Ivy

Bringing a Weighted-Mesh Representation to Bear on Generative Architectural Design Applications

Andrei Nejur
Technical University of Cluj-Napoca

Kyle Steinfeld
University of California, Berkeley

1

ABSTRACT

Mesh segmentation has become an important and well-researched topic in computational geometry in recent years (Agathos et al. 2008). As a result, a number of new approaches have been developed that have led to innovations in a diverse set of problems in computer graphics (CG) (Shamir 2008). Specifically, a range of effective methods for the division of a mesh have recently been proposed, including by K-means (Shlafman et al. 2002), graph cuts (Golovinskiy and Funkhouser 2008; Katz and Tal 2003), hierarchical clustering (Garland et al. 2001; Gelfand and Guibas 2004; Golovinskiy and Funkhouser 2008), primitive fitting (Athene et al. 2006), random walks (Lai et al.), core extraction (Katz et al.), tubular multi-scale analysis (Mortara et al. 2004), spectral clustering (Liu and Zhang 2004), and critical point analysis (Lin et al. 2007), all of which depend upon a weighted graph representation, typically the dual of the given mesh (Shamir 2008). While these approaches have been proven effective within the narrowly defined domains of application for which they have been developed (Chen 2009), they have not been brought to bear on wider classes of problems in fields outside of CG, specifically on problems relevant to generative architectural design (GAD).

Given the widespread use of meshes and the utility of segmentation in GAD, by surveying the relevant and recently matured approaches to mesh segmentation in CG that share a common representation of the mesh dual, this paper identifies and takes steps to address a heretofore unrealized transfer of technology that would resolve a missed opportunity for both subject areas. Meshes are often employed by architectural designers for purposes that are distinct from and present a unique

1 The Elephetus project by Anders Holden Deleuran (CITA/KADK) and David Reeves (Spatial Slur/ZHA Code) offers an example application of mesh segmentation in generative architectural design. Note that while this project exhibits many of the design qualities intended to be supported by the software proposed in this paper, and its development likely required some of the processes discussed here, the Ivy tool was not employed.

set of requirements in relation to similar applications that have enjoyed more focused study in computer science. This paper presents a survey of similar applications, including thin-sheet fabrication (Mitani and Suzuki 2004), rendering optimization (Garland et al. 2001), 3D mesh compression (Taubin et al. 1998), morphing (Shapira et al. 2008) and mesh simplification (Kalvin and Taylor 1996), and distinguish the requirements of these applications from those presented by GAD, including non-refinement in advance of the constraining of mesh geometry to planar-quad faces, and the ability to address a diversity of mesh features that may or may not be preserved.

Following this survey of existing approaches and unmet needs, the authors assert that if a generalized framework for working with graph representations of meshes is developed, allowing for the interactive adjustment of edge weights, then the recent developments in mesh segmentation may be better brought to bear on GAD problems. This paper presents recent work toward the development of just such a framework, implemented as a plug-in for the visual programming environment Grasshopper.

INTRODUCTION

This paper describes the motivations for the development of a platform for mesh segmentation suited for the requirements of contemporary generative architectural design (GAD). The context for this endeavor is the increased relevance of mesh-based form-finding and simulation techniques in architectural design and the maturation of programing and visual scripting, as the related software tools find more widespread use. In recent years, a number of new techniques for form-finding via mesh-based simulation have taken hold in GAD. These include spring-based physical simulation models, node-based structural simulations, and thermodynamic analysis. Such tools have increased the demand for approaches to mesh creation and manipulation, with meshes beginning to even challenge the relevance the now-dominant non-uniform rational Basis spline (NURBS) surface representation in GAD. It is also notable that the increased relevance of each of these examples has been enabled by the advent of visual programming, and by the Grasshopper programming environment in particular. While meshes have become more widespread for this audience, approaches to mesh segmentation are not well-studied in the context of GAD, nor are they well-supported by existing tools. This research seeks to identify existing relevant techniques in computer graphics (CG), adapt these techniques to the unique needs of the generative architectural design audience, and to produce a framework for their transfer such that they may be effectively applied in this new domain.

The first part of the paper reviews the relevant literature in both CG and GAD. It first presents an abbreviated survey of technical approaches to mesh segmentation in the context of CG, including a discussion of the applications for which these approaches were developed. As will become apparent, many of the relevant approaches rely on a common representation of the mesh dual, and proceed through the manipulation of a weighted graph. The common use of the weighted mesh dual allows for a number of complementary and overlapping approaches to be implemented via a generalized approach, and forms the basis of the development of the software framework proposed below. The same section details the extent to which mesh segmentation represents an unmet need in architectural design, and would benefit from the algorithms developed for CG applications. Several sympathetic approaches like weaving (Xing et al. 2011a) and mesh stripification (Xing et al. 2011b) are presented here as a survey of architectural projects that have employed mesh segmentation without the benefit of a supportive toolkit. From this survey, the requirements for mesh segmentation that are unique to GAD are derived, and an account of the specific tasks in design that would benefit from mesh segmentation is presented. The second part of the paper presents the methods by which the proposed software framework has been developed. First, the implementation details of the software tool created for mesh segmentation and fabrication are presented. The modular workflow of Ivy for Grasshopper is explained alongside the data structures and algorithms employed. A few typical workflows are detailed in this section, which, given the modular nature of the software and the common representation of the generalized weighted-graph representation, may be combined in a number of ways. Following this, we speculate upon the advantages of this modular technique in connection to the specific needs of GAD practice and research.

Mesh Segmentation in Computer Graphics

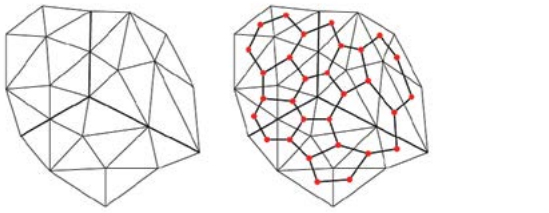
The boundary representation of the three-dimensional mesh has practically been a steady companion for the digital embodiment of form since the advent of computer generated imagery. In order to make use of this geometric data type on the computer screen, ways to meaningfully depict an otherwise featureless collection of mesh faces had to be devised. Among those, mesh segmentation stands as one of the most important. Its applications span the entire spectrum of mesh use in CG: mesh interpretation, feature detection, parametrization, multi-resolution modeling, mesh editing, morphing, animation, and compression all rely on some form of mesh segmentation to exist (Shamir 2008). Different applications that carry distinctive requirements have prompted a number of distinct techniques of mesh segmentation that have been well-articulated in previous work (Agathos et al. 2008; Shamir 2008; Chen et al. 2008). The first part of the literature review to follow functions as a meta-survey of these techniques, extracting the common

representations and procedures that suggest appropriate transfer to GAD.

