2014 Proceedings of the Symposium on Simulation for Architecture and Urban Design

Edited by Dr. David Gerber and Rhys Goldstein
A Freeform Surface Fabrication Method with 2D Cutting

Andres Sevtsuk\(^1\) and Raul Kalvo\(^2\)

City Form Lab, Singapore University of Technology and Design
20 Dover Drive,
Singapore 138682
\(^1\)asevtsuk@alum.mit.edu, \(^2\)raul@inphysica.com

Keywords: Gridshell, Fabrication, Free-Form Surface, Funicular Structures, Computation, RhinoPython.

Abstract

We introduce a method for creating free-form architectural structures out of 2D domain line networks. The resulting structure combines principles of thin shell and single-layer grid structures. The innovation lies in a three-dimensional geometrical arrangement, where all structural elements can be cut out of flat panels. The advantage of the proposed method is that structural support systems can be created for a wide variety of line networks using simple cutting technology (e.g. saws, laser-cutters, 3-axis CNC routers), making the construction of geometrically complex structures accessible to a wider audience at a significantly lower cost. We illustrate theoretical possibilities of the approach and demonstrate a full-scale application on a 200 square-meter pavilion built from plywood panels and clad with sheet-metal tiles at the Singapore University of Technology and Design. An analogous approach can be used with a high degree of flexibility to fabricate complex structures of different shapes and patterns for various building applications.

1. INTRODUCTION

Gridshells are part of a larger family of thin-shell structures that have a long history of structural investigation (Mungan and Abel, 2012, Schlaich 2002). Along with funicular vaults, monoliths and membrane shells, their effective structural properties have made gridshells an attractive solution for constructing bridges, hangars, domes, and pavilions that require uninterrupted covered space. Gridshells save material by using double-curved forms that follow the lines of structural thrust, thereby achieving economical, efficient and elegant structures. The geometric forms of membrane and funicular shell structures are dictated by the distribution of forces, where tensile structures work fully in compression. Whereas tensile structures almost always form anticlastic surfaces (Pront and Diminicus 2013), with exception of pneumatic shells, which are synclastic, then funicular shells form dominantly synclastic surfaces.

Gridshells and monolithic shells provide much more freedom since they can combine tension and compression into one surface. By definition, gridshells are double-curved structures (Douthe 2006). This geometrical aspect gives gridshells more global stability and reduces material usage in comparison to structures that work predominantly in bending (e.g. wall-ceiling or column-beam structures). In the context of this paper, we widen the definition of gridshells, treating them as 2D domain structures on a surface, where the surface can belong to any curvature type. We can also call our method a single-layer grid structure (Liu 2007), since every edge and every module (loop) forms an individual building block that can be fabricated separately. The advantages that result from the latter have been covered well by Canerapo (2014), and include benefits in fabrication, modularity, customization and material.

The use of double-curved forms introduces considerable challenges for the design and fabrication of such structures. Freeform gridshells tend to produce variable and complex joints between load-bearing beams. In the case of perfectly spherical shapes, such as the Bucky Ball, the convergence angles of all edges of the gridshell are identical, and the joints therefore economical to fabricate. But thrust lines of efficient spanning forms are not spherical; rather, they follow parabolic or otherwise variable curvature, requiring unique joints at every node of the gridshell. There are a number of existing ways of achieving variable curvature in such structures. The most commonly used method achieves the curvature of the gridshell through a large number of uniquely angled joints (i.e. MERO-Type), shown on the top
left of Figure 1. Constructing such joints is costly, however, requiring strong materials (e.g., steel) and advanced machinery capable of milling custom three-dimensional elements (Bo, Pottman et al. 2011).

A second approach subdivides a complex curved form into a grid of structural axes. A lattice of beams follow the axes and connect structurally at intersections (Figure 1, top right). If the members at joints meet at perpendicular angles, then this approach allows curved 3D gridshells to be achieved from 2D elements, using simple and economical joints (Sass 2005). With oblique angles—in the case of a triangulated lattice, for instance—the beams do not meet perpendicularly and their construction therefore requires 3D fabrication. In such a case, it is often more economical to tie the joints or connect them through the intersecting members with a pivoting fastener (e.g., a vertical bolt), without disrupting the continuity of the members (Figure 1, bottom left). Tie joints also allow non-orthogonal angles between beams, but the continuity of the members restricts the kinds of line networks that can be used in this method (Harris, Romer et al. 2003). The method is not suitable to line networks composed of discontinuous axes, for instance hexagonal line networks. There are also important constraints in the erection of such gridshells. Gridshells using the tied lattice structure are typically connected into a flat grid on the ground, and then gradually erected into shape by pushing in the supporting edges on site. This requires space and supporting ground, setting constraints on where such structures can be built.

