ABSTRACT
This paper presents progress in the development of practical applications for graph representations of meshes for a variety of problems relevant to generative architectural design (GAD). In previous work (Nejur and Steinfeld 2016), the authors demonstrated that while approaches to marrying mesh and graph representations drawn from computer graphics (CG) can be effective within the domains of applications for which they have been developed, they have not adequately addressed wider classes of problems in GAD. There, the authors asserted that a generalized framework for working with graph representations of meshes can effectively bring recent advances in mesh segmentation to bear on GAD problems, a utility demonstrated through the development of a plug-in for the visual programming environment Grasshopper. Here, we describe a number of implemented solutions to mesh segmentation and transformation problems, articulated as a series of additional features developed as a part of this same software. Included are problems of mesh segmentation approached through the creation of acyclic connected graphs (trees); problems of mesh transformations, such as those that unfold a segmented mesh in anticipation of fabrication; and problems of geometry generation in relation to a segmented mesh, as demonstrated through a generalized approach to mesh weaving. We present these features in the context of their potential applications in GAD and provide a limited set of examples for their use.
INTRODUCTION
In previous work (Nejur and Steinfeld 2016), a thorough review of
text of computer graphics (CG) revealed a number
of approaches to the segmentation of meshes that suggested
fruitful application in generative architectural design (GAD).
Examined individually, most of the approaches described there
could be found either in basic graph theory (Skiena 1998) or
in stand-alone tools already developed for CG. From these
precedents, the authors outlined and implemented a software
framework that targets applications in GAD. Other than the
synthesis of otherwise disparate routines in an environment
accessible to architectural designers, what sets this framework
apart is the modular versatility and customization that it affords.
This modular approach allows for a variety of possible work-
flows, and for multiple routines to be customized, juxtaposed,
and chained to create entirely new functionalities. Expanding
upon this previous work, this paper describes some of the
novel applications enabled by this characteristic of the frame-
work, and presents a selection of approaches that immediately
emerge from it. We conclude that the marriage of a mesh with a
weighted graph representing the dual of this mesh holds utility
beyond the initial purpose for which it was designed (mesh
segmentation), and may be employed to address a variety of
problems relevant to GAD.

In the pages that follow, we first discuss the unique features of
user interface and interaction that are required by the framework
approach, and that allow users to access the low-level data struc-
tures in a manner that may be easily understood by a community
of users with little direct experience with graph representations.
Next, we detail segmentation workflows for generating acyclic
connected graphs (trees) on meshes—a structure encapsulated
by a data type termed MeshGraph. These workflows lie at the
heart of the implemented tools, and comprise the core function-
ality for which our framework was originally developed.

Although mesh segmentation was the intended application
of this framework, the remainder of the paper demonstrates
the unexpected utility of the MeshGraph structure. We begin
by describing a range of geometric transformations of meshes
that rely on the pre-existence of related trees. These include
unfolding, as well as a number of other routines that anticipate
the needs of architectural fabrication. Finally, we present some
recent work on geometry generation in relation to meshes that,
while not anticipated at the outset of the development of the
tool, is suggestive of a promising territory for future work. Before
detailing the work performed in this scope, in the section below
we briefly recap some important concepts, terms, and data types
defined in a previous scope, and that continue to hold relevance
here.

Recap of Concepts and Terms

The dual graph is a concept central to graph theory, and is the
central operation of mesh segmentation using graph techniques.
On a triangular mesh, each face of the mesh becomes a node in
the graph, and each non-naked edge of the mesh becomes an
edge in the graph. The dual graph concept is implemented by Ivy
as a data object called MeshGraph.

A weighted graph is one in which nodes and/or edges are
assigned numeric values that are interpreted in cost func-
tions. All of the mesh segmentation routines discussed here
rely upon routines for determining node and edge weights in
specific configurations. In Ivy, there is a dedicated tool group
that contains routines for assigning and manipulating weights,
including assignment via dihedral angle, distance between face
center points, face size, and mesh color.

In graph theory, a tree is a special kind of graph that is both
directional and acyclicly connected, which is to say that any two
nodes are connected by exactly one path. This is a useful prop-
erty in GAD, in that any mesh dual that holds the properties of a
tree may be unfolded in a straightforward manner.

Borrowing from techniques in CG, segmentation routines in Ivy
are described as the process of converting a weighted dual graph
into a single tree or a “forest” of tree graphs in relation to a mesh.

