^{k+1}_j + \left(1 + \alpha \Delta t \sum_{v_j \in N(v_i)} L_{ij}\right)u^{k+1}_i$$  \hspace{1cm} (3.44)

Given the fact that $L_{ij} \geq 0$, we notice that scheme $3.44$ leads to a diagonally dominant sparse linear system that allows for iterative methods like Gauss-Seidel or Jacobi iteration$^{[3]}$. We can implement the cotan-laplace weights by the following code in the lean-and-mean library:

```java
void createCotanLaplaceMatrix() throws DEC_Exception{
    int numVerts = myComplex.numPrimalVertices();
    cotanLaplace = new SparseMatrix(numVerts,numVerts);
    DEC_Iterator vertexIterator = myComplex.createIterator(0,'p');
    while(vertexIterator.hasNext()){  
        DEC_PrimalObject vert = (DEC_PrimalObject) vertexIterator.next();
        int vertIndex = vert.getIndex();
        PVector vertexNormal = vert.getVectorContent("NORMAL_0");
        PVector vertexCenter = myContainer.getGeometricContent(vert).get(0);
        DEC_Iterator vertNeighborhood = myComplex.objectNeighborhood(vert);
        while(vertNeighborhood.hasNext()){  
            DEC_PrimalObject neighbor = (DEC_PrimalObject) vertNeighborhood.next();
            int neighIndex = neighbor.getIndex();
            PVector neighCenter = neighbor.getVectorContent("CENTER");
```
DEC_PrimalObject tempEdge = new DEC_PrimalObject(new IndexSet(vertIndex,neighIndex));
int edgeIndex = myComplex.objectIndexSearch(tempEdge);
DEC_PrimalObject edge = myComplex.getPrimalObject(1,edgeIndex);
DEC_DualObject dualEdge = myComplex.dual(edge);
int f0Index = dualEdge.getVertices().getIndex(0);
int f1Index = dualEdge.getVertices().getIndex(1);
DEC_PrimalObject f0 = myComplex.getPrimalObject(2,f0Index);
DEC_PrimalObject f1 = myComplex.getPrimalObject(2,f1Index);
ArrayList<PVector> f0Verts = myContainer.getGeometricContent(f0);
ArrayList<PVector> f1Verts = myContainer.getGeometricContent(f1);
int index0 = indexOfOtherVector(vertexCenter,neighCenter,f0Verts);
int index1 = indexOfOtherVector(vertexCenter,neighCenter,f0Verts);
if(index0 != -1 && index1!=-1){
PVector pivot0 = f0Verts.get(index0);
PVector pivot1 = f1Verts.get(index1);
PVector c00 = PVector.sub(vertexCenter,pivot0);
PVector c01 = PVector.sub(neighCenter,pivot0);
PVector c10 = PVector.sub(vertexCenter,pivot1);
PVector c11 = PVector.sub(neighCenter,pivot1);
float angle0 = PVector.angleBetween(c00,c01);
float angle1 = PVector.angleBetween(c10,c11);
float tan0 = tan(angle0);
float tan1 = tan(angle1);
if(tan0!=0 & tan1!=0){
    cotanLaplace.set(1/tan(angle0)+1/tan(angle1),vertIndex,neighIndex);
}else if(tan0==0 & tan1!=0){
    println("angle0 failed");
    cotanLaplace.set(1/tan(angle1),vertIndex,neighIndex);
}else if(tan0!=0 & tan1==0){
    println("angle1 failed");
    cotanLaplace.set(1/tan(angle0),vertIndex,neighIndex);
}else{
    println("both angles failed");
}
}
}

introducing function indexOfOtherVector(v0,v1,f0Verts); to get the other vertex in face besides v0 and v1.

3.2.3 DEC Related facts on Curvature

In Vector Calculus, the curvature of a parametrized surface \( S : D \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3 \) can be defined in terms of its fundamental forms that relate the partial derivatives of component functions of \( S \) to describe both the inner notion of metric of the surface (a (0,2)-tensor) and its shape operator[29]. Over a continuous
parametrized surface \( S(u, v) \) in \( \mathbb{R}^3 \), the \textit{normal} vector on \( S \) defines the \textit{orientation} over the surface, by defining a direction vector \( N(u, v) \) that defines the direction of the \textit{outwards} of surface \( S \):

\[
N(u, v) = \frac{\partial S}{\partial u} \times \frac{\partial S}{\partial v} \quad (3.45)
\]

The derivatives of both surface \( S \) and the normal vector define the \textit{fundamental forms} on the surface from which the notion of curvature arises. Indeed, given point \( p = (u, v) \) over the domain \( D \) of surface \( S \), we can define its \textit{normal curvature} \( \kappa_n \) as

\[
\kappa_n(p) = \frac{dS(p) \cdot dN(p)}{||dS(p)||^2} \quad (3.46)
\]

