4_ transformations

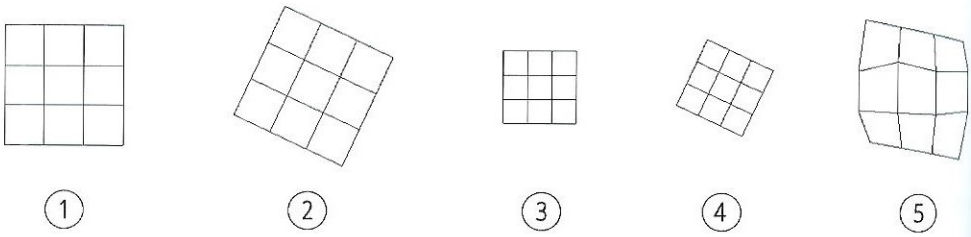
"The orthogonal features, when combined, can explode into complexity"

Yukihiro Matsumoto

Previous chapters have demonstrated how mathematics and logic are the basis of complex 3D models. In this section a new layer of complexity will be introduced, *geometric transformations*. In particular, the chapter will examine:

- **Euclidean transformations:**
  Euclidean transformations preserve lengths or angle measures. Euclidean transformations include translations, rotations, and reflections.
- **Affine transformations**
  Affine transformations do not preserve lengths or angle measures. Affine transformations include scaling, shearing and projections.
- **Other transformations**
  More complex transformations can be performed by combining transformations or by specific components such as *Morph* component.

Transformation	Maintain	Does not maintain
Euclidean	shape, size	position
Affine	parallelism	shape, size, position
Similarity	shape	size, position
Morph	topological relationships	geometrical properties

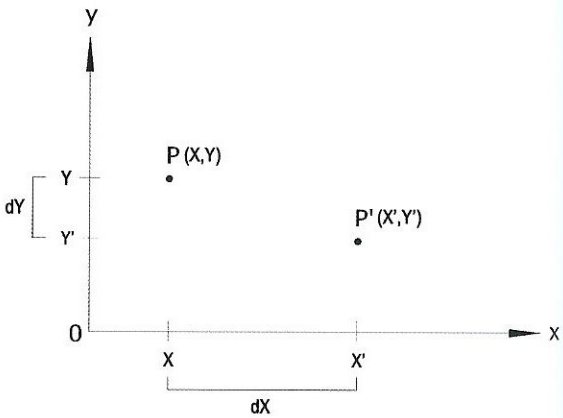


1. Original geometry;
2. Euclidean transformation: rotation;
3. Affine: scale. Scale is an affine transformation which maintains shape in addition to parallelism;
4. Similarity (scale + rotation);
5. Morph.

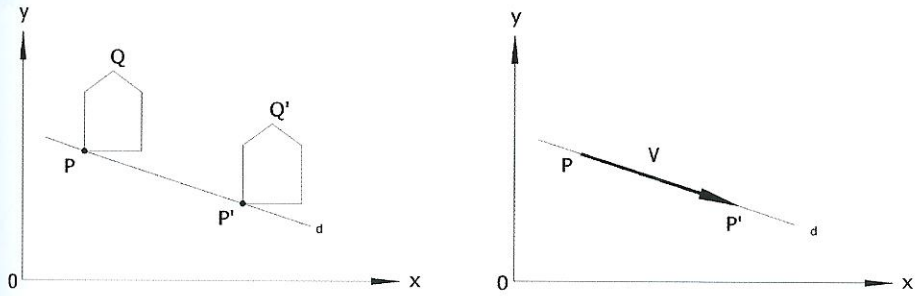
A transformation measures the change between two points P(x,y) and P'(x',y').
To translate a point to a different position a specific quantity is required to be to added or subtracted from its original coordinates.

$x' = x + dx$

$y' = y + dy$



An entire object can be translated performing the previous transformation to all its points or, more simply, to its vertices.



This means that a point reaches a new position by moving it of a certain distance, according to a specific direction and sense. These three characteristics define a "vector", that is a geometrical object characterized by ***direction, sense and magnitude (or length)***.

Therefore, a translation can be defined by specifying a start point P and a translation vector V.

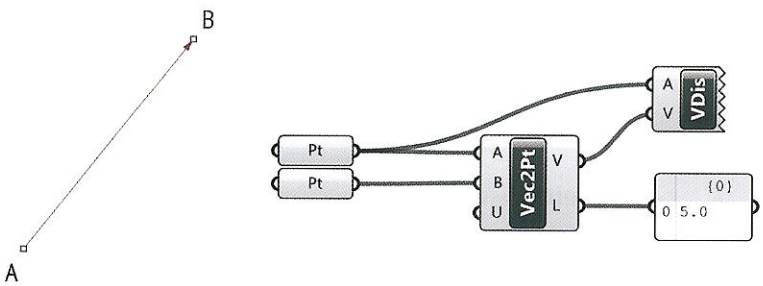
Grasshopper provides several methods to create vectors and perform operations on vectors.

4.1 Vectors

The panel (Vector > Vector) hosts the components used to create and modify vectors.

- **Vector 2Pt:**

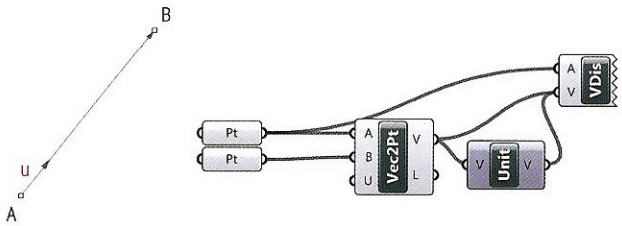
  The component *Vector 2Pt* (Vector > Vector) creates a vector between two described points: A (start point) and B (end point). The L-output of the *Vector 2Pt* component returns the magnitude of the vector and the V-output the vector. The *Vector Display* component can be used to visualize the vector in Rhino.



- **Unit Vector:**

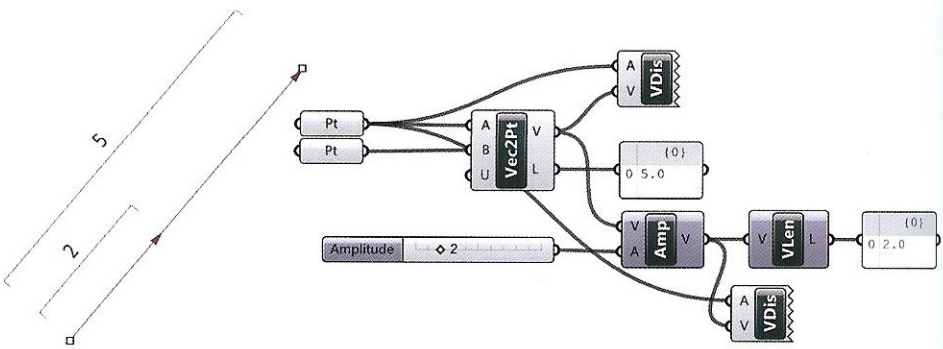
  Given a vector with a specified direction and magnitude (L) as described by two points, the component *Unit Vector* (Vector > Vector) can be used to unitize the vector or change the

magnitude to 1 unit, without modifying the direction. The component *Vector 2Pt* can be used to unitize an output vector by connecting a Boolean toggle set to True to the U-input.



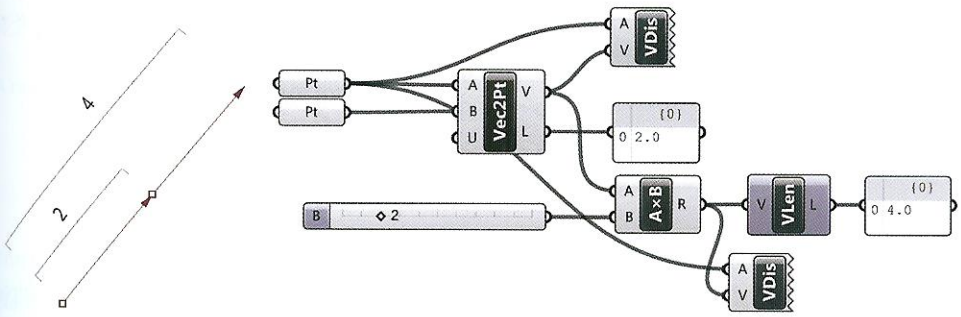
- **Amplitude:**

  The component *Amplitude* (Vector > Vector) sets the length of a given vector to an input value (A). For instance, a vector with an initial length equal to 5 is connected to the input (V) of the *Amplitude* component, the component will return a new vector with a magnitude (A) while maintaining the original direction and sense. Sense will change using a negative value as an input (A).



- **Scalar multiplication:**

  Scalar multiplication refers to the multiplication of a vector by a scalar number N. If N>0 a vector in the same sense will result, if N<0 a vector in the opposite sense will result. The vector's new length is equal to the product of the initial vector's length and N.



- **Unit X, Unit Y, Unit Z:**

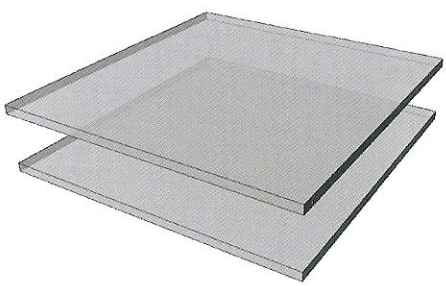
  The components *Unit X, Unit Y and Unit Z* (Vector > Vector) define unit vectors along the x, y and z axes. The input (F) can be used to set a length through embedded scalar multiplication.

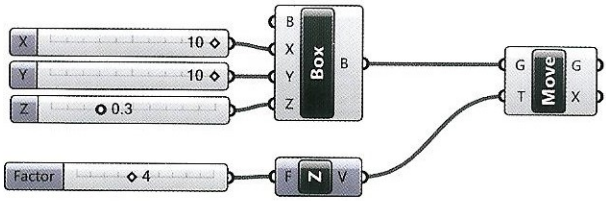
4.2 Euclidean transformations

Euclidean transformations (Transform > Euclidean) include translations, rotations, orientation, and mirror components.

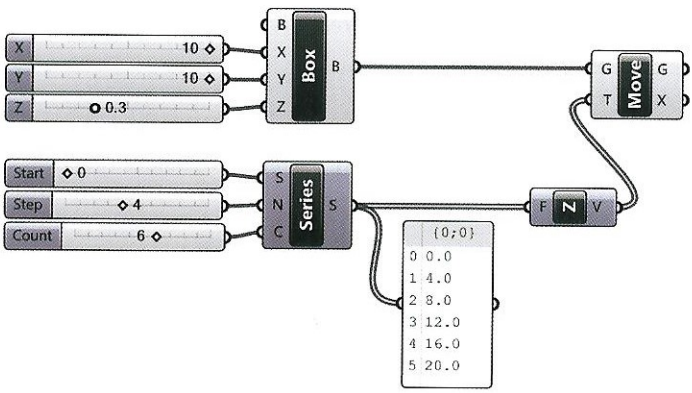
4.2.1 Translations: *Move* component

The component ***Move*** (Transform > Euclidean) translates a geometric entity (G) according to a vector (T). For example, a box created using the component *Center Box* (Surface > Primitive) is translated in the z direction using a *Unit Z* component.