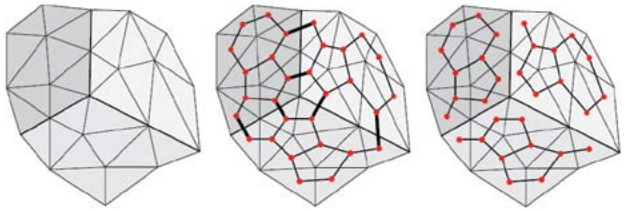
Mesh Segmentation in Generative Architectural Design

The architectural use of discrete surface descriptions in the service of form generation precedes the invention of the computer. As far back as the beginning of the last century, Antoni Gaudí used discrete surface representations in the physical computation of form. Later, Frei Otto brought the physical equivalent of a three-dimensional mesh to bear on the elaborate form-finding techniques that structured much of his design research. Early applications of the computer in GAD remained limited to academic research contexts, and regularly employed the mesh representation, as there were few other options for the representation of three-dimensional free-form geometry prior to the development of NURBS. With the advent of commercially-available CAD platforms, GAD transitioned from academic labs to applications in practice. As design practitioners were less likely to develop bespoke geometric routines than CAD researchers, many of the technologies employed were direct transfers from CG. This resulted in the occasional mismatch between the audience for which these technologies were developed and the manner in which they found application in GAD.

For example, a central concern (Pottmann et al. 2015) of GAD is free-form surface rationalization, a broad topic that includes panelization, surface approximation, and constraint-aware design. As such, those mesh operations destined for use in GAD must account for a host of additional properties directly derived from their eventual existence in the physical world. These concerns are qualitatively different than those applications most often targeted by CG, even those applications that involve physical fabrication. Geometric properties such as face size, dimensional proportion, an ability to produce offsets, number of faces, fairness, singularities, and node valence can often be overlooked by applications in CG, but have a tremendous impact in their architectural applications. As comprehensive surveys of mesh use in free-form architectural design have been well-articulated in previous work (Pottmann et al. 2015; Glymph et al. 2004; Liu et al. 2004), in the literature review to follow, we focus on the geometrical and topological traits of meshes that are most relevant to the present research. In summary, this survey finds that mesh segmentation remains relevant in the context of GAD. Whether the design manifests smooth surfaces or discrete collections of polyhedral flat surfaces, the ability to rationalize this geometry—that is to say, to reasonably realize the geometric design within the limits posed by a given a method of fabrication—is paramount. As is presented below, the present research relies heavily on the strong body of research dedicated to the panelization of architectural surfaces, and to the modeling of



2



3

2 Creating a dual graph on a triangle mesh. Each face becomes a node in the dual graph and each non-naked edge becomes an edge in the same dual graph.

3 A weighted graph diagram and its use in dual graph/mesh segmentation.

4 A simple diagram of the Kruskal's algorithm, the basis of the HMC.

5 Diagram of basic K-Means algorithm steps. 1) Select roots and calculate regions based on weight of graph edges. 2) Relocate roots to central points of regions. 3) Recalculate regions based on new roots.

meshes with fabrication-aware constraints (Glymph et al. 2004). Also brought to bear are the most recent developments related to the rationalization of meshes for fabrication, most notably related to papercraft (Mitani and Suzuki 2004), that suggest application in GAD. Taken together, these two surveys will allow us to discern the need for and provide the basis of a generalized framework for mesh segmentation in GAD.

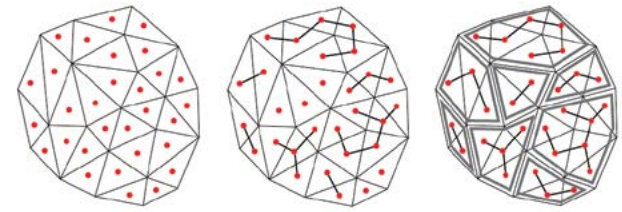
LITERATURE REVIEW

By surveying the most relevant approaches in CG, this section characterizes the state of the art in mesh segmentation from technical and theoretical points of view. Then, by surveying the existing use of meshes for surface rationalization in GAD, we articulate the requirements of the appropriate transfer of mesh segmentation techniques from applications adjacent to CG to those relevant to GAD. A number of algorithms are presented that were originally developed in CG and that have subsequently been adapted by the current scope of work. The expression of graph theory in each of these cases is highlighted, as the shared reliance on edge-weighted graphs forms the basis of the common framework proposed below.

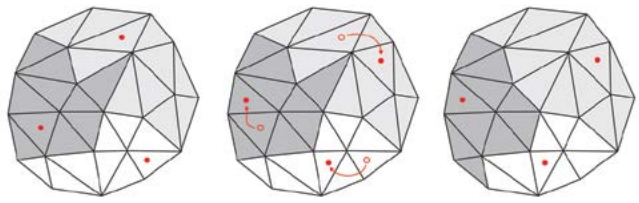
Survey of Mesh Segmentation Techniques in CG

While specific variants of routines for the segmentation of

meshes abound in computer science research—a testament to the widespread utility of segmentation in general—only a limited number of algorithms have achieved a strong status, variations of which recur often in the most current approaches. In the meta-survey conducted here, 45 papers were examined that either survey segmentation techniques in general in order to characterize the state of the art, or that compare two or more techniques in the context of a specific application. Nearly all those techniques surveyed implement a weighted-graph representation, and an overwhelming number relate strongly to applications in computer graphics, with the techniques demonstrated very often adapted to the specific needs of a relatively narrow set of concerns. As the common foundation of these techniques informs the development of the toolkit proposed below, thereby allowing for the coordinated application of different algorithms in the context of GAD, we highlight here the way in which each technique manifests a weighted graph representation. To illuminate the mesh segmentation algorithms discussed below, we employ a common language and notation for each routine presented. For the sake of clarity, some notions in the referenced papers will be renamed to fit this unifying convention. Among the central concepts are the dual graph and the weighted graph, which we define here. A *dual graph* (Figure 2) is a concept central to graph theory, and is the central operation of mesh segmentation using graph techniques. In the context of each example below, the faces of the mesh form the nodes of the graph, and the bounding edges between faces form the edges of the graph. A *weighted graph* (Figure 3) is one in which nodes and/or edges are assigned numeric values that are interpreted in cost functions. The "weight" or "cost" associated with a graph element is used in order to direct a walk on the graph, and thereby to prioritize certain paths over others. Any number of processes are used to map costs to elements. Only rarely would such values be set directly by an end-user. More often, specific processes are directed by the algorithm and rely on information found in the geometry of the mesh. In most of the algorithms examined here, the determination of weights is "hardwired" into the algorithm.



