Research Article

Márcio S. V. de Souza* and Ruy M. O. Pauletti

An overview of the natural force density method and its implementation on an efficient parametric computational framework**

https://doi.org/10.1515/cls-2021-0005
Received Sep 15, 2020; accepted Jan 03, 2021

Abstract: The new paradigms of parametric modelling have been proving promising on the advance of systems for analysis and design of taut (or tensile) structures. With this premise, the presented work consist on the development with a form-finding tool for Computer Aided Design(CAE) and Computer Aided Engineering (CAE) integration using VPL (Visual Programming Language), in the context of parametric modelling. The methods used in the implementation are the Force Density Method (FDM) and the Natural Force Density Method (NFDM), taking advantage of the linear solution approach provided, suitable for fast form-finding computational procedures.
The program is implemented as a Grasshopper plug-in and it is named BATS (Basic Analysis of Taut Structures), which enables parametric definition of boundary conditions for the form-finding. The program structure and benchmarks with other available Grasshopper plug-ins for taut structures form-finding are presented, showing considerably superior performance using BATS.

Keywords: form-finding, natural force density, force density, grasshopper, parametric modelling, taut structures, tensile structures, minimal surfaces

***Corresponding Author: Márcio S. V. de Souza:** Structural and Geotechnical Department, Polytechnic School, University of São Paulo, Av. Prof. Almeida Prado, 83 – Butantã, São Paulo, 05508-070, Brazil; Email: marcio.sartorelli.souza@usp.br
Ruy M. O. Pauletti: Structural and Geotechnical Department, Polytechnic School, University of São Paulo, Av. Prof. Almeida Prado, 83 – Butantã, São Paulo, 05508-070, Brazil, E-mail: pauletti@usp.br
** Paper included in the Special Issue entitled: Shell and Spatial Structures: Between New Developments and Historical Aspects

1 Introduction

Taut structures are defined by their characteristic state, in which the structural members work only in tension. They are usually flexible systems and are subjected to large displacements upon changes of the intensity and distribution of the loads.

Since Frei Otto's pioneering works in the late 1950s [1], they became an important research field for many architects and engineers. They provide lightweight structures, composed mainly of cables and membranes, which have no bending stiffness, and thus neither compression stiffness. Generally, it is not possible to define the shape of a taut structure *a priori*, but an appropriate shape is sought, compatible to an initial stress field imposed to the system. Once a viable configuration is found, the structural behavior under external loads can be determined considering geometrically non-linear analysis.

The Force Density Method (FDM) [2] provides the most convenient alternative for shape-finding, approximating a continuous surface by a network of linear elements. By its turn, the Natural Force Density Method (NFDM) [3] is an extension of the FDM that preserves the linearity of the original method, and overcomes some of its limitations to cope with irregular meshes which may arise from non mapped tessellations of surface geometries.

The imposition of natural force densities and the imposition of 2nd Piola-Kirchoff stresses on a reference configuration are equivalent [4], a property previously described for the original force densities by [5]. [4] also recognized that if the solution of the NFDM is recursively used as new reference configurations, the process might converge in few iterations to a configuration in which the resultant Cauchy stresses equal to the imposed 2nd Piola–Kirchhoff stresses. That means that for an isotropic homogeneous stress state, the final converged viable configuration will be a minimal surface. That approach is computationally more efficient for finding minimal shapes than other methods, such as pseudo-dynamic systems and geometrical optimizations. This makes NFDM well suited for structural design softwares, specially when the use of a parametric workflow is featured, as the user can quickly search for shapes, for instance changing the initial stress and then observing the final shape in real-time.A comparison between different methods on Ix-Cube 4.0 software, which has the NFDM implemented, was performed by [6] showing faster and more accurate results provided by NFDM. Although it is not studied in detail in this paper, the FDM and NFDM can also be applied for the form-finding of compression and mixed tension and compression structures (see [7]).

The new paradigms of parametric modelling have been showing promising on the advance of systems for analysis and design of tensile structures. The possibility to integrate Computer Aided Design (CAD) and Computer Aided Engineering (CAE) systems inside a parametric workflow allows better rationalization of the design process and automation of tedious tasks by linking CAD/CAE data for conception, analysis and detailing of structures. Finite Element Analysis (FEA) packages inside the visual programming language (VPL) Grasshopper have been showing outstanding results for structural design, as can be seen at [8], [9] and [10].

In this work, BATS (Basic Analysis of Taut Structures) is presented. BATS is a Grasshopper add-on which applies procedures for form-finding of taut structures and funicular shells inside parametric environment, and it was firstly prototyped for studies regarding the use of NFDM for funicular shell form-finding by [7]. The main code of BATS was updated from a pure C# code to a multi-language code with C# code for pre and post-processing using Grasshopper and Rhino API, which calls the solution from C++ functions, optimized for computational speed. This computational approach overcomes some issues presented on the previous code, as numerical inefficiency, mainly due to the use of the Open Source C# Library Math.NET, and non-optimized assembly of the FEA model from raw CAD geometry. The new code permits updates between many viable shapes in terms of milliseconds with an acceptable mesh refinement.

Benchmarks considering computational speed is presented considering saddle and catenoid minimal surface form-finding. BATS is compared to SATS (System for Analysis of Taut Structures, the first implementation of NFDM in MATLAB) [11], and Kangaroo solver [12], which is a multi-physics simulation Grasshopper add-on that uses pseudo-dynamic methods for many shape finding procedures, including minimal surfaces, with good computational performance.

2 Force Density Method

One of the first alternatives for form-finding was the definition of force densities, proposed by Linkwitz [2] and Sheck [13], in the context of cable nets.

The FDM is based on the equilibrium of each node in a cable net. With reference of the forces at the system in Figure 1, the resultant of internal forces acting on node i is

$$\vec{P_i} = \sum_{j=1}^{n} \vec{P_{ij}} = \sum_{j=1}^{n} N_{ij}\vec{v_{ij}} \tag{1}$$

Where N_{ij} is the interaction force among the nodes i and j, while $\vec{v_{ij}} = \vec{l_{ij}}/\left\|\vec{l_{ij}}\right\|$ is the unit vector oriented from i to j. Applying equilibrium conditions gives equation 2

$$\vec{F_i} + \sum_{j=1}^{n} N_{ij}\frac{\vec{x_j} - \vec{x_i}}{\|\vec{x_j} - \vec{x_i}\|} = \vec{0} \qquad i = 1, 2, \dots, n \tag{2}$$

Relating nodes with nodal displacements results in a non-linear equation system. However, defining at each element a force density n_{ij} (equation 3), a linear system with 3n equations is obtained (4), and with boundary conditions imposed, the system can be easily solved.

$$n_{ij} = \frac{N_{ij}}{\|\vec{x_j} - \vec{x_i}\|} \tag{3}$$

$$\sum_{j=1}^{n} n_{ij}(\vec{x_j} - \vec{x_i}) = \vec{F_i} \qquad i = 1, 2, \dots, n \tag{4}$$

Still, it is convenient for large meshes to adopt a matrix notation instead of the presented one. The Cartesian coordinates of the system, as the external and internal forces, can be expressed by 3 global vectors **X**, **F** and **P**, respectively

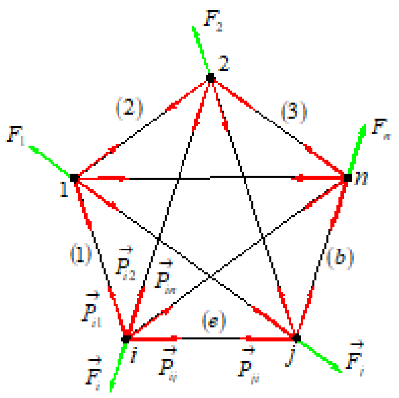

Figure 1: A system of central forces [3]

defined by equations 5

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}_{3nx1} ; \mathbf{F} = \begin{bmatrix} \mathbf{F}_1 \\ \mathbf{F}_2 \\ \vdots \\ \mathbf{F}_n \end{bmatrix}_{3nx1} ; \mathbf{P} = \begin{bmatrix} \mathbf{P}_1 \\ \mathbf{P}_2 \\ \vdots \\ \mathbf{P}_n \end{bmatrix}_{3nx1} \quad (5)$$