If input (F) of the *Unit Z* component is connected with multiple numeric values it will return multiple vectors, these vectors can be used to perform multiple translations. For instance, a multistorey building can be modeled by connecting a *Series* component to the F-input of *Unit Z* component. In this simplified model the N-input of the of the *Series* component sets the distance between floors, the C-input sets the total number of floors and the S-input defines the starting value of the numerical sequence. If (S) is set to 0 the first floor will coincide with the original box.

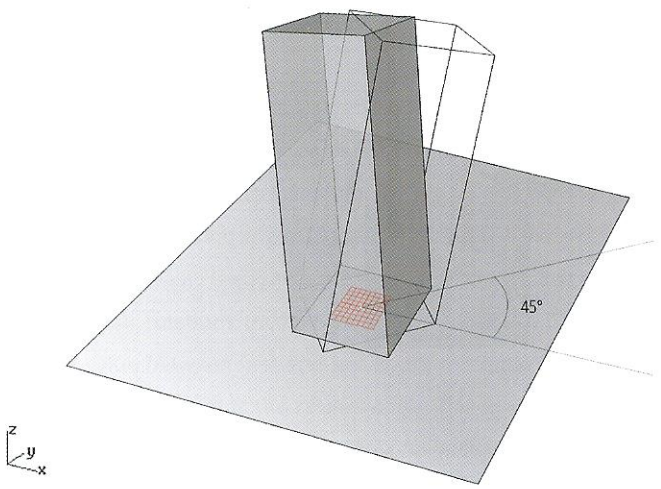


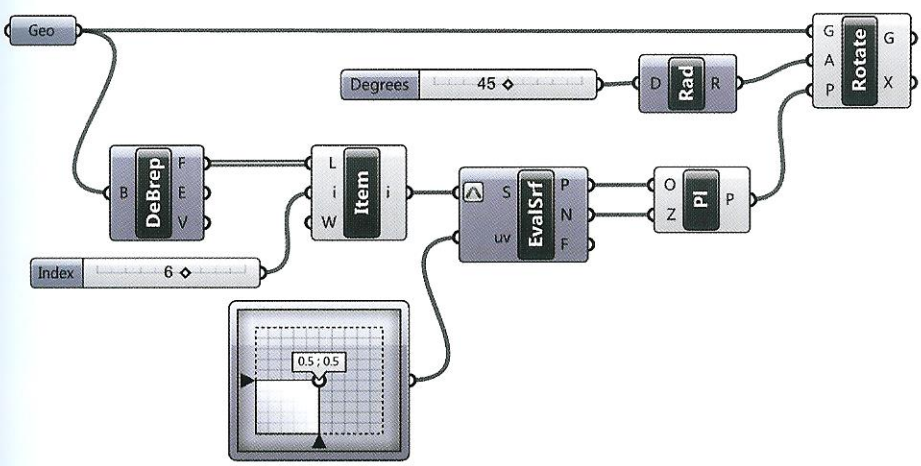
4.2.2 Rotations: *Rotate* and *Rotate Axis*

A rotation is the motion of a rigid body around:

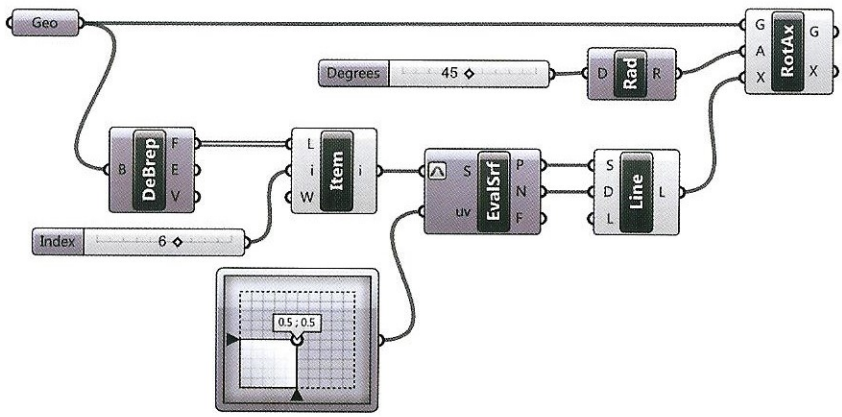
- a fixed point (called rotation center);
- a straight line (called rotation axis).

The components *Rotate* and *Rotate Axis* (Transform > Euclidean) rotate a geometric entity (G) in a plane or around an axis respectively. For instance, a prism can be rotated 45° with respect to its original position on an arbitrary-oriented surface by using the component *Rotate* in conjunction with other components that define a rotation plane.

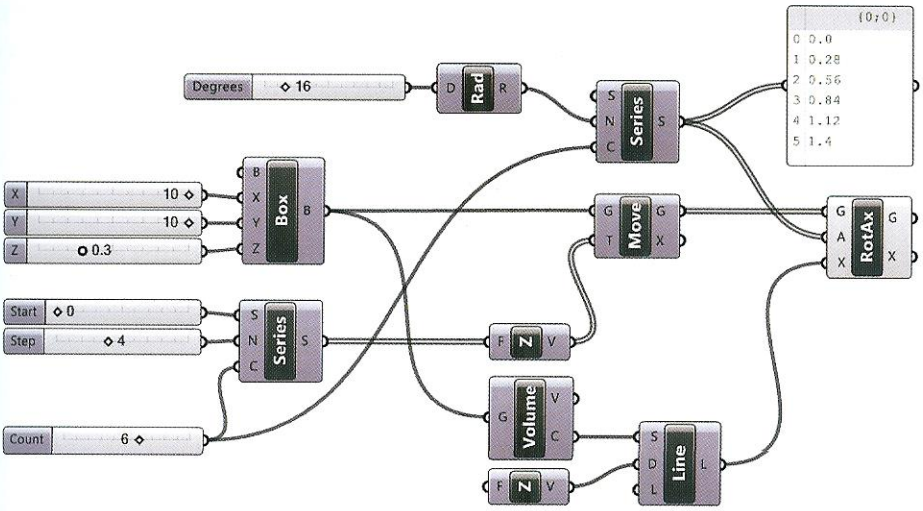




The *Geometry* container imports the prism. We can decompose it by using a *Deconstruct Brep* component and then we can extract the lower face by *List Item* set to 6. *Evaluate Surface* calculates the normal vector of the lower face at its center; then *Plane Normal* (Vector > Plane) returns the plane to rotate according with. Finally, ***Rotate*** is connected to the prism (G-input) and to the plane in (P). The rotation angle is set through a *Number Slider* ranging between 0° and 360°. Since *Rotate* accepts angle in radians, we have to convert degrees into radians by the Radians component (Maths > Trig).
In alternative, we can rely on the ***Rotate Axis*** component which needs a line (X-input) to perform a rotation. The line is created by a *Line SDL* component.



The component *Rotate Axis* used in conjunction with the *Series* component, can create progressive rotations.



With reference to the previous example (see 4.2.1), it is possible to define the rotation axis as a vertical line, starting from the gravity point of the initial box (by the *Volume* component). The progressive rotation is made using different rotation angles (*Series* outputs) instead of just one value. The C-input of *Series* must be set so that its value is equal to the number of objects to rotate.

To calculate the incremental rotations (*relative angle* between consecutive floors) as a function of the *twist angle* (angle value between first and last floor), this latter is divided by the number of floors minus one; the division's result is the *relative angle*. The following images display different incremental rotations referred to the wireframe model of a multistorey building: the angle between the first and last floor (twist angle) is 90°.

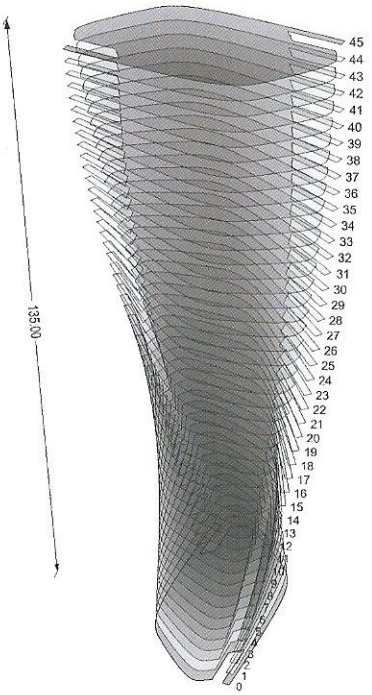


FIGURE 4.1

Progressive rotation applied to a simplified model of a multistorey building.

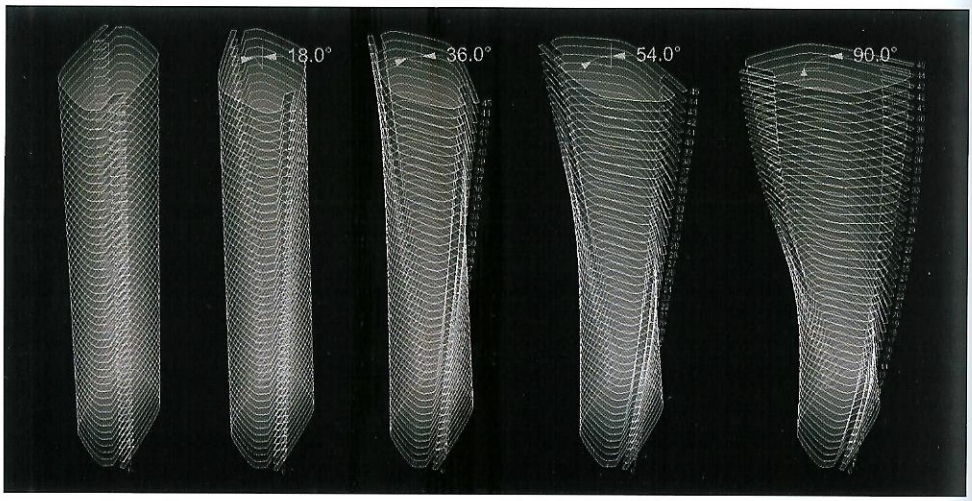
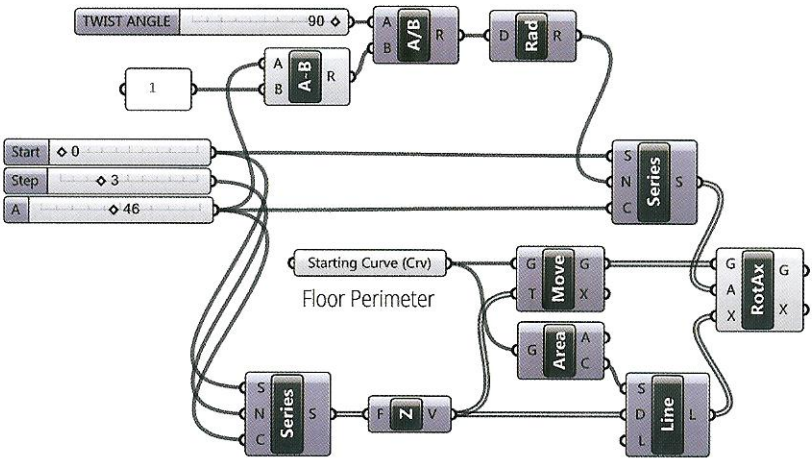