4

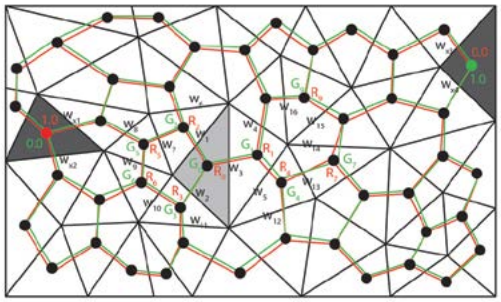


5

Hierarchical Mesh Clustering (Garland et al. 2001)

Hierarchical mesh clustering (HMC) (Figure 4) is among the simplest of the algorithms in this survey, and, like the others presented here, operates on the dual graph of a mesh. HMC applies a greedy clustering routine based simply on edge weights determined by planarity. The algorithm is a straightforward and relatively simple interpretation of a standard Kruskal or Disjoint Set minimum spanning tree algorithm on a graph (Skiena 1998). The most important difference from these standard routines is the iterative recalculation of the cost for the edges at each step. First, a Disjoint Set approach is employed, where the dual graph in the first step is atomized into its connecting nodes. In this step, each cluster has only one node. In subsequent steps, using a greedy approach, one edge at a time is contracted and the clusters separated by the edge are merged into a single cluster. As this proceeds, the algorithm evaluates each edge in a greedy fashion based on the cost of contraction. The cost of each edge is calculated at each step, based on a planarity measure function of the cluster post-merger. In a variant of this approach, a measure of the compactness of the cluster shape may be introduced in the cost calculation, using a value calculated by dividing the squared perimeter of the cluster by its area. This achieves a better approximation of the clustering intent. The time complexity of HMC is usually O(n×log (n)) depending on the test function.

Several spin-offs of this algorithm have been developed, each with some improvement or a change geared towards a specialized use. One of the most remarkable innovations is an HMC based on fitting primitives (Athene et al. 2006). The extended algorithm uses several geometric primitives described through mathematical functions to decide in the cluster joining process. From an architectural fabrication point of view, this is important because the primitives can be used at the end to approximate, and thus help fabricate, the surface through parts easily created in standard processes. Moving beyond the particularities of this routine, any other metric can be used to calculate the cost of contracting an edge. Such modifications have been applied in the service of faster radiosity calculation, collision detection, and surface simplification. A number of similar modifications have been employed as described below in the service of goals specific to GAD.



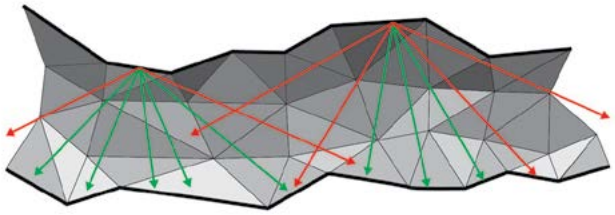
6

K-Means Clustering
(Shlafman et al. 2002; Funkhouser et al. 2004)
Like HMC, K-means (Figure 5) clustering is an iterative segmentation algorithm valued in CG applications for its relatively simple implementation and robust results. The input for the algorithm is typically an integer number defining the desired number of segments into which the original mesh will be split. Again, the dual graph comes into play at several crucial steps. First, in order to split the mesh, a set of edge weights is computed. Both papers referenced here make combined use of two properties, face-angle and edge-traversing-distance, in calculating the weight value for each edge. Based on these assigned weights, a central best-connected-point is computed using an all-pairs minimal-path algorithm, essentially a Dijkstra's algorithm run for every node in the original dual graph. Once the first root is selected, a second is identified as the farthest node from it. The distance is computed based on the stored values in the cached all-pair minimal path results. The process continues for any roots not yet selected, and the last step is repeated by looking for the most distant node from all the already selected root nodes. Then, after the root selection process is complete, each node is allotted to a mesh segment defined by one root, using the same all-pair weight-based distance calculation. After the initial segmentation is complete for each segment, a new root is computed. The new root is the best connected node in the segment nodes, based on the same all-pairs minimal-path algorithm, but this time, it is calculated only amongst the nodes in the segment. The last two steps are then repeated iteratively until the new root selection stabilizes for each segment on the same node. The time complexity of the algorithm is O(n^2×log(n)) where n is the number of faces in the mesh, for the initial all-pair calculation, plus the time required for the computation of the new roots for each segment at every subsequent step.

Random Walks (Lai et al. 2009)
Like K-Means and HMC, the random walk segmentation operates upon a weighted graph, but in contrast with these iterative approaches, the segmentation occurs in a single step. The algorithm requires the identification of a number of starting faces to be used as roots. Based on these, a weight value is computed for each non-root face that quantifies the likelihood that a random walk starting from the non-root face will eventually end up in a given root face (Figure 6). Each non-root face, then, stores one likelihood value for each root. Once these are computed, each non-root face is assigned to a cluster associated with the root with the highest likelihood. The calculation of each likelihood value proceeds by summing a weighted version of the likelihoods of each neighboring face. As we might expect, the weighting factor is directly related to the corresponding edge-weight, and is calculated in a similar fashion to the edge-costs in other algorithms. In the paper referenced here, the edge-weights are assigned by a combination of dihedral angle, edge-length, and traversal distance. Two variants are presented that respond to the needs of differing applications: one style of calculation is more suited for graphical models, while the other better addresses technical three-dimensional data. The authors also describe a number of methods for seed selection that add a supplemental degree of automation to the algorithm. One approach uses a similar method with the K-means algorithm of Shlafman et al. (2002), where just a number of desired seeds is required, which are then distributed according to the weight of the edges. The other employs a system of particles distributed across the mesh as connected by virtual springs, the energy of which are iteratively minimized and distributed, and seed faces are selected after the system converges. The strength and speed of this approach is rooted in the way values are calculated for all the faces of the mesh in one step. The algorithm solves a set of equations (one calculation for each seed face) through a sparse linear system. Solving the system yields the values for all the faces at once. This gives the algorithm a typical running time of O(n×m×log(m)), where n is the number of seeds and m the total number of faces in the mesh. As in the previous two algorithms, this segmentation employs a weighted dual graph at a number of stages and for assorted processes, and differs essentially in the particularities of weight calculation.

Feature Point and Core Extraction (Katz et al. 2005)
A different approach to mesh segmentation, albeit one based on the same ground level concepts from graph theory, is the Feature Point and Core Extraction technique (FP+CE) proposed by Sagi Katz, George Leifman and Ayellet Tal. The algorithm works in multiple steps, all of which make use of the weighted dual graph of the mesh.

Before the actual calculation starts, the model is preprocessed using multidimensional scaling (MDS), a technique that transforms the graph such that the Euclidean distances between vertices approximate their geodesic counterparts. The first formal step is the prominent feature detection. This happens



7

6 Diagram of likelihood calculation for a mesh face that a random walk started in the face will reach a certain root. Here the start face (light gray) and the two end faces (dark gray) are connected by two graphs, red and green, which show part of the chain of potential for a few adjacent faces.