Where $\mathbf{x}_i = \left[\vec{x}_i\right]_{3x1}$, $\mathbf{P}_i = \left[\vec{P}_i\right]_{3x1}$ $\mathbf{F}_i = \left[\vec{F}_i\right]_{3x1}$ are local coordinate and force vectors of each node. It is also convenient to substitute the sum inside the definition of **P** from the n nodes to b elements connecting those nodes. The vector of nodal coordinates and the vector of internal forces of the the element, as show in Figure 2, are related by equation 6.

$$\mathbf{x}^e = \mathbf{A}^e\mathbf{x} \quad ; \quad \mathbf{P}^e = \mathbf{A}^{eT}\mathbf{p}^e \quad (6)$$

Where $\mathbf{x}^e = \begin{bmatrix} \mathbf{x}_1^{eT} & \mathbf{x}_2^{eT} \end{bmatrix}^T$ and $\mathbf{p}^e = \begin{bmatrix} \mathbf{p}_1^{eT} & \mathbf{p}_2^{eT} \end{bmatrix}^T = \mathbf{N}^e \begin{bmatrix} \text{-}\mathbf{v}^{eT} & \mathbf{v}^{eT} \end{bmatrix}^T$, with $\mathbf{v}^e = \left[\vec{v}_{ij}\right]_{3x1}$ as the director cosine of the element inside the global Cartesian coordinate system. $\mathbf{A}^e$ is defined as the Boolean incidence matrix of the element.

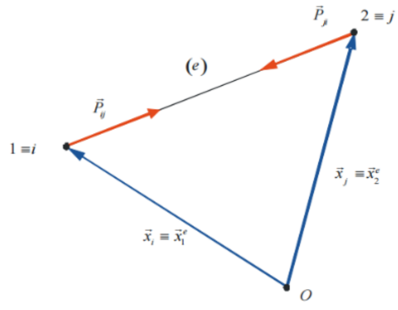
$\vec{P}_j$
$2 \equiv j$
(e)
$\vec{P}_{ij}$
$1 \equiv i$
$\vec{x}_j \equiv \vec{x}_2^e$
$\vec{x}_i \equiv \vec{x}_1^e$
O

Figure 2: Linear element with local and global indexes [3]

Once these definitions are given, the equation 4 can be rewritten as equation 7, where $\mathbf{K}_d = \sum_{b=1}^{e} \mathbf{A}^{eT}\mathbf{k}_d^e\mathbf{A}^e$ is the force density stiffness matrix of the system, with $\mathbf{k}_d^e$ defined in equation 8 is the element local stiffness matrix, with n^e as the element force density and $\mathbf{I}_3$ as an order 3 identity matrix.

$$\mathbf{K}_d\mathbf{X} = \mathbf{F} \quad (7)$$

$$\mathbf{k}_d^e = n^e \begin{bmatrix} \mathbf{I}_3 & -\mathbf{I}_3 \\ -\mathbf{I}_3 & \mathbf{I}_3 \end{bmatrix} \quad (8)$$

3 Natural Force Density Method

The NFDM preserves the linearity of the original FDM meanwhile issues related to irregular triangular meshes are overcome. The natural force density derives from the natural forces defined at the natural membrane finite element, first proposed by Argyris [14]. Pauletti [3] redefines the formulation of previous developments into a more concise notation.

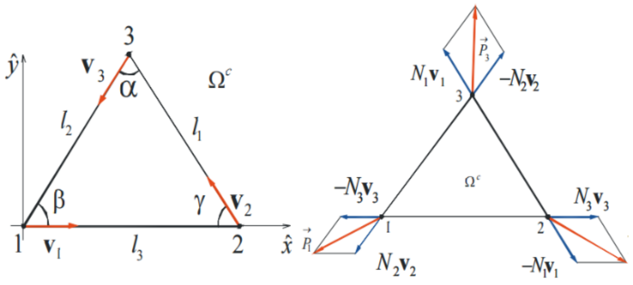

Figure 3: Natural Membrane Finite Element (a) unit vectors, (b) internal forces [3]

The nodes and sides at Figure 3 are numbered anticlockwise, where each node index is coincident with the face index in front of it. Node coordinates are referenced at both global and local Cartesian system, with the local representation expressed by " ∧ ".

To define the internal vector forces, it is convenient to define natural forces N_i , which are parallel to its faces. The relationship between the internal force vector of the element $\mathbf{p}^e$ and the internal forces are defined in equation 9. After some algebra (see [15]) it is possible to rewrite the natural loads vector $\mathbf{N} = [N_1N_2N_3]^T$ as in equation 10, where $V = At$ is the element volume, $\boldsymbol{L} = diag\,\{l_1, l_2, l_3\}$ a diagonal *"lenght matrix"* and **T** is a transformation matrix, given by equation 11.

$$\mathbf{p} = \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} = \begin{bmatrix} N_2\mathbf{v}_2 - N_3\mathbf{v}_3 \\ N_3\mathbf{v}_3 - N_1\mathbf{v}_1 \\ N_1\mathbf{v}_1 - N_2\mathbf{v}_2 \end{bmatrix} \quad (9)$$

$$\mathbf{N} = V\mathbf{L}^{-1}\mathbf{T}^{-T}\hat{\boldsymbol{\sigma}} \quad (10)$$

$$\mathbf{T} = \begin{bmatrix} \cos^2\gamma & \sin^2\gamma & -\sin\gamma\cos\gamma \\ \cos^2\beta & \sin^2\beta & -\sin\beta\cos\beta \\ 1 & 0 & 0 \end{bmatrix} \quad (11)$$

Comparing equations 9 with 6 suggest the definition of *natural force densities* $n_i = N_i/l_i$, and using the relation given at equation 10 gives us a *vector of natural force densities* (equation 12).

$$\mathbf{n} = \begin{bmatrix} \frac{N_1}{l_1} & \frac{N_2}{l_2} & \frac{N_3}{l_3} \end{bmatrix}^T = V\mathbf{L}^{-2}\mathbf{T}^{-T}\hat{\boldsymbol{\sigma}} \quad (12)$$

With the definition of **n** it is possible to assign the *natural force density stiffness matrix* $\mathbf{k}_{nd}$ as in equation 13, and the solution of a system with n linear elements and m triangular elements is given by equation 7 with $\mathbf{K}_d = \mathbf{K}$ (equation

14)

$$\mathbf{k}_{nd} = \begin{bmatrix} (n_2+n_3)\mathbf{I}_3 & -n_3\mathbf{I}_3 & -n_2\mathbf{I}_3 \\ -n_3\mathbf{I}_3 & (n_1+n_3)\mathbf{I}_3 & -n_1\mathbf{I}_3 \\ -n_2\mathbf{I}_3 & -n_1\mathbf{I}_3 & (n_1+n_2)\mathbf{I}_3 \end{bmatrix} \qquad (13)$$

$$\mathbf{K} = \mathbf{K}_d + \mathbf{K}_n d = \sum_{b=1}^{n} \mathbf{A}^{b^T}\mathbf{k}_d^b\mathbf{A}^b + \sum_{c=1}^{m} \mathbf{A}^{c^T}\mathbf{k}_{nd}^c\mathbf{A}^c \qquad (14)$$

Although this linear procedure produces a viable configuration of a bending-free shapes, the imposition of natural force densities to achieve a uniform stress field is nonviable. However, the NFDM process can be evaluated recursively by assigning a constant stress field $\hat{\boldsymbol{\sigma}}_0$ and repeatedly determining $\mathbf{K}$, using $\hat{\boldsymbol{\sigma}} = \hat{\boldsymbol{\sigma}}_0$ on Eq. 12, where the new reference configuration is updated every step. [4] shows that this process converges the solution to a minimal surface, as the Cauchy stress resultants for this converged solution are equal to the uniform second Piola–Kirchhoff stress resultants, which is an analogous solution to a mathematical minimal surface by the *soap bubble* analogy. This process finds minimal surface shapes in quite few linear steps, as will be discussed further, and is an interesting approach as requires low computational cost and produces a viable configuration at every step, feature that have clear advantages compared to non-linear methods that converges to a minimal solution but usually through a series of non-equilibrium configurations.

