This paper describes non-manifold topology (NMT) as it relates to the field of architecture and presents Topologic, an open-source software modelling library enabling hierarchical and topological representations of architectural spaces, buildings and artefacts through NMT. Topologic is designed as a core library and additional plugins to visual data flow programming (VDFP) software. The software architecture and class hierarchy are explained and two domain-specific demonstrative tools (TopologicEnergy and TopologicStructure) are presented to illustrate how third-party software developers could use Topologic to build their own solutions. The paper concludes with a reflection on the benefits and limitations of NMT in the design and simulation workflows and outlines future work.

**Keywords:** Non-manifold topology, Visual data flow programming, Building performance simulation, Structural analysis, Computational design, Building information modelling

**WHY NON-MANIFOLD TOPOLOGY**
While the field of topology is a vast and fascinating area of study in mathematics with precise concepts and terminology, in this paper we define terms and consider issues of topology and geometry narrowly and more simply as they apply to the field of architecture and computational design. Thus, our definitions of topological concepts will necessarily be less precise than those used by mathematicians, but more directly applicable to the field of architecture. The term ‘topology’ is derived directly from the Greek τόπος (place), and λόγος (study), and therefore can be defined as the study of place - or the study of space [1]. This is precisely our aim - mainly to enhance the representation of space in computational design systems through the use of topological concepts. Topology defines the relationships between entities. For example, two spaces can be thought as topologically adjacent, if they share a common face. In contrast to geometry, topology is concerned with the properties of space that remain constant when it is subjected to deformations. Yet, geometry and topology are fundamentally inter-linked. A geometric operation on an object fundamentally changes its topol-
ogy. For example, removing one outer face of a prismatic closed cell transforms it into an open shell and thus alters its topology.

**The difference between manifold and non-manifold geometry**

Computer-aided design (CAD), building information modelling (BIM), and parametric visual data flow programming (VDFP) software usually rely on low-level geometric engines (kernels) and software development kits (SDK). These geometric kernels are usually classified as manifold or non-manifold (Chatziwasileiadi, Wardhana, et al. 2018). Simply put, manifold kernels represent three-dimensional entities with a series of connected boundary elements that separate the outside world from its solid interior. Manifold entities (also called 2-manifold) with a boundary, such as a circle or a torus, can be unfolded into a continuous flat plane. In contrast, non-manifold Topology (NMT) entities (also called 3 or more manifold) such as a T-junction cannot be unfolded into a continuous flat wire or surface (see figure 1).

NMT can represent the space inside an object and allows subdivision of an outer boundary with inner zero-thickness boundaries. In addition, NMT allows entities with mixed dimensionalities to co-exist in the same entity. Manifold geometry kernels usually struggle to model and represent non-manifold entities and instead consider them modelling errors: “Some tools and actions in Maya cannot work properly with non-manifold geometry. For example, the legacy Boolean algorithm and the Reduce feature do not work with non-manifold polygon topology [...] Some types of polygon geometry will not work in Maya. Invalid geometry includes vertices that are not associated with a polygon edge and polygon edges that are not part of a face (dangling edges)” [2]. 3D manufacturing also struggles with non-manifold objects: “[...] Shapeways requires all objects to be 2-manifold. This means that each edge should be connected to exactly two faces. ‘Open’ objects are typically 1-manifold (or even 0-manifold for stray edges), models containing unwanted faces are 3- or more manifold” [3]. In addition, manifold kernels have traditionally focused on geometry far more than topology. These systems can rarely recognise, query and report on topological relationships and if they do, the processes and representations are particular rather than universal and ad hoc rather than formal.

![Figure 1](image.png)

At this point the reader may be wondering why anyone would use NMT if it causes such problems. The reason is that NMT provides many advantages over regular manifold modelling (Lee et al. 2009; Chang & Woodbury 1997; Nguyen 2011; Aish & Pratap 2013). As described earlier, NMT is well-suited to create a lightweight representation of a building as an external envelope and the subdivision of the enclosed space into separate spaces and zones using zero-thickness internal surfaces. Because NMT maintains topological consistency, a user can query these cellular spaces and surfaces regarding their topological data and thus conduct various analyses. For example, this lightweight and consistent representation was found to be well-matched with the input data requirements for energy analysis simulation software (Ellis et al. 2008; Jabi 2015).