A fourth approach achieves the gridshell curvature through curved structural beams, keeping all joints identical and standard (Figure 1, bottom right). Albeit repetitive, these joints too require 3D fabrication, which can be costly and restrictive (Bo, and Pottmann 2011). Only under rare curvature conditions and perpendicular line-networks can the elements of such structures be limited to two-dimensional cutting.

This paper introduces a method that allows gridshells to be structured from a curved line network that may be regular or irregular using arbitrary n-gons while keeping all structural elements and joints planar, allowing them to be fully fabricated on 3-axis cutting machines. The key benefit of the approach is that it offers great freedom in form and in the structural line-network design, while ensuring that all joints and beams can be fabricated economically from two-dimensional sheet material. An additional benefit is that the structure can be entirely prefabricated and assembled in modular components on site without large space constraints and without high-precision work on site.

To achieve this, two key innovations are necessary. First, the structure needs to be composed of two parallel walls around each network edge, and second, the adjoining non-parallel walls in each network loop need to be extruded at particular angles, such that straight intersection lines are achieved on the interior planes of n-gons. Both conditions are necessary to guarantee 2D fabrication.

An analogous gridshell that can be fabricated with strictly two-dimensional cutting is not feasible with a single-wall structure without creating gaps between elements or modifying the input line network, as illustrated in Figure 2. The figure shows in plan and axonometry how a continuous gridshell structure is achieved through a double-walled structure around each network edge, where the planes are extruded at such angles that straight intersection lines are achieved within all interior planes of the gridshell’s loops. In the case
of a single-walled solution (left), the pairs of panels on the left and right of the node cannot move any closer to the node along their own axes without starting to intersect. The single-walled solution therefore does not allow us to achieve straight intersection lines between all adjacent panels around a node with flat-bed cutting.

The joints are connected along a linear intersection line between two neighboring planes of gridshell beams, allowing any angles to be joined through a linear fastener (e.g. weld, fold, hinge etc.) as long as fasteners can fit between the planes. The vertical depth of the walls becomes a structural variable that can be increased for stronger linear connections. The proposed solution allows a wide variety of curved line networks to be turned into a gridshell structure in an economical way.

![Figure 3. Rendering of the SUTD library pavilion.](image)

We have implemented the approach in RhinoPython scripts and illustrate their implementation on a pavilion project at SUTD, manufactured from flat plywood sheets using off-the-shelf door hinges as linear fasteners (Figure 3). A similar method can be used to develop gridshells of different form, pattern, and material elsewhere.

2. USER INPUTS

The computation of a single-layer grid structure in our RhinoPython algorithm starts with two user inputs—an input line network and a NURBS surface. We call any curves, poly-curves, and polylines as lines since we only use the start and end points of any given curve. There can be alternative user inputs for line networks, for instance manifold n-gons, quads or triangular meshes. Requiring an input surface offers an easy way of obtaining normal vectors at the gridshell’s nodes. Users can assign multiple different surfaces to different lines or assign the vectors manually to each edge. Since the input surface is only used to obtain an approximate normal at each node, it is not critical to keep the input surface close to the line network.

A designer can start by modeling the desired surface of the structure, subsequently adding the line network onto the surface (Figure 4). The line network can be directly drawn in Rhino, generated using a Rhino3D plugin called Rhino Paneling Tools (Issa 2012), or modeled with third party tools like Netgen (Schöberl 1997). The form of the line network can be optimized using form finding tools like RhinoVault (Rippmen et al. 2012), Kangaroo (Piker 2013), or imported from other software packages. Line networks also offer a convenient geometric base for exchanging information between different analytic software environments, including finite element analysis (FEA) software for structural analysis.

![Figure 4. Left: a guiding surface. Right: Surface covered with a curved line network that defines the axes of the gridshell.](image)

The first important step in analysing the input line network is to detect which combinations of edges form closed loops. If three or more edges in the line network form a cycle, then they are detected as loops (Figure 5). It is critical to know the loops in the line network in order to find the different extrusion angles for structural walls, as explained below. The input surface is used to obtain the initial normal vectors at every network node.