FIGURE 4.2

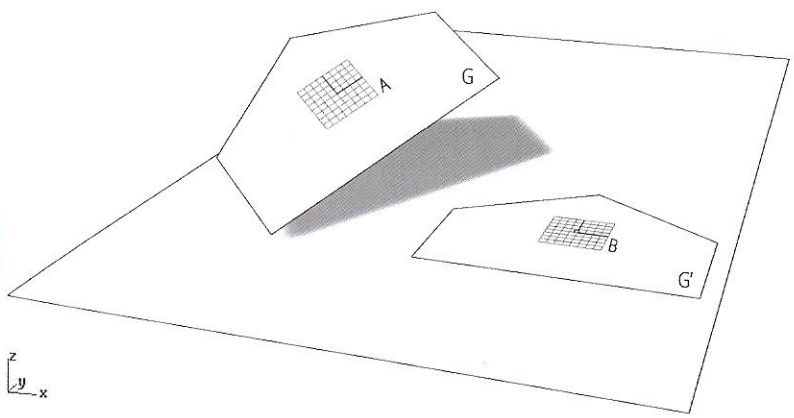
The image illustrate different configurations of the model as the *twist angle* changes.

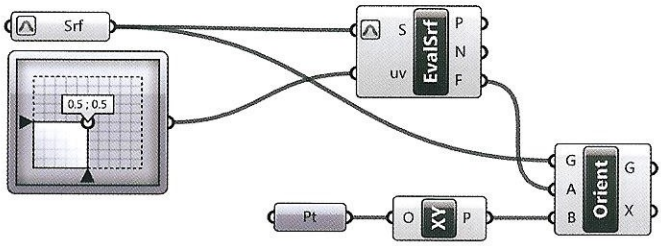
The Equation *relative angle = twist angle / (number of floors-1)* calculates incremental rotations for each floor.



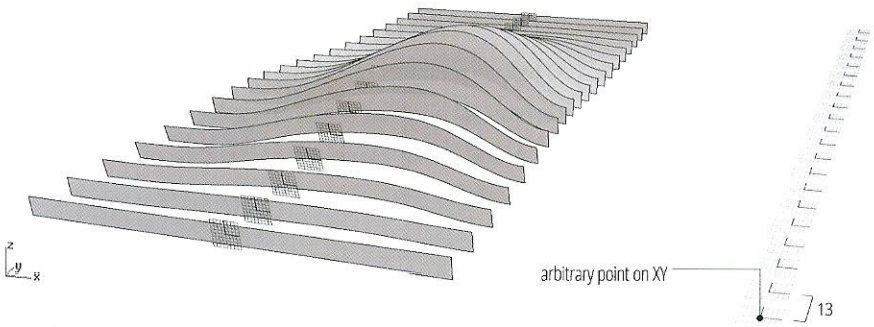
4.2.3 *Orient* component

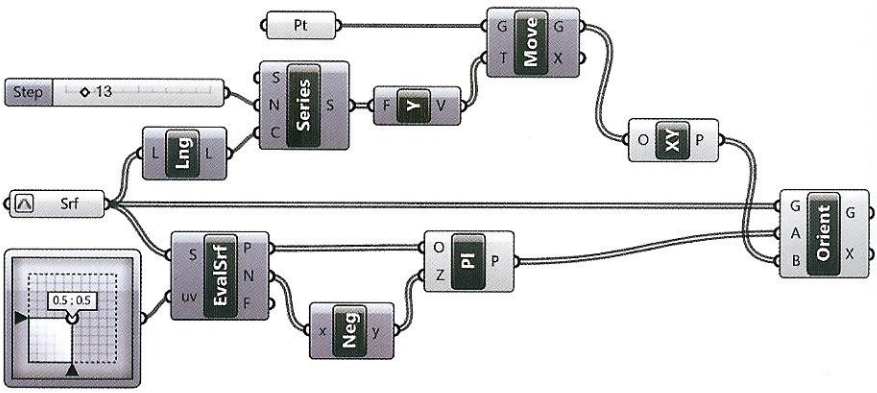
The component ***Orient*** (Transform > Euclidean) combines translations and rotations into a single transformation. A **geometric entity (G)** arbitrarily oriented in the space, can be oriented to new position (G') by specifying an **initial plane A** and a **inal plane B**. To orient geometry G to a new position G' the *Evaluate Surface* component is used to find an initial plane defined as a tangent plane output (F) at LCS point (0.5;0.5), and a final plane described by the component *XY Plane* (Plane > Vector). The orient component adjusts the initial plane to be coincident with the final plane.

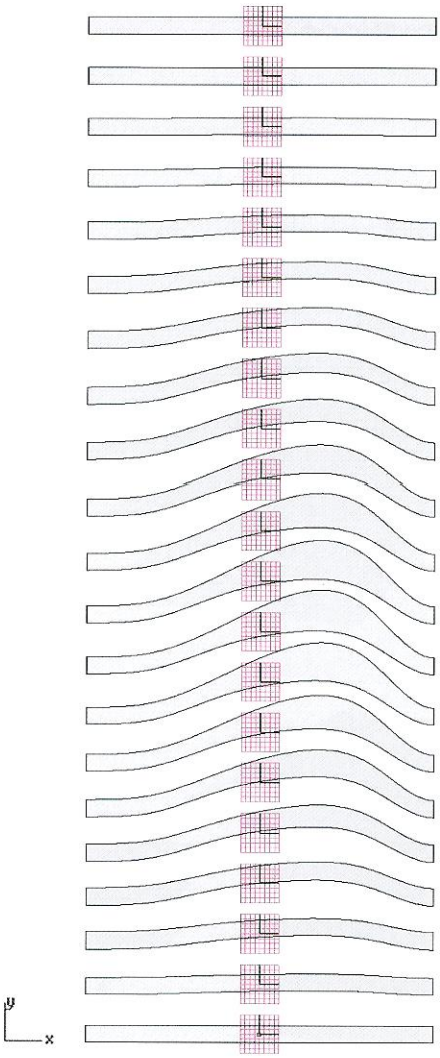




The orient component can also be used to orient a list of arbitrarily located ribs-surfaces to the XY plane. The *Evaluate Surface* component is used define a plane on each of the ribs. The *Evaluate Surface* normal vector output (N) is set as the z-axis direction (Z-input) of the *Plane Normal* component (Vector > Plane) defining the initial planes. To define the final planes, an arbitrary point is translated in the Y direction using the *Series* component as a scalar multiplier with: N-input set to the step size (13) and the C-input value coincident with the number of surface-ribs (calculated through *List Length*). At each point a *XY Plane* is defined. Lastly, the ribs-surfaces are oriented to the final planes.






y
x

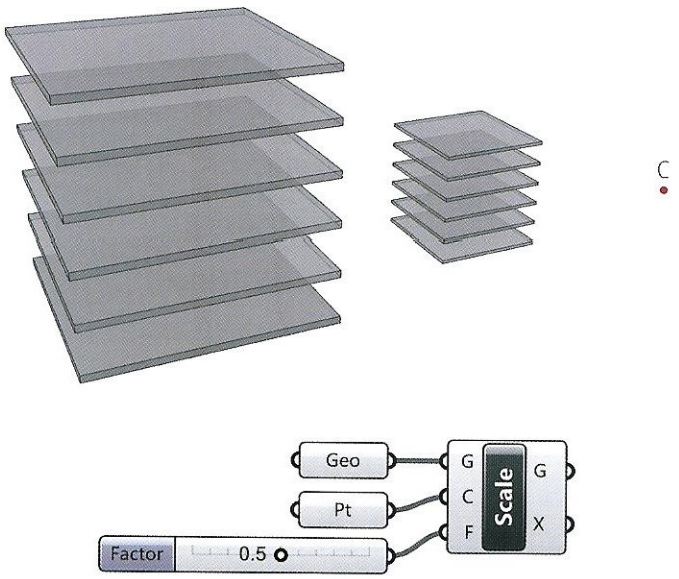
4.3 Affine transformations

Affine transformations (Transform > Affine) include scaling, shearing, projection and mapping components.

4.3.1 *Scale* component: uniform scaling

The component ***Scale*** (Transform > Affine) is a geometric transformation that enlarges or reduces objects uniformly in x, y and z directions. The ***Scale*** component resizes an initial geometry (G) by a scale factor (F) relative to a **center of scaling point (C)**. The center of scaling (C) can be any point. The **scale factor** is a positive number and **cannot be 0** (a null scale factor is a mathematical error). In particular, we have three main cases:

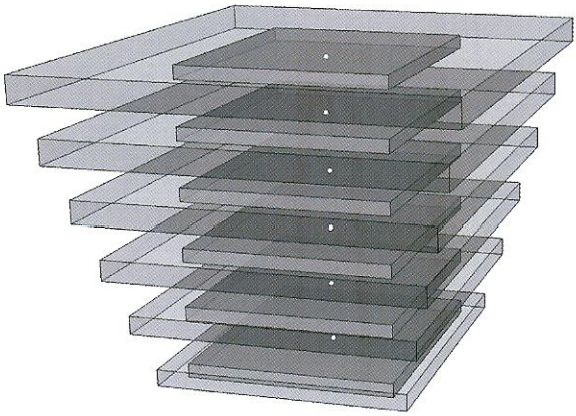
- **0 < F < 1**: objects reduce;
- **F = 1**: objects do not change;
- **F > 1**: objects enlarge.

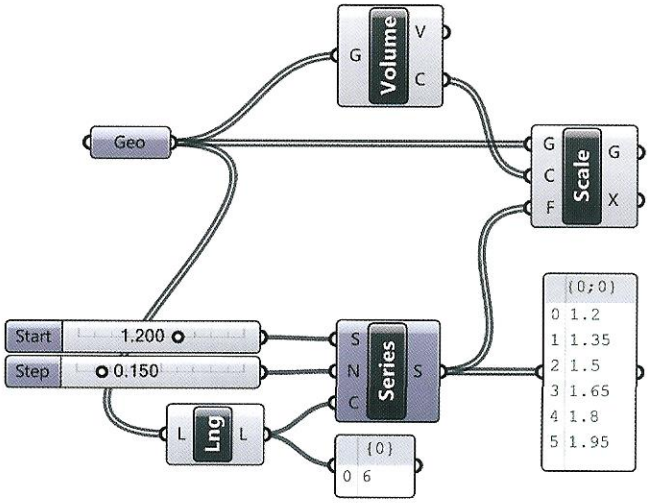


Scaling can also be performed on several objects. For instance, if the same scale factor is applied to the entire set of geometries the entire set of geometry will be scaled by the same proportion.