The NFDM can find also non-minimal shapes by imposition of non-isotropic and non-uniform stress fields. A obvious example is application of a single step of the NFDM, which gives a viable shape, yet have a non-uniform stress state. The imposition of non-isotropic solutions can be done

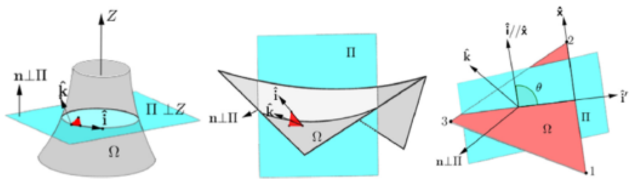

Figure 4: Definition of an orthotropic initial stress field by using director places [15].

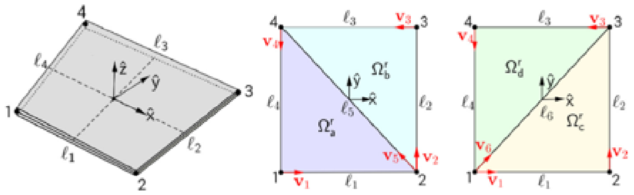

Figure 5: Definition of a quadrangular element by coupling multiple triangular elements [15].

by finding natural force densities by an initial orthotropic stress field using director planes on the elements, as detailed by [15]. [15] also proposed a method to extend the NFDM to quadrangular elements, by coupling multiple triangular elements on both directions to reduce biased stress results.

4 CAD/CAE integration by parametric workflow

The steps of design and analysis of taut structures presents some computational challenges, specially in the communication between CAD (Computer Aided Design) and CAE (Computer Aided Engineering) systems. CAD generally have resources for geometrical definition of complex shapes and details, where CAE is used for computational physical analysis. [16] defines the main issues on CAD/CAE integration, and address the main issues to loss of data, compatibility during the process and lack of automation. Those issues are due the different characteristics between the processes. A CAD model is mainly a computational representation of the geometry, not necessarily having attributes and properties. These features are crucial inside CAE environment, as it needs data such as material properties, physical interfaces and element types. Many CAE softwares offers CAD modelling features inside its interfaces, but still very limited compared to CAD specialized softwares. Also, some FEA packages are only code implementation, which need a external pre and post processor to work properly as a design tool.

On the other hand, specialized CAD software as Rhinoceros3D [17] have been delivering scripting features with API's (Application Programming Interface) giving the developer access to the CAD system geometrical objects inside an object-oriented-programming environment. With this features it is possible to develop CAE structural applications with full use of CAD capabilities, using geometrical objects as parameters for structural classes.

In CAD scripting context, mainly two types of programming are featured: Visual Programming Languages (VPL) and Textual Programming Languages (TPL). A VPL consists on visual block elements which contains algorithms and can be manipulated in a logical sequence of inputs and outputs. On other hand, TPL systems relies on a sequence of linear characters which describes the commands the program should execute [18]. VPL's are advantageous as do not require a broad knowledge on programming syntax, and are easier to use for general CAD users. The possibility of association of inputs and outputs in VPL's are interesting

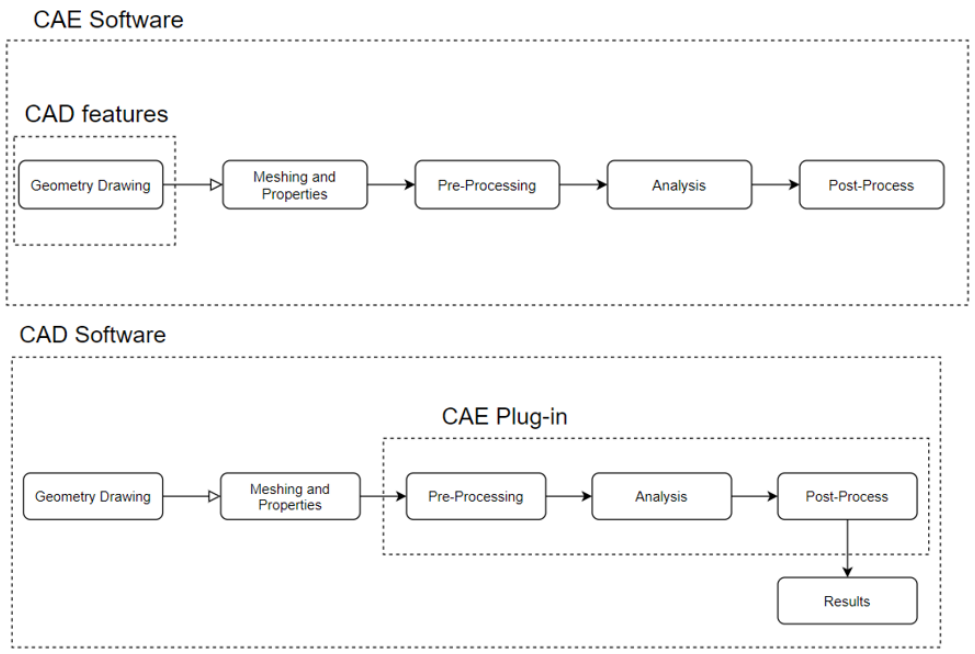

Figure 6: Possibilities in CAD/CAE integration

for CAE, as association between geometrical and analysis data can be made. However, VPL's are strictly dependent on TPL's, as each block component on the VPL is produced from a TPL code.

In terms of TPLs, applications that depends on numerical methods requires, besides code with low algorithmic complexity, a programming interface that offers nice memory allocation tools. Classical examples for high performance languages for linear algebra procedures are FORTRAN(with LAPACK and BIAS libraries) and C++ (with Eigen numerical library). On other hand, CAD software normally features APIs from more modern programming languages as C# and Phyton, as they are more versatile to work and delivers better features for object-oriented-programming and dynamic types, which fits well for API purposes as the developer can access and correlate all geometric classes and its methods given from the CAD software.

Working with multiples languages and types of programming can then lead towards a nice computer performance with user-friendly interface, that can be established in 3 levels: (1) User level, or the VPL itself; (2) Application level, for communication between the user and the numerical level; (3) Numerical processing level, which solves the problem desired with optimized code.

Rhinoceros is a CAD software and provides it's VPL Grasshopper, which have components for manipulation of many Rhino geometries and so forth enables a parametric workflow inside it's environment. The user can define a logical sequence of events describing the project in function of predefined parameters, and instantly retrieve geometry and analysis feedback within changes on them. Both Rhinoceros and Grasshopper have their own C# API's (*Rhinocommon (RC)* and *GrasshopperSDK (GS)*), Where *GS* depends on *RC*, and allow custom components and parameters development for many purposes. Many CAE extensions were developed as Karamba3D [8], Kiwi3D [9] and Beaver [19] for structural analysis, Ladybug [20] for Thermar/Solar Analysis, and Butterfly [20] and Eddy3d [21] for CFD analysis.

The use of a parametric workflow in FEA is then a promising candidate for solving CAD/CAE intercommunication problems, as it can automate processes converting geometric information and other relevant data for analysis. The conversion of data is automated by assembling series of algorithms, where any change in geometry returns a suitable new structural model, avoiding rework in mesh generation and other possible tedious tasks.

Thus, structural analysis software within a parametric environment enables a user-friendly yet powerful design and analysis process. The environment allows the user to implement routines for specific and interconnected tasks based on the geometry and properties of the membrane. In addition, The VPL routines pattern allows the user to perform the analysis and extract and reuse the results for other routines such cutting patterns, connection details and verification of structural components by technical standards [22].

5 BATS implementation

In order to implement an efficient and robust form-finding method inside parametric workflow environment, the Grasshopper plugin BATS was developed. It was firstly prototyped in [7], and was not concerned yet about computer efficiency. However, it's resulting code appointed for two main issues regarding computational performance.

The first issue is related to the linear system solution and other linear algebra processes required. Matrices can be highly memory consuming, considering that given n nodes, corresponding each one to 3 degrees of freedom, the number of variables inside the matrix is on the order of $(3n)^2$. This means that the complexity solving linear systems are in exponential order. Also, modern languages as C# and Python works with automatic memory management and dynamic types, which can be undesired for numerical purposes evolving matrices. In addition, it is well know in numerical analysis area that languages like FORTRAN and C++ provides tools for memory management which can optimize computational cost, and this naturally leads to more efficient numerical libraries available. In C#, a wide-used numerical library is the Open-Source library Math.NET. Although quite user-friendly and versatile, it shows high computational cost solving large linear system as also other linear algebra procedures.