This normal curvature can always be written in terms of the \textit{principal curvatures} \( \kappa_1, \kappa_2 \) of the surface as

\[
\kappa_n(S(p)) = \kappa_1 \cos^2(\theta) + \kappa_2 \sin^2(\theta) \quad (3.47)
\]

where \( \theta \) is the rotation angle for \( S(p) \) with regards to tangent vector \( \partial_u S \) at point \( p \)[21][29]. The principal curvatures describe the way in which the curvature of the surface varies with regards to the tangent vectors of the surface and are calculated as the \textit{eigenvalues} of its \textit{second fundamental form}[21] that depend on second-partial derivatives of component functions from \( S \). These principal curvatures define two kinds of curvature over surface \( S \), the \textit{Gaussian curvature} \( K \) and the \textit{mean curvature} \( H \) of a surface:

\[
H = \frac{\kappa_1 + \kappa_2}{2} \quad (3.48)
\]

\[
K = \kappa_1 \kappa_2
\]

The precise calculation of \( \kappa_1 \) and \( \kappa_2 \) requires us to go back to the concepts presented on section ?? and work our way through a complicated setting of tensors, vector fields and forms over a continuous atlas[13]. DEC theory proposes an alternate approach noticing some simple formulas that apply in continuous vector fields to obtain \textit{approximate} measures of curvature that can be defined solely in terms of composing elements of a mesh[21].

In order to do so, we start by noticing some facts about the \textit{toplogy} of polyhedra in \( \mathbb{R}^3 \) that relate facts about the \textit{Euler-Poincaré} formula for any polyhedron:

\[
\chi = V - E + F = 2 - 2g \quad (3.49)
\]

where \( V \) stands for the number of vertices, \( E \) the number of edges and \( F \) for the number of faces of a polyhedron and \( g \) is the \textit{genus} of the polyhedron (the number of handles in the polyhedron)[21]. Figure 3.13 relates the different values of \( \chi \) for models used in our tests with the \textit{lean-and-mean} library. The Euler characteristic of the torus is 0 since the torus can be regarded as a single \textit{handle}[21], the icosphere on the other hand has no handles at all, so that its genus \( g \) is zero. In the case of the hand model, the hand composes a manifold \textit{with} boundary and composes essentially a deformation of a \textit{polygonal disk} embedded in \( \mathbb{R}^3 \), to which the Euler-Poincaré characteristic is always 1[21].

The \textit{valence} of a polyhedron is the number of faces incident on every vertex. A vertex is said to be of regular valence when this number is even and of irregular valence when the number of faces is odd. The \textit{mean valence} of a surface should be 6 as the number of vertices tends to infinity[21]. We can easily calculate the valence at a vertex in the \textit{lean-and-mean} library by counting the number of faces incident at a given vertex and storing them into a \texttt{ScalarAssignment}:
void createValenceAssignment() throws DEC_Exception{
    DEC_Iterator vertexIterator = myComplex.createIterator(0,'p');
    while(vertexIterator.hasNext()){
        DEC_PrimalObject vert = (DEC_PrimalObject) vertexIterator.next();
        ArrayList<DEC_PrimalObject> contFaces = myComplex.primalFacesContaining(vert);
        int valenceNumber = contFaces.size();
        meanValence += (float) valenceNumber / (float) myComplex.numPrimalVertices();
        double valence = valenceNumber % 2 == 0 ? 0 : 1;
        valenceAssignment.assignScalar(vert, valence);
    }
}

See the results of some experiments done on various models using valence counting in Figure 3.14. We can calculate the mean curvature by calculating the angle defect at vertex \( v_i \), \( \angle(v_i) \) as the sum of the angles conformed by the edges incident at vertex \( v_i \).\[31,21\]. In DEC terms, this can be calculated in two ways: using the faces

\[
\angle(v) = \sum_{f \in R(v)} \theta(f, v) \tag{3.50}
\]

where \( \theta(f, v) \) is the angle of face \( f \) formed by the edges incident at vertex \( v \). Or, in terms solely of the vertices in the neighborhood of \( v \):

\[
\angle(v) = \sum_{w_1, w_2 \in N(v)} \theta(v, w_1, w_2) \tag{3.51}
\]

where \( \theta(v, w_1, w_2) \) identifies the angle formed by \( w_1 \) and \( w_2 \) using \( v \) as a pivot. The discrete mean curvature is defined as

\[
H(v) = 2\pi - \angle(v) \tag{3.52}
\]

Following the indications on Crane’s *Discrete Differential Geometry: An Applied Introduction*\[21\], we can translate this into the following *Processing* code. See some experimental results in Figure 3.15.

