6_smoothness

"...there are enough indications right now that subdivision software packages are going to be relevant in the next ten years".[14]

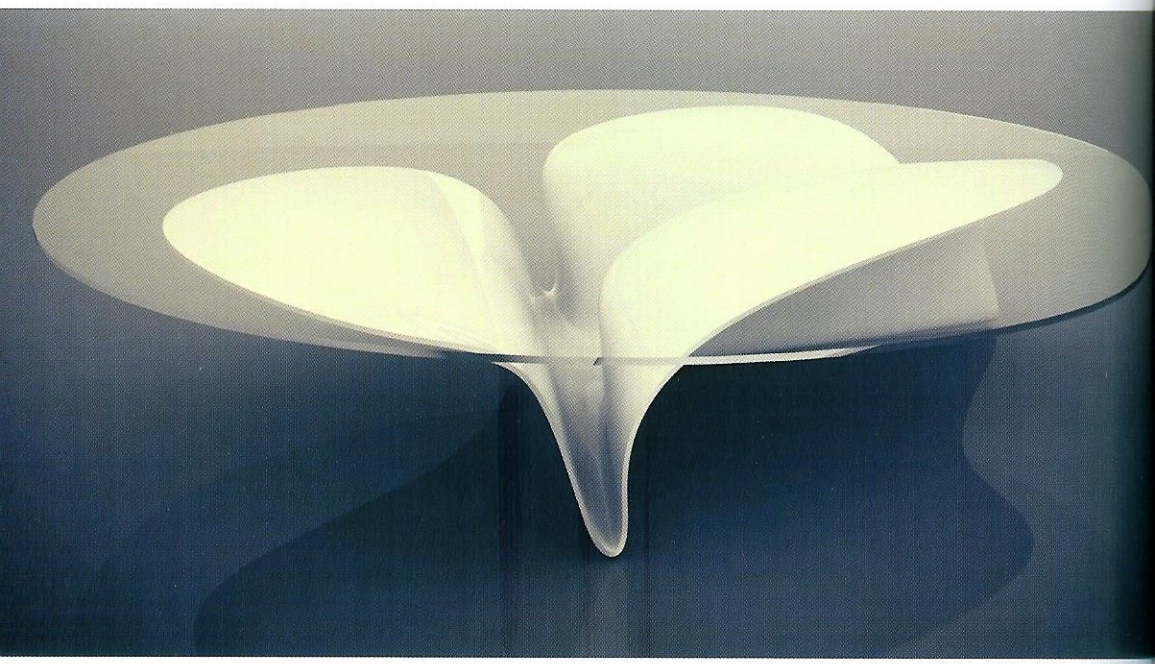
Greg Lynn

NURBS modeling is based on structuring mathematical curves or points to build surfaces. This method of surface creation is not always the best procedure, especially for freeform geometries. For example, the coffee table and the centerpiece shown on the following pages could be defined by NURBS surfaces; however, because of the surface complexity, another method relying on polygon meshes can be employed: the Subdivision Surface Modeling (SubD). This is a technique developed in the late 1970's and used extensively within the animation industry for character animation. The SubD method is based on the defining a schematic polygon mesh which is iteratively refined using specific algorithms. This technique allows users to model low-polygon meshes and generate smooth (high face count) meshes. The SubD method enables a level of smoothness which is quite difficult to obtain using NURBS techniques. Though NURBS are the dominant standard for CAD and engineering applications, SubD is becoming increasingly important in several fields such as industrial design.

NOTE 14

S. Converso, *Il progetto digitale per la costruzione. Cronache di un mutamento professionale* (Maggioli, 2010), 148–153.

FIGURE 6.1
Blossom Table, Arturo Tedeschi (2011).



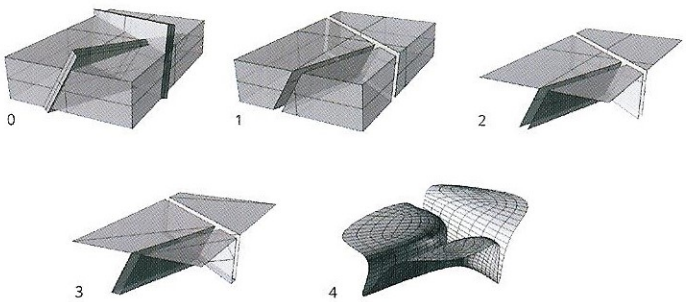
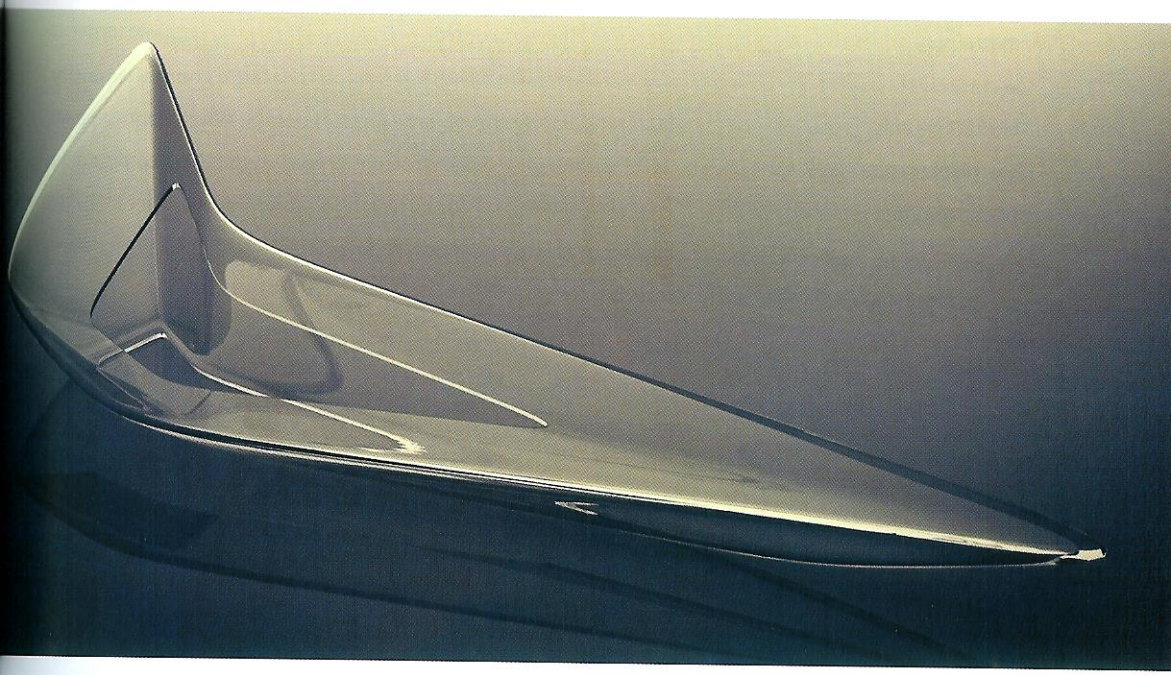
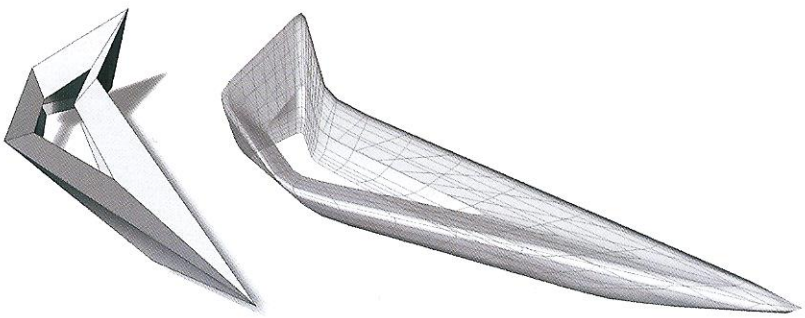


FIGURE 6.2
Centerpiece, Arturo Tedeschi (2011).





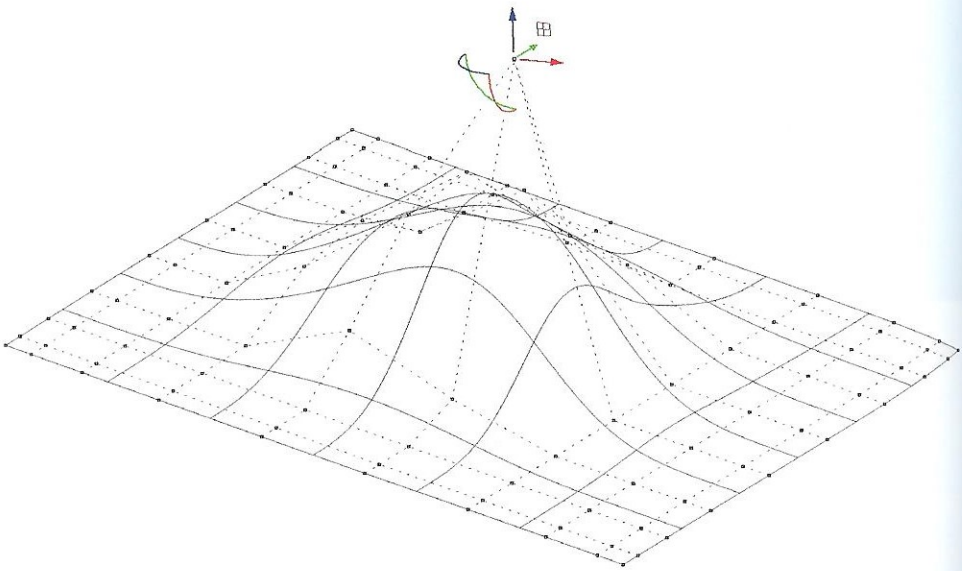
6.1 NURBS and Polygon Meshes

Polygon modeling is conceptualy different from NURBS modeling. Each method has strengths and weaknesses when defining tridimensional geometry.

Modeling software defines 3D objects by one of two methods:

- Representation by curves and NURBS surfaces;
- Approximation by polygon meshes.

Each curve, surface or freeform geometry discussed so far, whether created in Rhino or by Grasshopper components, are NURBS geometries (see chapter 3). NURBS are defined by a mathematical formulation which connects degree, control points, weights and knots in order to accurately represent any geometric object. Moreover, NURBS geometry is a **single geometric entity**. NURBS geometry can be modified by manipulating the object's control points. When control points are displaced or their relative weight is changed, a local deformation will result. The computation of coordinates is based on one or more parameters: *t* for curves and *u* and *v* for surfaces. NURBS surfaces, since they are based on mathematical formulations, define flexible and accurate freeform geometry; which are **inherently smooth**.



Alternatively, **polygon meshes** are not strictly defined by mathematical logic or curves. Polygon meshes are not single geometric entities, but they are made through a **set of adjacent polygons** which determines the global shape. A mesh is not actually smooth, the greater the number of polygon the smoother the mesh.

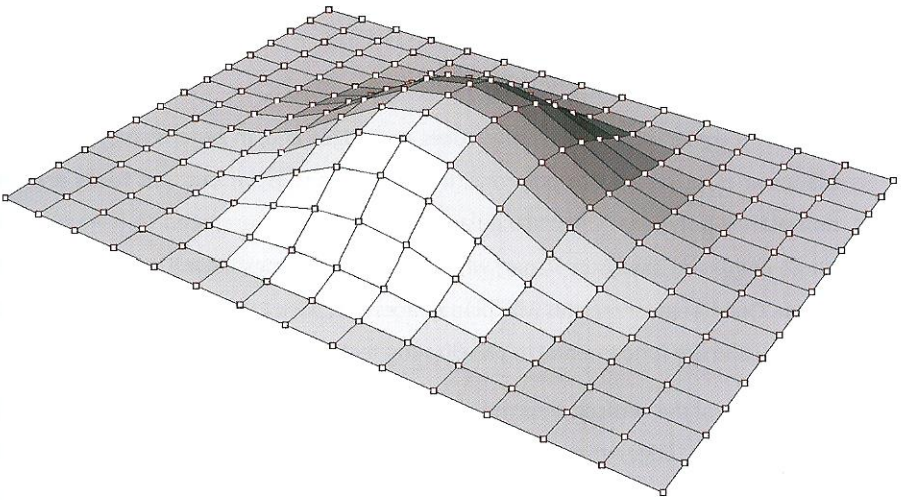
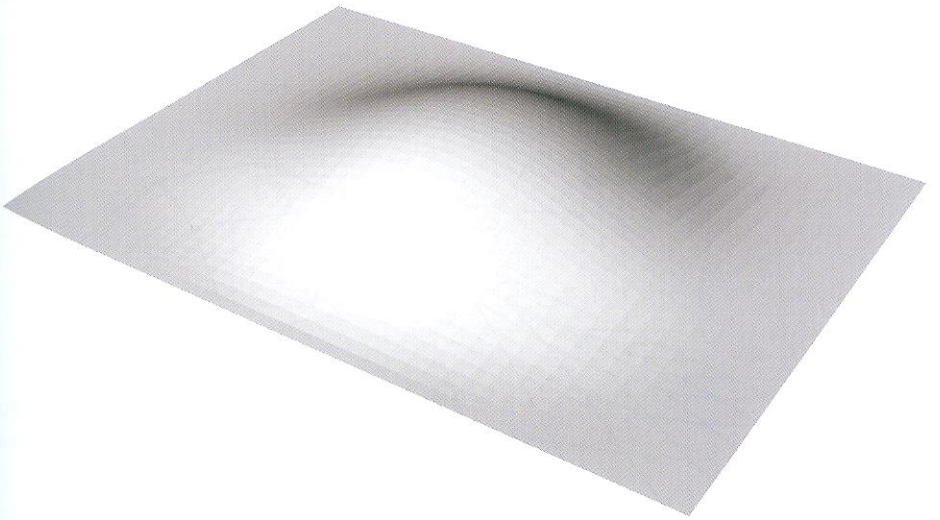


Figure 6.3
A polygon mesh is not actually a smooth geometry: the smaller are the polygons (below) the smoother is the mesh.