On other hand, the C++ open-source library Eigen have proven highly efficient, leading to fast results even for large systems. Benchmarks shows that Eigen performs almost the same as Math Kernel Library (MKL), a commercial numerical library well know by it's good performance [23]. That makes use of these resources highly desirable inside parametric workflow, as this can lead to practically real-time visualization of form-finding results. The advantage of C++ in Grasshopper development relies on great interoperability between C# and C++, where C++ functions in numerical level can be called by C# code.

The second issue is due the assembly of a FEA model based on geometrical information, where the program should index all geometrical nodes and assign the connectives between the elements. This is not a trivial task, as requires searching algorithms for equal points in 3D space. Linear search, which is the most obvious approach and was used in prototyping, provides a $O(n)$ complexity, and can be highly consuming when dealing with thousands of nodes. Other search algorithms can be performed in order to improve the complexity of the algorithm. Binary search and also R-Tree search can deal with the problem respectively with $O(\log n)$ and $O(\log_M n)$, where M denotes to the maximum entries in R-Tree nodes [24]. This task can be done in application level, as do not have matrices or linear algebra applications involved. Yet, it can still be a tedious process, taking more computational time than the solver, as evolves a search problem on a large list of 3D points.

The VPL components were developed in a C# application that uses C++ unmanaged functions from the developed numerical solver. Figure 7 shows the general workflow of the developed tool. First, raw Rhino/Grasshopper data is converted into structural analysis data, which is then used to assemble the structural model. All this data is structured in object-oriented-programming in C#.

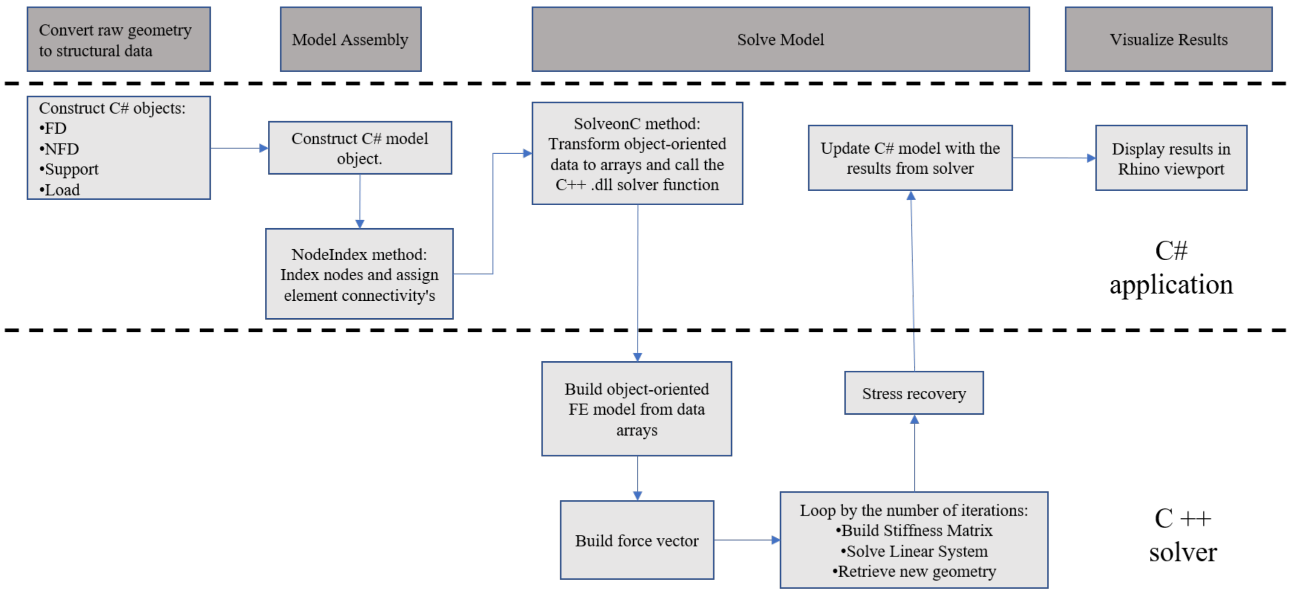
Convert raw geometry to structural data
Model Assembly
Solve Model
Visualize Results
Construct C# objects: •FD •NFD •Support •Load
Construct C# model object.
NodeIndex method: Index nodes and assign element connectivity's
SolveonC method: Transform object-oriented data to arrays and call the C++ .dll solver function
Update C# model with the results from solver
Display results in Rhino viewport
C# application
Build object-oriented FE model from data arrays
Stress recovery
Build force vector
Loop by the number of iterations: •Build Stiffness Matrix •Solve Linear System •Retrieve new geometry
C ++ solver

Figure 7: Workflow of the application and solver implementations

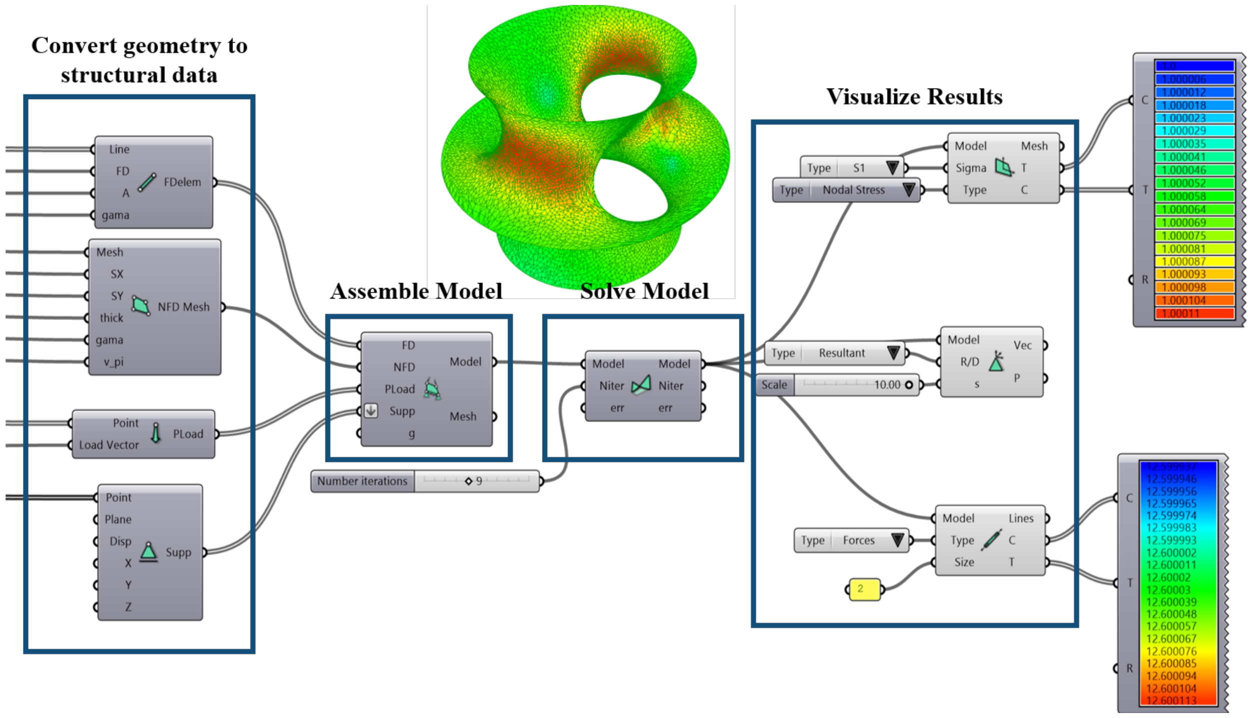

Figure 8: BATS components and results

The next step is to solve the model, where the C# application calls the solver function. In this step the structured data is transformed in a series of arrays, as the C# and C++ interoperability is straightforward only for native C types. In the C++ code, the data is then restructured in object-oriented style, but it have different purposes than the C# data. The C# application data aims at general structural data, that any FEA user would be familiar with, and the C++ solver data focus on attributes and methods for assembly of the external force vector and stiffness matrix. Although firstly required for simplicity in data transfer between two programming languages, the scheme of have independent data structures for the application and the solver showed to be of benefit, as each implementation can deal with their own needs separately.

Latter, the C++ code outputs the results also in arrays, and the C# application retrieves the results and prepare them for visualization inside the Rhino/Grasshopper interface. The details of each part of the developed tool, respectively the VPL components, the C# application code and the C++ numerical code, are discussed in detail the following subsections.