void createCurvatureAssignment() throws DEC_Exception{
    DEC_Iterator vertexIterator = myComplex.createIterator(0,'p');
    while(vertexIterator.hasNext()){
        DEC_PrimalObject vert = (DEC_PrimalObject) vertexIterator.next();
        PVector vertexNormal = vert.getVectorContent("NORMAL_0");
        PVector vertexCenter = myContainer.getGeometricContent(vert).get(0);
        double angleDefect = 0;
        DEC_Iterator vertNeighborhood = myComplex.objectNeighborhood(vert);
        ArrayList vertsAsList = vertNeighborhood.getList();
        int N = vertsAsList.size();
        for(int i=0;i<N;i++){
            DEC_PrimalObject v0 = (DEC_PrimalObject) vertsAsList.get(i);
            DEC_PrimalObject v1 = (DEC_PrimalObject) vertsAsList.get((i+1)\%N));
            PVector c0 = myContainer.getGeometricContent(v0).get(0);
            PVector c1 = myContainer.getGeometricContent(v1).get(0);
            double angle = PVector.angleBetween(PVector.sub(c1,vertexCenter),
            PVector.sub(c0,vertexCenter));
        }
    }
}
double meanCurvature = 2*Math.PI-angleDefect;
curvatureAssignment.assignScalar(vert, meanCurvature);
}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figures/figure3_13.png}
\caption{Calculated Euler Characteristic for different models used in our DEC tests.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figures/figure3_14.png}
\caption{Calculated mean valence for different models used in our DEC tests.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figures/figure3_15.png}
\caption{Calculated mean curvature using angle defect.}
\end{figure}
Chapter 4

Results, Conclusions and Future Work

The present chapter discusses the results presented throughout the document along with a small set of conclusions derived from this research on DEC theory as well as the future of DEC from our perspective.

4.1 Discussion of Results

Results will be discussed according to their order in the document.

4.1.1 On the topology of Triangulated Manifolds

Most DEC literature explains the approach to scalar functions as chains and differential forms as cochains and defines differential operators in terms of linear operators (sparse matrices), praising the fact that DEC theory separates topology from geometry which allows for a free approach of calculus that is coordinate-free, does not resort to limits or Riemann sums and is therefore completely discrete [10, 21]. The key concept here is topology over geometry: vector calculus over a triangulated surface is equivalent -via topological duality- to asking ourselves questions about the adjacency relations over a mesh.

With this consideration in mind, we decided to present the concepts presented on chapter 1. Our construction of the topology of triangulated surfaces allowed us to present DEC theory in chapter 4.1.1 in an easy and natural fashion, that made the construction of the lean-and-mean library a matter of translating formulas into code as we showed in chapter 3. We recognize that notation is key in the DEC context. Indeed, by defining the neighborhood and ring notation in our approach to DEC we were able to see the depth of the notion of duality in DEC theory and its true meaning regarding the relation between the geometry and the topology of triangulated surfaces.

Throughout the literature, a variety of notations is used that at times may do not necessarily suggest any computational implementation (see the contrast between the notation in A. Hirani’s Discrete Exterior Calculus [18] vs. Crane’s Discrete Differential Geometry: An applied introduction [21]). Notice that DEC relies heavily on summations over adjacent simplices and containing edges and faces. By first noticing this, we proposed rings and neighborhoods of simplices (definitions 1.1.7 and 1.1.10) that made this approach truly in-situ. The DECIterator class is in this sense fundamental throughout the lean-and-mean library and is where the connection between the Quad-Edge and Simplicial Complexes start.

Consequently, we had to make a decision concerning the lean-and-mean library: do we store the meshes using topologically based data-structures (i.e. the Quad-Edge) or do we maintain a polygon soup approach?
A Quad-Edge structure would enable us to attack DEC based local constructs in minimal performance time and would allow us to deal with the primal-dual relations with minimal effort, but would require additional work concerning the sums required in many parts of DEC operators and specially when constructing p-chains. Meanwhile, using a polygon soup approach makes the construction of p-chains, discrete differential forms and vector fields an almost trivial matter, due to their index-based nature, but could potentially hinder the time-performance of locally based constructs.

While deciding over the underlying data-structure in the DEC_Complex class, S.I. Goldberg's Curvature and Homology\[13\] gave us the final push towards using both approaches simultaneously. Both Goldberg and Guibas and Stoll\[24\] approach discrete surfaces as collections of objects (recall definition 1.1.1) that are linked together by relations that condition the properties of operators over them.

In this sense, we decided to concern us with defining operators ∂ and ∗ in the most efficient manner. By synchronizing the indices of in the primal and dual complexes we managed to make operator dual() to work in O(1), and the implementation of the boundary operator is done also in O(1) time, with subsequent O(log(n)) searches within the complex. Also the performance of the creation of neighborhoods and rings was also improved from O(n) to O(log(n)) by using binary search algorithms.

Further readings of Guibas and Stoll encouraged us to take an algebraic approach to the topology of simplicial complexes and motivated us to reduce all of the DEC constructs to the application of operators ∂ and ∗ over rings and neighborhoods, that allowed us to do a consistent presentation of all DEC-based constructs in chapters and \[3\] and greatly simplified the programming of lean-and-mean.

Moreover, the effect of our effort was two-fold: we have now an algebraic approach to triangulated surfaces that has not been seen in the DEC literature (in the scope of our research) and an alternate notation for DEC that encompasses both a local and a global view of a simplicial complex that reduces the effort when working with lean-and-mean for now we have a direct translation of DEC formulas into code (recall the examples presented in section 3.2).