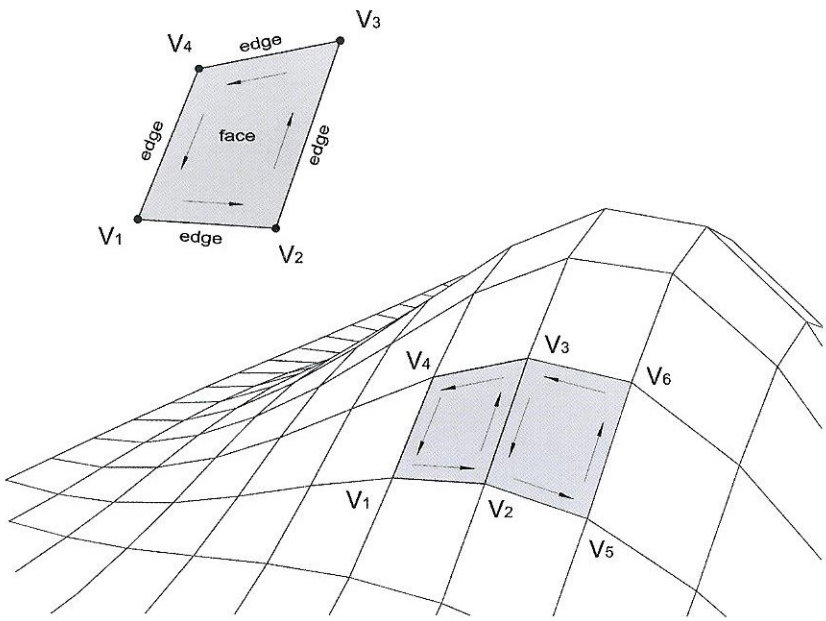


6.2 Polygon meshes

The elements of a polygon mesh are:

- **Vertices**: are points defined in the World Coordinate System by relative coordinates (x, y, z). The vertices provide positional information and affect the shape of the polygonal geometry;
- **Edges**: are segments that connect vertices;
- **Faces**: are constituted by a set of vertices and edges and, in general, they are formed by triangular or quadrangular polygons. It's important to point out that **faces are usually non-planar polygons**. Only triangular meshes can be guaranteed to have planar faces (three points are always coplanar) while quadrangular meshes are not guaranteed to be planar. Quadrangular meshes with planar faces are called **PQ Meshes** (Planar Quadrilateral Meshes) and are fundamental to panelization techniques of freeform architecture.

The connection-order of the vertices determines the orientation of the face, distinguishing the front face from the back face. Edges that connect vertices in the counterclockwise direction are front facing. Adjacent faces are defined as **compatible** if they have the same orientation. A mesh is called **orientable** if it is constituted by compatible faces.



For each face a **normal vector** can be defined. A front-facing mesh has an **outgoing** normal vector, while a back-facing mesh has an **ingoing** normal vector. As follows, an orientable mesh has normal vectors oriented in the same direction.

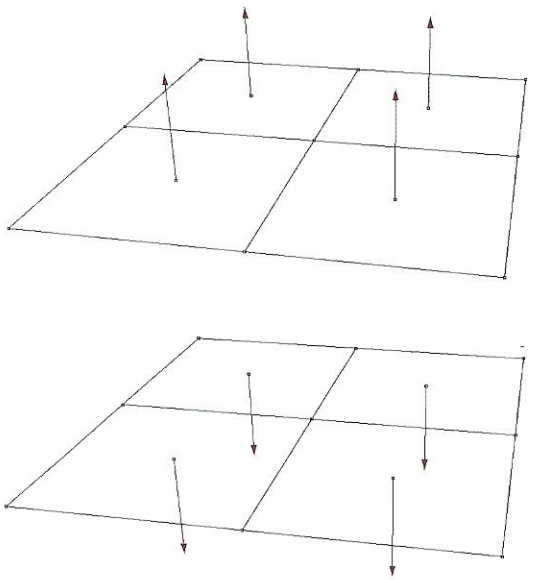


FIGURE 6.4
Two orientable meshes: the normals are oriented according to the same direction.

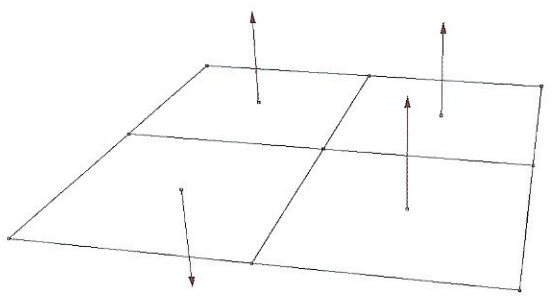


FIGURE 6.5
A non-orientable mesh.

6.2.1 Geometry and topology

A mesh is basically a geometric entity defined by a collection of points that determine its **geometry**, as well by the connection-logic that determine its **topology**. For instance, in figure 6.5, two mesh-boxes having the same vertices but a different connection-logic (topology). Conversely, image 6.6 illustrates two mesh-boxes with the same topology, but different geometry.

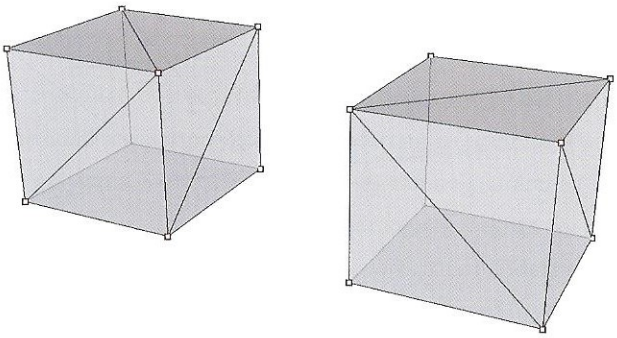


FIGURE 6.6

Two mesh-boxes with the same geometry but different topology.

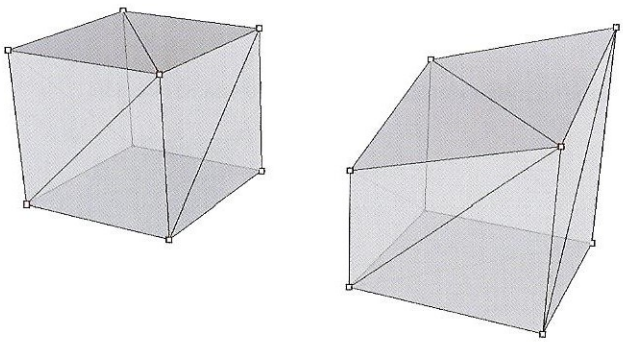


FIGURE 6.7

Two mesh-boxes with the same topology but different geometry.

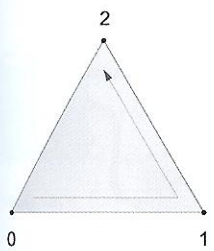
6.3 Creating meshes in Grasshopper

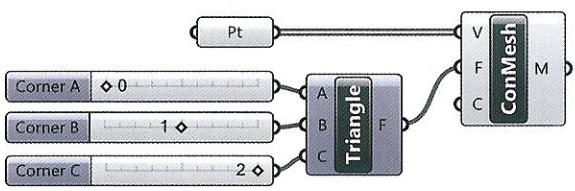
Several Grasshopper components are used to create meshes. Excluding mesh primitives (plane, box, sphere, ect.) which can be defined using a single component, there are three main strategies to define meshes in Grasshopper:

- Creating meshes by topology;
- Creating meshes by triangulation;
- Creating meshes by a NURBS to mesh conversion.

6.3.1 Creating meshes by topology

A mesh is defined by a collection of vertices connected sequentially, determining the meshes **topology**. The most basic mesh is a single triangular face with edges defined by three vertices: 0, 1, 2. The component *Construct Mesh* (Mesh > Primitive) is used to build a mesh. The component has two main inputs: (V) and (F). The V-input is defined by a set of three vertices, and F-input is defined by the component *Mesh Triangle* (Mesh > Primitive) which specifies the connection order i.e. the mesh topology. A is the first vertex specifying point index 0, B is the second vertex specifying point index 1 and C the third vertex specifying point index 2.



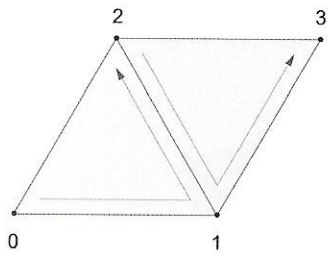


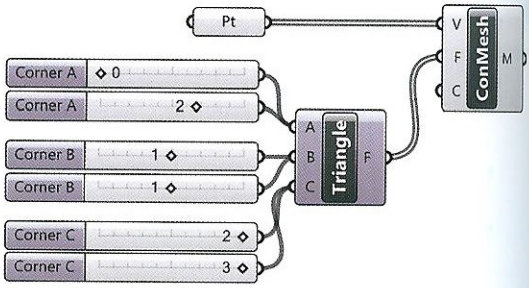
The above definition describes a three vertex triangular mesh with an anticlockwise direction. To visualize the mesh edges the *Preview Mesh Edges* option must be activated from the *Display* menu. If a fourth vertex is added to the list of set points, a new face will be generated by properly defining the mesh topology. In order to create two compatible faces the connection-order has to be set as follows:

Face 1: 0, 1, 2

Face 2: 2, 1, 3

In this case the A-input, B-input and C-input are all fed by two values as shown in the following algorithm:



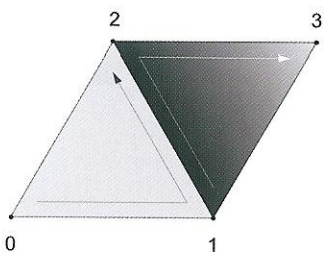


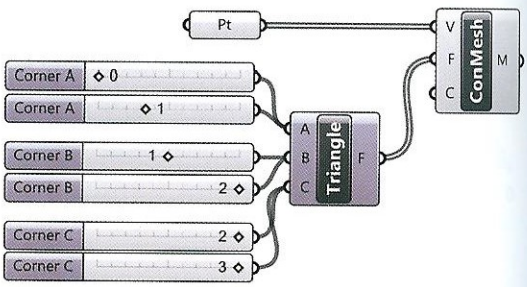
If the second face vertices are connected in the clockwise direction a non-orientable polygon mesh will result. Meaning, if the connection-order is set as follows, the face will be oriented in the opposite direction:

Face 1: 0, 1, 2

Face 2: 1, 2, 3

Grasshopper displays a non-compatibility warning by chromatically differentiating that two adjacent faces. The shading graphically represents the orientation: front (clear face) and back (dark face).





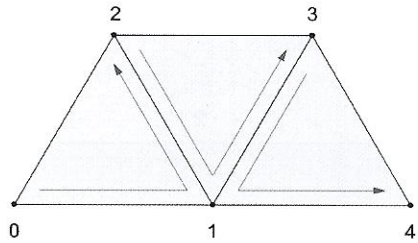
As follows, a triangular mesh with three faces can be created by setting five vertices topologically defined as follows.

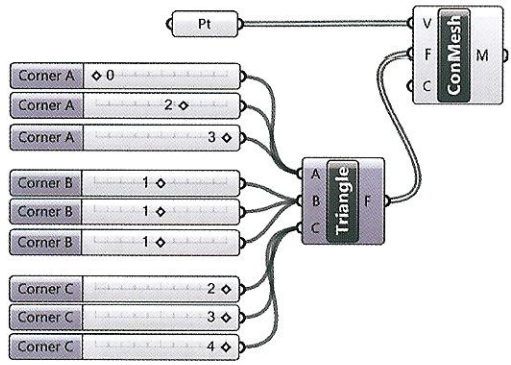
Face 1: 0, 1, 2

Face 2: 2, 1, 3

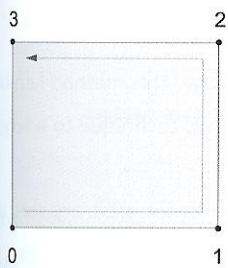
Face 2: 3, 1, 4

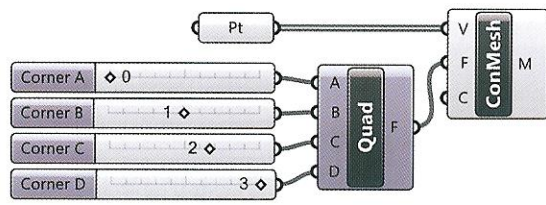
Since the three faces have anticlockwise-orientation an orientable mesh made up of three triangular faces results.





To create a quadrangular mesh face, the input (F) of the *Construct Mesh* component must be satisfied by the component *Mesh Quad* (Mesh > Primitive). The *Mesh Quad* component has four inputs to specify face topology and operates similar to the *Mesh Triangle* component. For example, to create a quadrangular mesh face, four vertices are set and connected according to the anticlockwise topology rule.