![Figure 5. A network consists of nodes, edges and loops. Different network topologies can be used as input geometry.](image)

The algorithm is able to detect loops in complex networks. Loops can be detected on regular or irregular polygons of any size (n-gons) and shape (Figure 5). The loop detection procedure is described in Figure 6. It is
important that before loop detection starts, the nodes should already have normal vectors attached to them. This is because if the angle between two adjacent node vectors is larger than 90 degrees, it is not possible to detect correct loops, as edges cannot be ordered in a counter-clockwise sequence. After loop detection is performed on every node in the network, all loops are cleaned and only unique loops are saved. The general aim of loop detection is to find a valid 2-manifold mesh structure (Botsch 2010). It is certainly possible to use a mesh as an input directly (which already has predefined loops), but many types of software used for architectural modeling do not support n-gon meshes; typical mesh structures are quads and triangles. We achieve greater flexibility when working with lines.

![Figure 6. Process of loop detection. A. Selection of node. B. Ordering all edges in a counter-clockwise sequence and picking one of the edges as the starting point. C. Walking along the chosen edge and finding its neighboring edge that shares the smallest angle with it. D. Repeating step C as long as the walk arrives back to the starting node to form a loop, or until the node-count quota for the search is exhausted.](image)

We have also investigated graph-based loop detection (Johnson 1975) but problems emerge in loops with a geometrically small area that have a longer perimeter than loops with a geometrically larger area, but shorter perimeter. The elegance of graph-based loop detection is that there is no need for geometrical information about each edge and loop detection is therefore computationally less intensive.

![Figure 7. Search quota is the number of nodes visited before loop detection is stopped.](image)

For better user control, we have added a search quota (maximum number of steps) that is used for loop detection. Figure 7 illustrates loop detection with quota of “5”. Loops longer than 5 steps are not counted as loops and instead categorized as “naked”. The script also makes it possible to set an edge naked explicitly. This provides an easy way of keeping certain edges of the network (e.g., edges of arches, ground support edge) naked for architectural reasons.

The algorithm also allows additional information to be stored about the line network. Lines can be named in model space in order to store information about special conditions and constraints they should follow during the computation process. For instance, we have reserved the name “horizontal” for lines whose normals need to be kept parallel to the ground plane. Depending on need, naming conventions can be used for many other purposes and each node can be given a custom user-defined normal vector that differs from the input surface for greater control. This can allow, for instance, the designer to introduce structural load analysis results as inputs to the node normal directions. Figure 8 illustrates how different parts of an input network can carry different names and meanings for the algorithm. Over time, we plan to introduce more user input options and potentially integrate the algorithm with engineering software for load calculations and form finding. It is possible to assign every edge a different material, height and thickness.

![Figure 8. Different lines in the network can carry names and information about these lines. Lines that are called “horizontal” in the figure are always extruded parallel to the ground plane in the algorithm.](image)

3. FINDING THE GEOMETRY OF STRUCTURAL PANELS

When loops have been detected and particular constraints set with naming conventions, then the central challenge of computing structural panels begins. These panels are eventually cut out of sheet material on a two-dimensional router. The structure forms double-walled panels around each input network line. Facing panels of two adjacent loops need to be parallel to each other and panels in the same loop that share a corner need to share a straight intersection edge, where a linear fastener can be placed, or where the walls can be folded. To achieve this, we need to find edge normals (e), loop-edge vectors (d), loop-edge planes (F), and loop-node vectors (l) that define the intersection lines between two adjacent panels, as shown in Figure 9.
First, node normals ($n$) are located at the intersection points ($P$) of the original input line network. The vectors of node normals are found by computing the normals at the locations of points ($P$). Second, the node normals at the opposite ends of each original network edge are used to derive the corresponding edge normal ($e$). Each edge normal is found as the average of its two endpoints’ normals: $e_i = (n_i + n_{i+1})/2$. Next we need to construct a loop-edge plane ($F$) from the original edge normal ($e$) and the loop-edge vector ($d$) so that $F.x = e_i$ and $F.y = d_i$. The normal vector of this plane ($F_z$) is used to offset the loop plane inside, by at least the thickness of the construction material. An additional gap can also be created between the parallel panels in order to leave space for fasteners on both sides of the structural planes.

The loop-node vector ($l$) is calculated as the intersection line of two adjacent loop-edge planes ($l_i = F_i$ intersection with $F_{i+1}$). Every loop-node vector ($l$) is in the same plane with both of its neighbouring loop-edge vectors ($l_i$ is coplanar with $l_{i-1}$ and $l_{i+1}$). These vectors ($l_i$ and $l_{i+1}$) and the loop-edge plane ($F_i$) are used to construct a structural panel, which represents the inner material surface of each loop. The surfaces are extruded outwards from each loop to achieve a desired material thickness. Since all structural panels share an inner edge with their neighbouring edges (along vector $l$), they can be joined to each other using linear fasteners that follow the vector ($l$) on the inner surface of a loop (Figure 11).