Alternatively, different scale factors can be applied to each geometric entity.

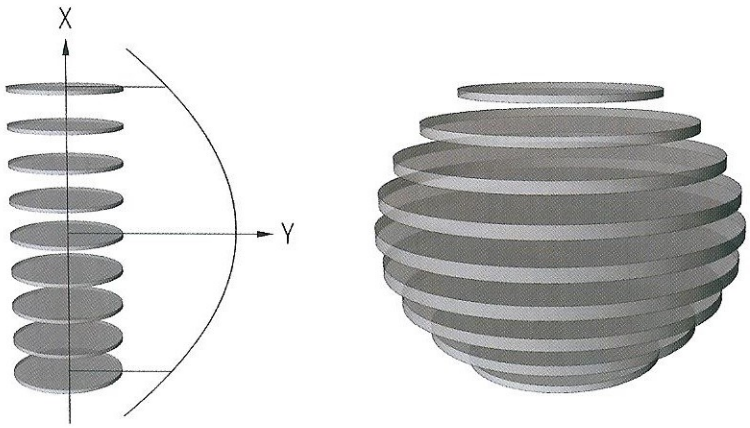
For example, to progressively enlarge a set of six boxes the input (F) of the *Scale* component is supplied with a list of increasing scaling factors by the *Series* component. The center of scaling input (C) is defined as the centroid of each box respectively.





The *Series* component generates a list of increasing or decreasing **linear** scaling values.

To generate more complex forms of scaling using mathematical functions the component *Graph Mapper* (Params > Input) can be used.

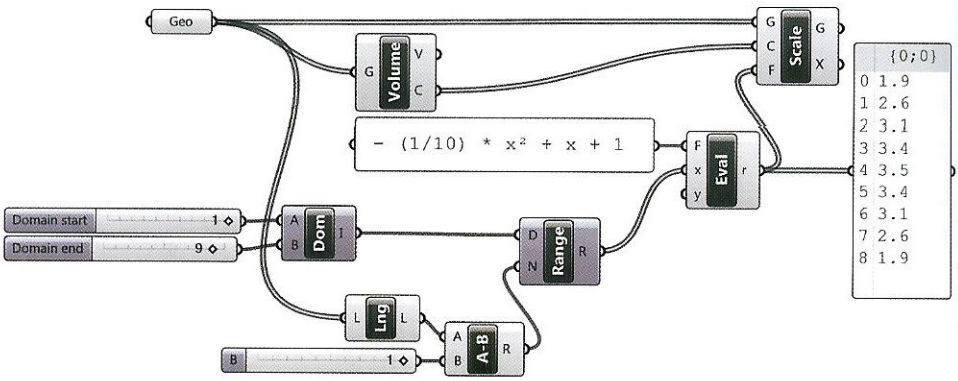


For instance, if the input (F) of *Scale* component is supplied with a **symmetrical** numerical sequence: (1, 2, 3, 4, 5, 4, 3, 2, 1) a parabolic form will result.

More specifically if the equation,

$$y = -(1/10) * x^2 + x + 1 \text{ with } 1 < x < 9,$$

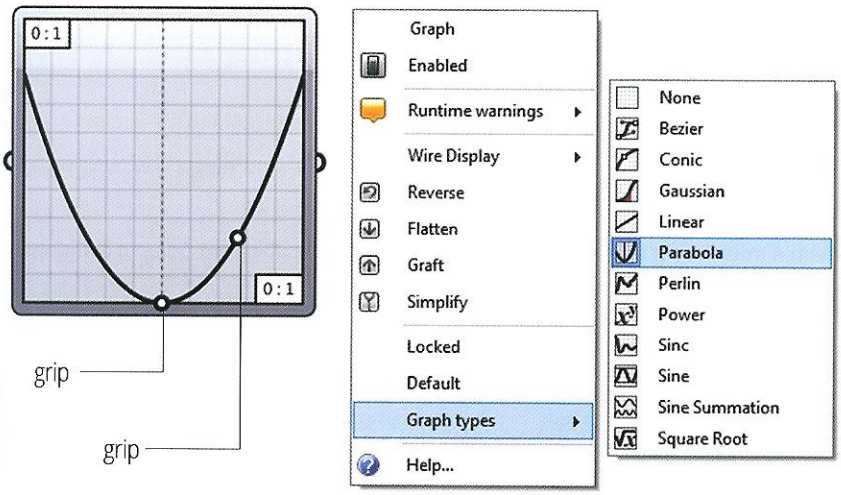
is supplied to the input (F) of the *Evaluate* component, the output (r) will return a symmetrical sequence co-domain which can be used as the scaling factor (F).



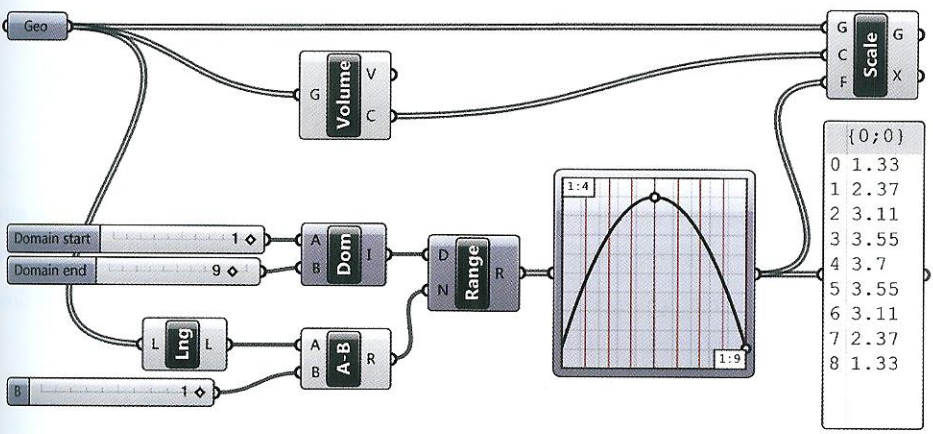
Of course it's not easy to manipulate complex equations to get a specific objective, but fortunately we can rely on an alternative tool available in Grasshopper: the *Graph Mapper*.

4.3.2 *Graph Mapper* component

To more easily define mathematical functions the component ***Graph Mapper*** (Params > Input) can be used to select a desired mathematical expression from a list.



The *Graph Mapper* component replaces the *Evaluate* component in the previous example. Instead of defining an equation, a desired function can be selected from the list provided by the *Graph Mapper* component. The function can be manipulated by adjusting the graphs grips.

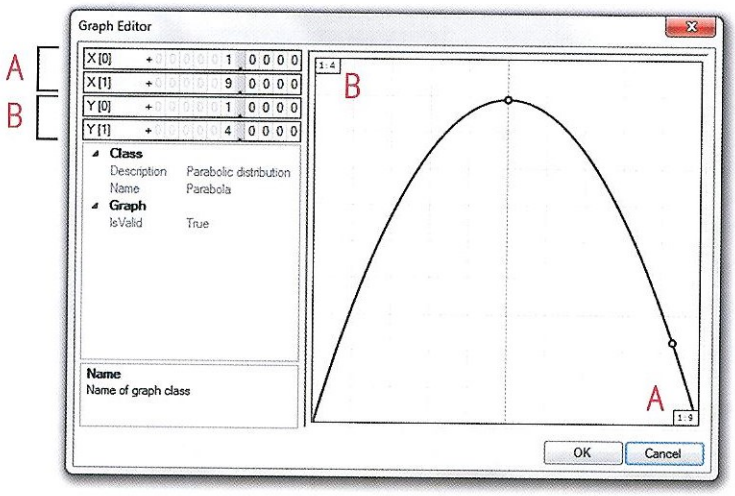


When you change the graph you actually change the mapping function, getting different results.

As for the *Evaluate* component so for the *Graph Mapper*, a proper domain must be supplied for the function using the *Construct Domain* component in conjunction with *Range*. It's important to point out that the *Graph Mapper* must be feed by a number of values equal to the number of geometries to scale. Since *Range* generates N+1 values (where N is the number of values defined in the N-input), we must subtract 1 from the number of objects coming from the *List Length* component connected to the initial geometries.

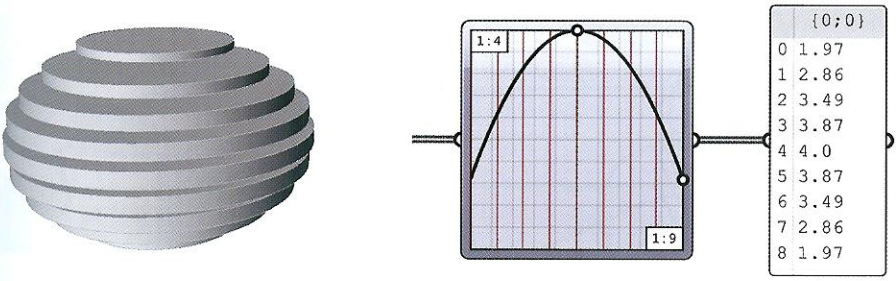
Nevertheless, the numeric domain is not the only thing to pay attention to. In fact, the graph has two "internal" domains to set. Double-clicking the *Graph Mapper*, a window appears and, with reference to the following image, we must change the two domains: A and B.

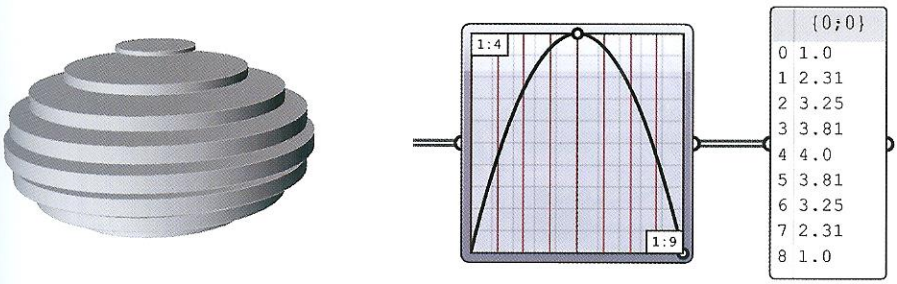
The domain A must be set according to the values used for the *Construct Domain* component (in our example 1 and 9).