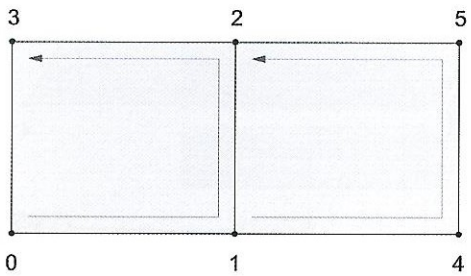


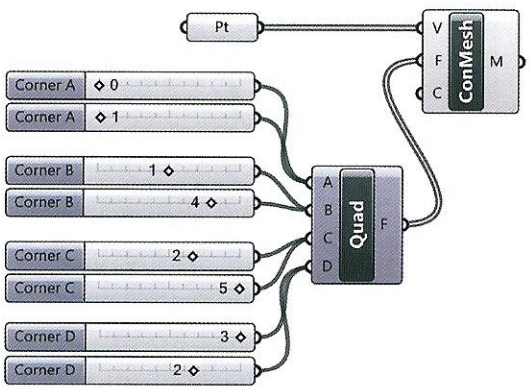


By adding two additional vertices to the set list of points, two mesh faces can define an orientable mesh by specifying the connection topology as follows:

Face 1: 0, 1, 2, 3

Face 2: 1, 4, 5, 2





6.3.2 Creating meshes by triangulation: Delaunay algorithm

Mesh topology specifies the connection order of vertices to define a mesh face. This method can be applied to small number of points or repeating logics. However, applying this technique to a large number of points is often not appropriate.

Alternatively, *triangulation algorithms* are used to define triangulated meshes for arbitrary sets of points. **Triangulation algorithms minimize differences between triangles in order to get a pseudo-regular mesh, avoiding skinny triangles.** In fact, skinny triangles can lead to inaccurate results if the mesh is used to calculate simulations or conduct analysis such as particle-spring

systems (see chapter 9) or finite element method (FEM) analysis.

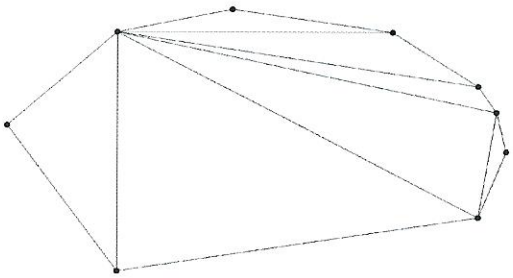
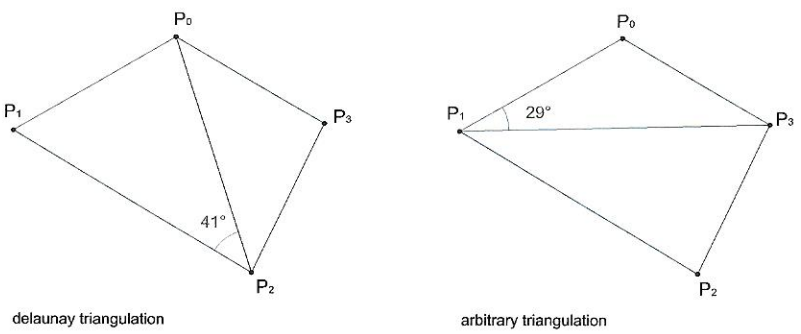


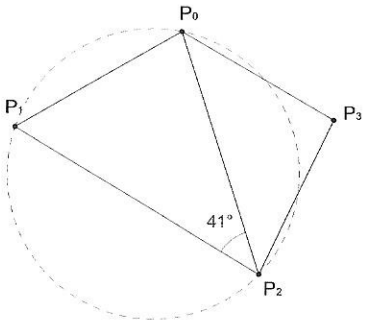
FIGURE 6.8
The image shows a mesh formed by skinny triangles. This kind of mesh is usually considered "bad" for simulations or analysis.

Several algorithms can be used to compute triangulations. The ***Delaunay algorithm***, named after the mathematician Boris Delaunay, also known as ***Delaunay triangulation*** is one of the most popular methods because of the outputs geometric properties.

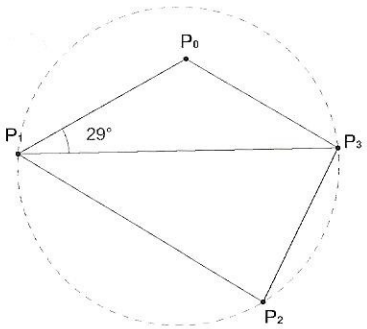
The *Delaunay triangulation* connects an arbitrary set of points by a geometric procedure that maximizes the minimum angle of triangulation, so that the resulting mesh tends to avoid "thin" triangles. The following figure compares the *Delaunay triangulation* logic to an arbitrary triangulation, applied to the same set of points.



The *Delaunay algorithm* creates the triangulation "choosing" triplets of points that meet the following rule: once defined the circumscribed circle (*Delaunay* circle) through three vertices of the same face (P_0, P_1, P_2) the *Delaunay* circle through P_0, P_1, P_2 cannot contains other points inside.

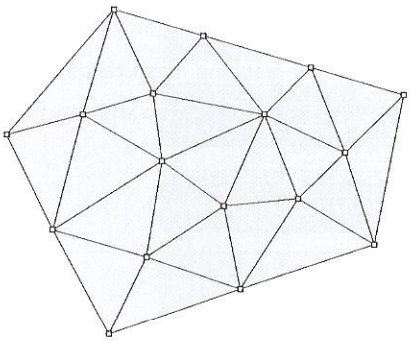


delaunay triangulation



arbitrary triangulation

The component *Delaunay Mesh* (Mesh > Triangulation) generates a triangular mesh according to the *Delaunay algorithm* for a series of defined points (P). The component's input (Pl) is the plane or planes where the algorithm operates.



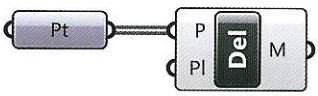
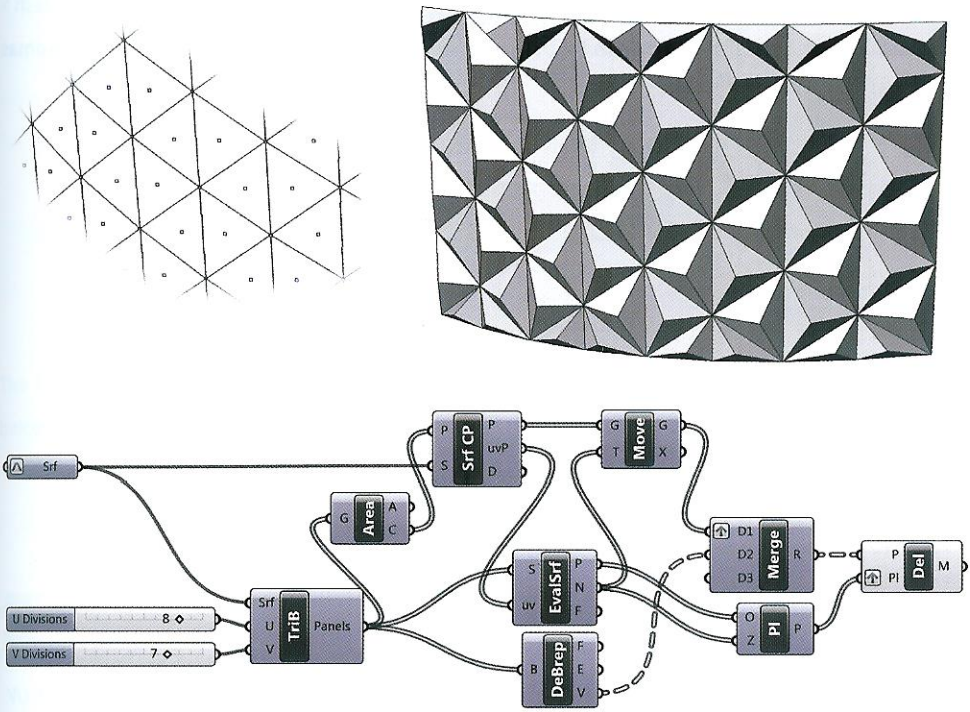


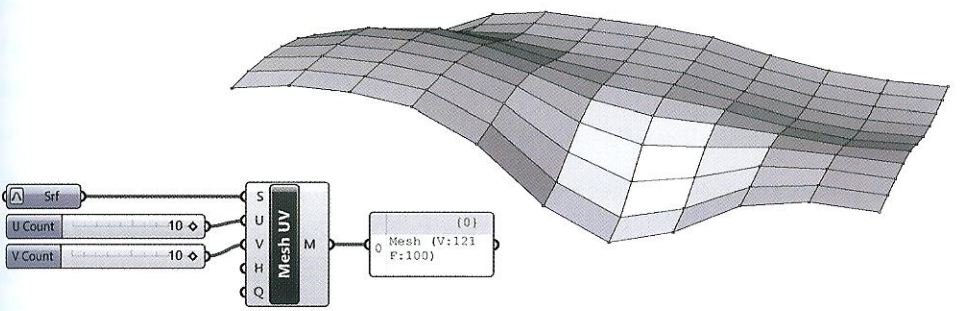
FIGURE 6.9
The Delaunay Mesh algorithm applied to a planar set of points.

In the following page you can see a tridimensional pattern get from a freeform surface using a *Delaunay algorithm*. As first step a triangular grid is created using the *Triangle panels B / TriB* component provided by the **Lunch Box** plug-in. For each triangle the centroid is calculated and subsequently moved according to the surface's normal vectors. The *Delaunay algorithm* is performed on the three vertices of each triangle merged with the translated centroids. The *Delaunay Mesh* component composes the vertices using the defined planes into a collection of meshes with 3 faces.

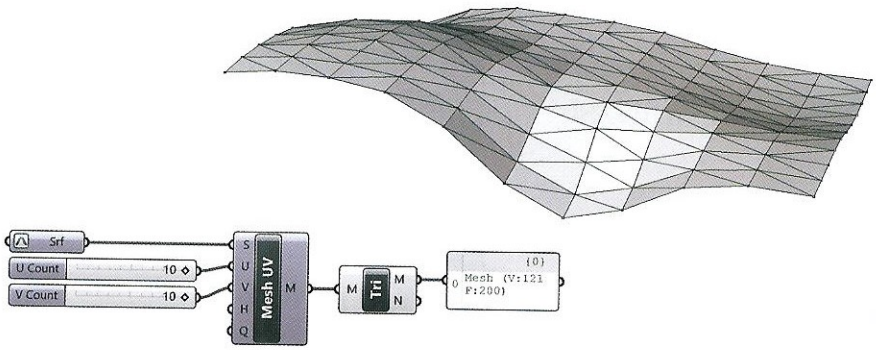


6.3.3 Creating meshes by a NURBS to mesh conversion

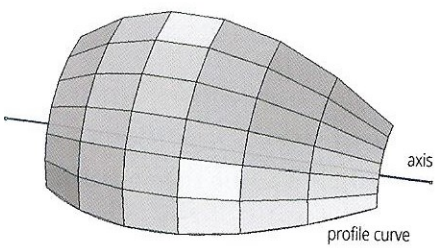
Meshes can also be defined from a base NURBS surfaces. The component *Mesh Surface /Mesh UV* (Mesh > Util) converts a NURBS surface into a **Quadrangular Mesh** by specifying the number of quads in U and V directions. To display the mesh edges enable the *Mesh Edges* option (Display > Preview Mesh Edges).



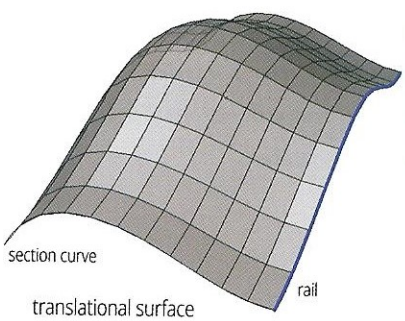
A **triangular mesh** can be obtained from a quad mesh using the *Triangulate* component (Mesh > Util) available after installing the **Mesh Edit plug-in, developed by [uto] (Ursula Frick and Thomas Grabner)**.



As discussed in section 6.2, triangular meshes are always planar, while Quadrangular meshes are planar only under certain circumstances. For instance, rotational and translational NURBS surfaces return **planar quadrilateral** meshes (or **PQ Meshes**) when converted using the component *Mesh UV*.