As we will see later in the paper, because NMT allows entities with mixed dimensionalities and those that are optionally independent (e.g. a line, a surface, a volume) to co-exist, structural models can be represented in a coherent manner where lines can represent columns and beams, surfaces can represent walls and slabs, and volumes can represent solids. In addition, non-building entities, such as structural loads can be efficiently attached to the structure. This creates a lightweight model that is well-matched with the input data requirements for structural analysis simulation software.

In another paper by the authors we illustrated how NMT can represent a design envelope and popu-
late it with bespoke conformal cellular units in preparation for digital fabrication. Access to topology information allowed us to create and follow rules about the shape of and connection between deposited cellular unit to create a more efficient and better connected conformal cellular structure (Jabi et al. 2017).

Finally, we successfully used NMT to spatially reason about and evaluate the social sustainability of vernacular courtyard houses. In that research project, topology information allowed us to build dual graphs and conduct space syntax analysis efficiently using lightweight models. The software represented rooms as spaces with zero-thickness dividing surfaces and with embedded apertures such as door. We then used that information to create a shape grammar and a computational tool to design socially sustainable tall buildings (Al-Jokhadar & Jabi 2017).

**RATIOCINATION VS. FABRICA**

Designing, representing and reasoning about architectural space is one of the unique and defining properties of architecture (Ching 2014). Accounts from the literature indicate that buildings are often first conceptualised as a hierarchical sequence of related spaces (Curtis 1996). Only once this spatial arrangement has been defined does the focus shift to how these spaces will be realised by the use of physical building components. However, in BIM systems, the most prevalent approach is to represent a building as a collection of 3D solid models with each solid representing an individual physical building component (Attia et al. 2011). Each of these solid models uses manifold topology to define the boundaries that separate the external void from the internal enclosed volume representing the material content of the component. Modern BIM systems do not require nor advocate the creation of a conceptual spatial model as the basis of the building fabric model. Consider, for example, this first sentence of the Autodesk Revit tutorial on how to get started with building a BIM model: “Start with the general building components (walls, floors, roofs). Then slowly refine the design, adding more detailed components (stairs, rooms, furniture) as you proceed.” [4]. Architects are thus often asked to postpone the derivation of the conceptual spatial model and other representations to a later stage and from an overly complex fabric model to conduct tasks such as energy analysis, structural analysis, spatial reasoning, and fabrication planning. This process leads to difficulties, errors and an increase in time and effort.

Alternatively, one can follow a Vitruvian approach by starting with a topological model as a conceptual regulating skeleton that sets out the ratio-cination - the rational and theoretical setting out of principles that can then support the creation of fab-rica - the physicality of fabricating architecture (Pont 2005). Similarly, other researchers have emphasised the importance of thinking abstractly and strategically by using a controller (something that controls something else) and a proxy (something that stands in for something else) in parametric design software (Woodbury et al. 2010). These strategies ensure resilience within the system in the face of change; thus a design system that forgoes this strategic definition of topological relationships risks brittleness and failure in later design stages (Jabi et al. 2017).

Therefore, one of the primary motivations for this research is to rethink the BIM design process to focus first on building a conceptual model that can act as an ordering framework and support for further design development. In this process, an NMT conceptual model would serve as a defining model for a derived building fabric model. The NMT conceptual model would define both the spatial configuration and topology as well as provide a skeletal framework that can be ‘thickened’, either manually or using computational algorithms and rules, into actual building fabric components (Aish & Pratap 2013).

This is, obviously, not a new concept for designers many of whom implement this methodology effectively, but in a bespoke and ad hoc manner due to the lack of formal support in BIM systems. A case in point is the Aviva Stadium in Dublin, Ireland design by Populous (formerly HOK Sports Architec-
ture) from 2005 to 2007. This project exemplified a transitional period that saw a shift from computer-aided solids modelling to fully parametric design processes. Initially, the design was first studied through static 3D models using McNeel’s Rhinoceros platform [5]. However, the designers quickly realised that this is an unsustainable approach due to the time and effort needed to implement changes. The stadium design was re-considered as a simplified, conceptual and parametric system, using Bentley’s Generative Components software [6] (see figure 2).