7 The shape diameter function. Rays are shot from every face of the mesh in the opposite direction of the normal face. Some of the rays are discarded based on the angle at which they land on the other side, or the length they travel inside the mesh. The length values of the kept rays are averaged as the shape diameter value.

by checking if the sum (computed using a version of Dijkstra's algorithm) of the geodesic distances of a node to all other nodes in the graph is larger than any of its neighbor's sums. If so, the node is considered a feature node. Second, the mesh "core" is extracted. The core is considered the body of the mesh devoid of features and protruding elements. For this, a spherical mirroring of the mesh is computed and a convex hull is created, such that the nodes close to the hull in the mirrored version of the mesh are considered "core" nodes. The initial collection of these core nodes is extended iteratively until all features have been separated into individual segments. Then, using a standard graph traversing technique (such as depth-first-search), all segments are walked and connected to the core if they don't contain at least one feature node.

Although created specifically for application in graphical model meshes, this method presents a number of elements easily applicable to more general uses, as is discussed in a section to follow.

Shape Diameter Function (Shapira et al. 2008)

The Shape Diameter Function (SDF) represents a unique take on the general approach of segmenting a mesh based on a set of values assigned to each face. The main differences are in the calculation of the values used for the segmentation. The segmentation algorithm itself is a k-way graph cut employed in other research projects related to mesh segmentation (Shlafman et al. 2002). The distinguishing feature is the two-step calculation of the segmentation values.

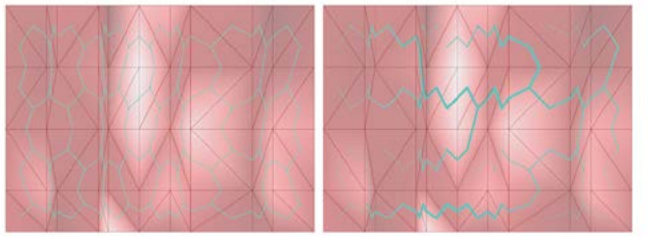
In the first step, a value is computed for each face of the mesh using a shape diameter function, which serves to describe how deep the mesh model is at a certain point. For this, a set of rays are constructed in a conical configuration centering on the evaluated point, and oriented in the direction of the inverted face normal. Based on the validity of the landing point on the other side of the model, rays are either stored or discarded. The mean value of any remaining valid rays is computed as the "shape diameter value" (Figure 7) of the face, which may then be clustered in a soft partition using a Gaussian mixture model and an expectation maximization algorithm. The second step uses a k-way graph cut to create hard boundaries and adapt the segmentation to the features of the mesh by factoring a series of geometrical determinants of the mesh (such as dihedral angle and edge length) into the calculated face values. The method described in the paper is again highly dependent on the type of mesh model presented, and can produce meaningful results only on closed and non-noisy meshes. However, within this limited subset of meshes, it is effective at producing meaningful partitions and, in subsequent steps, to extrapolate those partitions into mesh skeletons. In this way, SDF represents a potentially applicable approach to mesh segmentation in GAD, but the meshes in GAD do not necessarily belong to this limited subset, only insofar as a designer is able to understand the limits of its application.

Randomized Cuts (Golovinskiy and Funkhouser 2008)

Although not a novel technique per se, the process detailed in this paper is of interest because it proves the underlying compatibility of mesh segmentation techniques based on dual graph representation. The research employs a number of algorithms from the most commonly used ones (all of them already detailed above in the survey) and randomly switches between techniques in an attempt to find the most consistent cuts in a dual graph. The mesh is cut into functional parts using the same dual graph support and the multiple segmentations are analyzed by a function to determine the most consistent cuts. The process uses hierarchical clustering, k-means clustering, and min-cut clustering (a version of hierarchical clustering) to split the graph multiple times with different random input variables. Each of the cuts is evaluated by the function, and a score is assigned and in the end the best scoring cut is selected. This produces the most consistent possible cut, because no single algorithm and variable set can produce meaningful segmentations every time (Athene et al. 2006).

This statement acknowledges the fact that the wealth of mesh segmentation algorithms and the multitude of research undertaken in this field related to CG is made up mostly of individual specialized research geared toward very specific goals. Even if most of the algorithms employed are general, their implementation is based on the iterative exploration of their outcome. In order to produce meaningful results, the tools' development are

8



9

8 The tool menu in one of the wip versions of Ivy for Grasshopper. 1|Create and decompose MeshGraphs tools; 2| Add weight to MeshGraph; 3|Primary Segmentation (tree making); 4|Secondary Segmentation; 5|Iterative segmentation; 6|Special Segmentation; 7|Fabrication; 8|Mesh Info; 9|Other Tools.

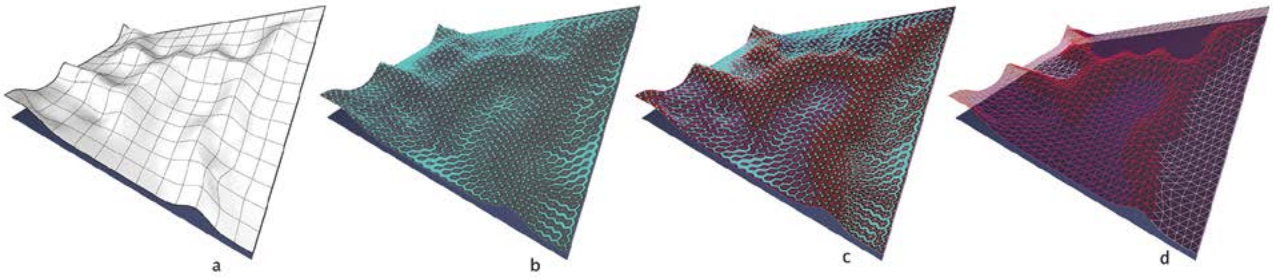
9 Mesh with attached MeshGraph (left) and tree MeshGraph on the same mesh calculated with Djikstra's algorithm using dihedral angle as edge weight.

often directed towards a certain behavior. In order to secure consistent results with every use, many of the values are hard-wired in the algorithm, thus forsaking exploration with the tool.