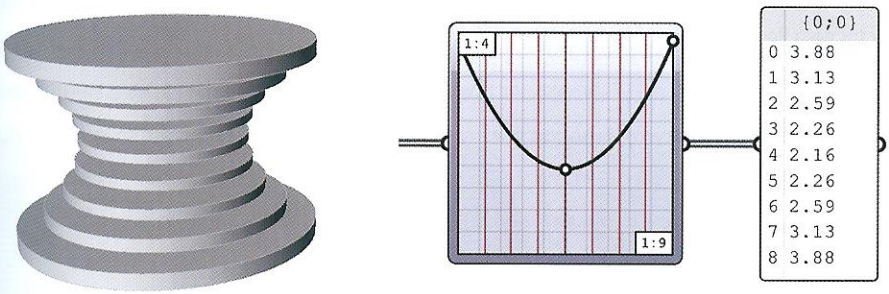


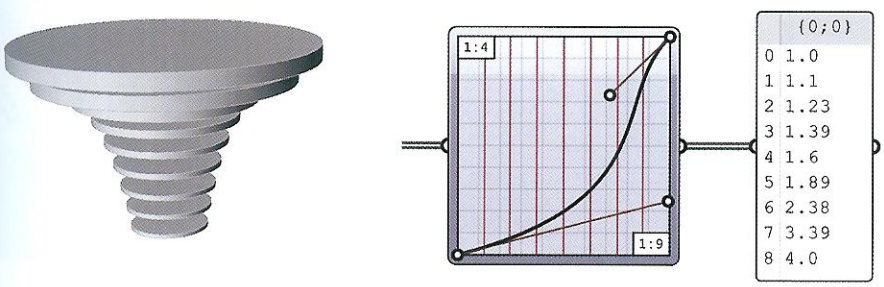
The domain B can be arbitrarily defined. It controls the extremes of the list of numbers. In other words, if B is set to [1,4] the lower scale factor will be 1 and the maximum scale factor will be 4. The two internal domains are displayed in the upper left and in the lower right corners of the *Graph Mapper* component contextual window.

The following images demonstrate how different graphs and different domains output varying geometric sets.









The *Graph Mapper* component can also be applied to the multistorey building example.

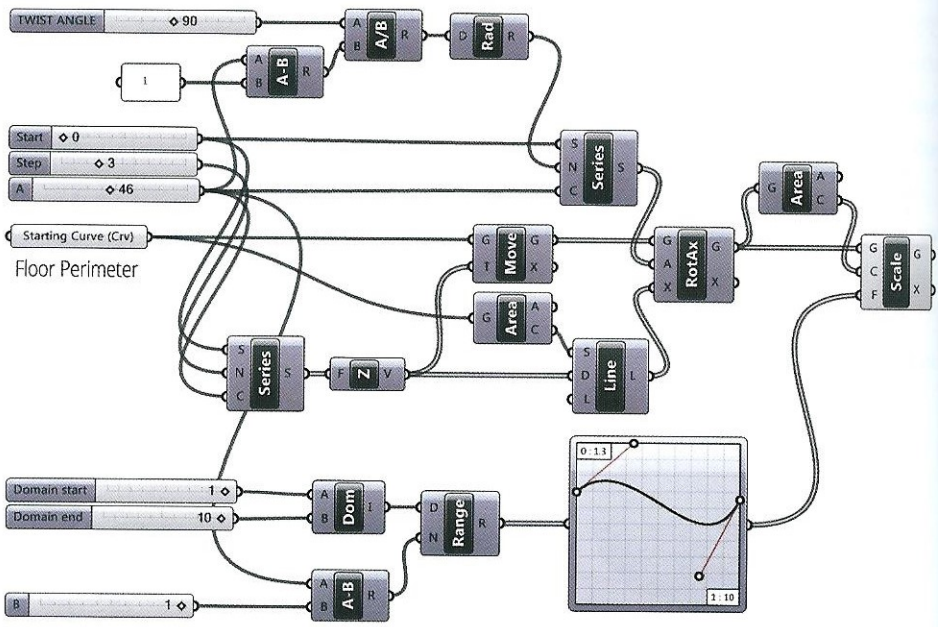
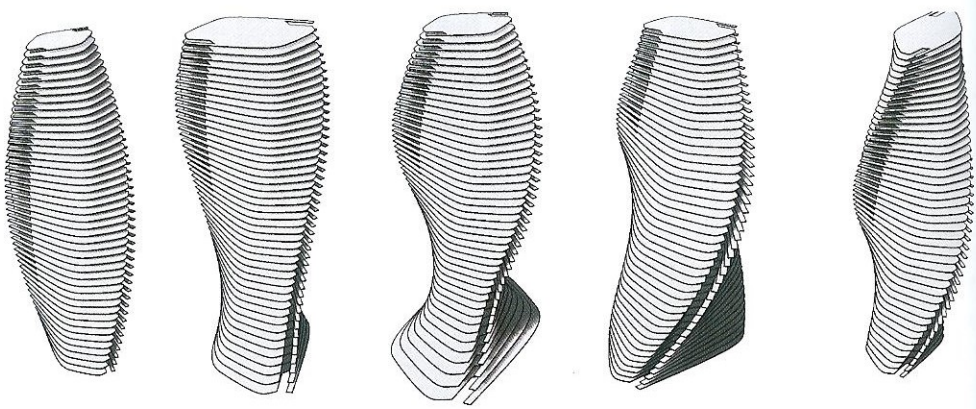


FIGURE 4.3

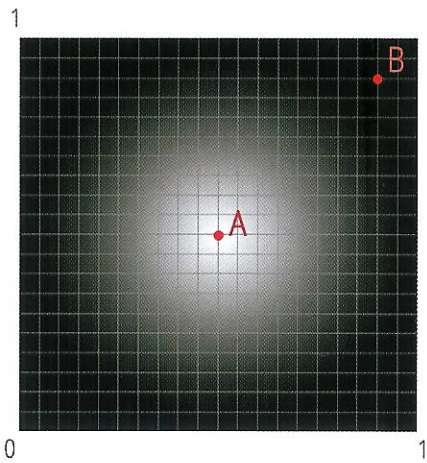
The image illustrate different configurations of a building model as the *Graph Mapper* changes, affecting the scale factor for each floor.



4.3.3 *Image Sampler* component

The component ***Image Sampler*** (Params > Input) is an input component which converts chromatic information of an image into numerical values.

Every rectangular picture can be imagined as a bidimensional domain ranging by default between 0 and 1. If a rectangular grid is superimposed onto the image each grid point P – defined by its UV coordinates in the LCS – provides an *intensity* value as an output.



point A

coordinates: (0.5, 0.5)

intensity: 1

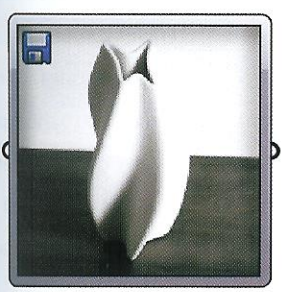
point B

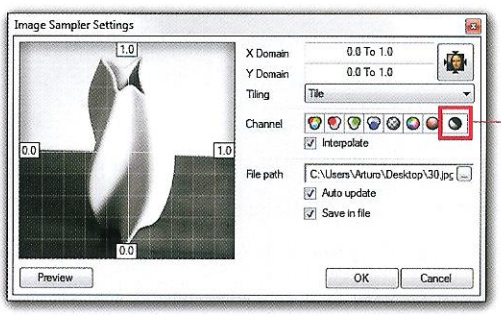
coordinates: (0.9, 0.9)

intensity: 0

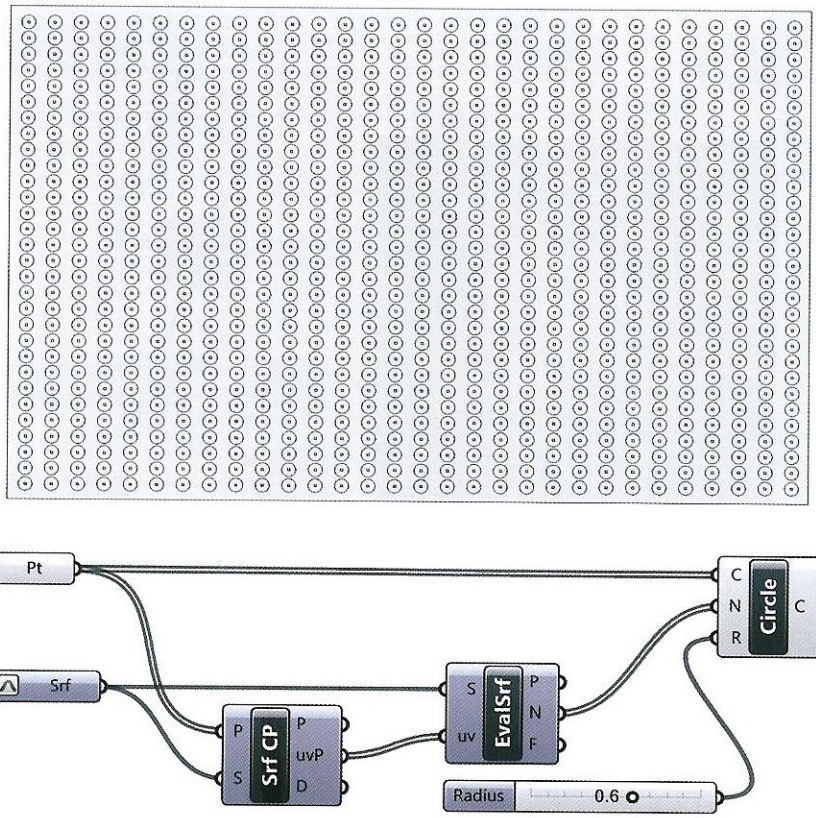
Intensity measures the color of the image at P dependent upon the specific *color model*: RGB, grayscale, etc., and outputs a value. For instance, the grayscale color model measures the intensity of a pixel and outputs values **ranging from 0 (black) to 1 (white) with fractional grayscale values in between.**

The *Image Sampler* editor – accessed by double left clicking on the component – is used to load an image, select a *color model* and specify whether to interpolate. The *Image Sampler* component requires a set of points with UV coordinates as input, and returns a list of numbers or *intensity* values.