The parametric model was made of three elements: “the footprint of the stadium, composed of eight tangential arcs; the plan of the inner roof or drip line, also composed of eight tangential arcs; and a radial structural grid that eventually became the supporting system of the stadium’s outer surface or skin.” (Jabi 2013). This controlling lightweight conceptual model, which could’ve been formalised as an NMT Cluster of vertices, edges, wires, and faces, was then shared by the architects and the structural consultants, through nothing more than a Microsoft Excel spreadsheet with an agreed format. The spreadsheet was then used as the basis to drive the solution of the structural model and the cladding system. In this example, the thickened fabric of the project was considered a derived model from the conceptual defining model of curve centrelines. The Aviva Stadium case study illustrates the need to formalise this inverted design process and enable designers to specify defining and derived models, and the relationships amongst them, more precisely and formally.

**NON-MANIFOLD TOPOLOGY**

Because NMT formalises the spatial relationships between the various entities in a model and describes how geometric entities are connected, this research hypothesises that NMT has potential to serve as the formal mechanism with which to build defining models and precisely translate them into derived building fabric models. This approach can serve a wide range of representations of architectural space with benefits for various design, analysis, and production tasks.

Based on a previous review published by the authors, a topological framework with the following eight entities, arranged from the highest level of dimensionality to the lowest, is proposed: Cluster, Cell-Complex, Cell, Shell, Face, Wire, Edge, and Vertex. Each entity may contain other lower-level entities - with the exception of a Cluster in which another Cluster can be contained (Chatzivasileiadi, Wardhana, et al. 2018).

In addition to allowing multiple faces to meet at an edge or multiple edges to meet at a vertex, coincident entities (within a user-specified tolerance) are merged and are ensured to be unique. Furthermore, imported mesh data, in standard vertex/index format, can be self-merged not only to remove duplicates, but to build the highest dimensional entities possible automatically. These entities share lower-dimensional entities where possible. This leads to efficient and consistent models that avoid the duplication problems found in regular manifold and polyhedral modelling that do not enforce such rules. For example, a mesh model (see left in figure 3) with 7 faces is converted into a non-manifold cluster object containing one (1) closed cell, one (1) open shell, nine (9) faces, nine (9) wires, nineteen (19) edges, and twelve (12) vertices (see right in figure 3). Any duplicated en-
tities are removed and topologically linked. Finally, notice that what used to be three (3) single rectangular faces (top and side faces) in the original mesh data are automatically segmented into two square faces each in the resulting self-merged NMT model.

NMT models can be combined and manipulated using regular Boolean operations. However, NMT Boolean operations extend the traditional regular operations of union, difference and intersection to include the following irregular Boolean operation: Merge, XOR, Impose, Imprint, Slice, Trim, and Unmerge (Aish & Pratap 2013). Generally, a regular Boolean operation removes any external faces of the input bodies that are within the resulting body, while a non-regular Boolean operation maintains these faces. As a result, regular operations lead to a manifold result, while non-regular operations lead to a non-manifold result.

Prior publications by the authors suggest a strong potential of using geometrical entities with NMT as a representation of architectural space that is also highly compatible with the input requirements of building performance simulation (BPS) engines, structural design, fabrication planning, and spatial reasoning (Jabi 2016; Jabi et al. 2017; Al-Jokhadar & Jabi 2017). The approach afforded by NMT provides topological clarity that has the potential to allow architects to better design, analyse, reason about, and produce their buildings.

TOPOLOGIC: A NON-MANIFOLD TOPOLOGY MODELLING TOOLKIT

This paper presents Topologic, an open-source software modelling library enabling hierarchical and topological representations of architectural spaces, buildings and artefacts through NMT. Topologic is designed as a core library and additional plugins to visual data flow programming (VDFP) applications (Hils 1992; Marttila-Kontio et al. 2009) and parametric modelling platforms commonly used in architectural design practice. These applications provide workspaces with visual programming nodes and connections for architects to interact with Topologic and perform architectural design and analysis tasks.

