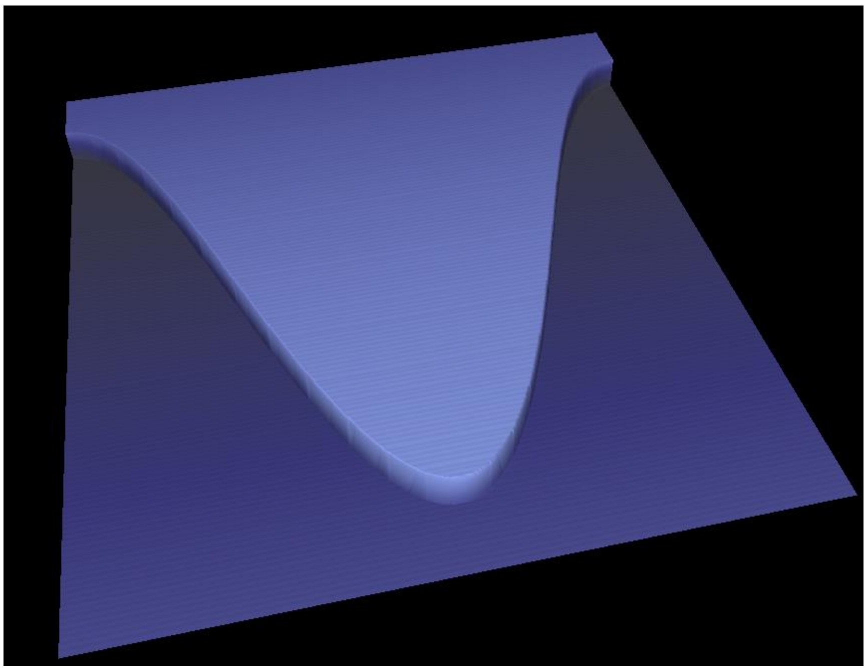
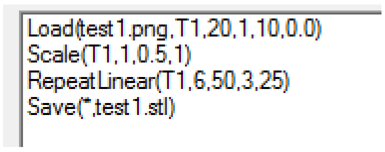
Example using an height map Image from GeoGen

Step 1: create depth map image using GeoGen (see later example for more details on using GeoGen). Here is the image displayed as a depth map:



Step 2: Create script in Texture Builder:

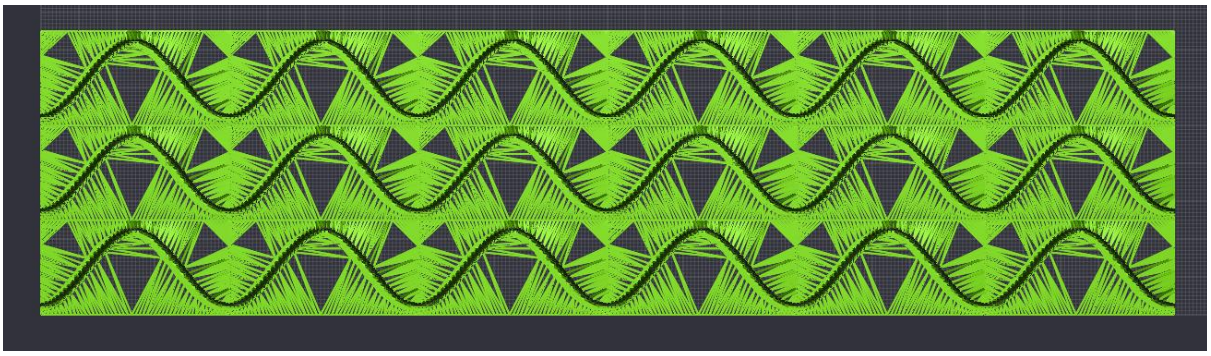


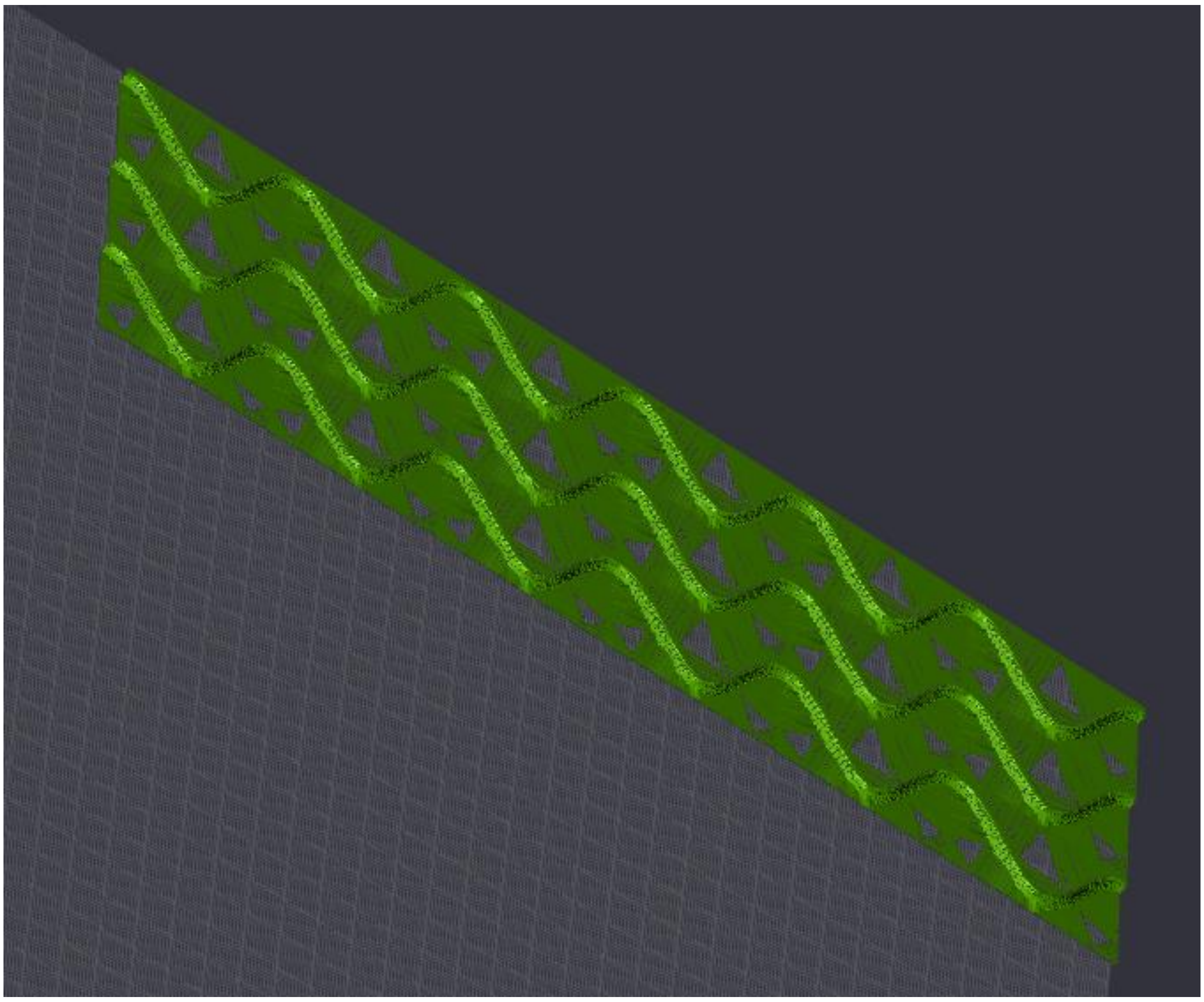
```
Load(test1.png,T1,20,1,10,0,0)
Scale(T1,1,0.5,1)
RepeatLinear(T1,6,50,3,25)
Save(*,test1.stl)
```

In this script:

- Load the image file (test1.png), to a tile named T1. Down size it by a factor of 20 (the 1000x1000 pixel image becomes an grid of 50 x 50 (X,Y) points. The simplification error is set at 1 (in the range 0 to 255), and a pixel grey scaled value of 255 is mapped to a Z value of 10.0.
- The tile T1 is scaled by a factor of 0.5 in the -direction
- The tile T1 is repeated 6 times in the X-direction at steps of 50 (mm) and 3 times in the Y-direction at steps of 25 (mm).
- All the resulting tiles are saved to the file test1.stl.