USER INTERFACE AND INTERACTION

Leitão, Santos, and Lopes (2012) observed that within the GAD
community, frameworks that allow access to low-level controls
are preferred over the “black boxes” of packaged software tools
or routines. This is due to the nature of the early stages of the
design process, in which techniques and approaches are revised
often, and multiple algorithms are often tested in combinations
that are difficult to be anticipated in advance. While a frame-
work approach offers significant advantages in this regard, it also
brings to bear a number of demands concerning user interface
and interaction. Prominent among these is the need to clearly
communicate and offer access to low-level data. In Grasshopper,
this happens by default for most geometrical data. For custom
data, however, visual information needs to be extracted by the
designer at every step and converted into visible geometry for
feedback. Ivy addresses this issue by providing visual repre-
sentations of important lower-level information, including the
weights of a MeshGraph, and of the spatial transformations of an
unfolding routine.

MeshGraph Visualization

To assist the user in understanding the nature of the mesh-to-
graph relationship, Ivy provides an enhanced preview for the
MeshGraph custom data object. This expands the normal mesh preview from Grasshopper with additional data. On top of the base mesh, a set of polylines and points are displayed that correspond to the geometric position of the graph edges and nodes. The graph preview offers modes that can graphically depict the weight of edges/nodes or the depth of a single element in a tree. In this way, a weight landscape of a mesh or the depth of a tree can be intuitively understood before these structures are put to use.

In addition to the visualization of a MeshGraph, a special component related to a mesh unfolding workflow (discussed below) offers the ability to animate the unfolding process in two different ways: coefficient-based and step based. The coefficient-based animation varies the angle used to rotate each mesh face using a slider. The step-based animation unfold applies transformations in a limited number of steps, starting from the root of the graph and proceeding toward the leaves. These two methods may be used in combination, which allows the designer to trace back and understand the relationship between fabrication data and a mesh form.

**Basic Segmentation Routines**

The largest group of routines in Ivy concern mesh segmentation, a process that includes graph processing, the assigning of edge and/or node weights, and the application of a weighted dual graph in order to identify one or more minimum-spanning trees on a mesh. As a brief overview of how rudimentary MeshGraph segmentation works was presented in a previous scope of work (Nejur and Steinfeld 2016), here we will discuss and classify the segmentation processes from the perspective of the expected outcomes they produce. All segmentation algorithms are housed in one of four sections on the Grasshopper ribbon, and may be chained in a variety of configurations in order to produce a wide variation of behaviors and results. The section names listed below are illustrative of the tools they contain:

- One-Step Segmentation components support instant MeshGraph slicing and segmentation through tree graph spanning.
- Two-Step Segmentation components support further decomposition of the spanned tree.
- Iterative Segmentation components support loop routines designed to tackle difficult-to-define or ambiguous segmentation criteria
- Special Segmentation components provide production-oriented routines that use data from other segmentations and validate it for other scopes, such as fabrication.

Besides the segmentation outcomes that may be directly related to architectural fabrication (as discussed in a section below), we may observe that the basic segmentation routines enabled by the modular nature of the Ivy framework allow for a number of other useful outcomes. Prominent among these are **feature extraction and separation**, and **feature reduction or shape simplification**. The former concerns the identification and extraction of important geometric surface features—such as finding and separating finger from hand, or limb from body—while the latter relates to reducing the number of polygons while maintaining quality. While neither of these applications (demonstrated in the nearby images) are novel, especially in the context of CG (Shamir 2008), neither have they been thoroughly considered in a GAD context, where they hold potentially significant implications for linking fabrication concerns to design models.

**TREE GENERATION AND SEGMENTATION**

The dominant approaches to mesh segmentation in CG center on the definition of acyclic connected graphs, otherwise known as “trees.” Just as the defining of a weighted graph dual of a mesh (and the later reconfiguration of this graph as a tree) forms the common basis of many of the routines surveyed from CG, so too do processes figure prominently in the core functionality of the Ivy plugin. In this section, we describe workflows for the generation of trees in Ivy in the service of mesh segmentation.

Another area where segmentation tools can bring added value is in topology exploration. A number of graph-spanning tools are implemented in Ivy (such as minimal path) that, in conjunction
with an intelligently defined weight landscape, can reveal otherwise imperceptible information about the configuration of a mesh, as demonstrated by the nearby image.