5.1 VPL components

Figure 8 describes the component workflow for the form-finding related to the workflow presented on Figure 7. A series of components retrieves geometrical data of meshes, lines and points and it's respective supplementary data, for force density linear elements, natural force densities mesh elements, supports and loads.

Each component outputs custom parameters which contain all data provided, and are used to assemble the model, subsequently sending the model parameter to the solver component, which updates the model object with the shape found and its respective stress field. Results can be retrieved by special components, where membrane view, cable view and support reaction view are available.

With that workflow, the user can modify the form-finding parameters (i.e. initial stress fields) as well as the initial geometry and its constraints. That makes the process of form-finding very versatile, as shape finding definition can be defined by geometrical and structural design parameters inside a CAD environment and with real-time updates within parameter changes.

5.2 Application level

Figure 9 shows the class diagram of the application code in C#, that is the core for the development of the VPL components. It is responsible to intercommunicate Rhino/Grasshopper with BATS numerical engine, and contains a collection of classes with geometric variables defined as Rhino geometrical objects in *RhinoCommon*, with supplementary information relevant for the analysis.

Node class collects variables regarding it's 3D geometrical point in *RhinoCommon* and have it's FEA index associated, which is determined on a further step inside Model class. For post-processing and colour plot purposes, it also have methods and fields for resultant nodal stress definition.

Support Class contains a Node class field, which also have only geometrical information before the pre-processing. More fields are needed for these objects, as support plane, prescribed displacements and constrained DOFs. Reactions field and methods are needed in order to post-process the model after solution.

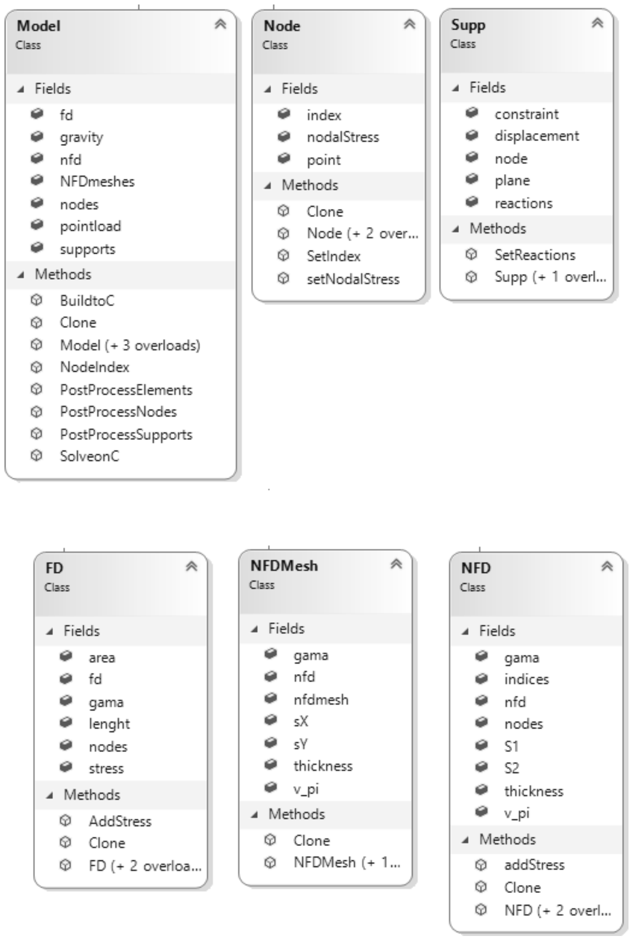

Figure 9: Class Diagram for application C# code

FD Class contain the geometrical line defined by two non-indexed nodes (i.e. geometrically informed only) which represents the element and other information described in Section 2. Also, a material density field is available for funicular shell purposes.

NFD classes contains a single triangular or quadrangular element defined by it's non-indexed nodes and other relevant information described in Section 3. Similar to FD class, it also contains fields regarding material density. For simplicity and assembly process, a auxiliary NFDMesh class is used, as this permits the definition of properties in a whole mesh as also speeds up model assembly as connectivity data of the mesh can be directly passed through the model. Elements have also fields for post-processing results, as the resultant stress fields.

Model class contains list field of all classes listed above, and has the relevant methods for pre and post-processing. It also contains the C++ calling method for the solve process. The pre-process is done by calling *NodeIndex* method, which retrieves all tridimensional geometrical information and provides index assignment of the global model nodes as also the reference global supports and element nodes indexes. *BuildtoC* and *SolveonC* methods grabs all structured data and assign series of data arrays to be sent to the C++ function, which will handle the numerical processing and retrieves the NFDM results. At last, post-process on each object type is made by *PostProcessElements*, *PostProcessNodes* and *PostProcessSupports* methods.

Every application class have it's corresponding parameter wrapper inside Grasshopper, which makes possible connection of objects data between components. That makes possible the assembly structure proposed in the VPL components, as the VPL do not recognize automatically custom classes that are not on *RhinoCommon* or *GrasshopperSDK*.

5.3 Numerical level

Numerical level refers to series of C++ functions related to a main function, which is called from application level. The main function must call it's variables considering interoperability between both C# and C++ variable types. C# provides compatibility of main C native data types when calling C++ functions using DllImport resources [25]. Arrays from the native types, that are needed due to size of data, can be passed considering them as pointers in C++, that will access directly the memory data called in C# code.

Sparse matrices are used as they storage only non-null values, considerably reducing memory allocation. However, sparse matrix build-up can be slow when applied changes on global indexes, as its structure is not directly correlated to dense matrix indexes [26]. Hopefully, sparce matrix assembly can be optimized by directly providing a set of indices and corresponding values, which are called *triplets*, defined by three values $(i, j, value)$.

For matrix assembly, the global stiffness matrix $\boldsymbol{K}$ is partitioned between free and constrained DOFs, resulting on four sub-matrices $\boldsymbol{K}_{ff}$, $\boldsymbol{K}_{fc}$, $\boldsymbol{K}_{cf}$, $\boldsymbol{K}_{cc}$, as the solution process requires only the unconstrained part, which can cause economy in highly constrained systems, and reaction forces can be directly obtained using the results with the other submatrices. Therefore, a list of triplets is needed for each submatrix, which is done by the *Partition* method, which stores each triplet list for further matrix assembly.

Figure 10 shows auxiliary classes for numerical model assembly. Arrays data are rebuilt in a scheme of classes $DOFs \rightarrow Nodes \rightarrow Elements$. This helps to organize matrix assembly of the linear system in a object-oriented scheme. Elements have information about force densities and also of its nodes, which defines its type (linear, triangular or quadrangular). Node objects contains geometrical position, its nodal index and correponding DOFs. DOF object then collects unpartitioned and partitioned indices.

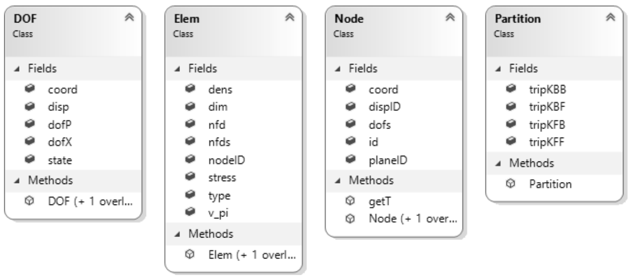

Figure 10: Class Diagram for numerical C++ code

That approach turns matrix assembly simple yet effective, as it avoid further searching algorithms for constrained DOFs during assembly, as they are directly considered before this step. The same partitioning process is done in the force vector $\boldsymbol{F}$ in $\boldsymbol{F}_f$ and $\boldsymbol{F}_c$. $\boldsymbol{LL}^T$ factorization with Cholesky Decomposition is used to solve linear systems, as $\boldsymbol{K}_{ff}$ preserves the symmetry and positive definitive properties required. At last, outputs delivers the new geometry with the resultant stress field and reactions.

Addressing these main issues retrieves good performance even for large systems,as is shown in Section 6.1.

6 Examples and benchmarks

Examples and benchmarks are shown in order to evaluate BATS reliability and computational performance. All tests were made with a personal computer with the following specifications: Intel i7 9750h processor (2.4Ghz), 16GB RAM. BATS is compared for minimal surface form-finding with two other solvers: Kangaroo [12] and SATS [11].