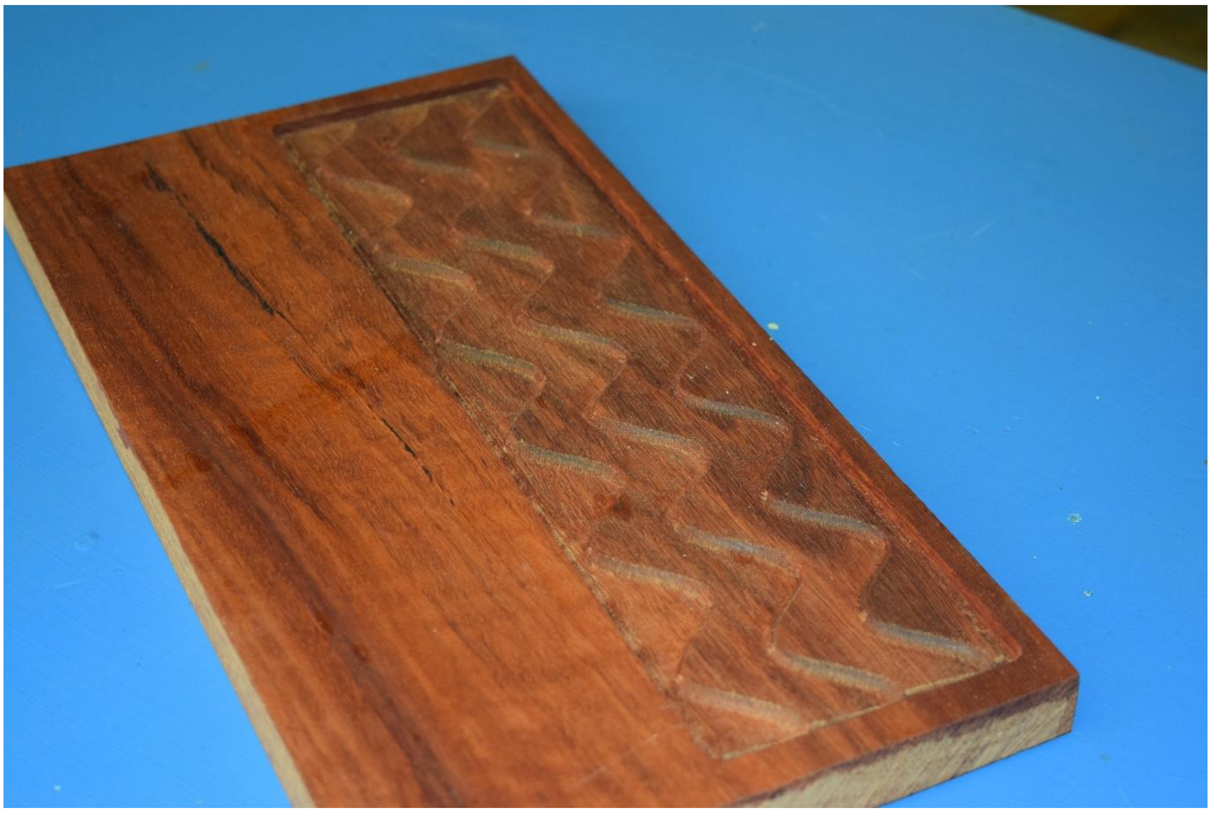
Step 3: Import result into CamBam:





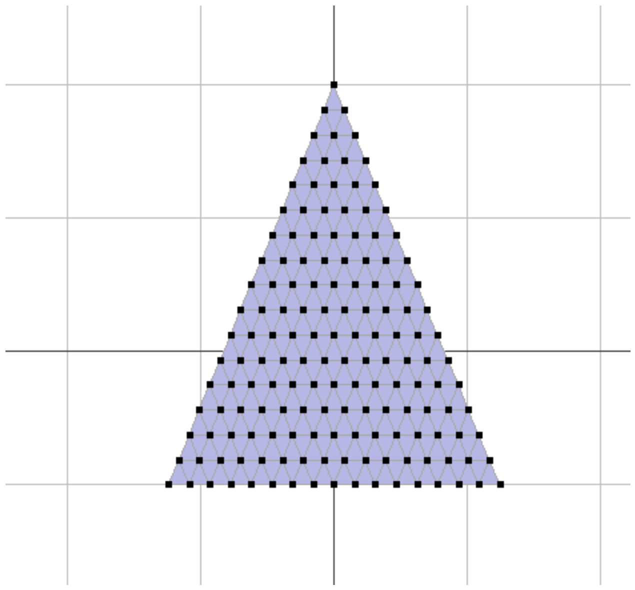
The effect of the surface simplification can be seen in this model