Furthermore, the dec_complex package of the lean-and-mean library can be freely used as a new data structure for storing meshes that is freely available for all Processing and Java users that and constitutes a major contribution to the Processing Environment providing a Quad-Edge like data-structure for mesh traversal.

4.1.2 On DEC as a theoretical framework

Differential forms are a setting into which Partial Differential Equations arise in a natural fashion, physical laws and principles can be stated in terms of the constituting elements of a problem domain so that the local geometry of the problem is included in the statement of principles and physical restrictions are naturally embedded into the equations derived\[17\]. Certainly, thinking physics in terms of discrete differential forms implies dealing with quantities that are intrinsic to the objects\[1,21,10\] and physical laws governing phenomena are -in terms of Differential Forms- to be understood as principles that basically appeal to the circulation of quantities and their flux throughout a manifold. In the discrete case, Operators in the DEC sense work by transferring these intrinsic quantities along neighborhoods and rings of simplices by means of operator d and the duality provided by operator ∗ defines the flux throughout a complex.

This approach of calculus proposed by DEC manages to bridge at a theoretical level two apparently
irreconcilable PDE solving techniques: Finite Element Methods and Finite Difference Schemes. Indeed, FEM methods derive from the Lax-Milgram theorem and treat differential equations as function approximation problems, while Finite Differences schemes derive from Taylor Polynomials and attempt to treat differential operators as weighted sums of functions at precise node locations (leading to a stencil based approach on differential equations). The presentation made regarding the solution to Laplace and the diffusion equation presented in subsection 3.2.2 proves essentially, that both methods can be reconciled and used under the DEC scope. In this sense, we regard DEC as a theoretical framework that provides a holistic approach to calculus in which many PDE solution methods (e.g. Mimetic Finite Differences, Finite Volume Methods) can coexist and are essentially points of view about the behavior of physical phenomena.

The DEC literature—though extent—does not point out strong enough the connections between topology and geometry in the case of simplicial complexes like we have approached in this document. We have shown that the DEC promise as a totally in-situ, ex-geometrica proposal for calculus is fulfilled. Moreover, DEC as a framework manages to unify apparently dissimilar operators (the laplacian, the divergence, etc.) under just one definition: $d$ and manages to relate common differential operators that work in distinct ways in physical problems to just subsequent applications of four operators: $\sharp, \flat, d$ and $\ast$. We find such a result both elegant and highly practical when noticing that the discrete definition of both $d$ and $\ast$ translate to the simplest of matrix operations: the transposition.

We can only single out the elegance of DEC as a construct in Discrete Differential Geometry and join reknown authors like Hirani, Desbrun and Schröder in their endeavour with DEC as a genuine discrete approach to geometry, derived not from numerical analysis but from computer sciences, and provide researchers and enthusiasts in Computer Graphics with a tool to rapidly test DEC based constructs. However, the results obtained from our numerical experiments make us weary of promoting DEC as a silver bullet. Indeed, though most of the literature focuses on the advantages of DEC as a theoretical framework and show examples in many classical problems in Computer Graphics, none of the authors revised provide actual numerical stability or accuracy tests. As a testament of our discontent, see the results related to the laplacian presented in Figure 3.12. All DEC-based Laplacian approximation schemes reveal significant differences within one another and in comparison to the actual value.

We must accept that a combination of numerical instabilities provided by having to exchange double and float values, a difference in the values obtained from within Processing and from the NetBeans environment and the gimbal-lock problem in angle calculation are partially to blame for the numerical inaccuracies observed in many of our experiments. We warn the users of lean-and-mean that the code has not been proven to be correct in the Computer Sciences sense. However, we recognize that we have tested DEC constructs under very uncertain conditions: .obj models do not constitute perfect Delaunay Triangulations. They are designed to be graphically attractive but do not necessarily constitute well-behaved meshes for DEC that the literature certainly favors. As a result, we had to make several executive decisions when performing numerical experiments.

First, we had to favor barycentric duality over circumcentric, since we had no control over the existence of skinny triangles in the models (The dual complex creation routine ended up defining vertices outside of the surface and created zero-area faces in many occasions). Secondly, tests performed on simple Blender created models revealed hidden faces within the models and strange loop cuts in even the simplest of geometric shapes. However, the lean-and-mean library is able to detect these features via the edge border definition provided in 3.1.1. And third, the presence of straight angles among edges of the models tested forced us to redefine the weights in the laplace-cotan formula (3.38) as shown in subsection 3.2.2 As it
seems, both Blender and MakeHuman triangulate models by splitting quad faces (faces with four vertices) leaving a large amount of straight angles throughout the model.

Upon revision of the topological properties of the meshes studied in comparison with the numerical results obtained, we dare say that the valence at a vertex has much to do with the error associated to differential quantities. Notice the results shown in Figure 3.12 of the laplacians presented in equations (3.19) and (3.38). For function \( F(x, y, z) = x^2 + z^2 - y \) its laplacian is constant of value \( \nabla^2 F = 4 \) (recall part (b) of Figure 3.12). Though the calculated values of the laplacian are far distant from number 4, the laplacian varies significantly around vertices of irregular valence.