Kangaroo is a multi-physics solver with a wide-range of physical simulations and features coupling between many physical objectives. In terms of membrane and cable design, it offer both minimal edge lengths and minimal surface solutions, where the latter is obtained by an iterative mathematical procedure imposing zero Gaussian's curvature on the mesh.

SATS is a MATLAB program which contains routines for form-finding and analysis of taut-structures and was the first program which NFDM was implemented. As BATS is a further development of SATS with better user-interface, only computer performance is analysed.

For all benchmarks it is assumed that the 200th iteration of NFDM is an acceptable minimal surface. [6] and [15] shows that for the shapes analysed the NFDM converges to these solutions in few steps. Then the geometrical error for each procedure can be evaluated and analysed at each iteration by the following relation:

$$err = \left\| \boldsymbol{X}_i - \boldsymbol{X}_{200}^{NFDM} \right\| \qquad (15)$$

6.1 Minimal saddle surface with boundary restriction

Figure 11 shows an initial mesh made from a simple NURBS patch. The model is defined by a square mesh of NFDM elements with FD elements in the mesh boundary, supported by the 4 corner edges. The addition of FD elements in the boundary adds a restriction to the minimal shape finding procedure which gives the best-fit solution for a minimal shape considering the tension in the FD elements. For the models assembled in Kangaroo, both considers an additional boundary restriction with linear spring elements at the border. This restriction is important as the surface degenerates if only the corner points are set fixed. More details on this matter can be seen at [27].

Performance analysis was made in three different mesh densities: 10x10, 25x25 and 50x50 grid. On all tests, the shape found for the saddle surface matches for the NFDM and minimal surface procedure with boundary restrictions on Kangaroo. However, the minimal edges procedure on Kangaroo presented a deviation from the exact solution.

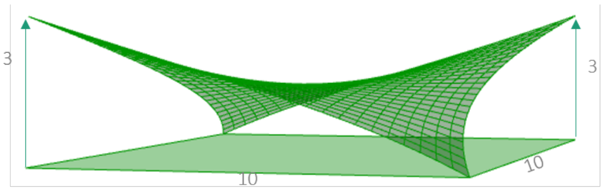

Figure 11: Initial mesh for form-finding

This is explained by the fact that the minimal edges procedure considers linear elements that are dependent on line length and mesh topology, and the correct definition for force densities is non intuitive. The definition for minimal edges on each line addresses the direction of principal stresses to the lines axis, and this constraint leads to the error observed. Therefore, the minimal edges length method, which is analogous to the original force density method, does not provides a correct solution.

The following graphs (Figure 12, 13 and 14) shows the evolution of the geometrical error as a function of both iteration number n and the time elapsed t.

With exception of the 10x10 grid solution, the mathematical minimal surface procedure on Kangaroo presented initial increase of the error, which starts to converge when the other methods are already or almost on convergence domain. It can be seen that in both terms of n and t BATS presented direct convergence to the solution, requiring a maximum of 20 iterations to converge (in the 50x50 grid case).

Both methods in Kangaroo presented an oscillatory behaviour for the error, and required more iterations to converge. Even that the minimal edge surface method required fewer time per iteration, the NFDM property of directly find viable shapes at each iteration, makes it more efficient as quite few steps are required to achieve convergence.

Interactive pseudo-dynamic methods presented on Kangaroo are explicit methods and so forth requires more steps to reach a solution, as they are dependent on the time step at each iteration. The procedure tries to find a equilibrium state in the pseudo-dynamic scheme by imposing constraints on the motion, which error function in

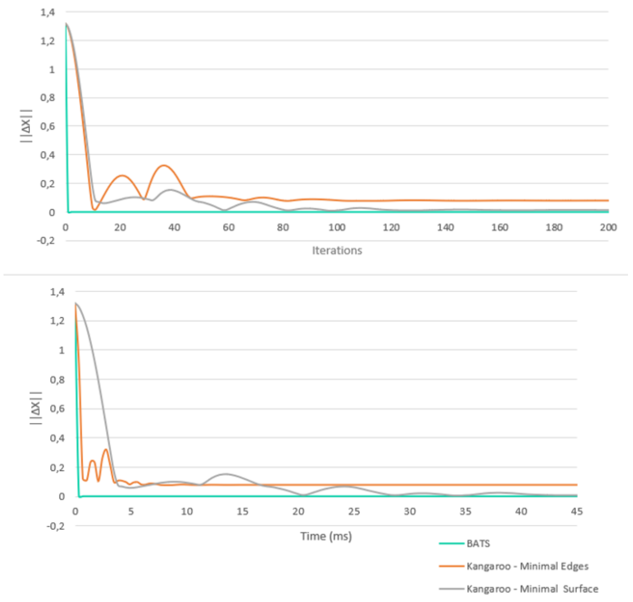

Figure 12: Iteration and time comparison for 10x10 grid

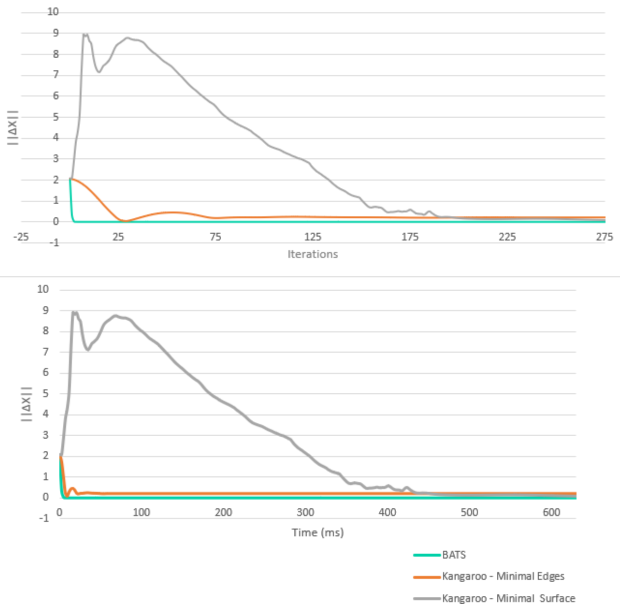

Figure 13: Iteration and time comparison for 25x25 grid

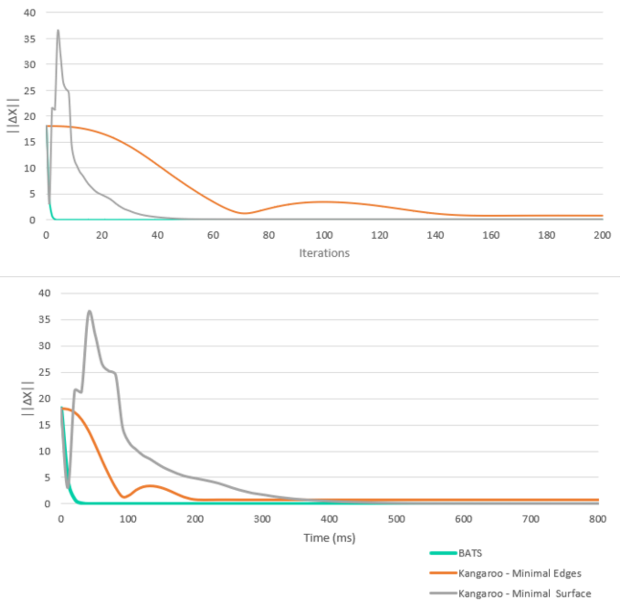

Figure 14: Iteration and time comparison for 50x50 grid

terms of time/iteration presents the behaviour of damped oscillations. However, these methods can be helpfull when dealing with non-linear structural analysis of tensile non-linear materials, as the problem can be derived only from the internal and external force vectors and avoids the use of the tangent stiffness matrix, that might be singular due to the local instabilities on the membrane. This method for non-linear systems solution is the so called Dynamic Relaxation method, that, although lacks computational efficiency compared to direct methods, it is more robust and achieve solutions which are quite difficult to using a Newton-Raphson solver. Further reading in this topic is addressed to [28] and [29].

Figure 15 gives the time elapsed on each procedure for convergence. BATS performs considerably quickly than other implementations.

SATS elapsed time is presented and requires the same iterations than BATS, as also uses the NFDM procedure, however presented lower performance. The main differences on the implementation of BATS and SATS is the language (C++ and MATLAB), numerical library (Eigen and MATLAB), sparse matrix assembly (optimized with triplets and no optimization) and linear system solving (factorization with Cholesky decomposition and generalized linear system solver).