Software architecture

Topologic is implemented using a multi-layer software architecture (see figure 4). At the lowest layer, we use Open CASCAIDE, an open-source NMT geometry software development kit (SDK) that provides data structures and modelling algorithms for 3D solid structures [7]. We also use ShapeOp, an open-source SDK for surface planarization [8]. Classes and methods in these two SDKs are encapsulated in the second software layer containing the TopologicCore and TopologicSupport libraries, written in C++. TopologicCore implements the core topologic classes and methods using an object-oriented programming (OOP) approach while TopologicSupport provides added utilities as needed. Above this layer, we implemented an interface layer, written in the .NET C++/CLI language, that connects the core and support libraries to the host geometric editor or visual data flow programming application. At present, this layer (Topologic(VDFP)) has been written for Autodesk Dynamo software [9] and is thus named TopologicDynamo.

Work is underway to implement a version for McNeel Rhino/Grasshopper 3D [10] (TopologicGH) which will be reported on in future publications. Additionally, we envisage plug-in developers will use this layer to
develop domain-specific applications. By strongly separating the code written for different platforms, this architecture ensures high modularity and code readability. In addition, the software can be easily extended to other platforms by writing a small library using the platform's conventions in the upper layer to encapsulate the core library.

**Class Hierarchy**

TopologicCore contains the following main classes (see figure 5):

- **Topology**: A Topology is an abstract super-class that stores constructors, properties and methods used by other subclasses that extend it.
- **Vertex**: A Vertex is a zero-dimensional entity equivalent to a geometry point.
- **Edge**: An Edge is a one-dimensional entity defined by two vertices. It is important to note that while a topologic edge is made of two vertices, its geometry can be a curve with multiple control vertices.
- **Wire**: A Wire is a contiguous collection of Edges where adjacent Edges are connected by shared Vertices. It may be open or closed and may be manifold or non-manifold.
- **Face**: A Face is a two-dimensional region defined by a collection of closed Wires. The geometry of a face can be flat or undulating.
- **Shell**: A Shell is a contiguous collection of Faces, where adjacent Faces are connected by shared Edges. It may be open or closed and may be manifold or non-manifold.
- **Cell**: A Cell is a three-dimensional region defined by a collection of closed Shells. It may be manifold or non-manifold.
- **CellComplex**: A CellComplex is a contiguous collection of Cells where adjacent Cells are connected by shared Faces. It is non-manifold.
- **Cluster**: A Cluster is a collection of any topologic entities. It may be contiguous or not and may be manifold or non-manifold. Clusters can be nested within other Clusters.

Several other classes are being actively developed. These include, but not limited to:

- **Context**: A Context defines a topological relationship between two otherwise independent Topologies.
- **Graph**: A Graph is a Wire that is defined by the topology of a CellComplex or a Shell. It can be manifold or non-manifold. A dual graph is a good example of this class.

Written using the Object-Oriented Programming (OOP) paradigm, at the top level of Topologic’s class hierarchy resides the Topology class, which is inherited by other classes representing the topological entities. Methods written in these classes can be generally classified into four categories, namely constructors, queries, Boolean operations, and other entity-specific methods. Constructors are used to construct a topological entity from geometric entities or lower-level topological entities. Query methods retrieve one or more entities from another entity. Queries can be further categorised into three: upward queries, to retrieve higher-level entities which constitute the argument entity; downward queries, to retrieve the constituent lower-level entities of an entity; and sideways query, to retrieve adjacent entities on the same level. Boolean operations combine two entities into one in various ways and are written inside the par-
ent Topology class. Finally, classes may have methods specially written for them. For example, the Wire and Shell classes have methods to test if they are closed. In addition, the Face class has operations to attach multiple faces as apertures, which can be used to model glazing as an example.

It should be noted that, in large part, Topologic does not introduce its own entities or geometric computations. Instead, whenever possible, it relies on those either from the underlying SDKs or from the host application to ensure compatibility. In addition, the topological classes can be used to represent their geometry counterparts. For example, a Topologic face may actually represent a planar or an undulating NURBS surface trimmed by a wire, thus providing an abstraction for a wide variety of surfaces.