We conclude that there exists a profound relation between topological invariants of surfaces (its valence) and the behavior of its differential operators and we suggest that the theoretical connections between algebra, topology and geometry present in the theory of continuous differential forms also occurs in the case of discrete triangulated surfaces. The key to understanding this connection is the notion of duality in DEC. Certainly, it is not gratuitous that the vertex neighborhood chain in equation in subsection 1.3.3 and the DEC representation of the laplacian in equation 3.16 are connected via duality. This duality between topological and differential operators is a deep result that we have exploited throughout this document and which we passionately promote.

4.1.3 Implementing DEC: The lean-and-mean Processing library

As we have stated before, our DEC implementation follows the example of PyDEC and Kahler[4] and the indications given in Building your own DEC at home by S.Elcott and A.Hirani[1]. However, the treatment of DEC operations and constructions as filters inspired by the ITK-VTK framework allowed us to organize functionalities in a much more orderly fashion. For the Processing community, the lean-and-mean library will surely be perceived as strange since it does not attempt to automatize most DEC related processes and produce highly code-economic Processing sketches.

The lean-and-mean library is something a bit over the bare-bones of DEC theory: it provides functional capabilities for vector calculus. Certainly, the inclusion of continuous differential forms and of Gaussian-Quadrature related capabilities in the lean-and-mean library will entice mathematicians working in the realm of differential geometry to use our library in learning environments to show examples the effects of vector and scalar fields in examples different from the usual quadric surfaces. While working on the code of the lean-and-mean library we kept reminding ourselves that exterior geometry is just an extension of vector calculus in a more general setting. Thus, we have just provided the Processing community with a library that is capable of performing vector calculus on triangulated .obj files.

The design of lean-and-mean followed the commandment of separating topology from geometry almost completely. Though all DEC_Complex calculations are done only using the indices of the simplices in the complex, we have also enriched the simplices with geometric features that belong to the .obj objects (texture vertices, normal vectors, centers). We did this to reduce the cohesion between the DEC_GeometricContainer and the DEC_Complex class and to improve performance times in the creation and rendering -particulary- of the dual complex. We can assure the user that the underlying representation of the mesh is almost irrelevant for the construction of a simplicial complex, and that both the topological and geometric aspects of a mesh needed for exploring meshes under the DEC scope can be fulfilled by inheritance on classes DEC_GeometricContainer and by creating newer MeshReader (a fact inspired by the ITK-VTK framework).
We are fully confident that upcoming versions of *lean-and-mean* will improve its numerical accuracy by solving the gimbal-lock related problems and that a selection of better behaved models will provide us with more insight into the behavior of DEC constructs. Moreover, we feel that releasing *lean-and-mean* to the Processing community will surely provide us a fresh look upon our code from which we will be able to mend our errors, and we are confident that *lean-and-mean* will strengthen public interest in both DEC theory and Discrete Differential Geometry.

### 4.2 Conclusions

The initial purpose of this research project was to find a new computer simulation scheme that would behave as a sort of *middle ground* between FEM and Finite Differences, that would hopefully combine the advantages of both (the wide range of applications of FEM and the simplicity of Finite Differences) avoiding the pitfalls of both (the theoretical complexity of FEM schemes and the reduced capacity of Finite Differences). In our experience, DEC keeps this promise partially but offers much more in return.

Despite the fact that DEC yields a consistent approach to Differential Geometry that is both computationally friendly that can be implemented without resorting to specialized computer resources, the numerical accuracy of DEC constructs presented for *comercial .obj* meshes is hardly enticing from a practical point of view. However, DEC does stand as a middle ground between FEM and Finite Differences, whose treatment of geometry and topology by means of *duality* is a promising feature that will ensure its survival in the realm of computational modelling. Furthermore, DEC is a completely computer-sciences-oriented theory that has shown us ways of using Numerical Analysis from the point of view not of a mathematician, but of a computer scientist.

The results of our numerical tests and the topological calculations done over the meshes studied suggest a strong and deep connection between discrete differential geometry and topology that can be studied from within the DEC scope. We feel that our approach to topology and our subsequent approach to DEC constructs solely in terms of operators $\partial$ and $\star$ and their *duals* $d$ and $\ast$ comprise promising evidence that DEC approach can lead to a unifying theoretical framework for both discrete topology and discrete differential geometry that is both computationally friendly and theoretically consistent.

We have tested DEC constructs virtually in uncharted territory: non-flat, non-regular, non-symmetric, complexes with a boundary. Though we cannot say we are *victorious*, our endeavour with DEC, its rejection of coordinates, its minimality of operators and its almost *mystical* use of differential forms leaves us with an enhanced intuition regarding PDE based problems and discrete differential geometry that we can only label as *mind-opening*. Indeed, DEC forces us now to think of topology and geometry not as complementary disciplines but as a part of a whole and stimulates our creativity so that in the future we can be aware of *homologies*, *Betti numbers*, the *Poincaré Conjecture* etc. as tools to enrich the solution of problems arising in both Computer Sciences and Computer Graphics.