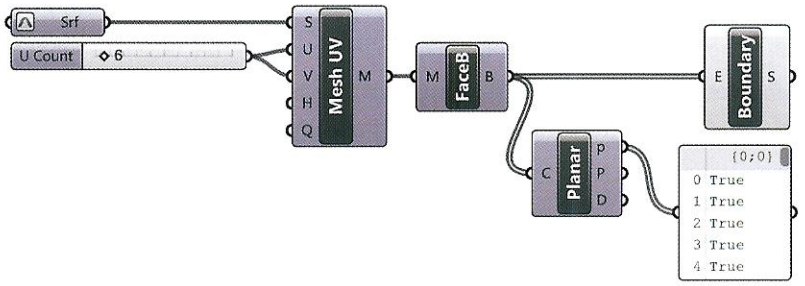


rotational surface

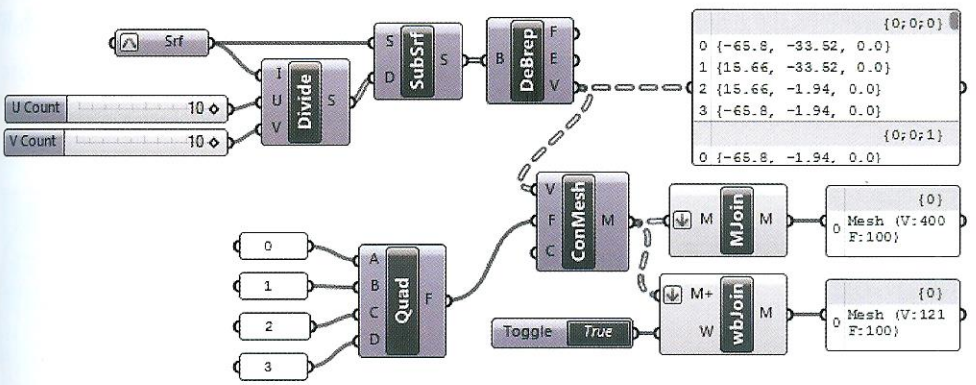


translational surface

To test for planarity the meshes edges are extracted using the component ***Face Boundaries*** (Mesh > Analysis) returning a boundary polyline for each face. Each boundary polyline is tested for planarity using the component ***Planar*** (Analysis > Curve). Alternatively, the *Boundary Surfaces* component can be used to test for planarity, since by definition the component will draw a surface only if the input curves form a planar boundary.



The NURBS to Mesh conversion can also be performed by specifying topology, as the definition below illustrates.

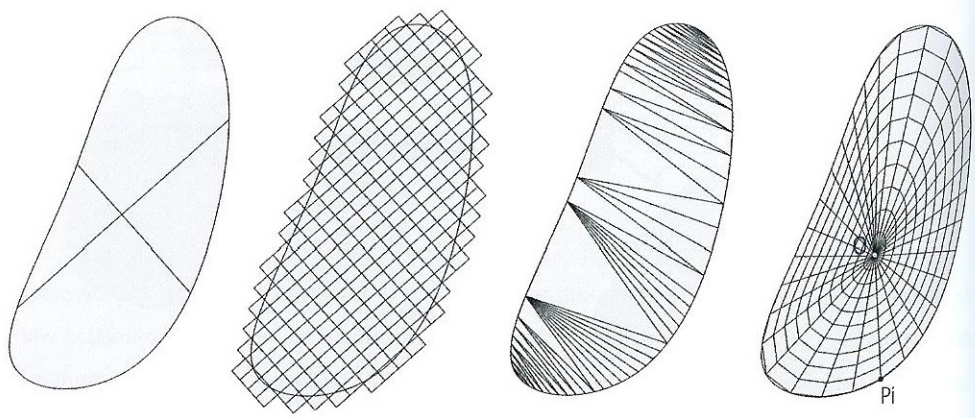


An initial A set surface is divided into sub-surfaces using the *Isotrim* component. The *Deconstruct Brep* component extracts the four vertices from each sub-surface, to define (in conjunction with a specified topological order) the *Construct Mesh* component, as discussed in 6.3.1. The result is set of quad meshes which can be joined into a single mesh by the component *Mesh Join* (Mesh > Util). Alternatively, *Join Meshes and Weld* (Wb > Extract) can be used to join meshes and weld coincident vertices, as specified by a Boolean toggle. *Join Meshes and Weld* is a part of the Weaverbird plug-in, discussed in next section.

NURBS to mesh conversion is compatible with **untrimmed** surfaces (see 3.5), since untrimmed surfaces contain a rectangular isogrid that can be converted into a set of quad meshes. Unlike untrimmed surfaces, **trimmed** *surfaces* are not compatible with the NURBS to mesh conversion method. Trimmed surfaces converted using the *Mesh Surface* component will return a polygon mesh

that approximates the original boundary geometry. Trimmed surfaces converted using the *Mesh Brep* (Mesh > Util) component will return a polygon mesh that preserves the original boundary geometry but allows for thin triangles. The NURBS to mesh conversion method for a trimmed surface must be considered on case by case basis; a possible strategy to convert a trimmed surface into a mesh is summarized as follows:

1. Adding an arbitrary point O;
2. Dividing the surface's boundary into N parts getting N points P_i;
3. Creating a set of lines from O to the subdivision points P_i;
4. Dividing the lines into equal parts and connecting the resulting points with a set of polylines;
5. Splitting the initial surface using the network of lines obtained in step 3 and step 4, to generate a set of trimmed sub-surfaces;
6. Extract the sub-surfaces' vertices and create the mesh relying on topology method.



Trimmed Surface Mesh Surface Mesh Brep

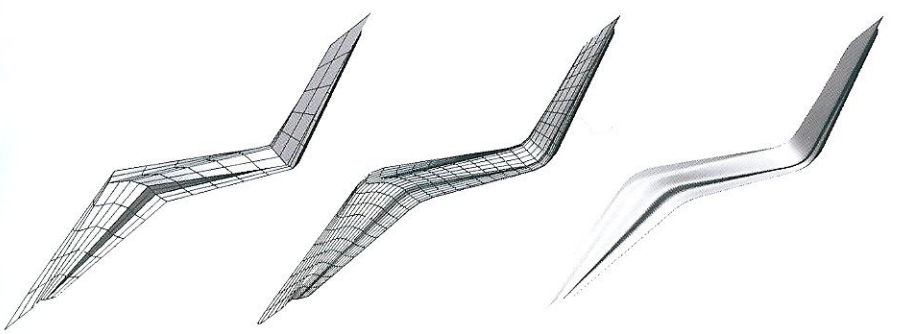
6.4 SubD in Grasshopper: Weaverbird plug-in

The basic idea behind Subdivision Surfaces is that we can get a smooth polygonal surface (limit surface) associated with any given input mesh. SubD algorithms process an input mesh and generate an output mesh that is closer to the limit surface. Depending on the type of the input mesh (triangular, quadrilateral etc.) we must use a different SubD algorithm. We will particularly focus on two method:

- Loop subdivision algorithm;
- Catmull-Clark subdivision algorithm.

Triangular meshes are commonly subdivided using the *Loop Subdivision* algorithm, while quadrilateral meshes are generally subdivided using the *Catmull-Clark* algorithm. Both algorithms are part of the plug-in **Weaverbird** developed by Giulio Piacentino[15]. The plug-in is available for free download from the following website: www.giuliopiacentino.com. After installation a new tab (Wb) will be accessible. Weaverbird's toolbar allows users to access components to perform subdivisions and transformations as well as define polygonal primitives and extract main elements from a mesh.

FIGURE 6.10
Weaverbird components toolbar.



NOTE 15
Giulio Piacentino, architect, graduated from Politecnico di Torino, continued his education at the University of Technology of Delft, Holland, then began a co-operation with NIO Architecten. He currently develops for McNeel and teaches at geometrydepth.com. He has trained students in geometry all around Europe.

6.5 Subdivision of triangular meshes: Loop algorithm

The Loop algorithm is an iterative subdivision method for **triangular** meshes, developed by Charles Loop[16] in 1987. For every edge in the source mesh, the algorithm adds a new vertex at the mid point. The mid points are connected and every triangle is replaced by 4 sub-triangles. Each iteration of the Loop algorithm approaches the limit surface, i.e. the surface defined by infinite iterations. The number of iterations is counter proportional to the change in geometry. As a result, only a few iterations are usually required to approximate a limit surface.

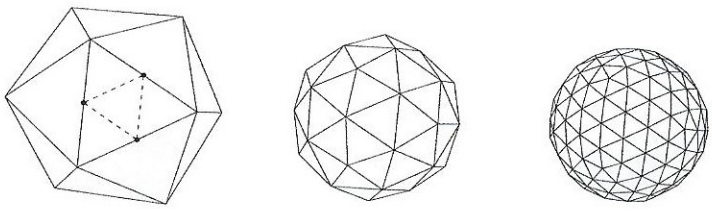
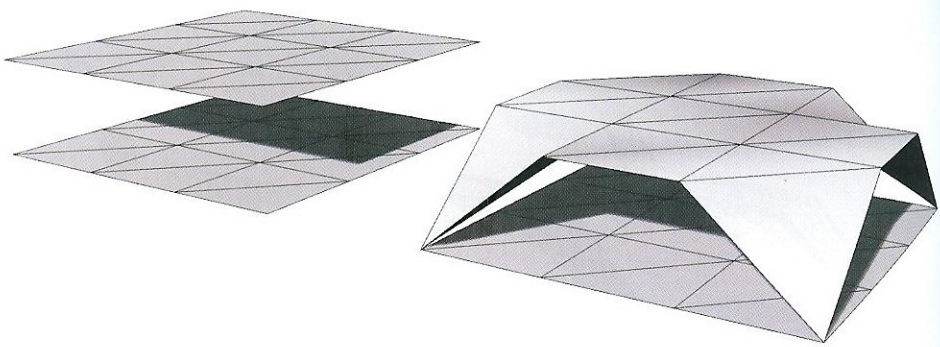


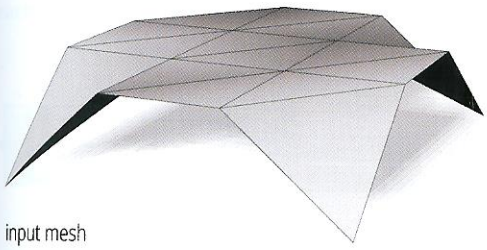
FIGURE 6.11
Loop subdivision scheme. Each triangle is divided into 4 sub-triangles, adding new vertices in the middle of each edge.

The Loop algorithm is performed by the component *wbLoop* (Wb > SubD). The M-input is the mesh to subdivide, the L-input the number of subdividing iterations and the S-input specifies how to treat the naked edges of the input mesh. The number of iterations can vary from 1 to 3. Generally, a simple and schematic input mesh will yield a more refined and polished output mesh.

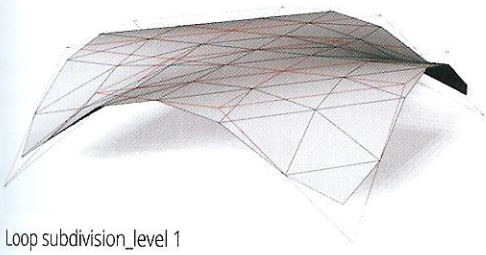


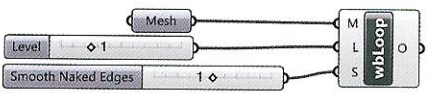
NOTE 16
C. Loop, "Smooth Subdivision Surfaces Based on Triangle", M.S. Mathematics thesis, University of Utah, 1987.

For example, a triangular mesh set from Rhino (previous image) can be subdivided using the following definition. The L-input varies from 1 to 3 illustrating the iterative refinement of the mesh, S-input is set to 1 (*Smooth*) and as a result the naked edges tend towards a spline.

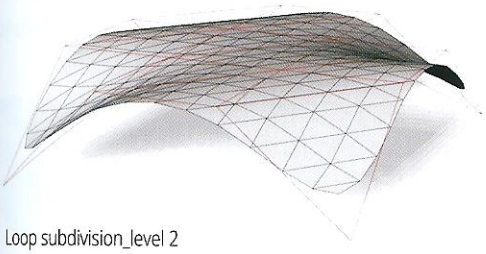


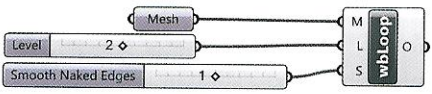
input mesh



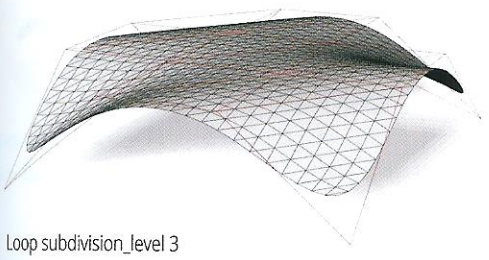


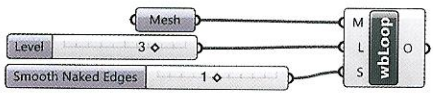
Loop subdivision_level 1





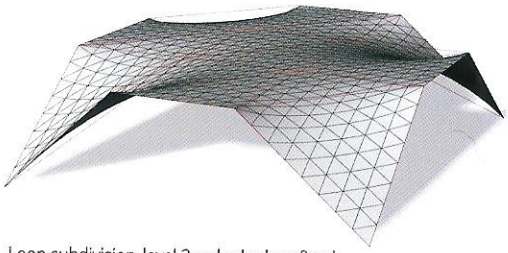
Loop subdivision_level 2

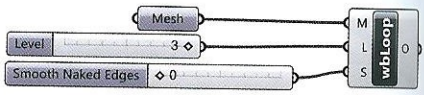




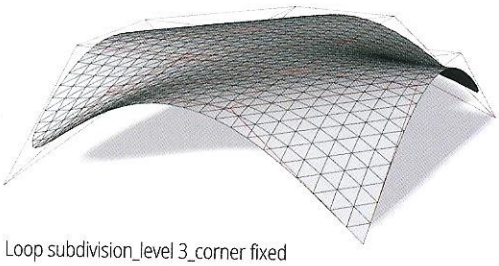
Loop subdivision_level 3

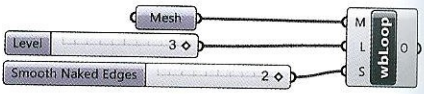
If the S-input of *Loop* component is set to 0 (*Fixed*) the naked edges will be fixed and do not move from the original position, 1 (*Smooth*) the naked edges tend towards a spline, or 2 (*Corner Fixed*) the naked edges tend towards a spline and two vertices are fixed. The three types of naked edge options are illustrated in the previous example or below.