Despite TopologicCore being object-oriented in its design, Topologic follows the VDFP methodology to ensure compatibility with its host visual data flow programming workspace. In VDFP, a program is represented visually by a directed graph made of nodes and arcs that connect them. Nodes represent functions and arcs represent the flow of data between them (Hils 1992; Marttila-Kontio 2011).

Topologic methods are represented as visual nodes with input and output ports. Topologic entities are immutable at the host application levels. Thus, modifying the attributes of an entity after the fact, as is usually done in an OOP environment, is not possible. Instead, the user has to trace the constructor of the object and modify its input parameters or create a brand new deep copy of the entity with different attributes.

A Topologic node is designed to accept topological instances from the library and the native geometric counterparts from the host application. In addition, any node that produces a topological entity also has an extra output port that outputs the geometric counterpart to allow display using the native host application as well as additional workflows external to Topologic. These design principles guarantee effortless connection from and to indigenous nodes in a single workflow.

**DOMAIN-SPECIFIC APPLICATIONS**

As explained above, we envisage that our software will be used by plug-in developers to create domain-specific applications. To illustrate this process, we created two demonstrative applications. The first application, TopologicEnergy allows the user to quickly build models that can be sent to EnergyPlus (Crawley et al. 2000) for energy simulation using the OpenStudio toolkit (Guglielmetti et al. 2011). The second application, TopologicStructure demonstrates how a mixed-dimensional structural model can be created with structural loads applied to it.

**TopologicEnergy**

A comparative study with traditional workflows was conducted and reported in previous publications (Chatzivasileiadis, Lannon, et al. 2018) and thus will only be summarised here. The experiment analysed four pathways to the energy modelling of a building with relatively complex geometry including curved surfaces and bespoke glazing. From the four pathways explored, the NMT pathway using TopologicEnergy was able to model and handle complex geometry and produce reliable results, while benefitting from the advantages of NMT. As shown in figure 6, the workflow consisted of (a) modelling the external envelope and the glazing design on a flat surface, (b) mapping the glazing unto the curved wall, (c) subdividing and planarizing the wall and mapped glazing into a set of wall panels and windows, (d) slicing the building into multiple stories, and finally (e) sending the model to OpenStudio/EnergyPlus for energy analysis (Wardhana et al. 2018).

**Modelling the external envelope and the glazing design on a flat surface.** We start by creating a series of circular wires that are then lofted into a surface. This is converted to a Topologic face (see figure 6-1). The glazing is created as a rectangular face with internal wires that represent the glazing apertures (see figure 6-2).

**Mapping the glazing onto the curved wall.** The flat glazing design is sampled using a user-specified number of rows and columns and mapped onto the
UV parametric space of the curved wall (see figure 6-3).

Subdividing and planarizing the wall and mapped glazing into a set of wall panels and windows. The curved wall and glazing are then segmented into quadrilaterals using a user-specified number of rows and columns (see figure 6-4). The resulting mesh is then planarized using the ShapeOp library [8]. The glazing is further triangulated and scaled down slightly in keeping with the requirements of OpenStudio and EnergyPlus.

Slicing the building into multiple stories. A series of planes are created to represent floor slabs and converted into Topologic faces (see figure 6-5). Using the non-manifold slice boolean operation, the building is sliced into separate floors and a Topologic CellComplex is created (see figure 6-6).

Sending the model to OpenStudio/EnergyPlus. The Topologic CellComplex is queried for its constituent Topologic Cells, faces, and sub-faces and those are translated into OpenStudio spaces and thermal zones (see figure 6-7). The TopologicEnergy software then sends the resulting energy model to EnergyPlus and automatically triggers an energy simulation.

**TopologicStructure**

The second demonstrative application, TopologicStructure, allows the user to quickly build structural models and apply structural loads in preparation for structural analysis. The overall aim is to create a mixed-dimensional topological shape by a list of vertices and indices and apply structural loads to it. The overall workflow consisted of: (a) creating the model using an array of vertices and an array of vertex indices, (b) defining a list of locator points, and use them to find the closest simplest sub-shape of the model, (c) Apply various kinds of structural loads to these sub-shapes (see figure 7).