### 4.3 Future Work

The decision to work with triangulated surfaces was driven by a practical interest: introducing vector calculus to usable *assets* in video-games and computational art. However, many of the concepts in DEC require us to work with *flat* complexes. Most of the works presenting DEC related simulation experiments
involving the solution to Partial Differential Equations (see for example S.Elcott et al. *Discrete, Vorticity-Preserving, and Stable Simplicial Fluids* chapter 8 of [1]) require flatness. Further research is needed to develop algorithms and operator representation that are both stable and accurate for surfaces embedded in \( \mathbb{R}^3 \).

Though the *lean-and-mean* library is still at its early stages of development, the iterator approach on the topology of triangulated meshes worked in our favor greatly, giving us the possibility to traverse the mesh in a fashion similar to the *Quad-Edge* structure, yet retaining many of the features of classical polygon soups, providing the user of the *lean-and-mean* a hybrid approach to traversing a mesh. In this sense our DEC_Complex class may be used in future endeavors in triangulation algorithms -like the random insertion algorithm for Delaunay Triangulation- perhaps in a simpler fashion than with the *quad-edge*.

The inclusion of \( p \)-chains over simplicial complexes is a promising venture for research into the the realm of algebraic topology of triangulated surfaces. Indeed, Some examples in the literature venture to establish the grounds for a fully discrete algebraic topology approach to the study of surfaces with promising results in deep mathematical waters (see for example Desbrun et al. *Discrete Poincarré Lemma* [11]). Further study of the matrix representation of operators presented throughout this document may provide novel ways of studying the topology and the geometry of discrete surfaces perhaps in the field of invariant calculation, offering a discrete version of for example the *index theorem* or Gauss-Bonnet [13].

As shown, DEC-based relation between differential forms and vector fields are not as present as in the the continuous case. DEC theory centers itself around clever usages of Stokes’ theorem over already constituted differential forms that may or may not relate to vector fields. Further study is required in DEC theory to exploit the duality relation between differential forms and vector fields. We believe that a DEC based approach to differential forms that deal with vector fields can produce simpler approaches to illumination models and can be used for easier schemes of texture application (see for example chapter 6 of [1]).

As a theoretical framework, DEC manages to include the flexibility of differential forms following the same principles of the continuous theory avoiding abstractly defined tensors and replaces the tangent-cotangent space construction with the chain-cochain approach by a very intelligent application of Stokes’ Theorem (recall 2.1.7). In this sense, we can only admire the elegance in which this whole theoretical exercise has been constructed and celebrate the work done by A.Hirani, M.Desbrun, P.Schröder and many others that have ventured into the DEC dimension of computer modelling with significant results. We can only hope to keep contributing to the development of DEC as a theory to establish DEC as an alternative method in computational modelling with the same relevance, stability and maturity as for example the Finite Element Method.
Bibliography


ANEXO 2

CARTA DE AUTORIZACIÓN DE LOS AUTORES
(Licencia de uso)

Bogotá, D.C., 16 de Junio de 2016

Señores
Biblioteca Alfonso Borroto Cabral S.J.
Pontificia Universidad Javeriana
Cuidad

Los suscritos:

Carlos Camilo Rey Torres, con C.C. No 80793710

En mi (nuestra) calidad de autor (es) exclusivo (s) de la obra titulada:

Discrete Exterior Calculus (DEC): A Theoretical Framework for Computer Simulation and Geometric Analysis based on Advanced Mathematical Concepts

(por favor señale con una “x” las opciones que apliquen)

<table>
<thead>
<tr>
<th>Tesis doctoral</th>
<th>Trabajo de grado</th>
<th>X</th>
<th>Premio o distinción: Si</th>
<th>No</th>
</tr>
</thead>
</table>

cual:

presentado y aprobado en el año 2016, por medio del presente escrito autorizo (autorizamos) a la Pontificia Universidad Javeriana para que, en desarrollo de la presente licencia de uso parcial, pueda ejercer sobre mi (nuestra) obra las atribuciones que se indican a continuación, teniendo en cuenta que en cualquier caso, la finalidad perseguida será facilitar, difundir y promover el aprendizaje, la enseñanza y la investigación.