Both the desired vertical depth, which is an approximation since depth on both ends of the panels is different due to their trapezoidal shape that generates the gridshell’s curvature, and the offset distance between two parallel panels are design variables that a user can control. User input for the depth of the shell sets the depth at the lower end of the trapezoid. Since the paths of forces in gridshells are generally designed to follow through the midpoints of structural elements, then the loop edges need to be extruded vertically in both directions above and below their original axis in the input line network. Our algorithm allows the user to decide where to set the centre lines relative to the structural depth of the gridshell. Once the height of each trapezoidal panel is computed, the trapezoids are offset towards the centre of each loop to form the inner surfaces of the structural loops. The choice of panel thickness and gap size can limit the geometry of line networks that can be used, as explained in Section 3 above.

The gridshell’s curvature can be achieved in one of two ways, depending on user preference. First, if the top and bottom edges of the panels are kept parallel, the curvature can be achieved at angles between neighbouring panels (Figure 10, top). The benefit here is that edge members can be cut from constant-width boards using automated and rotating saw mills, for instance. The downside, however, is that the resulting inside and outside surfaces of the gridshell obtain complex angles at joints, where significant forces transcend. It can be hard to transfer significant forces through such angled joints, requiring additional enforcement for load paths (this was the case in the pavilion, discussed below) or reducing the gap size to zero, such that adjacent modules touch each other at nodes. The second option is to keep panels at intersection points flat, achieving the shell’s curvature instead by curving or angling individual panels themselves (Figure 9, bottom). This allows the inner and outer surfaces of the gridshell to remain flat around joints, which can be covered by flat panels, which can be used structurally, to transfer loads between edges around a node. Each panel can be fastened into the spacer blocks or edges at the opposite ends of every node to carry the forces across the node via a flat surface. We adopted this latter approach for the SUTD library pavilion.

4. ADJUSTMENTS

User inputs to the script may produce a few important constraints on the feasibility of the structure. The most important constraint concerns assigning a zero thickness to the gridshell’s panels and a zero distance to the gaps
between parallel panels, which can easily produce unwanted intersections between adjacent loops. It is still possible to use the algorithm successfully with zero wall thickness and zero gap size between parallel walls, but the geometry of the network and the normal vectors at nodes are then constrained to limited solutions. This is a widely studied problem (Glymph et al. 2004, Pottman et al. 2007, and Liu et al. 2006), where the easiest solution involves keeping all node normals pointing towards a single convergence point. There is a way to ignore the normal convergence point constraint by limiting the line network to a valence of three (every node has a maximum of three edges connected to it, as in a hexagonal line network for instance). There is also a solution with a valence of four, where two diagonal edges are co-planar. But precision is critical here—when two adjacent edges are \textit{almost} co-linear and their normals \textit{almost} co-planar, then the intersection between them is not suitable for the structure since loop-node normals (Figure 9) deviate too much from node normals. It is therefore recommended that adjacent lines should have a smaller angle than 180 degrees. Note that the above geometric constraints only concern the zero panel and gap distance scenario and can be resolved by increasing the panel and gap thicknesses. However, there is also a constraint with panel thicknesses—if panel thickness is larger than the length of one of its adjacent edges, then an error occurs. The current state of the algorithm assumes that all these constraints are addressed by the user; they are not automatically detected.

The above method of lattice generation is suitable to most types of input line networks. Occasionally, however, the input network may contain sharp curvature peaks or concavity areas, which tend to produce sudden changes in the direction of surface normals. Sudden changes between neighbouring node vectors can produce colliding corner conditions for panels (Figure 11).

In order to avoid such collisions and sharp angular changes in nodes, the underlying node normals can be relaxed. The process of relaxation adjusts the node normal such that rapid angular changes are dispersed via Laplacian smoothing across multiple neighbouring nodes, reducing sudden normal changes in any one node (Cannan, Joseph et al. 1999). Relaxation provides better and smoother transition between node normals. Figure 11 (left) illustrates the before and after relaxation results on an arch with a relatively sharp peak. Grey normal vectors in the figure indicate initial stage node vectors; blue, the relaxed vectors. Note that “horizontal” edges are constrained to remain parallel to ground. Collisions can also be reduced by decreasing the height of structural panels, increasing panel thickness, and increasing the gap between parallel panels.