Agent Segmentation
As discussed elsewhere (Nejur and Steinfeld 2016), the base data object in Ivy is the MeshGraph, a data object that adds information to the classical Rhino/Grasshopper mesh, and is primarily used for segmentation via the generation of trees following work in CG. However, other approaches to segmentation may take advantage of this same structure. Here we describe such an alternative approach that employs a multi-agent system.

In Ivy, MAgents are extensions of MeshGraph objects that include behavioural traits and awareness towards other nearby agents on the same mesh. Technically, the MAgent object stores a reference to a MeshGraph segment as a member, as well as a reference to an unsegmented MeshGraph, and miscellaneous other routines for tracking graph nodes already processed by other MAgent instances. This structure allows for segmentation behaviours beyond those that directly follow a weight landscape. At the time of writing, two base behaviours have been defined in Ivy, an explore behavior and a consume behavior, as discussed below. Further, an API has been provided to allow for the bespoke definition of additional behaviors.

The two base behaviours of the MAgent object are based upon a simplified model of slime mold (Physarum polycephalum) growth. The explore behaviour of an agent extends with one node of the MeshGraph segment, starting from each leaf of the segment. Here, growth is attempted in divergent directions so that the growing tendrils do not intersect. The consume behaviour works in a similar way, but its marginal nodes expand at each step in all available directions, thereby producing a blanket growth effect similar to a weighted breadth-first search. Any combination of MAgents may be deployed simultaneously or in succession, where each agent has a lifespan and new agents are born after some expire. All behaviours respect the weight landscape and can be constrained to arbitrary weight limits set per agent, a feature that adds yet another layer of variation to the possible agent segmentation outcomes.

The set of tools packaged with Ivy contains two compiled components that work with MAgent construct, but are intended as examples of simple behaviours. The real potential of agents comes from the Ivy API, however, as this approach permits complex chains of behavioural decisions and even the addition of new behaviours.

GEOMETRIC TRANSFORMATIONS BASED ON TREES
Unfolding polygon meshes using acyclic connected graphs (trees) demonstrates that this data structure holds utility beyond mesh segmentation, and is one of the main applications for which Ivy was designed. The aim of the routines described in this section is the generation of meaningful two-dimensional fabrication data from any given three-dimensional polyhedral two-manifold mesh with a reasonable number of faces. In Ivy, unfolding is described as an extension of the segmentation workflow, and may be applied following any segmentation described above.
Unfolding Routines

To perform a dedicated unfolding of a mesh without prior segmentation, the only prerequisite is the construction of a spanning tree. As detailed in Nejur and Steinfeld (2016), although this step may involve graph-edge slicing, it does not necessarily qualify as segmentation, because the cuts do not produce discrete pieces. Instead, this process merely ensures that the MeshGraph is acyclicly connected, which is to say that there are no circuits in the graph, and that between any given pair of nodes there is only one possible connection route. Since this is the only condition, a simple breadth-first search algorithm may suffice to prepare a mesh for unfolding. Given this simple prerequisite, even if there is no other preparation for the flattening of a given mesh, the implemented unfold component in Ivy can handle a range of potential problems, and a usable unfold is reliably produced regardless of the previous segmentation strategy.

The process of unfolding is as follows: starting from the leaves of the graph tree, each leaf mesh face is rotated about its edge toward the neighbouring face in order to bring the two into alignment; the two faces are then rotated about the edge of the next connected face; this process repeats until the root of the graph (or another strand of already unfolded faces) is reached. Even though any polyhedral shape expressed as a tree graph is unfoldable, it is rare to happen upon a case in which an unfold with all the properties required for fabrication are met: almost all trees become overlapped in their unfolded state. To minimize such cuts, and maximize segment size, the algorithm splits the tree as far away as possible (measured topologically) from the colliding faces. In tree topology terms, this is at the earliest common ancestor of the colliding faces. This approach to addressing unfolded collisions offers the useful property of resolving multiple overlaps simultaneously.

While the above process will suffice for a rough unfold, as established by Pottmann et al. (2015) and Attene, Falcidieno, and Spagnuolo (2006) through primitive-based segmentation, a weight-based tree segmentation is much more appropriate for a controlled unfold. Ivy offers a combination of these approaches: through a weighted guidance of tree growth, and/or secondary segmentation in a subsequent step, edges or edge chains that might produce overlaps can be identified and removed from the start. This results in less splitting to be handled by the unroll routine, and a more predictable set of resulting flat segments.