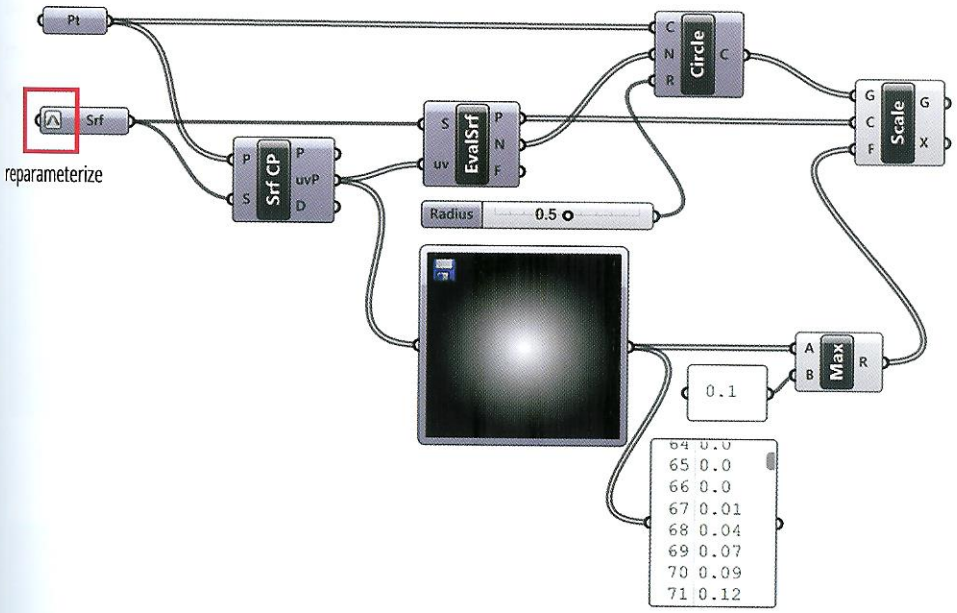




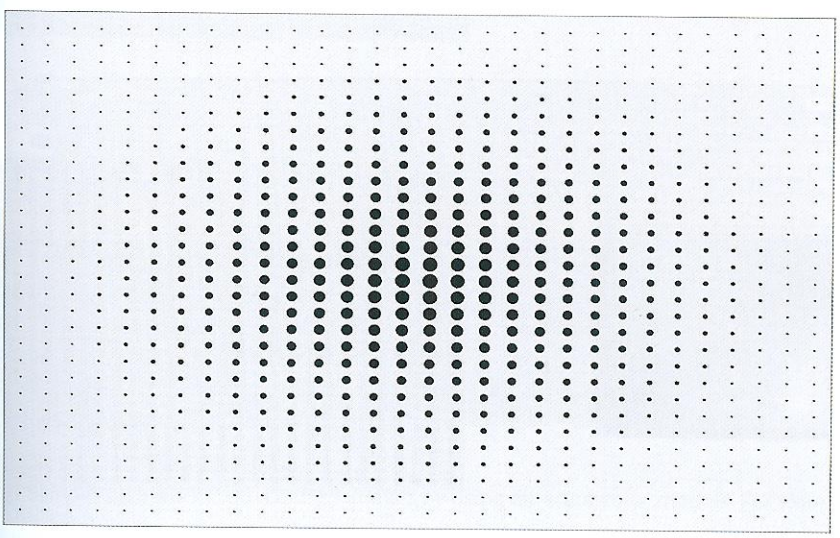
For example, a freeform surface and an array of points is set from Rhino. The component *Surface CP* is used to convert points coordinates from WCS to LCS, allowing the surface is evaluated at the uv resulting coordinates. The normal vectors (N) are connected to the N-input of *Circle CNR*. The radius of the circles is set to a constant value defined by a slider.



A geometrical pattern can be obtained by scaling the current circles using a scale factor that reduces geometries, i.e. a scale factor between 0 and 1. Since the output of the *Image Sampler* component is a list of *intensity* values ranging between 0 and 1, the *Image Sampler* output can be used as scale factors. To generate the scale factors the output (uvP) of the *Surface CP* component is connected to the input of the *Image Sampler*. The *Image Sampler* component outputs a list of values ranging between 0 and 1 according to the sampling of the set grayscale image. The *Image Sampler* output is connected to the scale factor input (F) of the scale component generating a pattern of circles. The initial surface must be reparameterized.



Images with pure black areas return null **intensity** values which are null scale factors. To cull null scale factors the component *Maximum* is used to replace every 0 with a different value such as (0.1).



The following images compare generated patterns to their corresponding grayscale images.

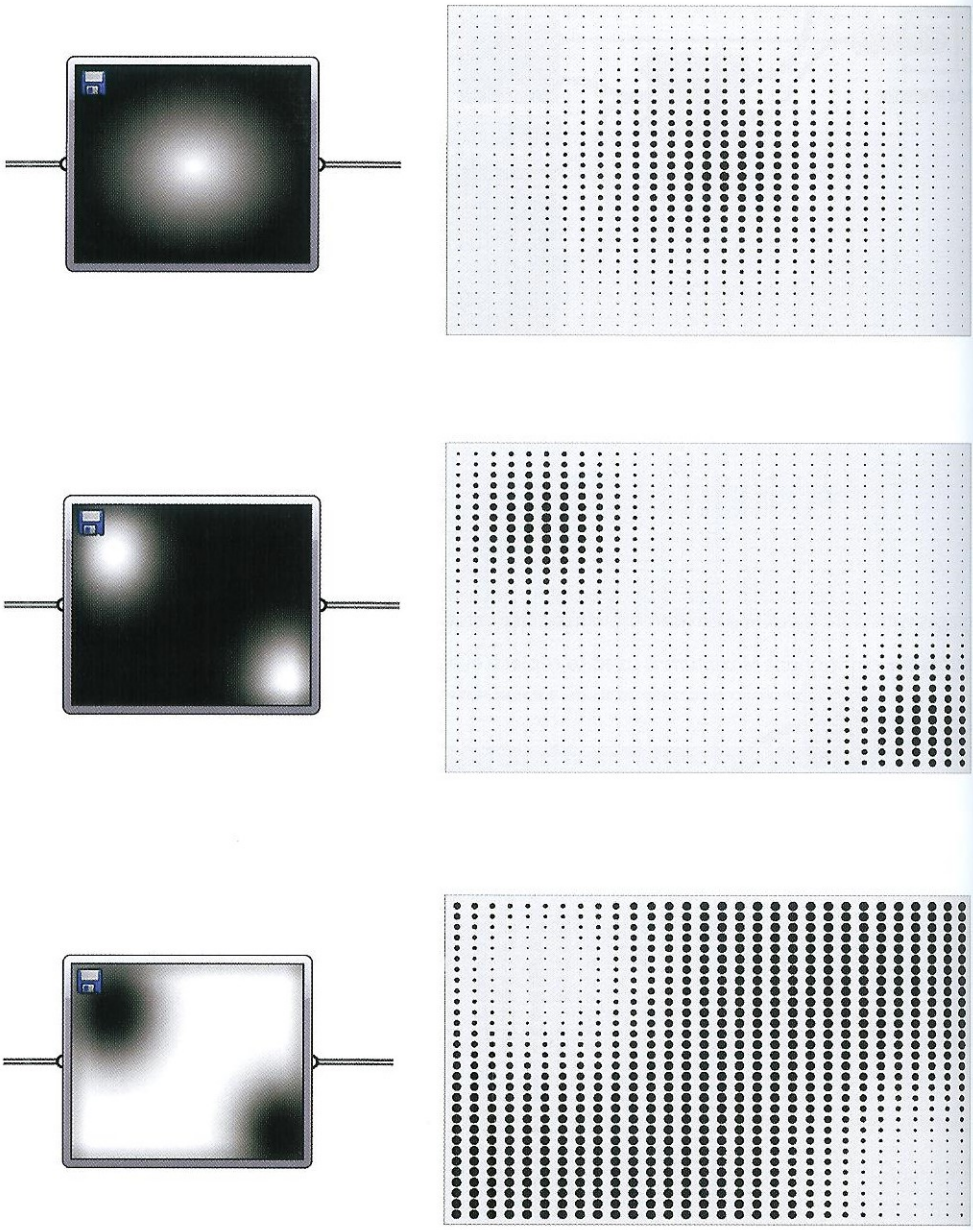
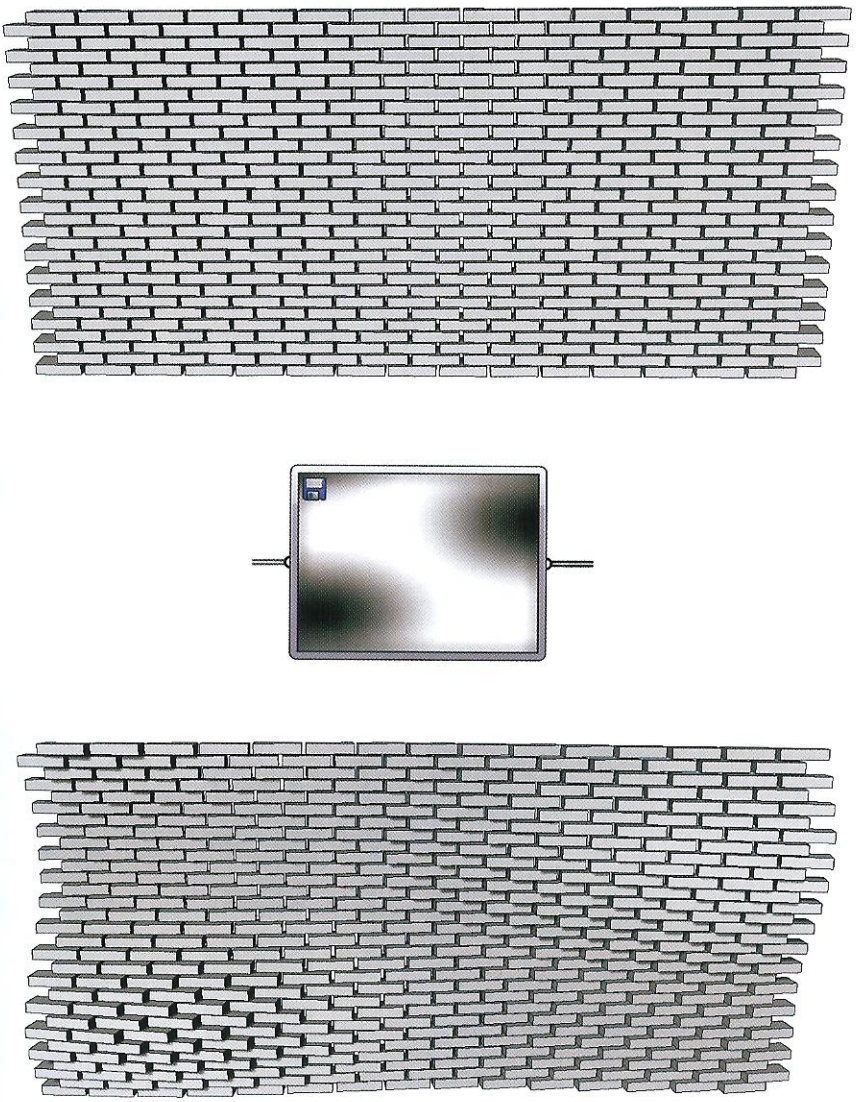
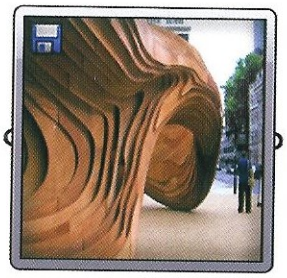


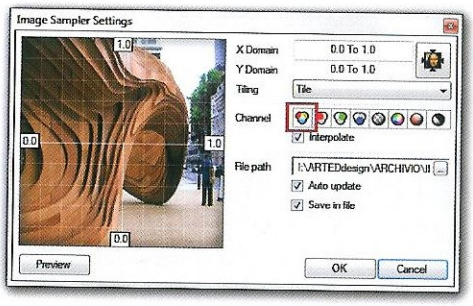
FIGURE 4.4
Different configurations corresponding to different grayscale images. Bright areas (close to white) return values close to 1 (no scaling), while dark areas (close to black) give values close to 0 (maximum reduction).