En consecuencia, las atribuciones de usos temporales y parciales que por virtud de la presente licencia se autorizan a la Pontificia Universidad Javeriana, a los usuarios de la Biblioteca Alfonso Borroto Cabral S.J., así como a los usuarios de las redes, bases de datos y demás sitios web con los que la Universidad tenga perfeccionado un convenio, son:

<table>
<thead>
<tr>
<th>AUTORIZO (AUTORIZAMOS)</th>
<th>SI</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. La conservación de los ejemplares necesarios en la sala de tesis y trabajos de grado de la Biblioteca.</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2. La consulta física (sólo en las instalaciones de la Biblioteca)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3. La consulta electrónica - on line (a través del catálogo Biblos y el Repositorio Institucional)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>4. La reproducción por cualquier formato conocido o por conocer</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>5. La comunicación pública por cualquier procedimiento o medio físico o electrónico, así como su puesta a disposición en Internet</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>6. La inclusión en bases de datos y en sitios web sean éstos onerosos o gratuitos, existiendo con ellos previo convenio perfeccionado con la</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
De acuerdo con la naturaleza del uso concedido, la presente licencia parcial se otorga a título gratuito por el máximo tiempo legal colombiano, con el propósito de que en dicho lapso mi (nuestra) obra sea explotada en las condiciones aquí estipuladas y para los fines indicados, respetando siempre a titularidad de los derechos patrimoniales y morales correspondientes, de acuerdo con los usos honrados, de manera proporcional y justificada a la finalidad perseguida, sin ánimo de lucro ni de comercialización.

De manera complementaria, garantizo (garantizamos) en mi (nuestra) calidad de estudiante (s) y por ende autor (es) exclusivo (s), que la Tesis o Trabajo de Grado en cuestión, es producto de mi (nuestra) plena autoría, de mi (nuestro) esfuerzo personal intelectual, como consecuencia de mi (nuestra) creación original particular y, por tanto, soy (somos) el (los) único (s) titular (es) de la misma. Además, aseguro (aseguramos) que no contiene citas, ni transcripciones de otras obras protegidas, por fuera de los límites autorizados por la ley, según los usos honrados, y en proporción a los fines previstos; ni tampoco contempla declaraciones difamatorias contra terceros; respetando el derecho a la imagen, intimidad, buen nombre y demás derechos constitucionales. Adicionalmente, manifiesto (manifestamos) que no se incluyen expresiones contrarias al orden público ni a las buenas costumbres. En consecuencia, la responsabilidad directa en la elaboración, presentación, investigación y, en general, contenidos de la Tesis o Trabajo de Grado es de mi (nuestro) competencia exclusiva, eximiendo de toda responsabilidad a la Pontificia Universidad Javeriana por tales aspectos.

Sin perjuicio de los usos y atribuciones otorgadas en virtud de este documento, continuaré (continuaremos) conservando los correspondientes derechos patrimoniales sin modificación o restricción alguna, puesto que de acuerdo con la legislación colombiana aplicable, el presente es un acuerdo jurídico que en ningún caso conlleva la enajenación de los derechos patrimoniales derivados del régimen del Derecho de Autor.

De conformidad con lo establecido en el artículo 30 de la Ley 23 de 1982 y el artículo 11 de la Decisión Andina 351 de 1993, “Los derechos morales sobre el trabajo son propiedad de los autores”, los cuales son irrenunciables, imprescriptibles, inembargables e inalienables. En consecuencia, la Pontificia Universidad Javeriana está en la obligación de RESPETARLOS Y HACERLOS RESPETAR, para lo cual tomará las medidas correspondientes para garantizar su observancia.

NOTA: Información Confidencial:
Esta Tesis o Trabajo de Grado contiene información privilegiada, estratégica, secreta, confidencial y demás similar, o hace parte de una investigación que se adelanta y cuyos resultados finales no se han publicado.  

Si ☐  No ☒

En caso afirmativo expresamente indicaré (indicaremos), en carta adjunta, tal situación con el fin de que se mantenga la restricción de acceso.

<table>
<thead>
<tr>
<th>NOMBRE COMPLETO</th>
<th>No. del documento de identidad</th>
<th>FIRMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carlos Camilo Rey Torres</td>
<td>80793710</td>
<td></td>
</tr>
</tbody>
</table>
FACULTAD: Ingeniería
PROGRAMA ACADÉMICO: Maestría en Ingeniería de Sistemas y Computación
ANEXO 3
BIBLIOTECA ALFONSO BORRERO CABAL, S.J.
DESCRIPCIÓN DE LA TESIS O DEL TRABAJO DE GRADO
FORMULARIO

<table>
<thead>
<tr>
<th>TITULC COMPLETO DE LA TESIS DOCTORAL O TRABAJO DE GRADO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discrete Exterior Calculus (DEC): A Theoretical Framework for Computer Simulation and Geometric Analysis based on Advanced Mathematical Concepts</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SUBTÍTULO, SI LO TIENE</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>AUTOR O AUTORES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apellidos Completos</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td>Rey Torres</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DIRECTOR (ES) TESIS O DEL TRABAJO DE GRADO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apellidos Completos</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td>Florez Valencia</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FACULTAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ingeniería</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROGRAMA ACADÉMICO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tipo de programa (selecione con “x”)</td>
</tr>
<tr>
<td>-------------------------------------</td>
</tr>
<tr>
<td>Pregrado</td>
</tr>
<tr>
<td>Especialización</td>
</tr>
<tr>
<td>Maestría</td>
</tr>
<tr>
<td>Doctorado</td>
</tr>
<tr>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nombre del programa académico</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ingeniería de Sistemas y Computación</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nombres y apellidos del director del programa académico</th>
</tr>
</thead>
<tbody>
<tr>
<td>Angela Carrillo</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TRABAJO PARA OPTAR AL TITULO DE:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Magister en Ingeniería de Sistemas y Computación</td>
</tr>
</tbody>
</table>