Any discussion of segmentation for fabrication must address the topic of stripification. Stripification, a process by which single face mesh strands are identified, is generally regarded as one of the best ways to decompose a mesh while avoiding overlaps in the unfolded state. Ivy implements research presented in Taubin and Rossignac (1998) as an orange-peel algorithm, and also accommodates agent behaviours. This process is similar to the approach taken by work of Anders Holden Deleuran (2015) at CITA and others (Fornes 2014). In Ivy, this well-worn approach is described as a specific case of general segmentation: a strip-like segmentation may be achieved by constructing a tree (via any
means described above, including agent segmentation or the Kruskal-Valence algorithm) with no branching permitted.

**Anticipating Fabrication: Flaps**
The MeshGraph data type is designed to maintain an active connection between the start and end states of the working geometry. Because of this, even in the linear environment of a parametric model such as Grasshopper, it is possible to define auxiliary geometries related to the original mesh, and to carry these geometries through to the flat fabrication data created by the unfold.

The first and most elemental application of this feature may be observed in the SimpleFlap tool. This tool creates glue flaps or tabs for each edge of the segmented mesh. Since this tool is designed for ease of use, and does not offer many variations from the standard tapered shape of the glue flap, a second component has been developed that generalizes the definition of a flap to include any planar geometry. The Custom Flap tool allows for the use of any set of planar curves as flaps, a feature that accommodates for any number of fabrication strategies, including riveted connections, snap connections, and entangled connections. Because each cut edge receives separate left- and right-hand flaps, the assembly strategy can be tuned according to specific geometric traits, such as the angle of incidence between connected faces. In this way, the Ivy unfolding routine can produce connection mechanisms that are responsive to fabrication material and local geometry.

**Unrolling Additional Geometry**
Just as we are able to define auxiliary geometry related to each unfolded edge of a mesh, so too can we define such geometry related to each mesh face. Like the definition of flaps, this functionality holds ramifications for problems in fabrication and assembly.

The MeshNode data object is capable of storing arbitrary planar curve geometry that may be related to a mesh face, and, when unrolled, may be subsequently transformed together with the underlying mesh until it finds a final flat state. This
functionality holds relevance in GAD for two reasons. First, it enables enhancements related to fabrication, especially if used in conjunction with the custom flap component. Unrolled additional geometry may be used in conjunction with flap geometry while also extending beyond the contour of the related face—a useful feature when designing for assembly. Second, since the addition of a layer of geometric information to unfolded mesh models does not increase the number of faces transformed, certain geometric routines may be handled without the additional calculation.

GEOMETRY GENERATION BASED ON TREES
As demonstrated in the section above, acyclic connected graphs (trees) related to polygon meshes hold utility beyond mesh segmentation, and may be effectively applied in the service of certain classes of geometric transformation relevant to GAD, including unfolding. This data structure holds still further ramifications for GAD. In this section we discuss the use of the MeshGraph type in two applications concerning geometry creation: mesh weaving and mesh creasing.

General Mesh Weaving
The generation of geometry through a weaving process on arbitrary two-manifold polygon meshes is a well-researched topic in CG. Recent advances have been made in this area (Xing et al. 2010; Akleman et al. 2009), some of which have begun to address concerns of fabrication at architectural scales (Xing et al. 2011). The approach taken by many such researchers relies on mesh face or mesh edge subdivision and projection on individual planes. The produced subdivisions in a second step can be joined into strands of weaved geometry. Our approach to mesh weaving, although also based also on graph theory, is fundamentally different. The weaving enabled by Ivy is not the result of subdivision and projection; rather it is more closely related to analog weaving processes.

In the first step, two separate segmentations are created on a given mesh, each of which is grown based on a different weight landscape. Ideally, these weight landscapes should be near to the inverse of one another. Alternatively, the second segmentation could be created by using the cut/uncut state of the graph edges from the first. Next, the segmentation geometry is modified in order to allow the individual geometries to negotiate one another without collision. Here, each graph node becomes an offset of the original mesh face, and each graph edge produces a mesh quadrilateral that connects the offset to the original mesh edge. This step is similar to an established technique (Hernandez et al. 2013), but is employed here for a different purpose. Finally, we walk the individual segmentations of the graph, offsetting individual faces either up or down along the normal based on the cuts found in the two trees. Essential in this process are the individual segmentations of the base graph, and in turn, the two weight landscapes that drive them. The weaving is designed to work with singular spanning trees over the whole graph, or with segmentations that result in multiple smaller trees (or stripes). When the weaving process is complete, MeshGraphs are created using the entangled meshes that result—MeshGraphs that can later be unfolded using the processes described in a section above. Since fabrication of such a weave requires entangling the two unfolded strips, to facilitate assembly two pieces of
information are carried forward from the original MeshGraph: a reference from woven node to the original node, and a flag indicating the relative position (up or down).