Identification of Need in Generative Architectural Design

Applications in CG regard the mesh quite differently than applications in GAD. In general, CG applications tend to treat a mesh as a nominally smooth surface that happens to be described using discrete faces in order to take advantage of the related discrete computational methods. In contrast, GAD applications more often rely on the network of connected faces and edges of the mesh as a simplified representation of architectonic elements, such as structural framing or facade panels. This basic distinction leads to a number of important requirements for mesh segmentation that are unique to GAD. A major exception to this distinction is physical simulation models, such as energy models or spring-systems, which are treated as discrete descriptions of nominally smooth surfaces in GAD. Insofar as GAD regards meshes as simplified representations of architectonic elements, many of the requirements for subdivision concern fabrication. Take, for example, the work surrounding the definition of planar quadrilateral (PQ) meshes (Glymph et al. 2004; Pottmann et al. 2015; 2008; 2007; Liu et al. 2004), in which a constrained mesh finds significance in its ability to represent the curtain wall panels of a glass building. Some of the routines related to PQ meshes rely on the discovery of developable strips, a procedure that is described as a subset of mesh segmentation in CG, and may be handled using a weighted-graph approach. Another more modest example in scope of execution may be found in the work of Marc Fornes, whose creative practice specializes in artistic installations. While there are no publications that detail the processes employed in the design of these projects, we can surmise that the subdivision of meshes is central to their realization in that the number of mesh faces (along with the number of "stripes," which we may presume to be a unit of subdivision) is listed prominently in the project credits. A related concern unique to GAD regards the preservation of the specific features of a mesh, such as folds, creases, and geometric textures. These concerns are not well addressed in the existing CG literature, but as is demonstrated below, are possible to support using a generalized weighted graph approach. Using mesh segmentation to enable architectural production (both at full-scale and in the service of architectural scale models) is an important defining feature of GAD. This particular issue is rarely a goal in CG, and as a result, there are hardly any tools explicitly developed for it in the larger field of computer science. Notable exceptions to the lack of existing segmentation routines that address the unique needs of GAD include primitive-fitting routines and applications in papercraft. The primitive fitting technique (Athene et al. 2006) suggests application to full-scale architectural fabrication, as does the mesh-segmentation used for part-grafting pieces of a mesh model using elements selected from a given library of forms (Funkhouser et al. 2004). Routines in CG designed for the fabrication of meshes center on small-scale papercraft (Mitani and Suzuki 2004; Massarwi et al. 2007), which includes small models made of thin-sheet materials such as paper, cloth (Julius et al. 2005), metal, or plastic. The aim of this research is to reproduce a natural "look" without the constraint of preserving the exact input geometry.

Perhaps the most significant mismatch between existing tools developed for CG and GAD concerns the difference in the expected cultures of use, and the exploratory nature of the design process. In GAD applications, software tools are used primarily as a means for exploration well before any production takes place. In early stages of the design process, techniques and approaches are revised often, and multiple algorithms are often employed in novel combinations to produce results far beyond what could be anticipated in advance. For this reason, frameworks that allow access to low-level controls are preferred by this community over packaged software tools or routines presented as black-boxes.



10 One step segmentation of hyperbolic paraboloid with added large-scale surface noise. Step (a) the surface and the guide surface; (b) the mesh version of the surface with a forest of trees resulted from a Kruskal's algorithm calculation. The weight of the MeshGraph is based on the dihedral angle and the proximity to the base (blue-gray) surface. In (c) the same forest of graph trees is separated in single-node trees and multi-node trees. Step (d) shows the mesh segmentation in 4 parts and a separation area of faces in red.

IVY

Presented above are two surveys: the first details the existing approaches to mesh segmentation routines in CG, while the second describes the unique requirements and unmet needs in GAD for such routines. Here, we bring the common foundational representation of the weighted graph dual—identified by the first survey—to bear in the construction of a framework that addresses the needs and requirements identified by the second survey.

We assert that if mesh segmentation routines are to be brought to bear on the unique requirements of GAD, then a comprehensive and modular framework is needed. The wide range of specialized algorithms already used for specific narrow tasks in CG need to be generalized and adapted such that the resulting framework may be applied more situationally. For this, a common representation is required so that the different algorithms may be coordinated and combined by the end-user. Ensuring data transference between algorithms enables custom aggregation of tools that go beyond simple input value changes, and into new and unimagined uses of mesh segmentation. Detailed in this section are the steps taken towards creating a weighted-graph mesh segmentation framework for use in GAD, implemented as an extension for the popular visual scripting platform Grasshopper. We describe here the core data structures, routines, and expected workflows supported by this extension, which is named Ivy, and discuss the extension's unique synthesis of the mesh segmentation routines presented above.

Implemented Data Structures & Routines

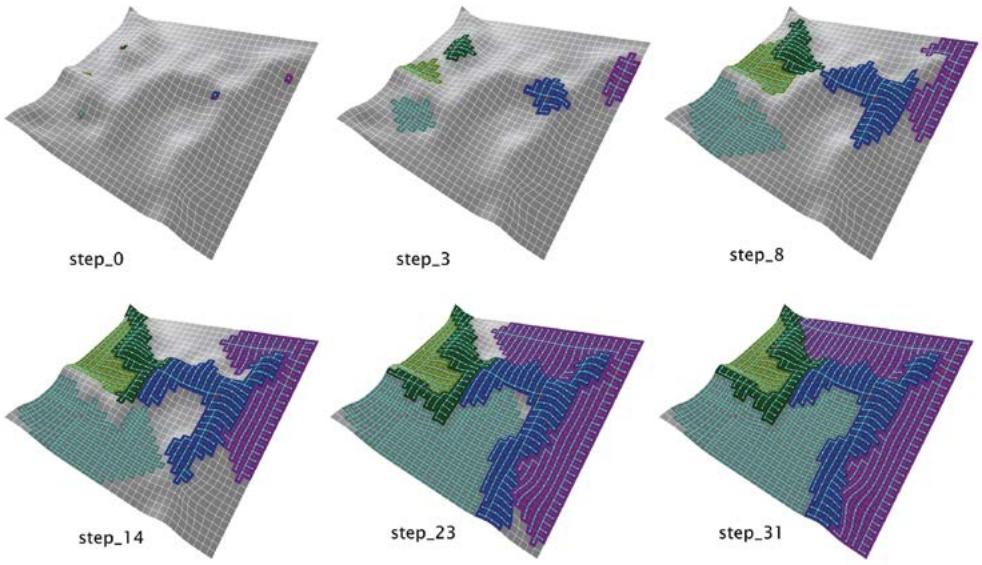
A weighted graph dual of a mesh, and its reconfiguration as a minimum-spanning tree via various routines, forms the common basis of many of the routines surveyed from CG. Here we describe the implementation of the required data-structures to represent and manipulate this graph, including the MeshGraph, MNode, and MEdge types, and the related routines that assign weights and perform segmentations in a generalized way that is able to reproduce the routines discussed above. The toolset, as it was implemented in Grasshopper, is organized according to an expected workflow for mesh segmentation. This is reflected by the ordering of the nine groups of components seen in the nearby diagram (Figure 8). Following the tool groups on the Grasshopper ribbon in Ivy equals following the intended general chaining of tools for an expected mesh segmentation.

MeshGraph

The dual graph concept is implemented by Ivy as a data object called MeshGraph. This provides the core functionality of the toolset and allows the interconnection of different tools into a coherent linear flow. A MeshGraph stores a collection of nodes (MNode) and edges (MEdge) that typically correspond to the mesh faces and non-naked edges of a given mesh. It also stores a copy of the given mesh geometry at construction, which allows for operations such as unfolding and fabrication to occur after segmentation.