Here is a sample machined result:

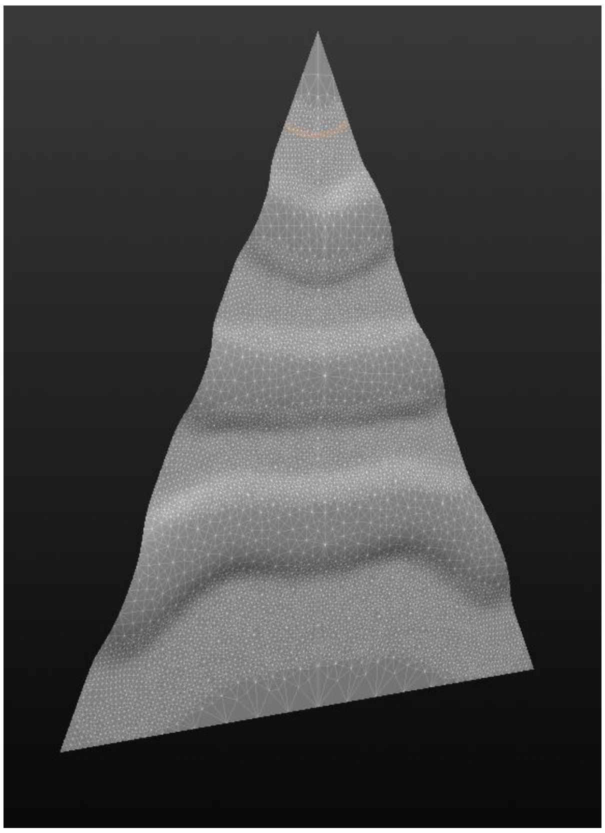


Rotation Example Using AOI & Sculptris

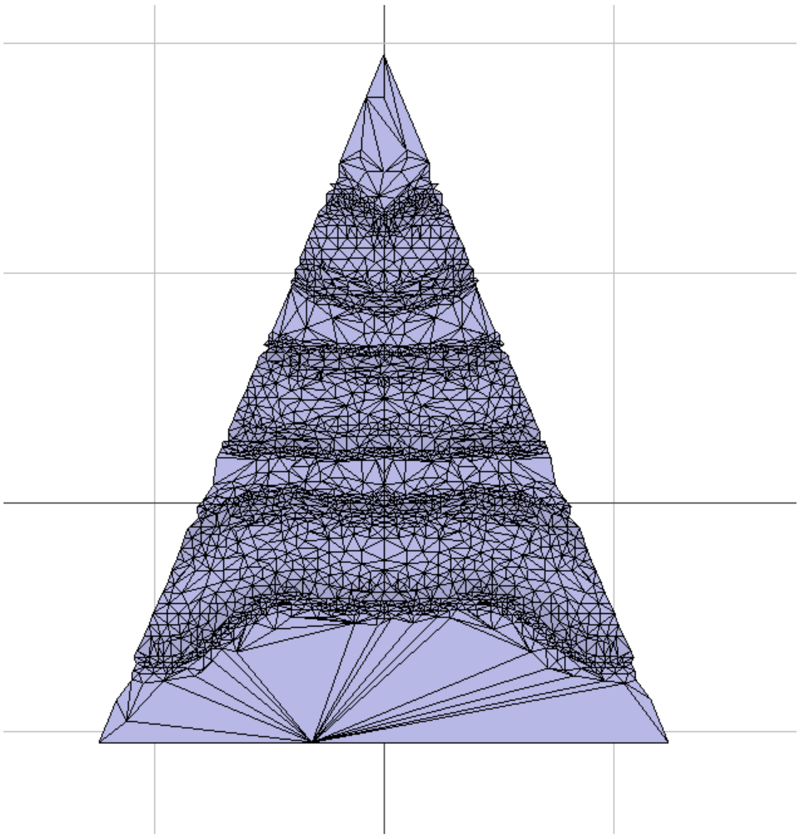
Step 1: Basic flat triangulated shape crated in AOI, saved as OBJ file.