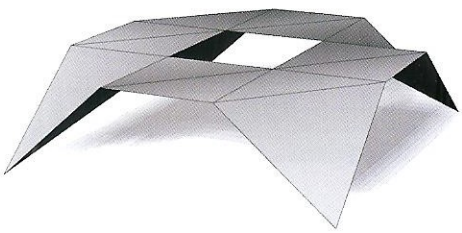
Loop subdivision_level 3_naked edges fixed

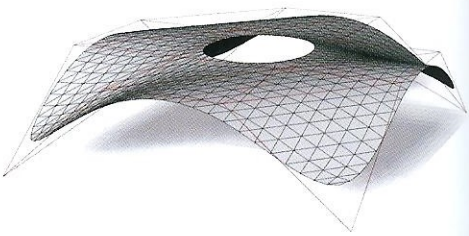




Loop subdivision_level 3_corner fixed

As we previously discussed a refined mesh can be generated from a simple input mesh using the Loop subdivision algorithm. The Loop SubD algorithm also works for meshes with holes. Once the mesh is subdivided the hole, as demonstrate below, will become pseudo rounded.





6.6 Subdivision of quadrangular meshes: Catmull-Clark algorithm

The Catmull-Clark is an iterative subdivision algorithm for **quadrangular** and **triangular** meshes, developed by Edwin Catmull and Jim Clark[17] in 1978. The algorithm iteratively adds new vertices for each face to approximate a smooth surface.

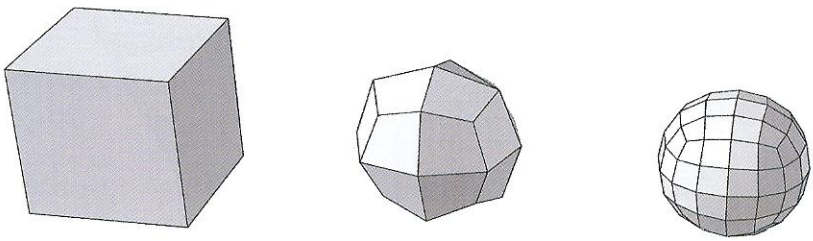


FIGURE 6.12
Catmull-Clark subdivision scheme.

The following image, displays the final result of converting the tridimensional NURBS skin – created in the chapter 5 – into a smooth and continuous polygon mesh using the Catmull-Clark algorithm. As explained in 5.2.1 the final skin is composed of three sets of surfaces: the frame (*surfaces set 01*), the lateral faces (*surfaces set 02*) and the lower faces (*surfaces set 03*). Before subdividing the surfaces, each of these surfaces must be converted into a mesh using the component *Mesh Surface*.

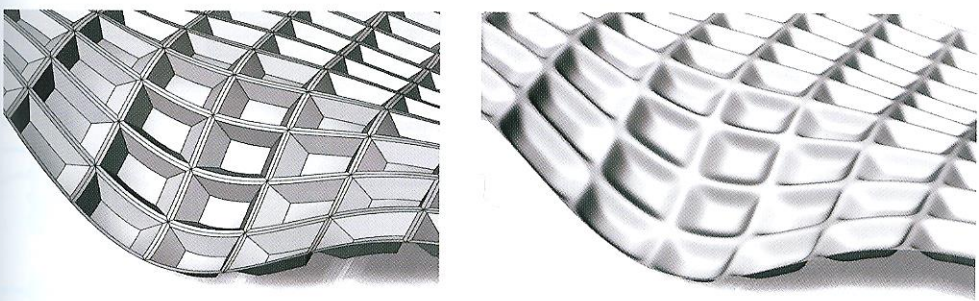
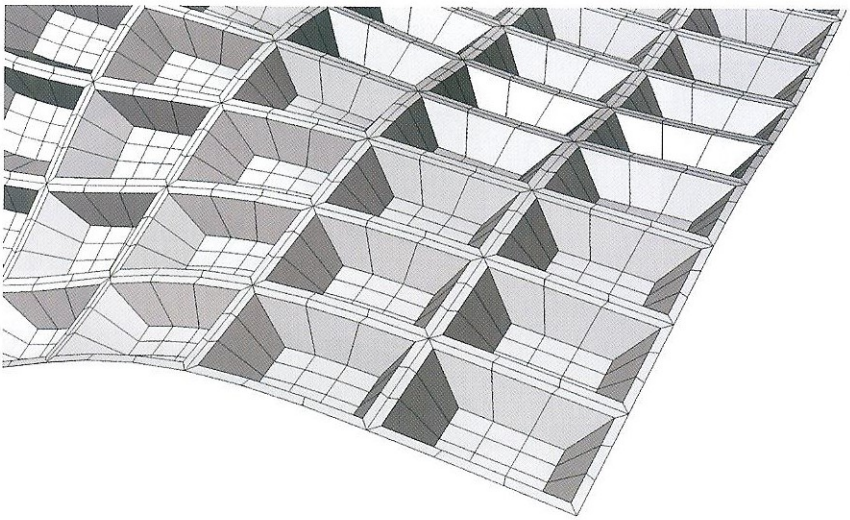
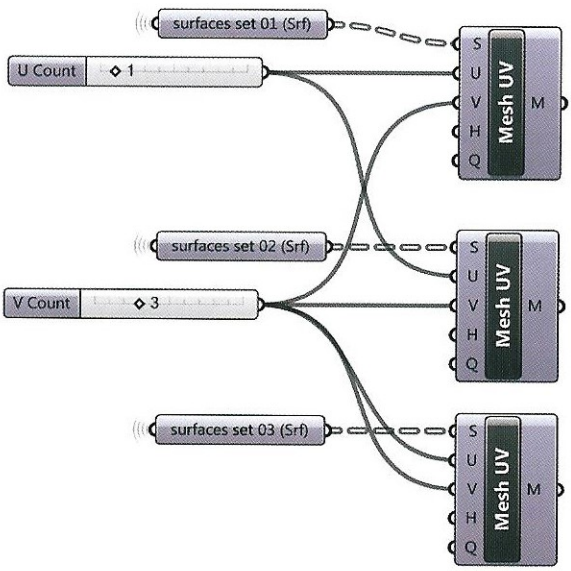


FIGURE 6.13
Smoothing a tridimensional skin by a subdivision strategy based on the Catmull-Clark algorithm.

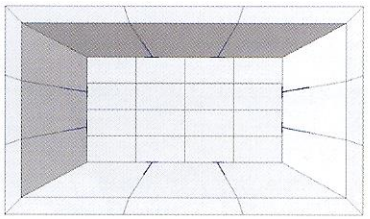
NOTE 17
E. Catmull and J. Clark, *Recursively generated B-Spline surfaces on arbitrary topological meshes*, Computer Aided Design, 1978.

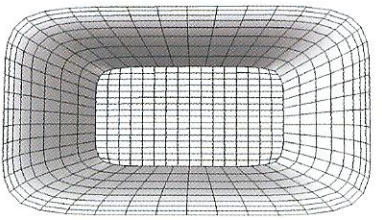
The component *Mesh Surface* requires a specified number of quads in U and V direction. When converting a single surface the number of U and V subdivisions is arbitrary. Conversely, for multiple merged surfaces, **the number of quads in U and V direction should be set so that edges always meet in the vertices, avoiding T-nodes**.



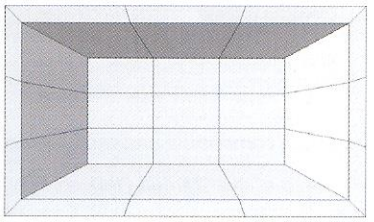


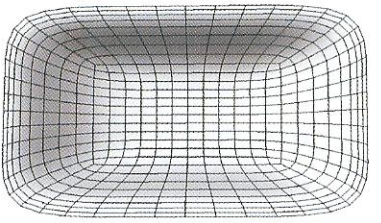
When a set of contiguous meshes have T-nodes (image below), they cannot be **welded** correctly. Under this condition, the subdivision algorithm operates on each individual disconnected mesh.



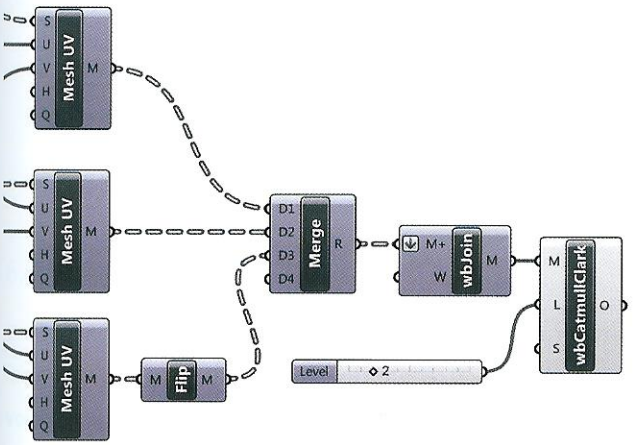


If the U and V count is consistent such that the nodes can be welded, the subdivision algorithm will operate as expected.





If the three mesh nodes align and can be welded, the component ***Join Meshes and Weld*** (Weaverbird > Extract) can be used to create a single mesh that can be smoothed by the component *Weaverbird Catmull-Clark Subdivision* (Weaverbird > SubD).

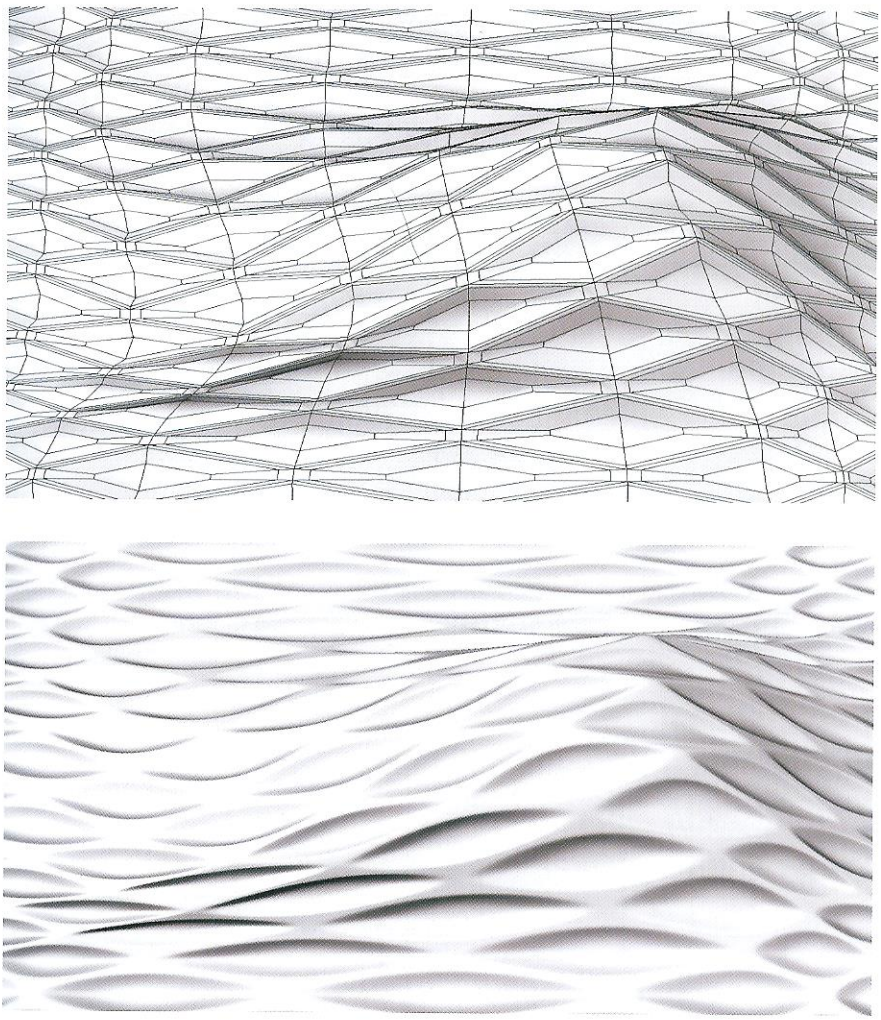


The component *Mesh Flip* (Mesh > Util) reverses the direction of the faces created from the *surfaces set 03*. Mesh Surfaces must be compatible in order to be joined and welded, as discussed in section (6.2). If the surfaces are not compatible a shadow will appear along the mutual edge, graphically displaying an error.

By flipping the faces created from *surfaces set 03*, the faces will be compatible and can be joined, welded and then smoothed. Similar to the *Loop* component the M-input of the *Catmull-Clark* component is the mesh to subdivide, the L-input the number of subdividing iterations and the S-input specifies how to treat the naked edges of the input mesh.

FIGURE 6.14
The final mesh resulting from the subdivision operated by the Catmull-Clark component.

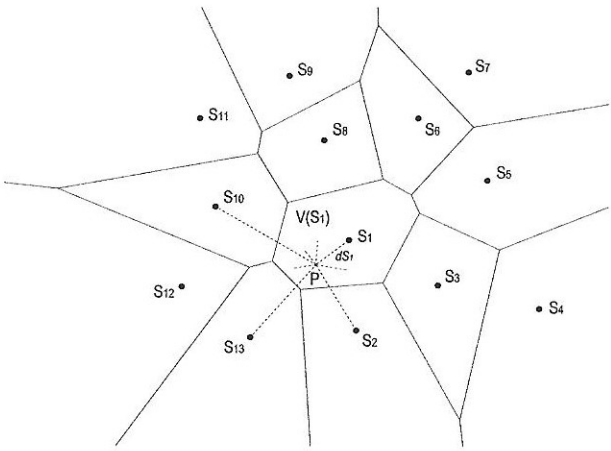
The subdivision strategy developed in the previous example can be applied to the hexagonal skin discussed in 5.2.2. The following image displays the hexagonal skin before and after subdividing using the Catmull-Clark algorithm.