5. FABRICATION

Once the edge surfaces are extruded and populated through the structure, a double curved surface can be formed from flat edge panels. A few important steps, specific to the fastening solution chosen by the user, need to be addressed before fabricating the structure. In the SUTD pavilion case, hinges had to be fitted to the inside corners of each loop and adjacent loops had to be fixed to each other through a shared parallel plane with bolts through spacer blocks. In order to achieve this, connection holes for both hinges and bolts had to be found on every edge surface in three-dimensional space (Figure 12).

We introduced a \textit{hinge class} in the algorithm that takes several user inputs to determine the dimensions and hole patterns of hinges specified by the user. Depending on the pivot radius of the hinge, for instance, an additional gap may be needed in the inner corners of each loop to fit the hinge (Figure 12). The user input automatically populates line drawings on all panels for fixing the hinges between adjacent edges. We also included a \textit{bolting class} in the algorithm to determine the exact locations for bolts in the parallel walls of two adjacent loops. The \textit{bolting class} takes variables, such as desired number of bolts, distances from edges and bolt diameters, and automatically populates bolt holes onto all necessary panels. Solving for these hole locations algorithmically allows the user to detect any potential issues in the model and greatly improves on-site fabrication precision. All holes are predrilled during the CNC cutting process, and little error is left for fabrication on site. This reduces the site work to just an assembly process where no additional cutting or drilling is required in situ, allowing the structure to be assembled without highly specialized labor.
Next, after the fixtures and their desired holes have been computed, then all edge surfaces of the structure need to be flattened and fitted onto standard material sheets for two-dimensional cutting. The algorithm performs the flattening process by taking the outline of each edge surface, rotating it parallel to the ground plane, and packing numerous outlines as densely as possible to specified sizes of sheet material. In case of plywood or sheet metal, for instance, the flattening and packing can be fit onto standard 4x8 foot sheets. The packing procedure allows the user to specify the desired minimum distance between any two outlines on the layout, which helps the designer account for the size of the drill-head or laser-beam in cutting the elements. This packing procedure is further described by Dritsas, Kalvo and Sevtsuk (2013).

For outdoor use, the gridshells can also be clad with panelling tiles to prevent water, light or heat from entering the inside of the shell. Flat panelling approaches have been widely discussed elsewhere (Schiftner, 2010; Issa 2012) and for the sake of brevity we refrain from discussing them at length here. From a structural perspective, we should mention, however, that relatively strong cladding materials (e.g. rolled steel) can also be used structurally to distribute loads between gridshell panels around every node. The cladding tiles can be fixed directly to the underlying edge surfaces or spacer blocks between them, generating straight force paths through the metal cladding sheet from one edge to another. This approach was used with rolled steel cladding on the SUTD gridshell.

6. DISCUSSION

The method we introduce allows a designer to generate free-form gridshell structures from two-dimensional elements that can be cut using regular flatbed routers or laser-cutters. This approach has been implemented on a gridshell pavilion at SUTD (Figure 13). The pavilion uses 12mm plywood elements as structural edges, with a vertical depth of 200mm. Neighbouring panels are fixed with an off-the-shelf 4-inch door hinge, forming triangular loops. Two adjacent loops are fixed to each other via a parallel plane that is offset 25mm to fit a flat spacer block between the two walls. The structure is covered with 2mm rolled steel cladding tiles, which form an overlapping hexagonal “fish skin” over every node (Figure 14). The cladding sheets are screwed directly into spacer blocks between plywood walls, thereby distributing the axial loads from edge to edge via the sheet metal tiles. All plywood elements and spacer blocks are flat, cut on standard flatbed CNC routers.

The same algorithm can be used to generate gridshell lattices for different surfaces and different line networks. Figure 15 illustrates four different lattice patterns on an identical (anticlastic) surface. Figure 16 demonstrates the application of the algorithm on different types of surfaces using three types of line networks on each.

Beyond allowing complex double-curved gridshells to be constructed out of flat material elements, the proposed approach also offers an additional benefit—it allows the
design of the structure to be modified and fine-tuned until a very late stage in the design process.

A designer can simply change the geometry of the underlying line network or update the input parameters for material thickness, offset spacing, vertical depth or hinge and bolt dimensions, and the entire gridshell lattice can be regenerated instantaneously, complete with 2D fabrication layouts. Constructing the algorithms on custom scripts from multiple different classes, each of which have a set of input parameters, makes adjustments considerably easier than a large parametric or BIM model, where all parts share a long dependency chain, would allow. The process thus offers considerable flexibility and freedom for design. It can be used to develop gridshell structures of different form, material or pattern in a fast feedback loop between a designer and the algorithm.

We plan to release the software for public use by summer 2014, downloadable from the City Form Lab website (http://cityform.mit.edu).

REFERENCES