PREMIO O DISTINCIÓN (En caso de ser LAUREADAS o tener una mención especial):

<table>
<thead>
<tr>
<th>CIUDAD</th>
<th>ANO DE PRESENTACIÓN DE LA TESIS O DEL TRABAJO DE GRADO</th>
<th>NUMERO DE PÁGINAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bogotá</td>
<td>2016</td>
<td>71</td>
</tr>
</tbody>
</table>

TIPO DE ILUSTRACIONES (seleccione con “x”)

<table>
<thead>
<tr>
<th>Dibujos</th>
<th>Pinturas</th>
<th>Tablas, gráficos y diagramas</th>
<th>Planos</th>
<th>Mapas</th>
<th>Fotografías</th>
<th>Partituras</th>
</tr>
</thead>
</table>

SOFTWARE REQUERIDO O ESPECIALIZADO PARA LA LECTURA DEL DOCUMENTO
Nota: En caso de que el software (programa especializado requerido) no se encuentre licenciado por la Universidad a través de la Biblioteca (previa consulta al estudiante), el texto de la Tesis o Trabajo de Grado quedará solamente en formato PDF.
<table>
<thead>
<tr>
<th>TIPO</th>
<th>DURACIÓN (minutos)</th>
<th>CANTIDAD</th>
<th>FORMATO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Video</td>
<td></td>
<td></td>
<td>CD</td>
</tr>
<tr>
<td>Audio</td>
<td></td>
<td></td>
<td>DVD</td>
</tr>
<tr>
<td>Multimedia</td>
<td></td>
<td></td>
<td>Otro ¿Cuál?</td>
</tr>
<tr>
<td>Producción eléctrica</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Otro Cuál?</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DESCRIPTORES O PALABRAS CLAVE EN ESPAÑOL E INGLÉS**
Son los términos que definen los temas que identifican el contenido. (En caso de duda para designar estos descriptor, se recomienda consultar con la Sección de Desarrollo de Colecciones de la Biblioteca Alfonso Borreto Cabal S.J en el correo biblioteca@javeriana.edu.co, donde se les orientará).

<table>
<thead>
<tr>
<th>ESPAÑOL</th>
<th>INGLÉS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cálculo Exterior Discreto</td>
<td>Discrete Exterior Calculus</td>
</tr>
<tr>
<td>Formas Diferenciales discretas</td>
<td>Discrete Differential Forms</td>
</tr>
<tr>
<td>Mallas Trianguladas</td>
<td>Triangulated Meshes</td>
</tr>
<tr>
<td>Simulación Computacional</td>
<td>Computer Simulation</td>
</tr>
<tr>
<td>Geometría Differencial Discreta</td>
<td>Discrete Differential Geometry</td>
</tr>
</tbody>
</table>

**RESUMEN DEL CONTENIDO EN ESPAÑOL E INGLÉS**
(Máximo 250 palabras - 1530 caracteres)
El presente documento resume los resultados de un proyecto de investigación sobre la construcción e implementación del cálculo exterior discreto (DEC) aplicado al contexto de superficies trianguladas (mallas .obj) utilizando el IDE Processing, que presenta una construcción completa de la teoría DEC a partir de una aproximación simple a la teoría de las mallas trianguladas, que permite una construcción natural de cálculo completamente discreto sobre superficies trianguladas repitiendo muchos de los resultados de la teoría de la geometría de formas diferenciales sobre variedades diferenciables a partir de elementos simples, computacionalmente amables que aprovecha las relaciones topológicas de las superficies trianguladas para definir operadores diferenciales aprovechando la noción de dualidad a varios niveles. El documento muestra tanto el diseño de algoritmos basados en DEC como código de Processing para la solución de ecuaciones diferenciales parciales, análisis de mallas y cálculo de curvatura media, sobre modelos creados con MakeHuman y Blender 2.74b.

The following document summarizes the results of a research project regarding the construction and implementation of Discrete Exterior Calculus (DEC) applied to the context of commercially made triangulated surfaces (.obj meshes) using the Processing IDE. The document presents a full construction of the foundations of DEC theory starting from a simple approach to the topology of triangulated surfaces which allows us to build a theory of calculus that borrow from the theory of exterior calculus on differentiable manifolds to build a fully discrete theory of calculus that manages to relate topology to geometry introducing the notion of

PUJ- 6G. Normas para la entrega de Tesis y Trabajos de grado a la Biblioteca General - Junio de 2013
duality at several levels. We show both algorithm designs and Processing code for Partial Differential Equation solution, mesh analysis and mean discrete curvature calculation as examples of applications of this theory and provide actual tests performed on meshes created with MakeHuman and Blender 2.74b.