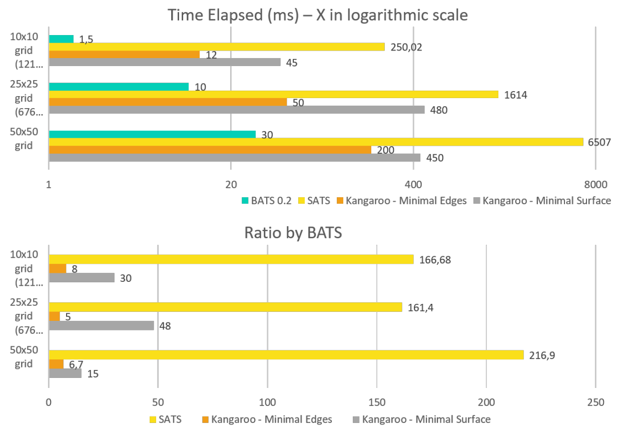

Figure 15: Comparison with SATS included

6.2 Minimal catenoid surface

This example is defined from an initial mesh of a cylinder of radius r and height h, for the form-finding of a minimal catenoid shape. For this class of shapes, a minimal surface can be found analytically with the Goldschmidt solution, which states that a minimal surface of a catenoid, with two equal radius rings as boundary, only have a solution for $h < 1.3254868r$ [30].

As explained before, the minimal edges procedure is not equivalent to a minimal surface form-finding and will be neglected in this example. Three different heights are defined in order to check the Goldschmidt limit: $h = 1.30r$ (Figure 16), $h = 1.32r$ (Figure 17) and $h = 1.34r$ (Figure 18). Besides a apparent gradient of stresses showed on the stress colour plot in the catenoid, the maximum stress variation is on order of $10^{-}4$, and then it can be clearly assumed that the stress distribution is uniform for all converged BATS results.

Expected results arises for h bellow the Goldschmidt limit, where both procedures converges to the same shape.Above the limit, both solutions diverge into a degenerative mesh as no solution is viable. However, it can be seen that for $h = 1.32r$, which is quite close to the limit, BATS converges to a solution whereas Kangaroo don't, showing better sensitivity of results using the NDFM.

The graph of geometrical error in function of time elapsed is presented on Figures 19 and 20. It can be observed that for $h = 1.30r$ both solutions behaves in the

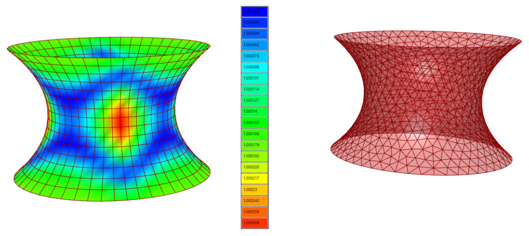

Figure 16: Catenoid solution for $h = 1.30r$. BATS on the left, Kangaroo on the right

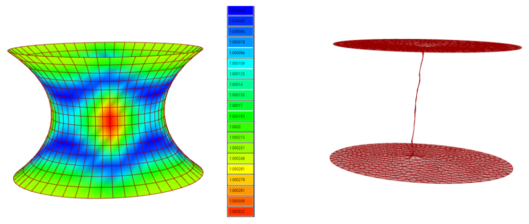

Figure 17: σ_1 plot of the catenoid solution for $h = 1.32r$. BATS on the left, Kangaroo on the right

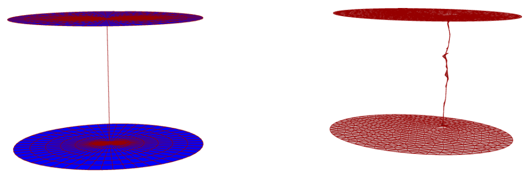

Figure 18: Conoid solution for $h = 1.34r$. BATS on the left, Kangaroo on the right

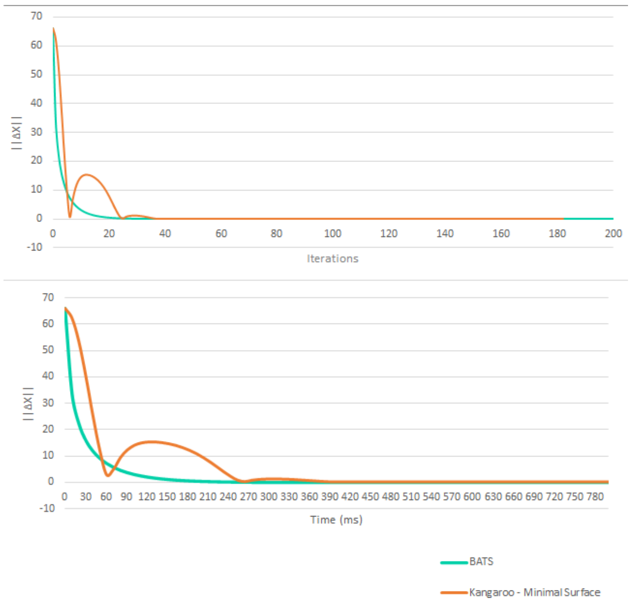

Figure 19: Iteration and time comparison for $h = 1.30r$.

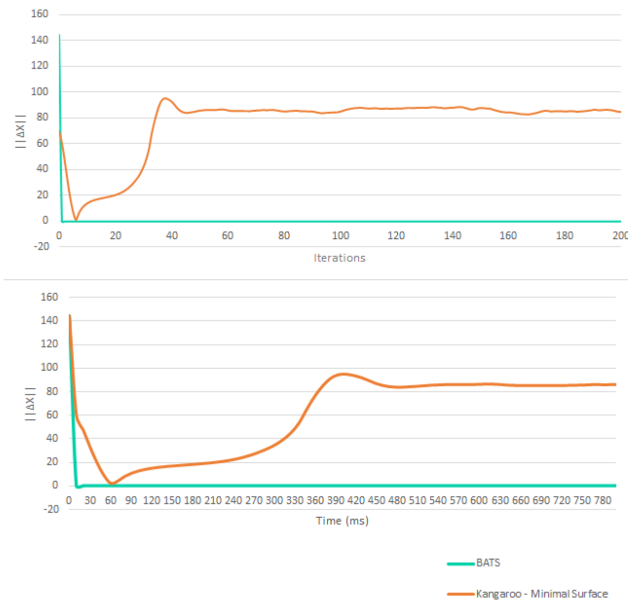

Figure 20: Iteration and time comparison for $h = 1.32r$.

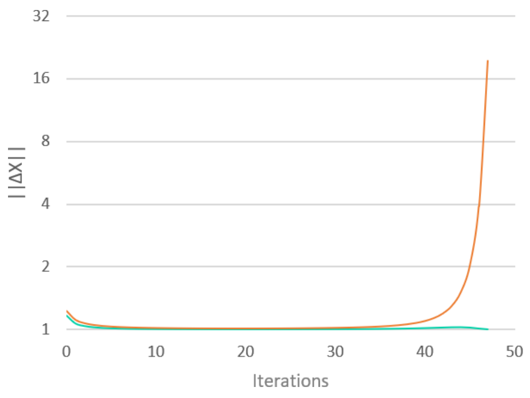

Figure 21: Minimal and Maximum resultant stress per iteration, for $h = 1.34r$

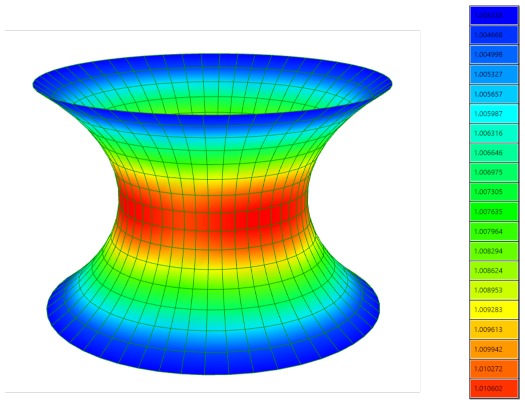
Figure 22: Last viable shape before solution diverges.