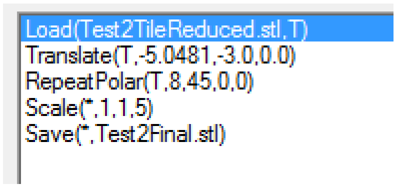
Step 2: Import into Sculptris as OBJ file, then created surface with symmetry line down centre, then export to an OBJ file.



Step 3: Import back into AOI and simplify mesh (the Sculptris mesh contains far too many triangles to be useful), export as STL.



Step 4: Create script in Texture Builder:

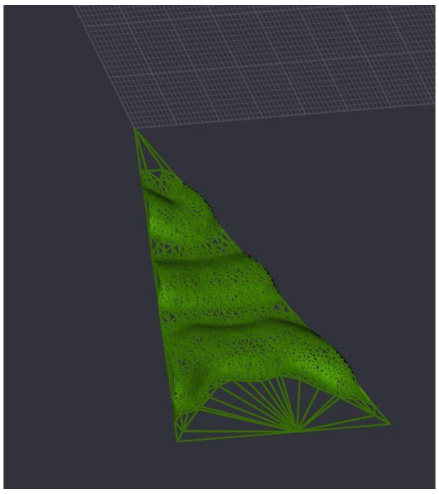


```
Load(Test2TileReduced.stl,T)
Translate(T,-5.0481,-3.0,0.0)
RepeatPolar(T,8,45,0,0)
Scale(*,1,1,5)
Save(*,Test2Final.stl)
```

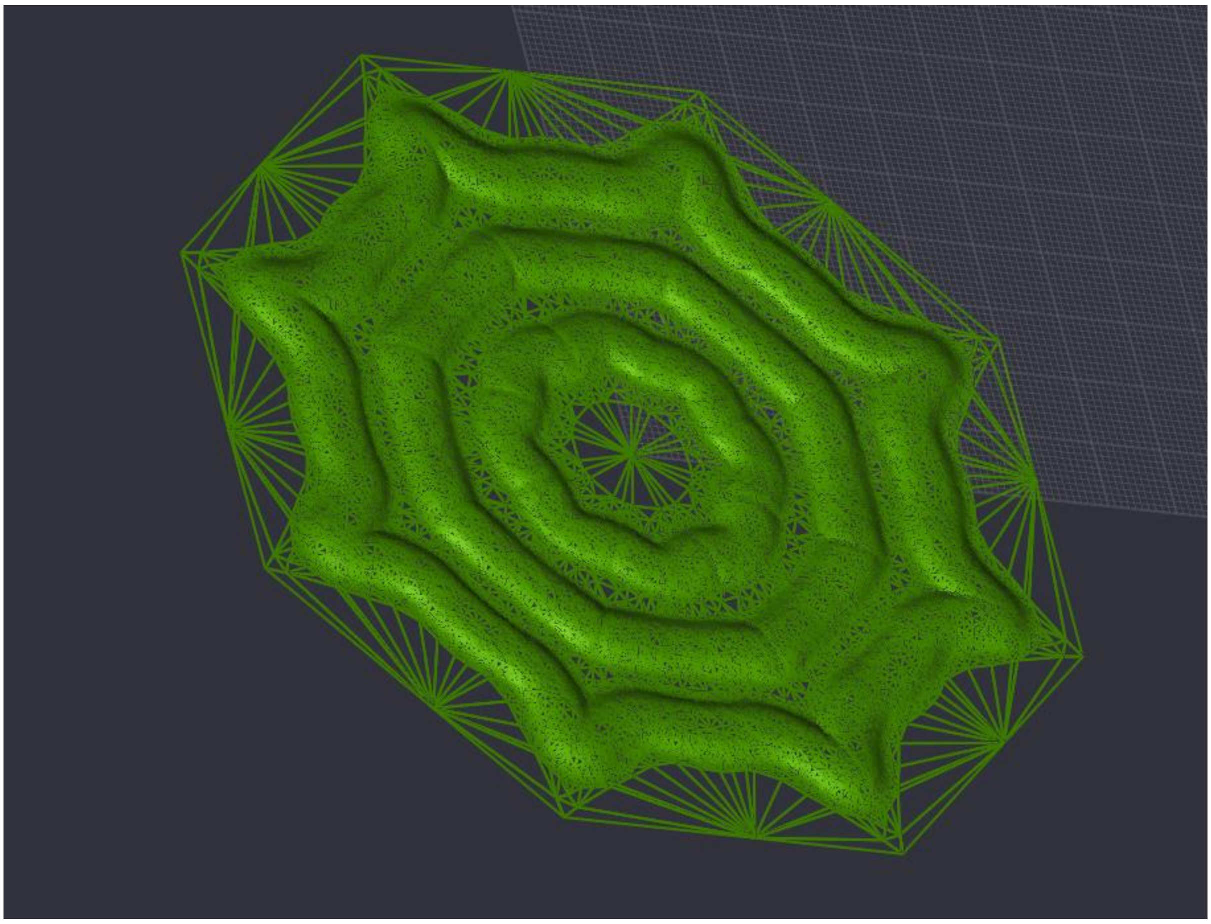
This scrip:

- Loads the file Test2RileReduced.stl, and creates a tile T;
- Translates the tile (T) so that the top of the triangle is at the origin.
- Makes 8 copies of tile T at 45 deg intervals around a circle at the origin.
- Scales all tiles by a factor of 20.
- Saves all tiles to a STL model file Test2Final.slt.

Step 5: Comment out <RepeatPolar> command for the moment and <Run> to see a single tile located at origin (this step is often useful just to see what you get from the loaded model):

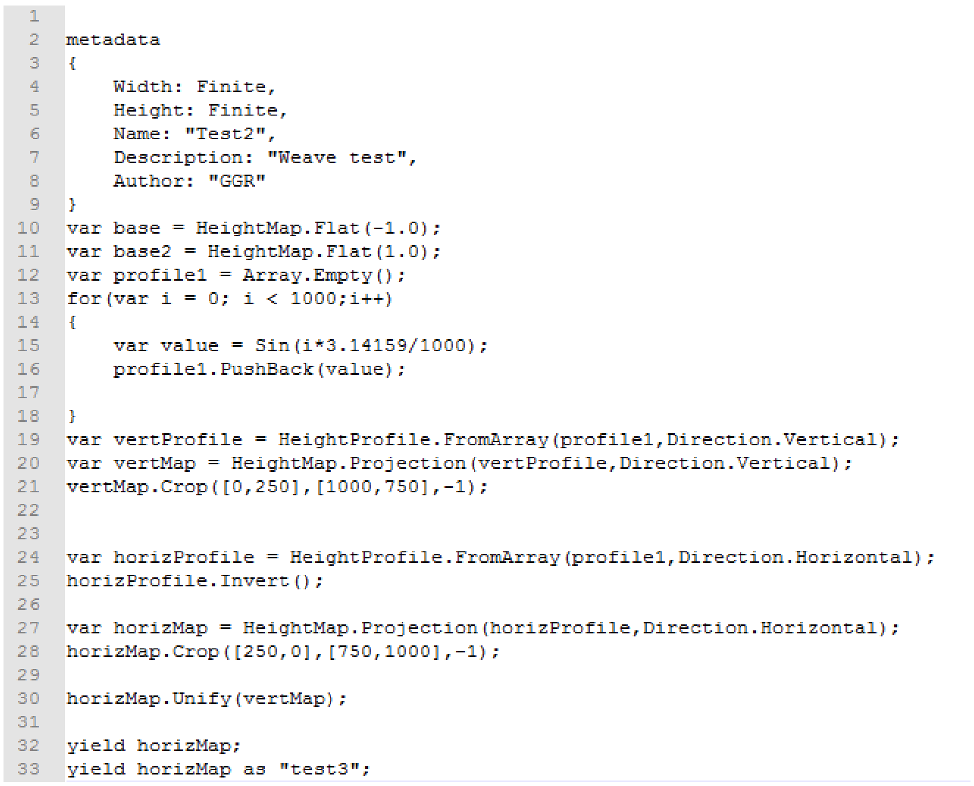


Step 6: Uncomment <RepeatPolar> command and <Run> again to give the complete model in CamBam:



Weave Texture Example From an Image Using GeoGen:

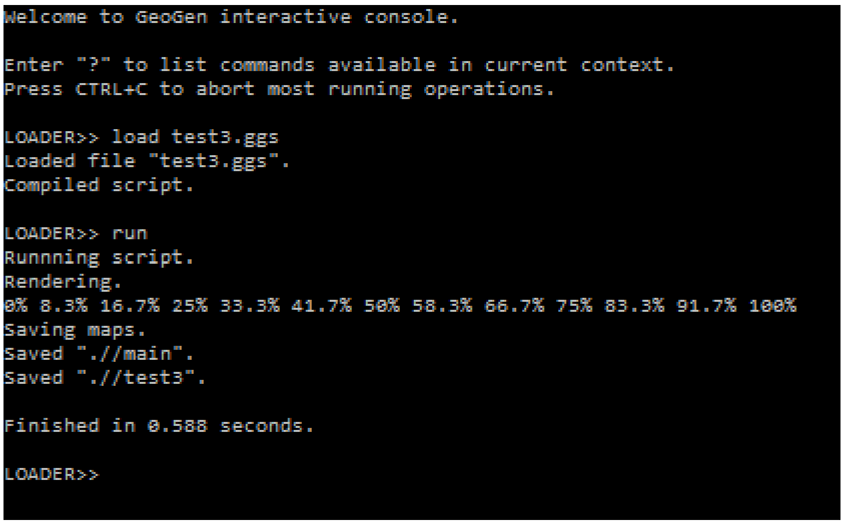
Step 1: Create a GeoGen script (GGS):



```

metadata
{
    Width: Finite,
    Height: Finite,
    Name: "Test2",
    Description: "Weave test",
    Author: "GGR"
}
var base = HeightMap.Flat(-1.0);
var base2 = HeightMap.Flat(1.0);
var profile1 = Array.Empty();
for(var i = 0; i < 1000;i++)
{
    var value = Sin(i*3.14159/1000);
    profile1.PushBack(value);

}
var vertProfile = HeightProfile.FromArray(profile1,Direction.Vertical);
var vertMap = HeightMap.Projection(vertProfile,Direction.Vertical);
vertMap.Crop([0,250],[1000,750],-1);


var horizProfile = HeightProfile.FromArray(profile1,Direction.Horizontal);
horizProfile.Invert();

var horizMap = HeightMap.Projection(horizProfile,Direction.Horizontal);
horizMap.Crop([250,0],[750,1000],-1);

horizMap.Unify(vertMap);

yield horizMap;
yield horizMap as "test3";
```

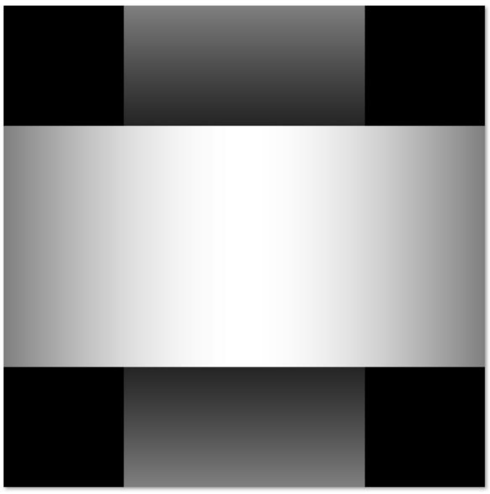
The basic shape is created using a Sine function (other shapes could be used)

Step 2: Execute script in GeoGen to create a PNG image file (1000 x 1000 pixels) representing the height map:

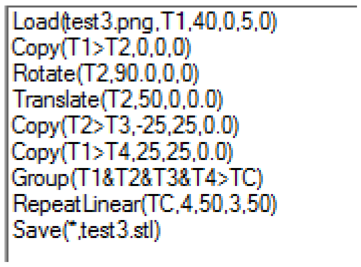


```
Welcome to GeoGen interactive console.

Enter "?" to list commands available in current context.
Press CTRL+C to abort most running operations.

LOADER>> load test3.ggs
Loaded file "test3.ggs".
Compiled script.

LOADER>> run
Runnning script.
Rendering.
0% 8.3% 16.7% 25% 33.3% 41.7% 50% 58.3% 66.7% 75% 83.3% 91.7% 100%
Saving maps.
Saved ".//main".
Saved ".//test3".

Finished in 0.588 seconds.

LOADER>>
```