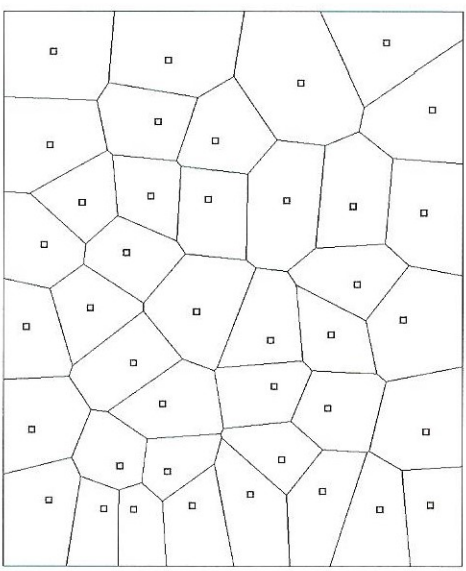
6.6.1 Voronoi skin

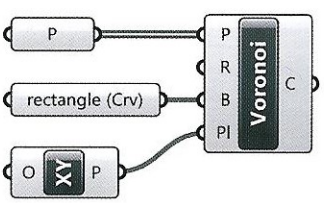
The ***Voronoi diagram***, named after the mathematician Georgii Voronoi, is a decomposition of a metric space according to proximity criteria. Given a specified set of *n* points S (S_1, S_2, ..., S_n), the Voronoi diagram for S is the decomposition of the bidimensional space which associates a region

$V(S_i)$ called Voronoi cell, to each point of S, so that all the points of $V(S_i)$ are closer to S_i than any other point of S. The Voronoi diagram has practical applications in different fields, from physics to city planning (territorial division based on distances from a specific center). Such a diagram has also fascinated designers for its intrinsic beauty.

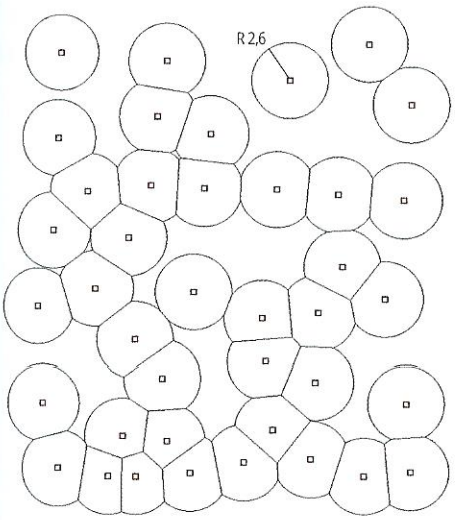


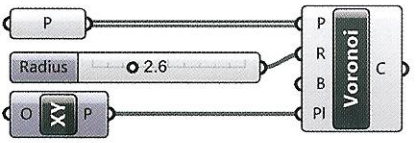
The component *Voronoi* (Mesh > Triangulation) creates a planar Voronoi diagram given a set of points (P), on a plane (Pl) within a containment boundary (B).



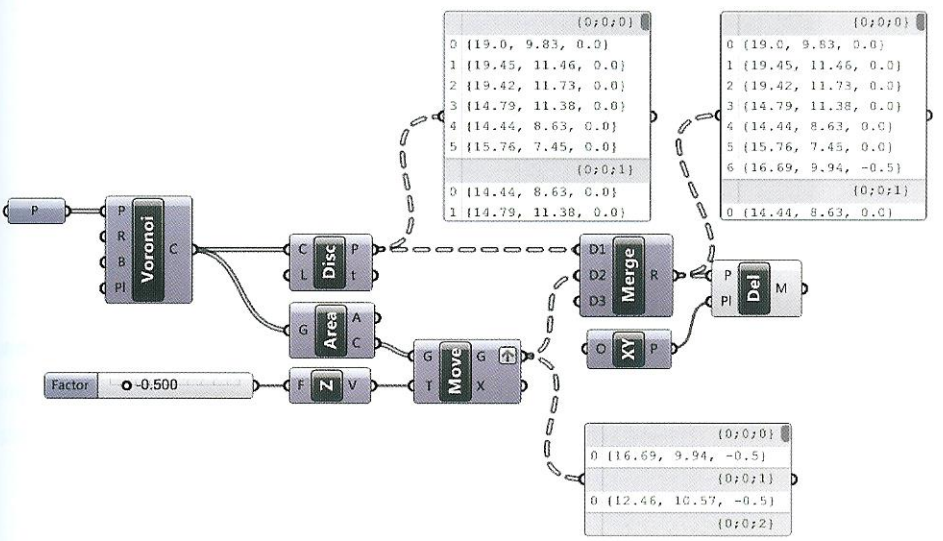


An optional radius R can be set for the diagram, as shown below. If no radius is specified the Voronoi component defaults to an infinite radius input (R) resulting in merged cells.

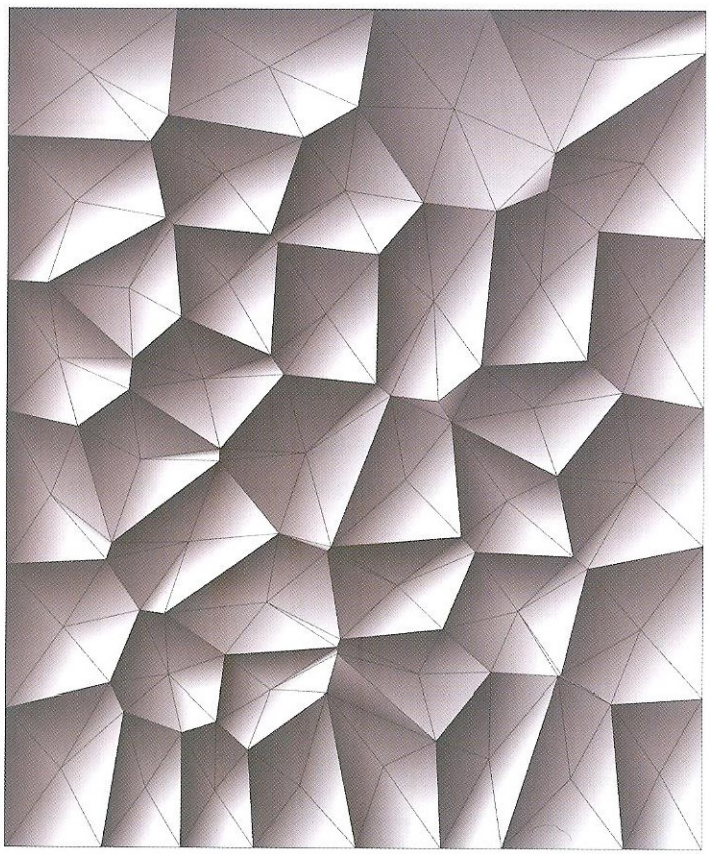




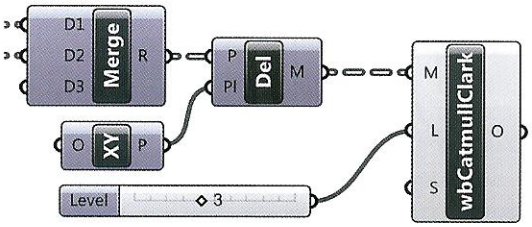
Similar to previous examples a Voronoi skin can be created from a planar Voronoi diagram. The procedure is to define a set of pyramids to smooth by a subdivision algorithm. Each pyramid has a Voronoi cells as a basis.

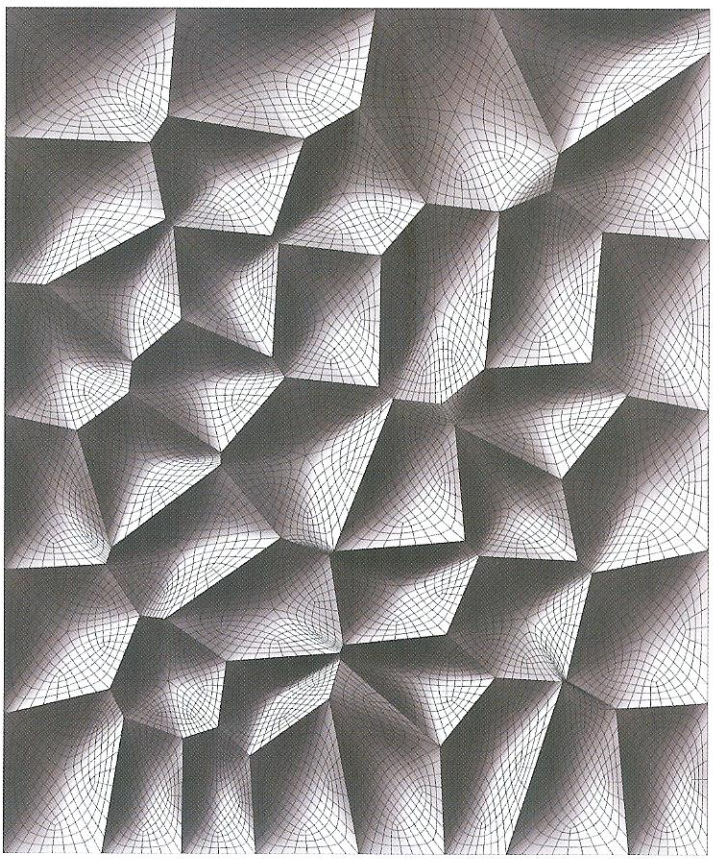


The first step in the Voronoi skin procedure is to use the component *Discontinuity* (Curve > Analysis) to return the discontinuity points or the vertices that define each Voronoi cell. The discontinuity component creates a *branch* for each cell with n items depending on the number of discontinuity points for each cell. Next, the centroid of every cell is translated in the negative Z direction. The discontinuity points and the translated centroid points are merged into a single list, with input (D1) and (D2) set to *graft*.

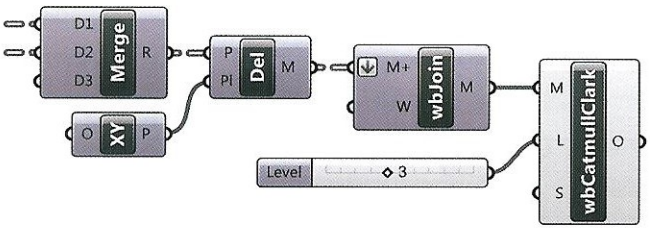


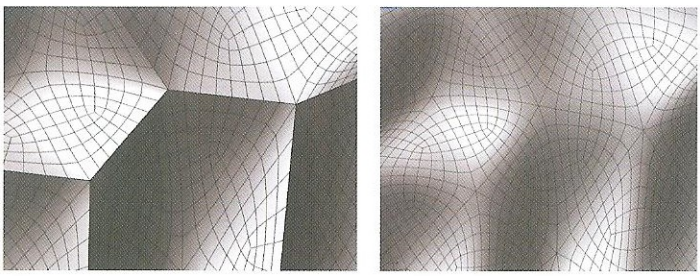
The polygon mesh defined after triangulation is not a single joined and welded mesh, instead it is formed by 40 unique discontinuous meshes.

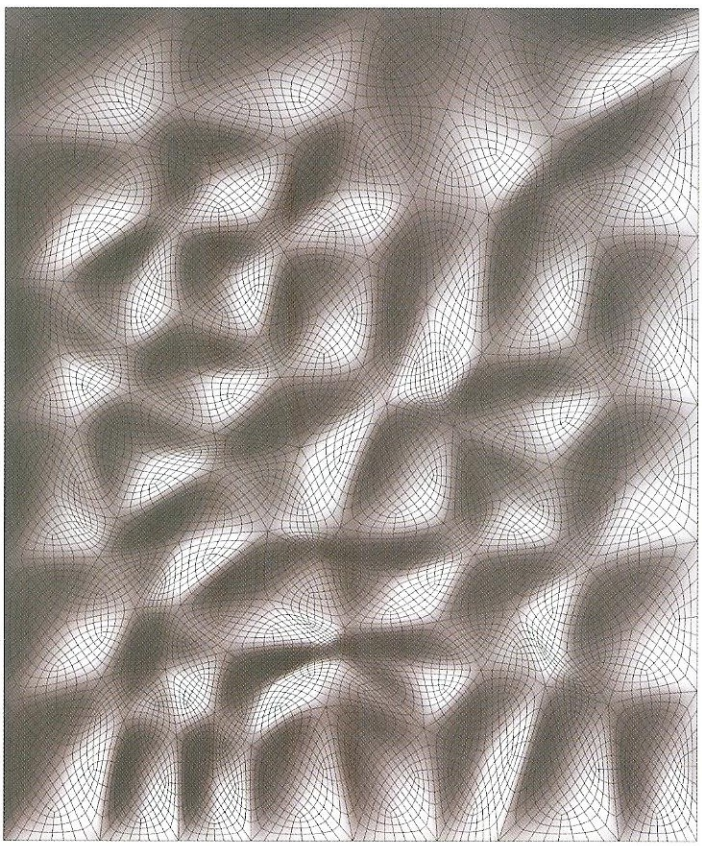




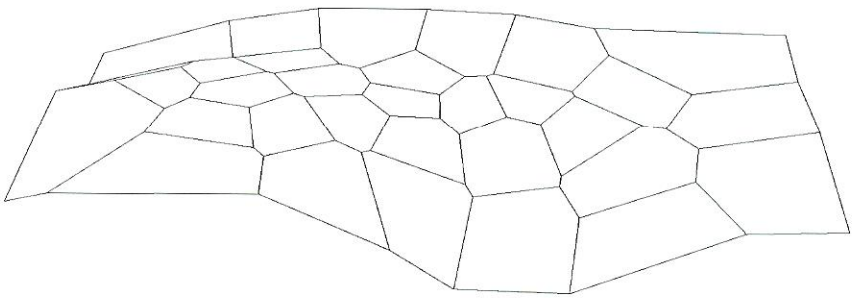
To create continuity between the unique meshes the M-output of the *Delaunay* triangulation component must be joined and welded using the component *Weaverbird Join Meshes and Weld*, with input (M+) set to flatten. The final result is shown in the following images.