Below the results of the combination of the *Image Sampler* with different transformations. Each brick moves and rotates according to a grayscale image. The bricks' gravity points (whose coordinates were reparameterized between 0 and 1) were used as UV points for the *Image Sampler*.

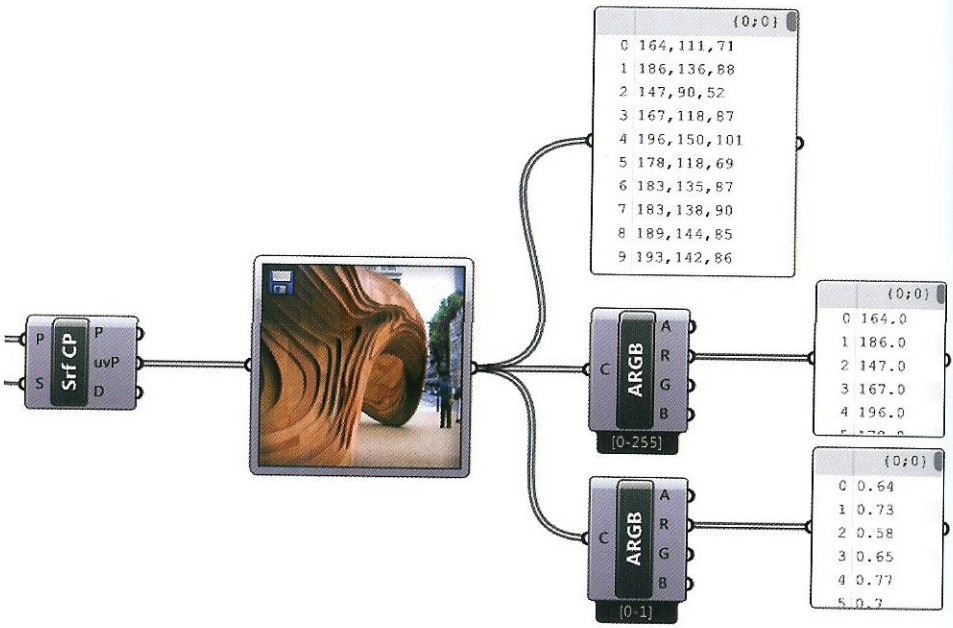


The *Image Sampler* also works with colored images by using different color models. For example, The RGB color model returns a list of RGB values – or a triplet of values (channels) – ranging between 0 and 255.





RGB values can be split into four channels using the component *ARGB* (Display > Colour). The component returns the Alpha (A), Red (R), Green (G) and Blue (B) channels. By default *ARGB* outputs values range between 0 and 1, the range can be set to [0,255] within the context menu by right-clicking on the component and selecting *Integer Channels*.



The following example demonstrates how to generate a variable offset from a flat surface using a colored image. In particular, the algorithm offsets parts of the surface that correspond to the red areas of the image.

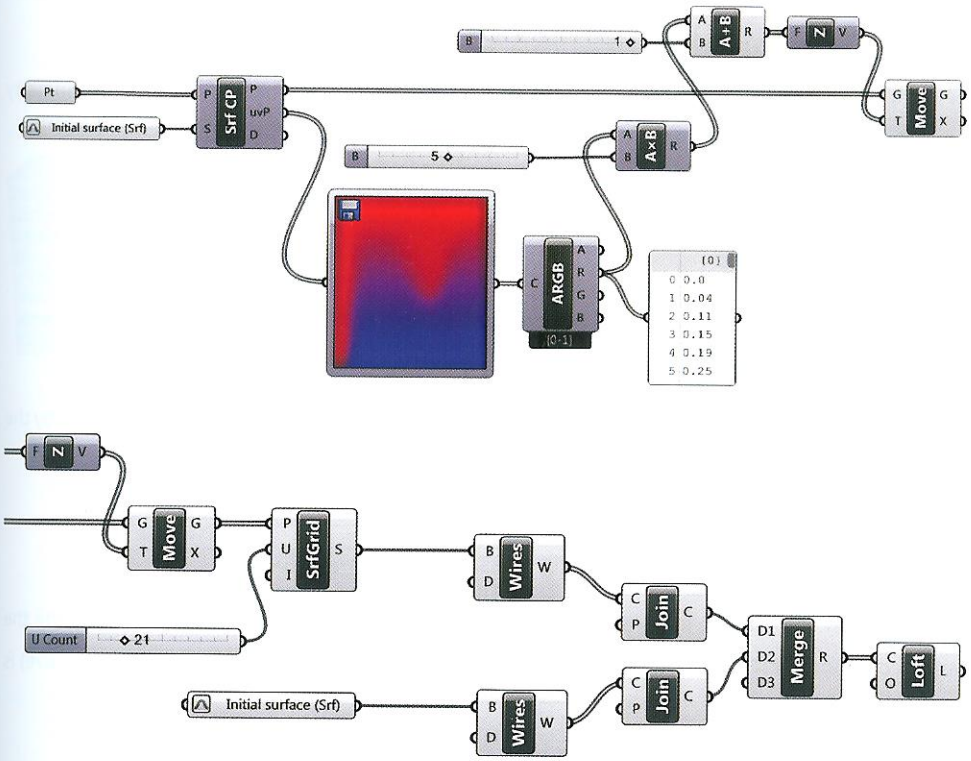
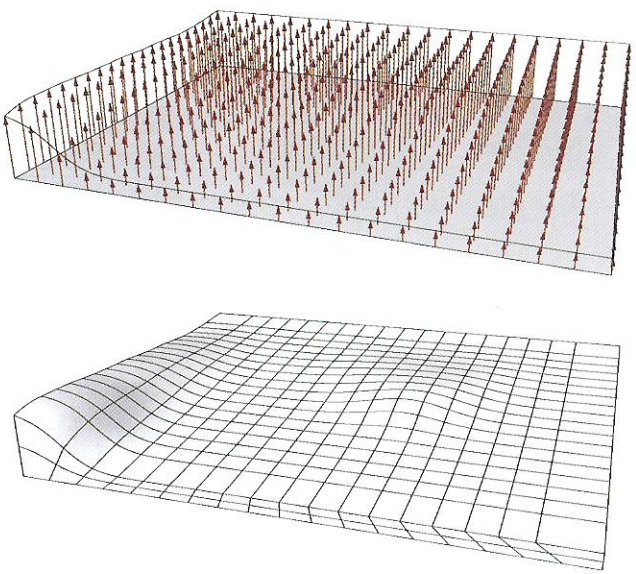


FIGURE 4.5
Variable offset obtained through the *Image Sampler*. The algorithm performs the offset for those parts of the surface corresponding to red areas in the image.



4.4 Other transformations: *Box Morph*

The component ***Box Morph*** (Transform > Morph) uses a reference "pliable" box to deform geometry to a target box. *Box Morph* is likened to Rhino's *Cage* and *CageEdit* commands.

The *Box Morph* requires three data inputs:

1. **A geometry to morph (G)**

   A geometric entity or a set of geometric entities;

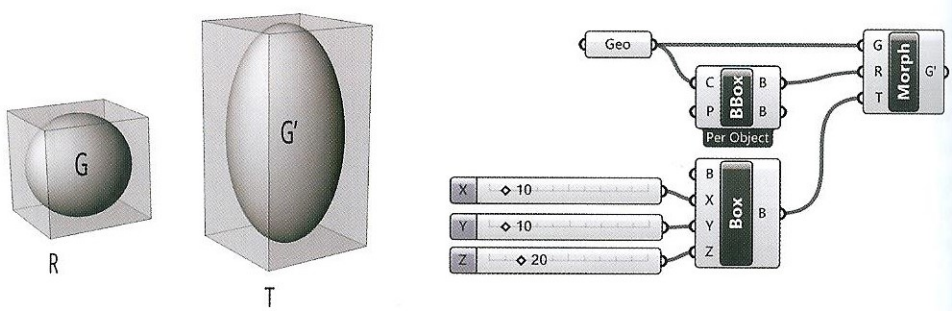
2. **A *reference box* (R)**

   A box which encompasses the geometry to morph. The *reference box* can be defined by the *Bounding Box* component (Surface > Primitive);

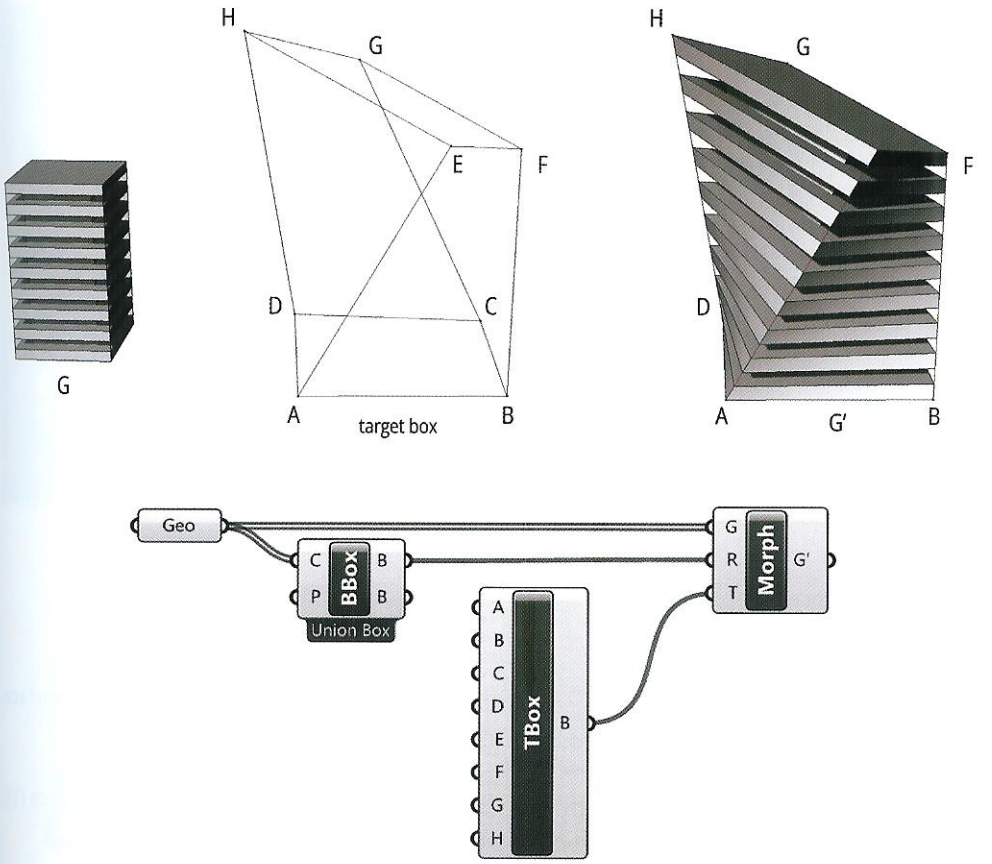
3. **A *target box* (T)**

   Any box that can be created using Grasshoppers box-components.