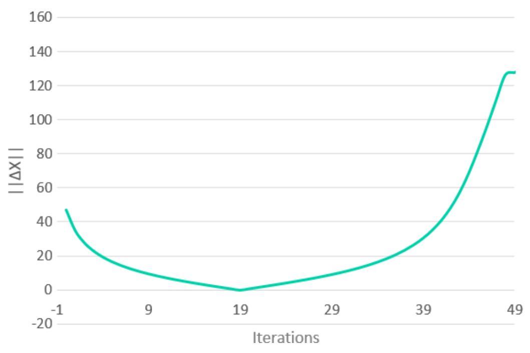

Figure 23: Comparison between the last viable shape and corresponding iteration resultant geometry $h = 1.34r$.

same manner as the saddle surface example. On other hand, $h = 1.32r$ shows Kangaroo almost reaching a solution but diverging the minimum error is achieved, which happens at both BATS and Kangaroo solution for $h = 1.34r$.

However, as NFDM produces at each iteration an viable solution, it's possible to retrieve the last iteration in which the stress field deviance decreases. For $h = 1.34r$ case, Figure 21 shows the maximum and minimum stress acting on the surface, where when the values are equal a uniform stress field is achieved. It was observed that the shape with the stress field closest possible to a uniform state is found in the 20^{th} iteration, and it can be seen on Figure 22.

The error between each iteration and this solution is plotted in Figure 23, where the convergence domain and divergence domain of the error function can be observed.

7 Conclusion

This work showed a implementation of the Natural Force Density Method in a novel framework composed by Grasshopper components, application code in C# for component programming, and a numerical code in C++.

Comparison with the first implementation showed the importance of efficient memory management and the choice of numerical libraries and methods. Comparison with other form-finding methods in Kangaroo shows how an efficient implemented implicit method can perform highly better than a nicely implemented explicit method. Besides it, the developed tool delivers viable configurations at all iterations as also deliver the resultant stress field associated with the shape.

The combination of a robust form-finding method as NFDM with adequate numerical programming techniques and efficient numerical libraries inside a parametric environment produced a reliable, fast and versatile form-finding tool for the shape finding of taut structures and funicular shells. Future research is on applying the computational framework for non-linear analysis and cutting pattern routines for taut structures, taking advantage of the numerical efficiency presented.

Funding information: The authors state no funding involved.

Author contributions: All authors have accepted responsibility for the entire content of this manuscript and approved its submission.

Conflict of interest: The authors state no conflict of interest.

References

[1] Aldingerm IL. Frei Otto: Heritage and Prospect. Int J Space Struct. 2016;31(1):3. DOI: https://doi.org/10.1177/0266351116649079.
[2] Linkwitz K, Schek HJ. Einige Bemerkungen zurBerechnung von vorgespannten Seilnetzkonstruktionen. Ingenieur-archiv. 1971;40(3):145. DOI: https://doi.org/10.1007/BF00532146.
[3] Pauletti RM. An extension of the force density proce-dure to membrane structures. In: Int Assoc Shell Spatial Struct (IASS) Symp. Beijing, China; 2006.
[4] Pauletti RM, Pimenta PM. The natural force density method for the shape finding of taut structures. Comput Methods Appl Mech Eng. 2008;197(49-50):49–50. DOI: https://doi.org/10.1016/j.cma.2008.05.017.
[5] Bletzinger KU, Ramm E. General finite element approach to the form finding of tensile structures by the Updated Reference Strategy. Int J Space Struct. 1999;14(2):131. DOI: https://doi.org/10.1260/0266351991494759. .
[6] Prandini R, Pauletti RM. A comparison of alternative form finding methods in ixCube 4.10 program. In: Int Assoc Shell Spatial Struct (IASS) Symp. Tokyo, Japan; 2016.
[7] Souza MS, Pauletti RM. Parametric design and optimization of shell structures using the Natural Force Density Method. In: Int Assoc Shell Spatial Struct (IASS) Symp. Tokyo, Japan; 2016.
[8] Preisinger C. Linking structure and parametric geometry. Architectural Design. 2013;83(2):110. DOI: https://doi.org/10.1002/ad.1564.
[9] Bauer AM, et al. Exploring software approaches for the design and simulation of bending active systems. In: Int Assoc Shell Spatial Struct (IASS) Symp. Boston, USA; 2018.
[10] Pini JT, Souza MS. Beaver: A computational parametric approach for conception, analysis and design of timber structures. In: World Conf Timber Eng (WCTE). Santiago, Chile; 2020.
[11] Guirardi D. The dynamic relaxation method applied to the analysis of cable and membrane structures. PhD thesis. University of São Paulo; 2011. DOI: https://doi.org/10.11606/T.3.2011.tde-29062012-151842.
[12] Piker D. Kangaroo: Form Finding with Computational. Physics Architectural Design, 2013, 83(2):136. DOI: https://doi.org/10.1002/ad.1569.
[13] Schek HJ. The force density method for form finding and computation of general networks. Comp Meth Appl Mech Eng. 1974;3(1):115. DOI: https://doi.org/10.1016/0045-7825(74)90045-0.
[14] Argyris JH, Dunne PC, Angelopoulos T, Bichat B. Large natural strains and some special difficulties due to non-linearity and incompressibility infinite elements. Comp Meth Appl Mech Eng. 1974;4(2):219. DOI: https://doi.org/10.1016/0045-7825(74)90035-8.
[15] Pauletti RMO, Fernandes FL. An outline of the natural force density method and its extension to quadrilateral elements. Int J Solids Struct. 2020;185-186:423. DOI: https://doi.org/10.1016/j.ijsolstr.
[16] Gujarathi G, Ma YS. Parametric CAD/CAE integration using a common data model. J Manuf Syst. 2011;30(3):118–32. DOI: https://doi.org/10.1016/j.jmsy.2011.01.002.
[17] McNell, Rhino3D, https://www.rhino3d.com.
[18] Leitão A., Santos L. Lopes J. Programming languages for generative design: A comparative study. Int J Arch Comput. 2012;10(1):139. DOI: https://doi.org/10.1260/1478-0771.10.1.139.
[19] Souza MSV, Pini JT. Beaver. https://www.food4rhino.com/app/beaver.
[20] Roudsari M, Pak M. Ladybug: A parametric environmental plugin for grasshopper to help designers create an environmentally-conscious design. In: Proc BS 2013: 13th Conf Int Build Perf Simul Assoc. Chambéry, France; 2013.
[21] Kastner P, Dogan T. A cylindrical meshing methodologyfor annual urban computational fluid dynamics simulations. J Build Perf Simul. 2019;13(1):59. DOI: https://doi.org/10.1080/19401493.2019.1692906.
[22] Cruz AC, Pini JT, Souza MSV. Project and analysis of glued laminated timber using parametric modelling. Barchelor thesis. University of São Paulo; 2018 (In Portuguese). DOI: https://doi.org/10.13140/RG.2.2.21317.68323.
[23] Jacob B, Guennebaud G. Eigen. http://eigen.tuxfamily.org.

[24] Guttman A. R-trees: A dynamic index structure for spatial searching. In: Proc ACM SIGMOD Int Conf Manag Data. New York, USA; 1984. DOI: https://doi.org/10.1145/602259.602266.

[25] Microsoft, DllImportAttribute Class, https://docs.microsoft.com/en-us/dotnet/api/system.runtime.interopservices.dllimportattribute?view=netcore-3.1.

[26] Dhatt G, Touzot G. Lefrançois E. Finite Element Method. John Wiley & Sons, Inc. USA; 2012. DOI: https://doi.org/10.1002/9781118569764.

[27] Souza D, Pauletti R. Almeida Neto E. Finding minimal surfaces by direct area minimization. IASS-SLTE Int Symp. Acapulco, Mexico; 2008.

[28] Barnes MR. Form-finding and analysis of prestressed nets and membranes. Comp Struct. 1988;30(3):685. DOI: https://doi.org/10.1016/0045-7949(88)90304-5.

[29] Pauletti RM, Guirardi D, Adriaenssens S, Rhode-Barbarigos L. An efficient mass-tuning algorithm for the Dynamic Relaxation Method applied to cable and membrane structures. In: VII Int Conf Text Compos Inflat Struct.Barcelona, Spain; 2015.

[30] Goldschmidt C., Determinatio superficiei minimae rotatione curvae data duo puncta jungentis circa datumaxem ortae, Gottingae, Typis Di- eterichianis. 1831. https://hdl.handle.net/2027/hvd.hnym3n.