Edgeweights and Nodeweights

The essence of mesh segmentation is the identification of variable properties of the mesh for the purpose of assigning values to each mesh edge or mesh face. These values, often called weights or costs, serve as a guide for the segmentation routines. 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. Importantly, there are also tools for assigning arbitrarily calculated weights, which enable the use of any numerically expressible property.



11

11 One-step segmentation of hyperbolic paraboloid with added surface noise. (a) the surface and the guide surface; (b) the mesh version of the surface with a forest of trees resulted from a Kruskal's algorithm calculation. The weight is based on the dihedral angle and the proximity to the base surface. In (c) the same forest of graph trees is separated in single-node trees and multi-node trees. Step (d) shows the mesh segmentation in 4 parts and a separation area of faces in red.

12 A two-step segmentation routine. The base mesh (a) is used to create a MeshGraph with edges weighted according to the dihedral angle between the faces. Making a tree from the dual MeshGraph (b) results in cup-like surfaces connected with one edge (the least sharp edge from the rims connecting the cups). By removing the edges with the largest weight in the tree (c) the parts are segmented. No faces from the original mesh (d) remain isolated.

Segmentation Routines

Routines that segment a mesh given its weighted graph dual represent the largest group of tools in the framework, and also the most important. Virtually all tools that operate and modify a graph in Ivy are considered MeshGraph segmentators. Here we find graph-processing algorithms that make use of already-defined edge or node weights in the service of selecting two MeshGraph nodes, disconnecting them by removing an edge, and then updating the graph structure. At times, this requires operation on a tree, which is expressed as a special case of a graph. This is why the tool sections in Ivy are divided between Primary Segmentation, Secondary Segmentation, Iterative Segmentation, and Special Segmentation.

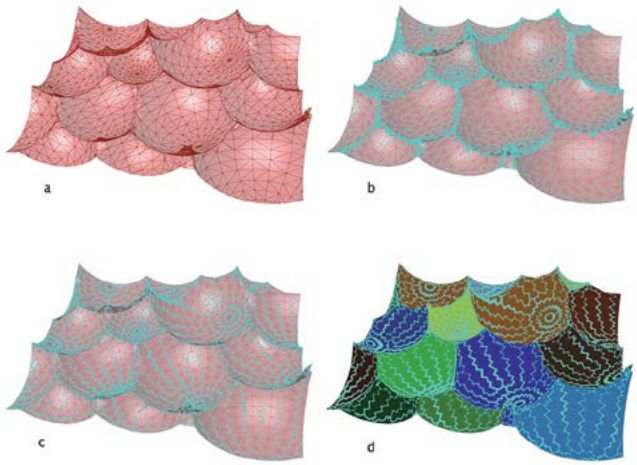
Primary Segmentation: Tree-Making Routines

The tree-making routines are the standard graph algorithms often employed for this purpose, including Prim's, Kruskal's, Djikstra's (Figure 9), and Depth First Search. The construction of a tree graph is the first step in either splitting a MeshGraph, or preparing it for segmentation in a subsequent step, as discussed in the section below on one-step segmentation. This is the main reason why in the Ivy workflow, converting a dual graph into a tree is considered primary segmentation. Another reason consists in the fact that as a result of the dual graph to tree conversion, a forest of tree graphs can emerge, thus effectively creating a segmentation of the original mesh (Nejur 2016, 4–8). Figure 10 shows an example of Disjoint Set tree making and resulted segmentation and Figure 11 depicts a primary segmentation example based on multiple root minimum span tree algorithms (MRMSTs).

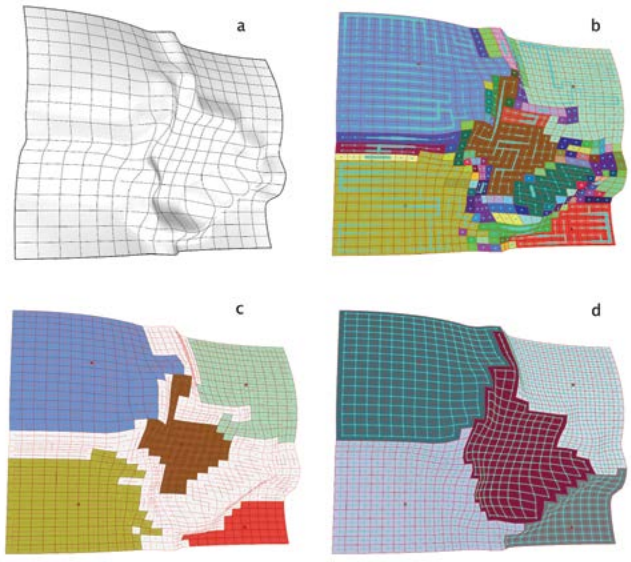
Secondary and Iterative Segmentation

The next section of segmentation routines includes tools that segment readymade tree graphs. Secondary segmentation happens through the selection and removal of tree edges (Ivy Manual, 9). Through a singular edge removal—a simply connected graph—a tree is split into two segments (Figure 12).

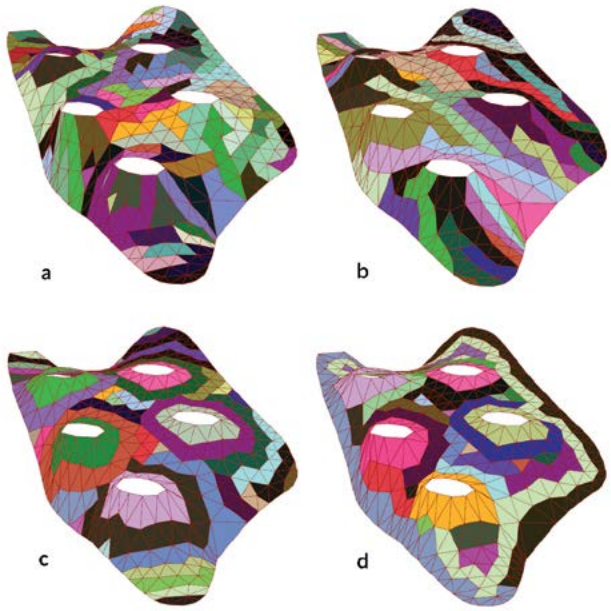
Another section is dedicated to self-contained iterative algorithms that operate on the original dual MeshGraph. Here we find the implementation of many of the segmentation routines identified in the literature review above, including K-Means



12



13



14

13 An enhanced K-Means segmentation setup in Ivy with automatic detection of the number of regions. The base, a doubly curved surface with a large overlapped noise (a) is overly segmented with a disjoint set (Kruskal's) algorithm with a weight limit. This produces a forest of trees with a diverse number of nodes (b). The weight limit is set so that low weight landscapes can coagulate in larger trees. By extracting those large trees and calculating a weight center for each of them we get a very good approximation of the final placement of the k-means seeds (c). Starting from those seeds, the iterative part of the classic K-means algorithm reaches a stable state in very few steps. This drastically cuts the run time of the algorithm, depending on the size and the features of the mesh, sometimes to less than a third.