The *Box Morph* component transforms the *reference box* (R) into the *target box* (T) modifying the contained geometry (G). In the following image, the component *Center Box* (Surface > Primitive) is used as the *target box*.



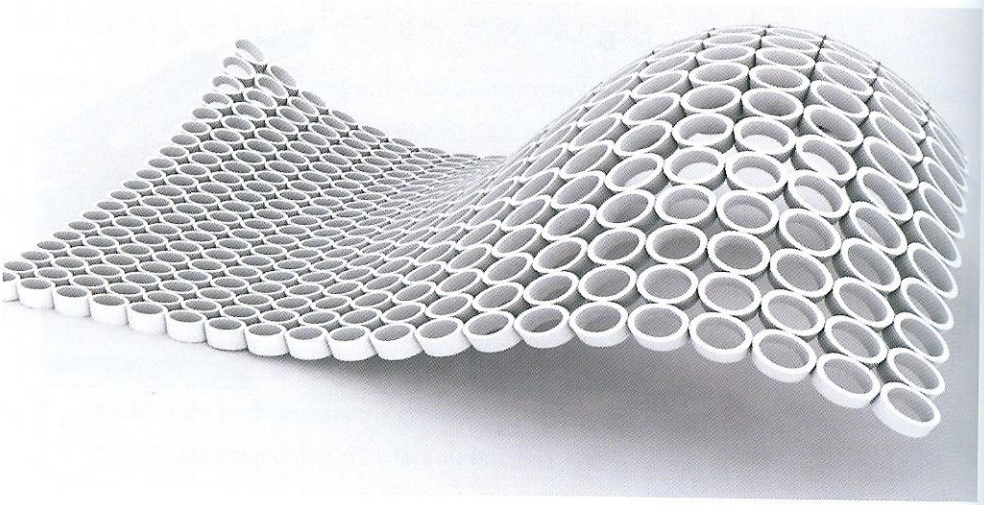
Multiple geometries can also be simultaneously morphed to primitive or non-primitive boxes. To morph multiple objects the *Bounding Box* must be set to *Union Box* mode through the contextual menu. The target box is defined by the component *Twisted Box* (Transform > Morph) which generates a twisted box from corner points. The bounding points can be defined in Grasshopper or set in Rhino. In the following example, the points are set from Rhino using the *Twisted Box* local setting.



In this case, multiple geometries are set from Rhino using the *Geometry* component. The *Bounding Box* component set to *Union Box* mode encompasses the geometries. The *target box* is created by setting eight points drawn in Rhino using the local setting, if these points are adjusted the geometry will deform accordingly.

4.4.1 *Paneling*

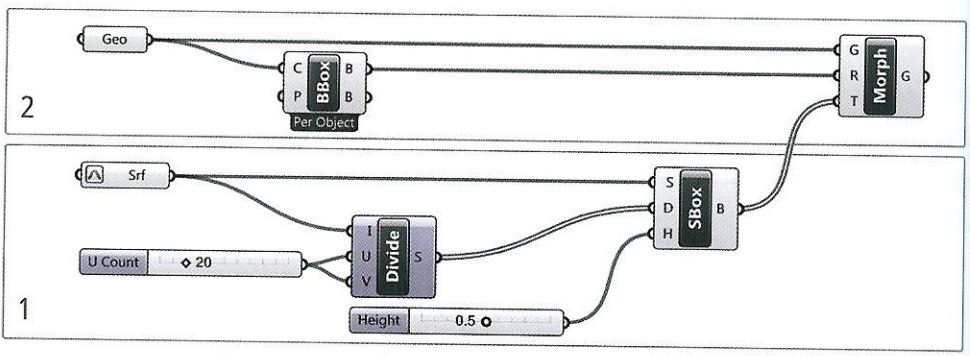
Paneling extends the *Box Morph* logic to multiple *target boxes.* Given an arbitrary surface with a set of twisted *target* boxes a surface-panelization can be defined using a single or a set of merged geometries.



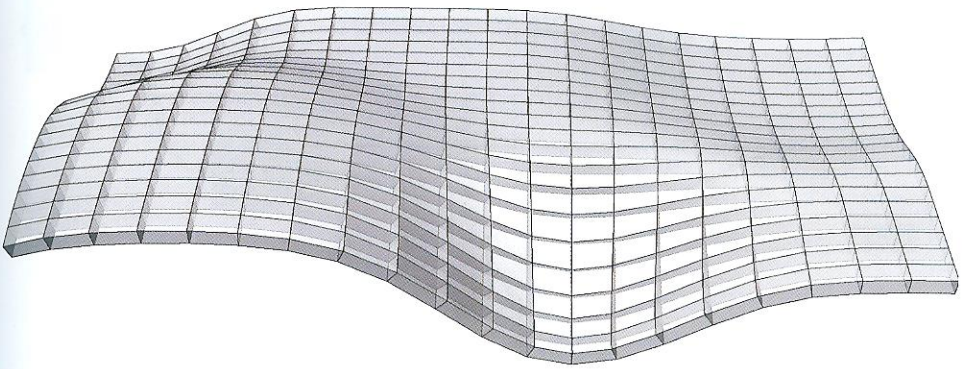
The *paneling* procedure consists of two steps:

1. Creating a set of twisted boxes (T) on an arbitrary freeform surface.
   The component *Surface Box* (Transform > Morph) generates twisted boxes on a surface with a defined bidimensional domain (D) and a height (H).

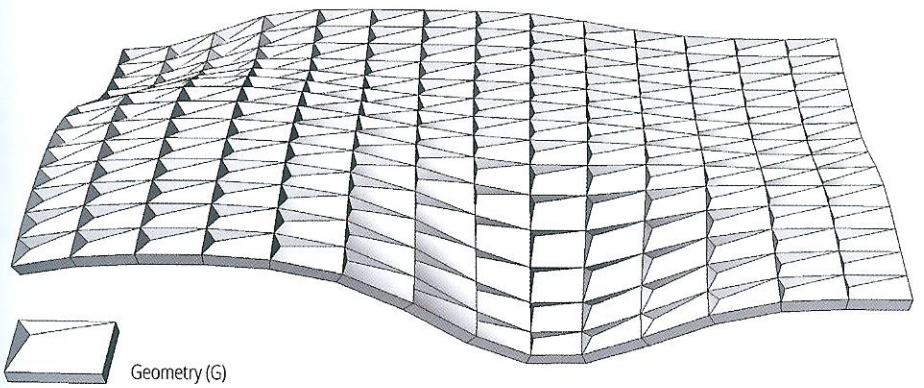
2. Morphing a geometry (G) – encompassed by a *Bounding Box* (R) – to twisted *target* boxes (T).



For example, a grid of 20 x 20 surface boxes – as specified by the bidimensional domain – with a height of 0.5 units are created on the surface using the component *Surface Box*. The *Surface Boxes* are the *Target Boxes* for morphing.



Morphing manipulates the base geometry (G) to match the target boxes (T), creating panels.



If random values are connected to the H-input of *Surface Box* different heights for the target boxes will result; yielding a panelization of random height base geometries (see following images).

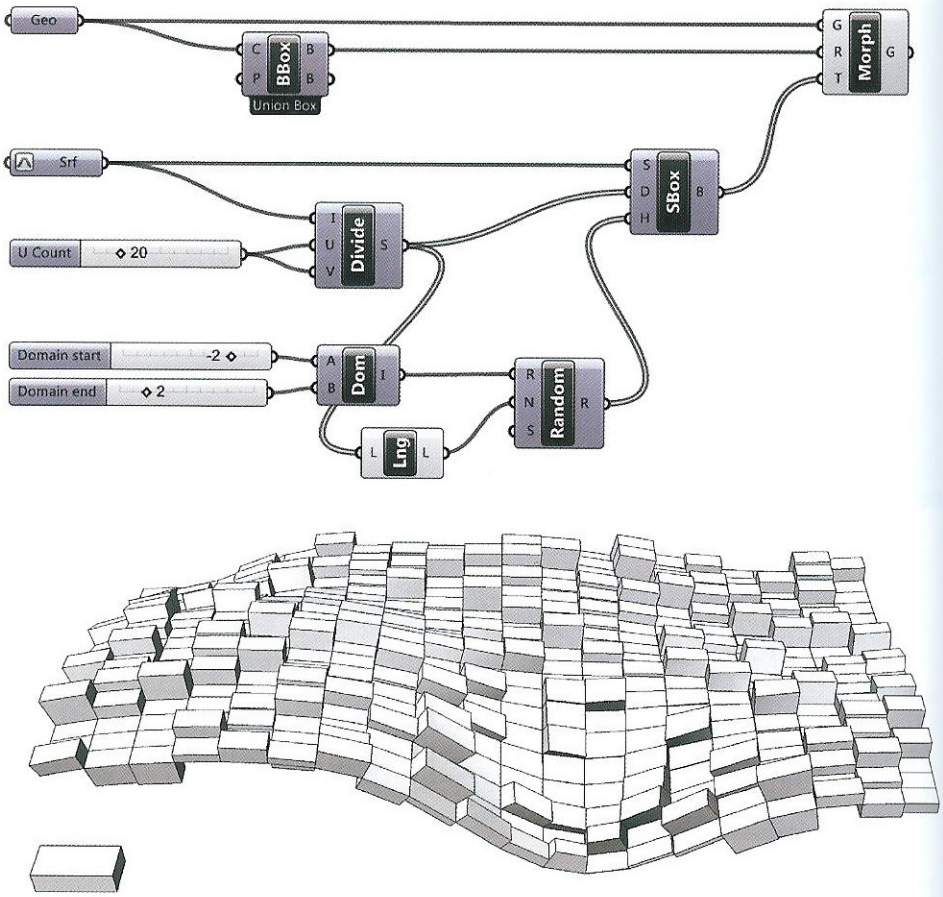


FIGURE 4.6

Interesting output can be achieved using a non constant value for the H-input of *Surface Box*.

Herzog & De Meuron, Beijing National Stadium (Bird's nest). Picture by Jorge Láscar.