Creating the model using an array of vertices and an array of vertex indices. We start by creating a list of vertices, and a list of vertex indices (see figure 7-1). This is akin to creating a mesh from the same set of inputs. However, here we are creating a cluster of mixed-dimensional topology, consisting of columns, walls, and spaces. We save the vertices and vertex indices as CSV files and load them in Dynamo using its built-in CSV file reader. We then pass the list of vertices and vertex indices to the Topology.ClosestSimplestSubshape() node, to find which “simplest” sub-shapes are closest to the locator points (see figure 7-4). A sub-shape A is simpler than another sub-shape B if it has a lower dimension (e.g. a vertex is simpler than an edge, and an edge is simpler than a surface etc).

Selecting sub-shapes to which the loads will be applied. To identify sub-shapes to which we would like to apply structural loads, we use a collection of point locators that fall on or near the desired sub-shapes (see figure 7-3). We connect the constructed model and the locator points to the Topology.ClosestSimplestSubshape() node, to find which “simplest” sub-shapes are closest to the locator points (see figure 7-4). A sub-shape A is simpler than another sub-shape B if it has a lower dimension (e.g. a vertex is simpler than an edge, and an edge is simpler than a surface etc).

Applying structural loads to the model. In Topo-
logicStructure, we implemented a mechanism to apply loads to a topological shape (see figure 7-5). A single load is applied to a point location on an entity. A group of loads is called a LoadCluster, and it can be applied to an edge (linear loads) or to a face (surface loads). A LoadCluster is only valid if all of its loads are within the valid range of the edge/face, otherwise an error message is generated. Loads can have direction and magnitude and are applied to a parametric location on the target entity. For linear loads we use both a u translation and a u scale to apply the loadCluster to a region of the edge. Similarly, surface loads use a u and v translation, scale, and rotation applied to the loadCluster to locate it within a specific region of the target surface. Topologic models without forces (see figure 7-6) can then be combined with those that have forces applied to them (see figure 7-7). The resulting topological model is highly compatible with structural analysis software input requirements (see figure 7-8). We are currently implementing a link to such software and will report on our progress in a future publication.

Topologic and its associated tools are under active development. The core toolkit is designed to be platform-independent and allow third-party developers to build domain-specific software. TopologicEnergy and TopologicStructure, as described above, serve as demonstrative domain-specific applications for energy analysis and structural analysis that can act as templates for others to follow. We are in discussions with our industrial partners to integrate Topologic with their workflows and tools and will report on that in future publications.

CONCLUSION

Manifold modelling, while needed in later stages of design to create the models of the fabric and components of buildings, is too complex and cumbersome in the early stages of design and hinders the use of building performance simulation. Topologic aims to address these issues by introducing a rigorous class hierarchy and a set of methods that are able to manage both manifold and non-manifold topologies. The combination of non-manifold topology and a versatile 3D software kernel has the potential to provide a comprehensive solution for architects while maintaining design creativity and flexibility. The development of Topologic has made it clear that there is a need to further investigate and conduct user-testing of innovative methods for creating, displaying and interacting with geometric, topological data using advanced interfaces and information theory. Ultimately, our aim is to help architects and engineers to: “[…] build the lightest possible model using the least effort that gives the most accurate feedback about their design and engineering concepts” (Aish & Pratap 2013).

ACKNOWLEDGEMENTS

This research project is a collaboration between Cardiff University and University College, London and is funded by a Leverhulme Trust Research Project Grant (Grant No. RPG-2016-016). We would like to thank the National Renewable Energy Laboratory (Daniel Macumber), BuroHappold Engineering (Al Fisher), and Grimshaw architects (Radu Gidei) for their help with and contribution to this project.
REFERENCES


Ching, FDK 2014, Architecture: Form, Space, and Order, John Wiley & Sons


Curtis, WJR 1996, Modern Architecture Since 1900, Prentice Hall PTR


Jabi, W 2013, Parametric Design for Architecture, Laurence King Publishing


Woodbury, R 2010, Elements of Parametric Design, Routledge