14 Different unrolling strategies of a parametrically generated architectural mesh. The meshes are segmented in Ivy with the aim to reduce the final number of pieces needed to unroll the geometry on a flat surface. Shown are four variants resulted from different weight settings for the MeshGraph. From top to bottom, left to right: dihedral angle, edge distance from a median curve, orange peel edge classification starting from the holes, and finally the same orange peel classification started from the edge.

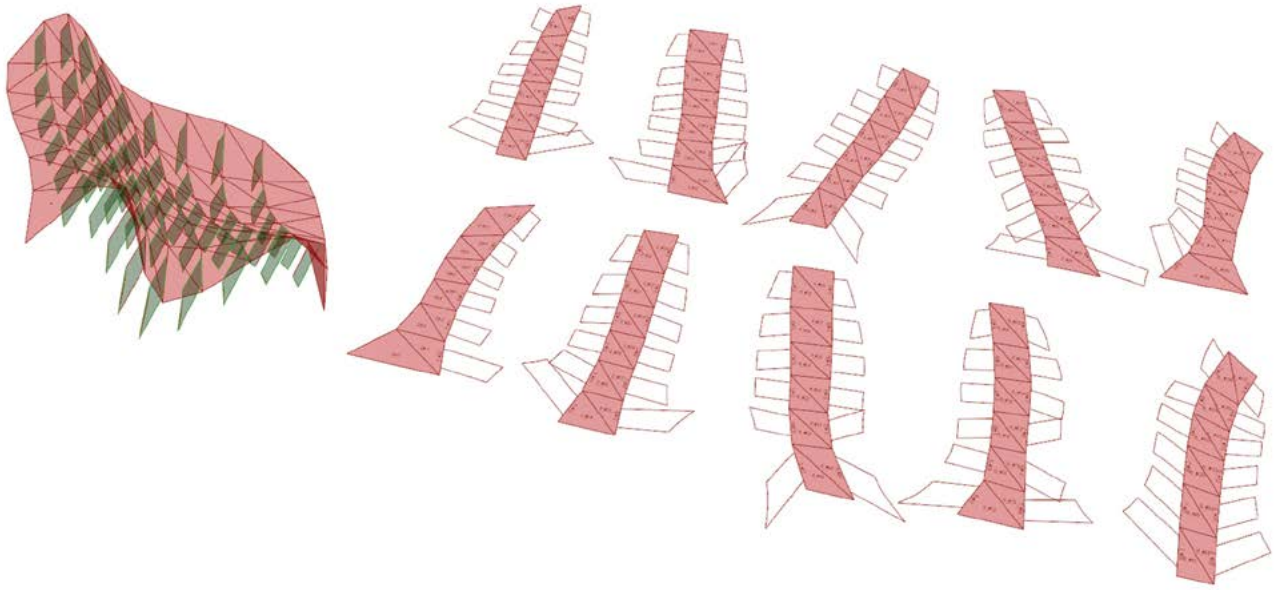
and HMC, each of which has been adapted to modular visual programming environment of Grasshopper. Figure 13 shows an example of the iterative segmentation workflow.

DISCUSSION

In this paper we demonstrated that generative architectural design, although deeply embedded in the contemporary techniques of digital form-making and representation, still lacks adapted tools for mesh segmentation. It is clear though that mesh segmentation is useful and needed in the context of GAD. However, the specific conditions of generative design claim a different approach than those of CG, for instance where a lot of research has already been conducted.

As a result of the identified need for specific mesh segmentation in GAD, we introduce Ivy, a platform based on the underlying mathematical concepts that power most of the research in the field. Ivy brings together a number of tools ported from CG-based research or developed specifically for the tasks in GAD. The separate tools and technologies are compatible through the graph representation of the mesh, which is already the default data object for most of the research in the field.

The platform implementation for mesh segmentation in Grasshopper brings many benefits to generative design beyond mere availability of tools. The possible aggregation of functionalities facilitated by the visual scripting platform of Grasshopper and the common graph language proves that the sum of tools working together is really much more potent than the simple addition of individual algorithmic benefits. For now, implementation of mesh segmentation tools in GAD in general, and Grasshopper specifically, is fairly new, and Ivy was started mostly as a proof of concept. As a result, there are still speed issues and limitations when it comes to meshes with many faces. Optimization of the code to extend usability is one of the goals for the authors. At the time when this paper was written, only a few of the algorithms developed for CG research had been ported to Ivy. The authors hope to extend the range of components that bring useful mesh segmentation to GAD. Along with new segmentation algorithms, the focus of future research and subsequent papers will fall on mesh segmentation usage in GAD. Within this more practical area of investigation, a special place will be dedicated to mesh unrolling. For reasons pertaining to the size and scope of this paper, the tools and workflow descriptions



15 A speculative study of the use of Ivy in the service of mesh unrolling. Connecting tabs of arbitrary geometry may be added at each connection edge.

were rather brief. A more in-depth explanation and detailing of the tools and functionalities of Ivy can be found in the Ivy User Manual (Nejur 2016). Because Ivy was designed to be a platform, it enables future research into mesh evaluation, segmentation, and fabrication as part of the Grasshopper ecosystem. This has manifested fabrication techniques and mesh exploration strategies that are included with Ivy, but are only briefly mentioned in this paper. Aside from a few standalone uses mentioned above, a number of workflows, examples, and practical applications remain to be presented. This positions the present body of text as an introduction to the tool and a sample of practical mesh segmentation use inside the GAD paradigm.

ACKNOWLEDGEMENTS

The paper presents research related to a Fulbright grant that supported Andrei Nejur as a visiting scholar hosted by Kyle Steinfeld at UC Berkeley. The development work for the toolset was undertaken using student feedback from the class of Simon Schleicher.

REFERENCES

Agathos, Alexander, Ioannis Pratikakis, Stavros Perantonis, Nikolaos Sapidis, and Philip Azariadis. 2007. "3D Mesh Segmentation Methodologies for CAD Applications." *Computer-Aided Design and Applications* 4 (6): 827–841.

Attene, Marco, Bianca Falcidieno, and Michela Spagnuolo. 2006. "Hierarchical Mesh Segmentation Based on Fitting Primitives." *The Visual Computer* 22 (3): 181–193.

Chen, Xiaobai, Aleksey Golovinskiy, and Thomas Funkhouser. 2009. "A Benchmark for 3D Mesh Segmentation." *ACM Transactions on Graphics* 28 (3): Article 73.

Fornes, Marc. "MARC FORNES & THEVERYMANY." Accessed April 26, 2016. https://theverymany.com/.