**Mesh Crease and Structural Support Generation**

Ivy offers two other routines for geometry generation based on trees. As in our approach to general mesh weaving, these routines rely on mesh segmentation through spanning tree graphs, but are applied in the service of outcomes that address concerns unique to GAD. While these tools are less developed than others described above, they begin to suggest the sort of applications that are possible when specific problems in architecture are addressed using the MeshGraph data type.

The Crease Mesh tool uses the tree graph(s) resulting from a segmentation and produces subdivisions in the base mesh by adding vertices in the middle of each face and in the middle of each edge that finds a correspondence in the graph. These new vertices are moved some distance along the face normals and averaged edge normals. The final offset values applied to the new vertices are calculated using the tree graph hierarchy, as well as a linear variation of the numbers between the provided root and leaf values. The result resembles a set of folded creases (or ridges) in the original mesh, and suggests a flow or pattern of erosion on a topographical surface. We speculate that this tool might offer an ability to add selective structural reinforcement to certain areas of an architectural form.

The Graph Structure tool uses the same principles, but instead creates a network of lines that start on the mesh, though are offset from it by some amount. The offset is calculated, as with the crease component, in a linear fashion as a variation from the specified root value to the given leaf value. The result is a structure that we speculate could work as a load distribution system based on the position of a node in the hierarchy of the tree graph. Depending on the segmentation of a mesh, multiple supporting structures could be generated using this tool. Alternatively, by using the same segmentation, multiple versions of a structure could be found simply by changing the root of the tree graph.

**CONCLUSION AND FUTURE WORK**

This paper has detailed a number of recently developed features of Ivy, a software framework for working with acyclic connected graphs (trees) related to polygon meshes—a structure encapsulated by the MeshGraph data type. While this framework was first conceived to address problems in mesh segmentation, we have demonstrated here that the approach of describing and manipulating polygon meshes through weight landscapes expressed on their dual graph holds utility beyond segmentation,
and extends into a number of applications relevant to generative architectural design.

We envision future work to extend in two directions. Foremost, as the Ivy tool has remained in beta development for the duration of this research, we recognise the need to validate the expected utility of many of these methods, and to calibrate the routines to the needs thereof. Additionally, we see potential in extending the functionality of the features mentioned above in one area in particular: the generation of geometry based on trees.

Validation of the research presented here against the requirements of architectural practice promises to provide a host of opportunities for further development. Most immediately visible challenges include refining the workflow for defining custom flaps and connections, and in producing custom scoring lattice hinges. Further, since at present the folding routines in Ivy hold utility for working with relatively thin inelastic materials, we see a number of opportunities for working in contexts that challenge these limitations, and that require closer connections with 1:1 prototyping. Thin sheet materials, while rigid, fold easily by hand or with minimal mechanical assistance. Thicker sheet material would require a modification of the cutting patterns in the unfolding algorithms currently available in Ivy, for example the use of lattice hinges.

Further, such modifications could enable a connection between the desired three-dimensional form and the creation of the cut pattern. Because this future work is closely tied to material research, these particular avenues of development will require physical prototyping. As a result, we see potential in workshops or pavilion installations that will serve to test these ideas.

Another direction that we expect to be fruitful is the elaboration of tree-based geometry generation to better address a range of applications. These include an expansion of the weaving algorithm to include partial weaving strategies, support for non-manifold or pseudo non-manifold meshes, and, in particular, a potential double use of the assembly flaps that could provide structural stiffness. As described above, the Custom Flaps component of Ivy allows for arbitrary two-dimensional curves, a feature that enables a range of flap strategies (such as glue flaps, dry entangled flaps, riveted flaps, and snap joints). This feature could be extended to allow flaps to serve a structural purpose, thereby enhancing surface stability and overall structural stiffness.

REFERENCES


**IMAGE CREDITS**
All drawings and images by the authors.

__Andrei Nejur__ is an Assistant Lecturer in the Architecture Department at the Technical University of Cluj-Napoca.

__Kyle Steinfeld__ is an Assistant Professor specializing in digital design technologies in the Department of Architecture at the University of California, Berkeley.