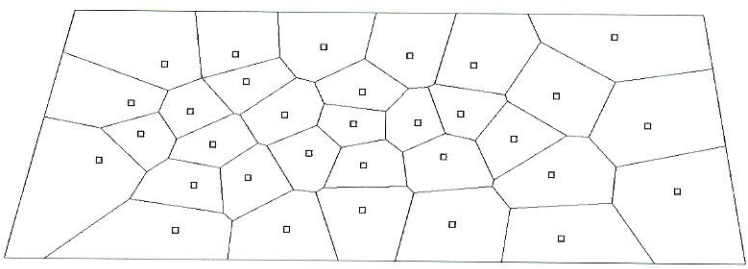

D1
D2
D3
Merge
R
O
XY
P
P
PI
Del
M
M+
W
wbJoin
M
M
L
S
wbCatmullClark
O
Level
3

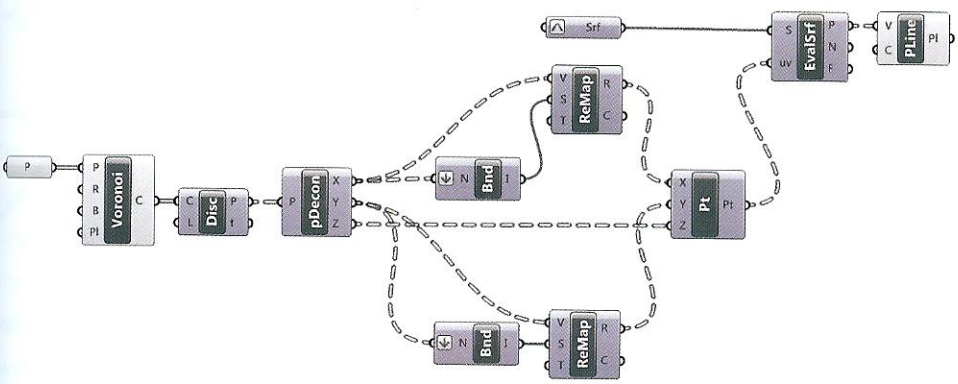




Such a strategy can be also applied to non-planar Voronoi diagrams which can be achieved using plug-in software or relying on several strategies. One of them is based on *remapping* (see 2.5). As first step we create the Voronoi on the plan and then we remap Voronoi's discontinuity points (*Discontinuity* component) in a range between 0 and 1 (*Remap Numbers* component). Since the points of a reparameterized target surface range between the same domain, we can use *Evaluate Surface* and *Polyline* to reproduce the Voronoi diagram onto a freeform surface. The complete algorithm is shown below.

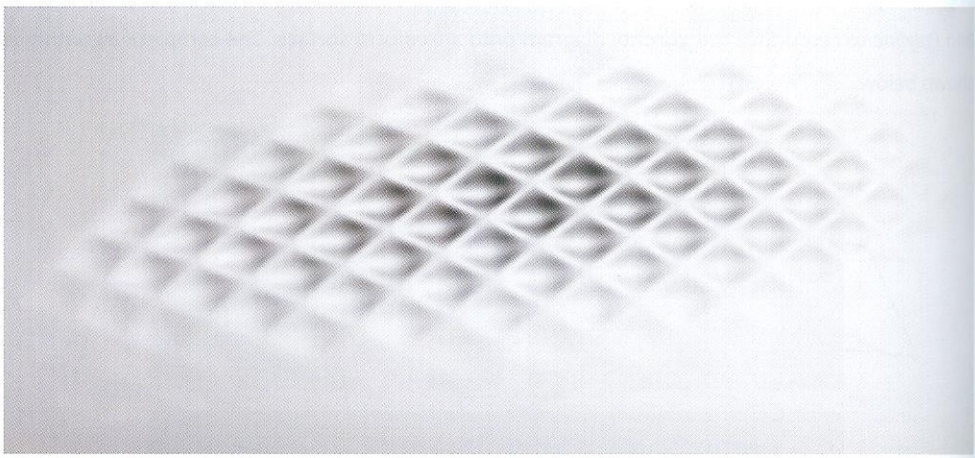




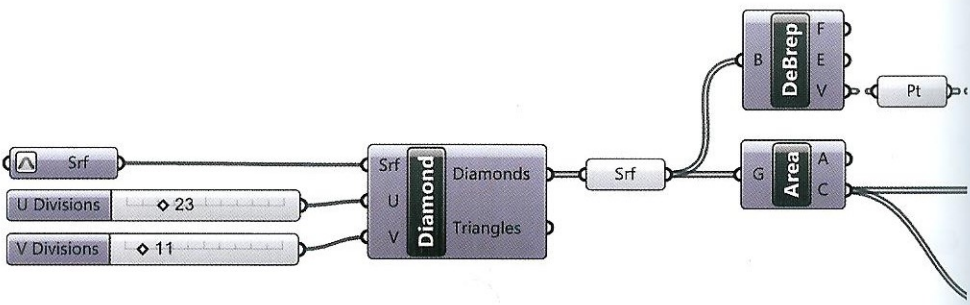


6.6.2 Fading pattern

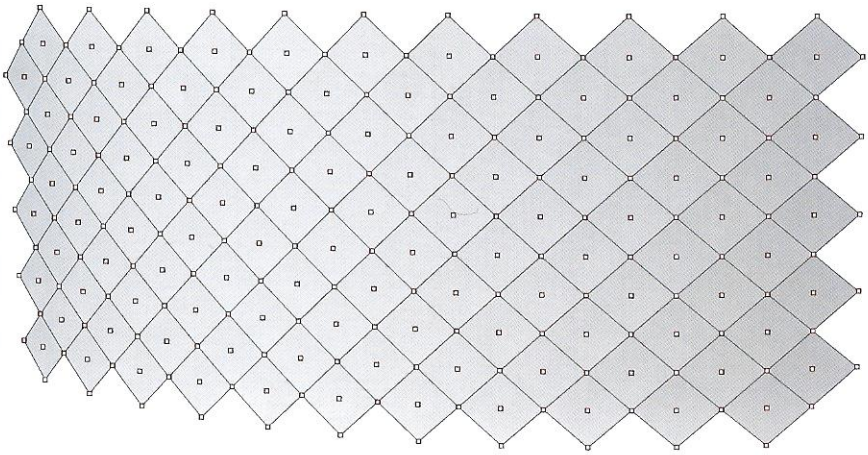
The final example combines a subdivision algorithm with the **Image Sampler** component (4.3.3), to generate a fading tridimensional pattern informed by a grayscale image



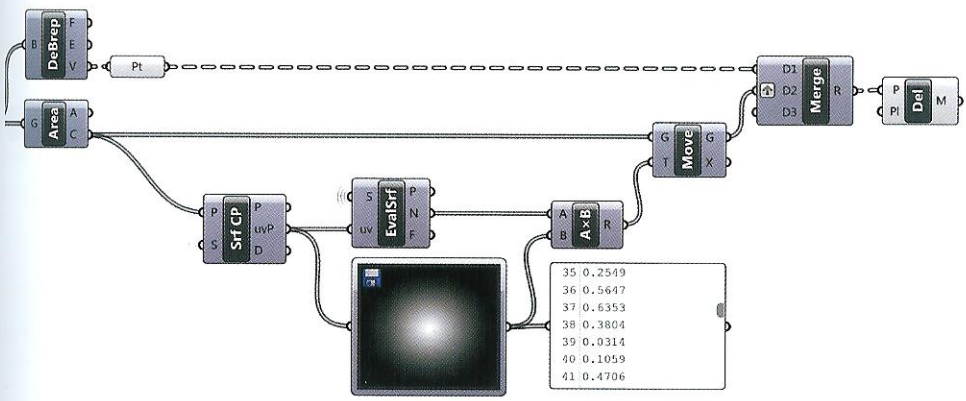
Similar to previous exercises, a bi-dimensional grid is generated from a NURBS surface. The **Lunch Box** plug-in component *Diamond Panels* (LunchBox > Panels) outputs quadrangular and triangular surfaces; the first step in creating a fading tridimensional pattern is to extract the surface vertices and calculate the centroid of every quadrangular (or diamond) surface.



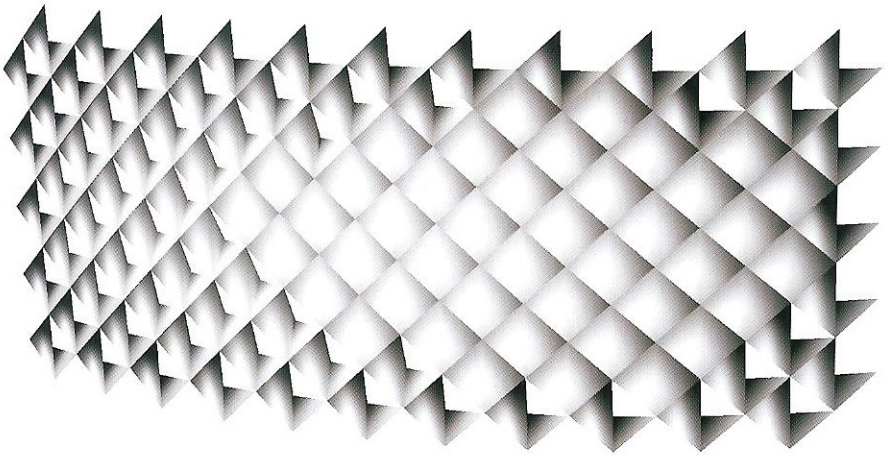
Next, the centroid of each surface is translated according to a scalar factor multiplied by the surface normal unit vector. The four bounding vertices and the translated centroid for each face are merged into a single Data Tree branch. The component *Delaunay Mesh* creates a mesh pyramid from each branch.

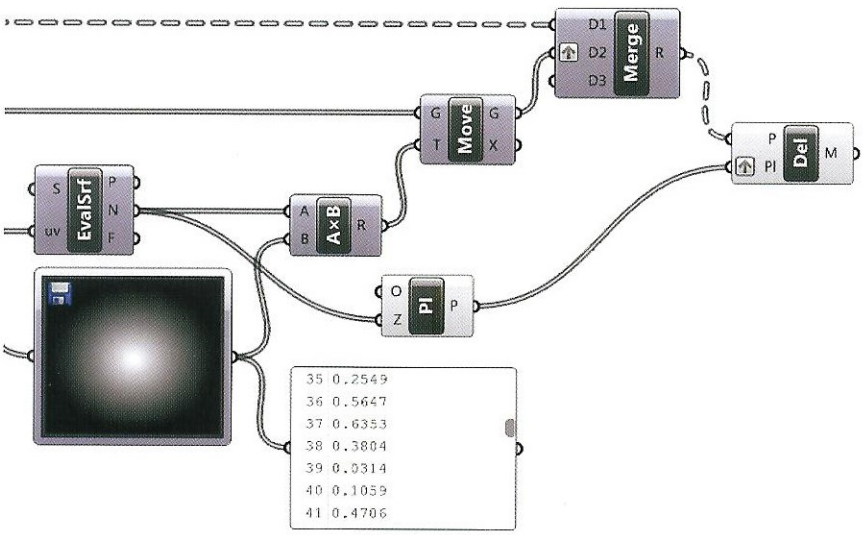


To create a fading tridimensional pattern a non-constant centroid translation is required. As displayed below the *Image Sampler* component can be used to provide non-constant scalar factors.