Funkhouser, Thomas, Michael Kazhdan, Philip Shilane, Patrick Min, William Kiefer, Ayellet Tal, Szymon Rusinkiewicz, and David Dobkin. 2004. "Modeling by Example." In *Proceedings of the 31st International Conference on Computer Graphics and Interactive Techniques*, edited by Joe Marks. Los Angeles, CA: SIGGRAPH. 652–663.

Garland, Michael, Andrew Willmott, and Paul S. Heckbert. 2001. "Hierarchical Face Clustering on Polygonal Surfaces." In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*. Research Triangle Park, NC: SI3D. 49–58.

Gelfand, Natasha, and Leonidas J. Guibas. "Shape Segmentation Using Local Slippage Analysis." In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*. Nice, France: SGP. 214–223.

Glymph, James, Dennis Shelden, Cristiano Ceccato, Judith Mussel, and Hans Schober. 2004. "A Parametric Strategy for Free-form Glass Structures Using Quadrilateral Planar Facets." *Automation in Construction* 13 (2): 187–202.

Golovinskiy, Aleksey, and Thomas Funkhouser. 2008. "Randomized Cuts for 3D Mesh Analysis." *ACM Transactions on Graphics* 27 (5): Article 145.

Julius, Dan, Vladislav Kraevoy, and Alla Sheffer. 2005. "D-Charts: Quasi-Developable Mesh Segmentation." *Computer Graphics Forum* 24 (3): 581–590.

Kalvin, Alan D, and Russell H. Taylor. 1996. "Superfaces: Polygonal Mesh Simplification with Bounded Error." IEEE Computer Graphics and Applications 16 (3): 64–77.

Katz, Sagi, and Ayellet Tal. 2003. "Hierarchical Mesh Decomposition Using Fuzzy Clustering and Cuts." *ACM Transactions on Graphics* 22 (3): 954–961.

Katz, Sagi, George Leifman, and Ayellet Tal. 2005. "Mesh Segmentation Using Feature Point and Core Extraction." *The Visual Computer* 21 (8): 649–658.

Lai, Yu-Kun, Shi-Min Hu, Ralph R. Martin, and Paul L. Rosin. 2009. "Rapid and Effective Segmentation of 3D Models Using Random Walks." *Computer Aided Geometric Design* 26 (6): 665–679.

Lin, Hsueh-Yi Sean, Hong-Yuan Mark Liao, and Ja-Chen Lin. 2007. "Visual Salience-Guided Mesh Decomposition." *IEEE Transactions on Multimedia* 9 (1): 46–57.

Liu, Rong, and Hao Zhang. 2004. "Segmentation of 3D Meshes through Spectral Clustering." In *Proceedings of the 12th Pacific Conference on Computer Graphics and Applications*, Seoul: PG. 298–305.

Mangan, Alan P., and Ross T. Whitaker. 1999. "Partitioning 3D Surface Meshes Using Watershed Segmentation." *IEEE Transactions on Visualization and Computer Graphics* 5 (4): 308–321.

Massarwi, Fady, Craig Gotsman, and Gershon Elber. 2007. "Papercraft Models Using Generalized Cylinders." In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*. Maui, Hawaii: PG. 148–157.

Mitani, Jun, and Hiromasa Suzuki. 2004. "Making Papercraft Toys from Meshes Using Strip-based Approximate Unfolding." *ACM Transactions on Graphics* 23 (3): 259–263.

Mortara, Michela, Giuseppe Patané, Michela Spagnuolo, Bianca Falcidieno, and Jarek Rossignac. 2004. "Plumber: A Method For a Multi-Scale Decomposition of 3D Shapes Into Tubular Primitives and Bodies." In *Proceedings of the Ninth ACM Symposium on Solid Modeling and Applications*, Genova, Italy: SM. 339–344.

Nejur, Andrei. 2016. "Ivy for Grasshopper Manual: Version 0.802." *Digital Design Research Repository Andrei Nejur*. Accessed July 01, 2016. http://research.n2arh.ro/ivy/manual.

Pottmann, Helmut, Michael Eigensatz, Amir Vaxman, and Johannes Wallner. 2015. "Architectural Geometry." *Computers & Graphics* 47: 145–164.

Pottmann, Helmut, Alexander Schiftner, Pengbo Bo, Heinz Schmiedhofer, Wenping Wang, Niccolo Baldassini, and Johannes Wallner. 2008. "Freeform Surfaces from Single Curved Panels." *ACM Transactions on Graphics* 27 (3): Article 76.

Pottmann, Helmut, Yang Liu, Johannes Wallner, Alexander Bobenko, and Wenping Wang. 2007. "Geometry of Multi-layer Freeform Structures for Architecture." *ACM Transactions on Graphics* 26 (3): Article 65.

Shamir, Ariel. 2008. "A Survey on Mesh Segmentation Techniques." *Computer Graphics Forum* 27 (6): 1539–1556.

Shapira, Lior, Ariel Shamir, and Daniel Cohen-Or. 2008. "Consistent Mesh Partitioning and Skeletonisation Using the Shape Diameter Function." *The Visual Computer* 24 (4): 249–259.

Shlafman, Shymon, Ayellet Tal, and Sagi Katz. 2002. "Metamorphosis of Polyhedral Surfaces Using Decomposition." *Computer Graphics Forum* 21 (3): 219–228.

Skiena, Steven S. 1998. *The Algorithm Design Manual.* Santa Clara, CA: TELOS—the Electronic Library of Science.

Stork, David G., Richard O. Duda, and Elad Yom-Tov. 2004. *Computer Manual in MATLAB to Accompany Pattern Classification*. Hoboken, NJ: Wiley-Interscience.

Taubin, Gabriel, and Jarek Rossignac. 1998. "Geometric Compression through Topological Surgery." *ACM Transactions on Graphics* 17 (2): 84–115.

Xing, Qing, Gabriel Esquivel, Ergun Akleman, Jianer Chen, and Jonathan Gross. 2011. "Band Decomposition of 2-Manifold Meshes for Physical Construction of Large Structures." In *Posters of the 38th International Conference and Exhibition on Computer Graphics and Interactive Techniques*. Vancouver, BC: SIGGRAPH.

Xing, Qing, Gabriel Esquivel, Ryan Collier, Michael Tomaso, and Ergun Akleman. 2011. "Spulenkorb: Utilize Weaving Methods in Architectural Design." In *Proceedings of the 14th Annual Bridges Conference: Mathematics, Music, Art, Architecture, Culture.*, edited by Reza Sarhangi and Carlo H. Séquin. Coimbra, Portugal: Bridges. 163–170.

Yamauchi, Hitoshi, Stefan Gumhold, Rhaleb Zayer, and Hans-Peter Seidel. 2005. "Mesh Segmentation Driven by Gaussian Curvature." *The Visual Computer* 21 (8–10): 659–668.

IMAGE CREDITS

Figure 1: Deleuran and Reeves, 2015
Figures 2–15: © Nejur, 2016

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.