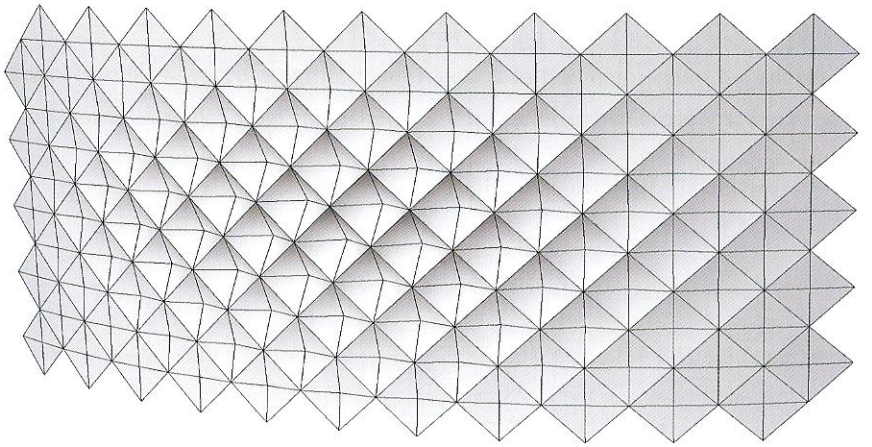


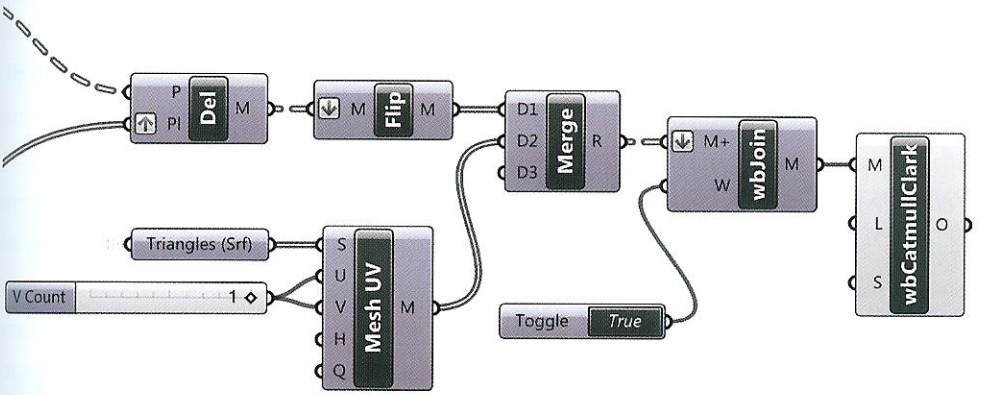
The resulting pyramids display an error because of the different orientations of each pyramid. To correct this fault a plane must be defined for each diamond-surface using the component *Plane Normal* (Vector > Plane); with input (Z) defined by the output (N) of the *Evaluate Surface* component. The calculated plane for each surface output (P) is connected to the grafted input (Pl) of the *Delaunay Mesh* component. Grafting is required to match the outgoing data of the *Merge* component.

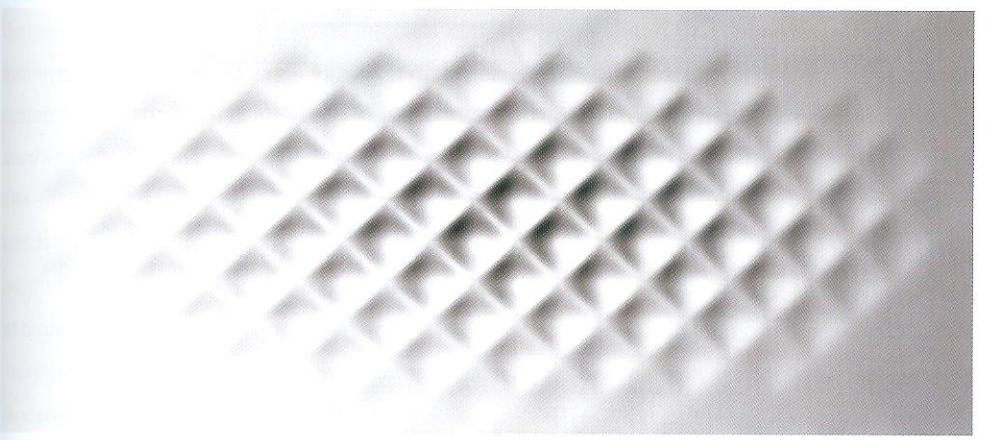




After properly structuring the pyramid data to triangulate, the output of *Delaunay Mesh* component is merged with the second output (Triangles) of the *Diamond Panels* component, to define regular edges. The final step is to smooth the flattened list of joined and welded meshes using the *Catmull-Clark* component.




P
PI
Del
M
M
Flip
M
D1
D2
D3
Merge
R
M+
W
wbJoin
M
M
L
S
wbCatmullClark
O
Triangles (Srf)
V Count
1
S
U
V
H
Q
Mesh UV
M
Toggle
True



6.6.3 Strategy: cull adjacent faces

Complex shapes are often composed of an aggregation of mesh-boxes, as shown in the image below. Once the boxes are joined together, internal overlapping faces can be removed according to the following algorithm.

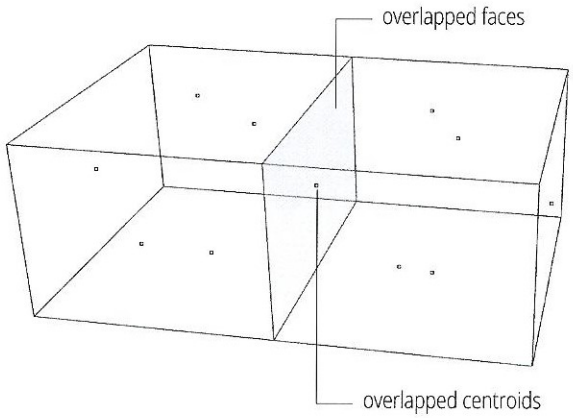


FIGURE 6.15
Often we need to remove the internal faces of a joined mesh made up of mesh-boxes.

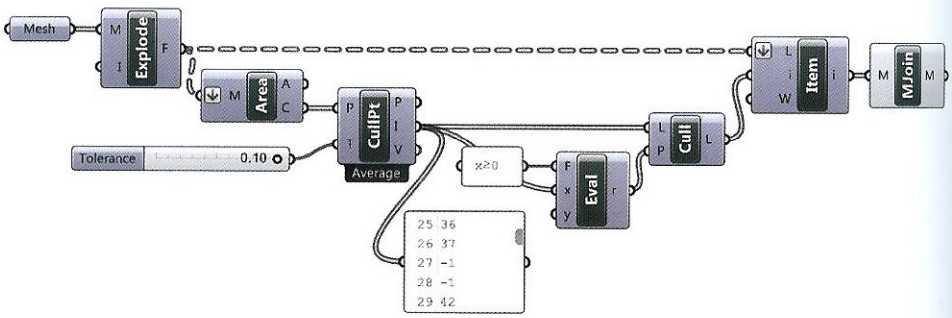


FIGURE 6.16
Cull Duplicates component culls points that are coincident and gives us the index of non-overlapped faces.

Digital informing creativity

Ludovico Lombardi

Lead Architect at Zaha Hadid Architects

The relationship between the tools used to investigate space and the way we understand space, and therefore the way our mind engages the creative process of designing, is a constant dialectic and dynamic process. The influence of the tools, understood both as theoretical researches and, in this case, as computational platforms, is marking the avant-garde experimentations in today's architecture, product design and fashion scenes.

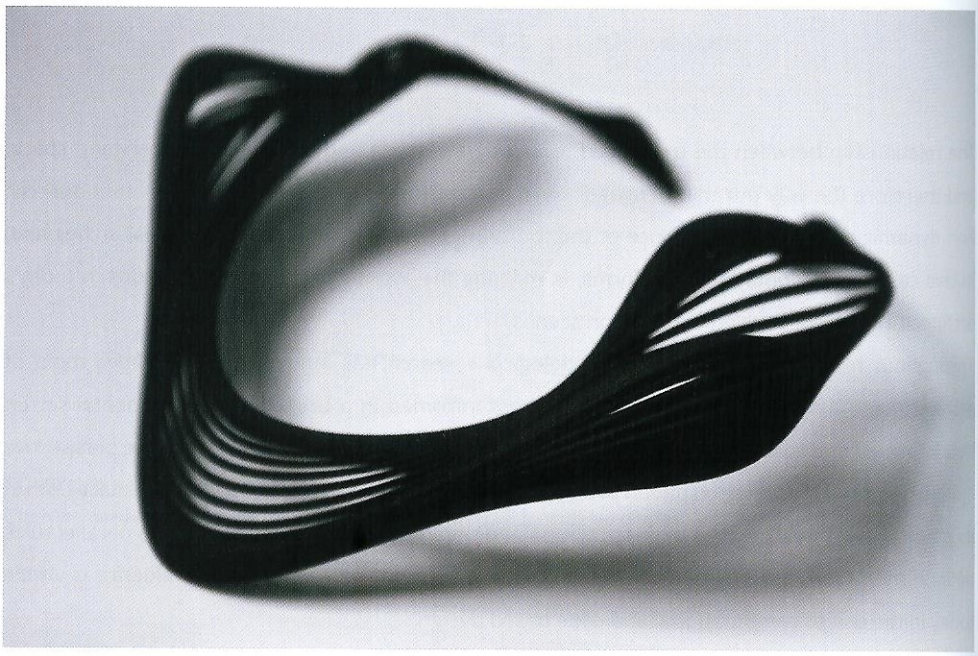
Throughout history theoretical and technological research has influenced the creative mind of architects and artists, and the technical progress informed in a broader way in a mental set up that informed and reconfigured artist and architect's creativity. The theorizations of the perspective system triggered a visual and spatial revolution that defined the Renaissance period spatially both in its physical and intellectual spatial articulation and organization. In the same way the parallel rules defined the modernist way of understanding and designing spaces. The same influence is visible today in the use of computational and code based design.

The great advantage of computational design is not cad based systems, but the possibility to engage with generative systems, understood as generic code based system, able to manifest themselves in a dynamic and adaptive way based on the basic codes and parameters that define them. If generative systems allow designer's creativity to redefine shapes, forms, configurations and organization systems as relational, it is also reflects in a new aesthetic register, the aesthetic of relations or the aesthetic of sublime (understood as an aesthetic of legible forces).

The sublime is derived by an aesthetic that is the result of forces applied into a generative system, which are legible in the configurations and variations resulting by the system itself. Examples of this aesthetic are present in different fields and different domains, from the sublime aesthetic of a flock of birds in the sky constantly reconfiguring and morphing in new formations, to the gothic constant morphing of the same element from column to vault nodes or ornament in a continuous way, or the landscape representations of Ruskin.

While none of the examples described above necessitate computational design to manifest themselves, the legibility of their sublime aesthetic remain true to what I was trying to define as the

new aesthetic informed by computation design. Today's avant-garde designers use computational tools to manage the complexity present in natural system, as in the gothic or as in Ruskin paintings. It is also the tool that allows to negotiate the complexity of contemporary heterogeneous and dynamic society of multitude.



Black Swan necklace, Ludovico Lombardi (2014).

On the other hand there is a superficial and Manneristic overproduction of computational based designs that are not relevant to the above discourse, but that can be defined as a simplistic experimentations triggered by the easy access of the new code based computational platforms. This is probably the most pertinent way to explicitate the need and the role of a designer, with its sensibility and its rational. The creative process is enriched and informed by computational tools, but those do not substitute the role of the creative mind. The misinterpretation of generative system, as not driven by the designer creativity, but in the specific just by computational tools, is the triggering factor of most of today's criticism to computational based design. Understanding that design is relational first than even its physical manifestation, is the paradigm that brings back the role itself of the designer creativity while coding the relations that generates the outcomes of those complex system.

The repositioning of the role of the designer in the above statement hints also to the counterpart of the new design approach, which is the distinction between designer and technical virtuoso. While both have different merits, one does not implies the other and the distinction is still valid as it always was. There is the need of a common understanding and knowledge sharing, but the two roles are not always or don't always to coincide. The specificity and technicality of code based design is still something that necessitates a strong logic and mathematical background, while the understanding of its set up and rules is what a designer today needs to be exposed and engage with. Variations, inflection, self-similarity and integrations become parameters of evaluation of computational design aesthetic, and their definitions is an a-priori from the their formal representation, embedded in the set-up of generative systems and the relations within the system itself. The creative process is readable at different stages, from the moment when the designer is establishing those relations within the set-up of the system, to the moment when the manifestation in the physical domain gets informed by materiality, engineering and production methods. All of those factors are now integrated in what can be defined as a complete creative moment. Today's designers are borrowing tools and referring to a vast number of fields tangent to design, allowing for a re-definition of both aesthetic and logic of space and forms. There is a strong common research manifesting in parallel in different fields, from architecture to design to fashion, and in the most progressive courses in universities and design schools. This legible academic direction is something that will shape the future generation of designers and will define a new type of creativity and aesthetic, which will reflect in our daily life in a constantly increasingly way.

Ludovico Lombardi (LDVC) promotes design by research through computational and material processes. The avant-garde design research operates on multiple scale, from urban design, landscape design, architecture and industrial design. The articulation of complexity and organization through elegance and sophistication become the paradigm to negotiate contemporary heterogeneous and dynamic society of multitude.
Ludovico's work experience in Spain, Italy and England includes Carlos Ferrater, Arata Isozaki and Zaha Hadid architects, where he currently works. In 2008 he graduated with a Master's degree in Architecture and Urbanism from the DRL Design Research Lab of the Architectural Association in London after having previously studied in Italy at the Politecnico of Milan and in England at the Bartlett. His work has been featured on a series of international magazines, such as AD, AJ, Abitare among others. His works have been presented and exhibited at the Architecture Foundation in London and in touring exhibitions in Italy and in the US. Ludovico has appeared as guest critic at the Architectural Association, at the Bartlett UCL and for the DRL and has been invited to lead workshop and lecture in Turin, Bologna, Florence and internationally at the Tate Britain in London and at RISD Rhode Island School of design. Ludovico was recently invited to collaborate with Abitare magazine and was appointed unit master for the M.arch in Urban Design program at the Bartlett UCL for the academic year 2009/10. He was recently appointed Faculty Professor at RiSD architecture department.

http://www.ldvc.net/