That looks like this, i.e. the basic tile pattern for the required texture:



Step 3: Create script in Texture Builder to process the image and create a tessellation:

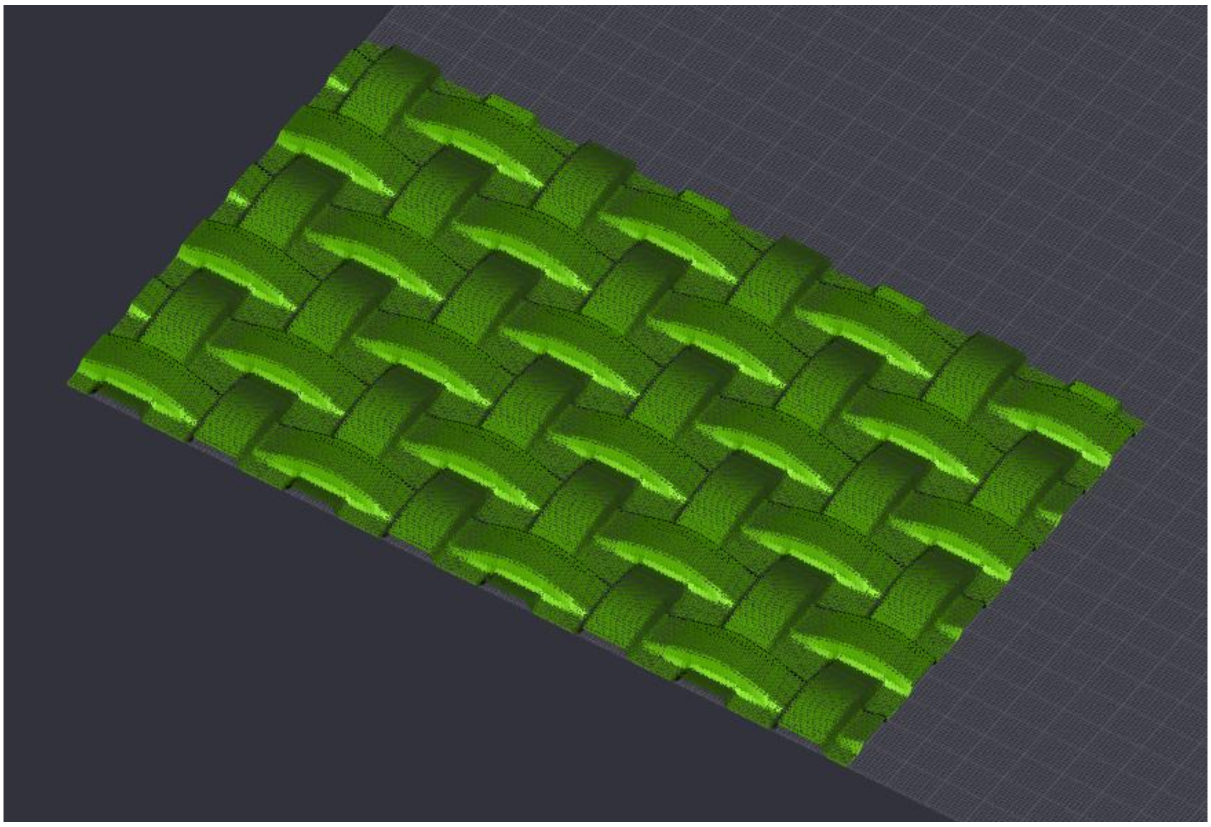


```
Load(test3.png,T1,40,0,5,0)
Copy(T1>T2,0,0,0)
Rotate(T2,90,0,0,0)
Translate(T2,50,0,0,0)
Copy(T2>T3,-25,25,0,0)
Copy(T1>T4,25,25,0,0)
Group(T1&T2&T3&T4>TC)
RepeatLinear(TC,4,50,3,50)
Save(*,test3.stl)
```

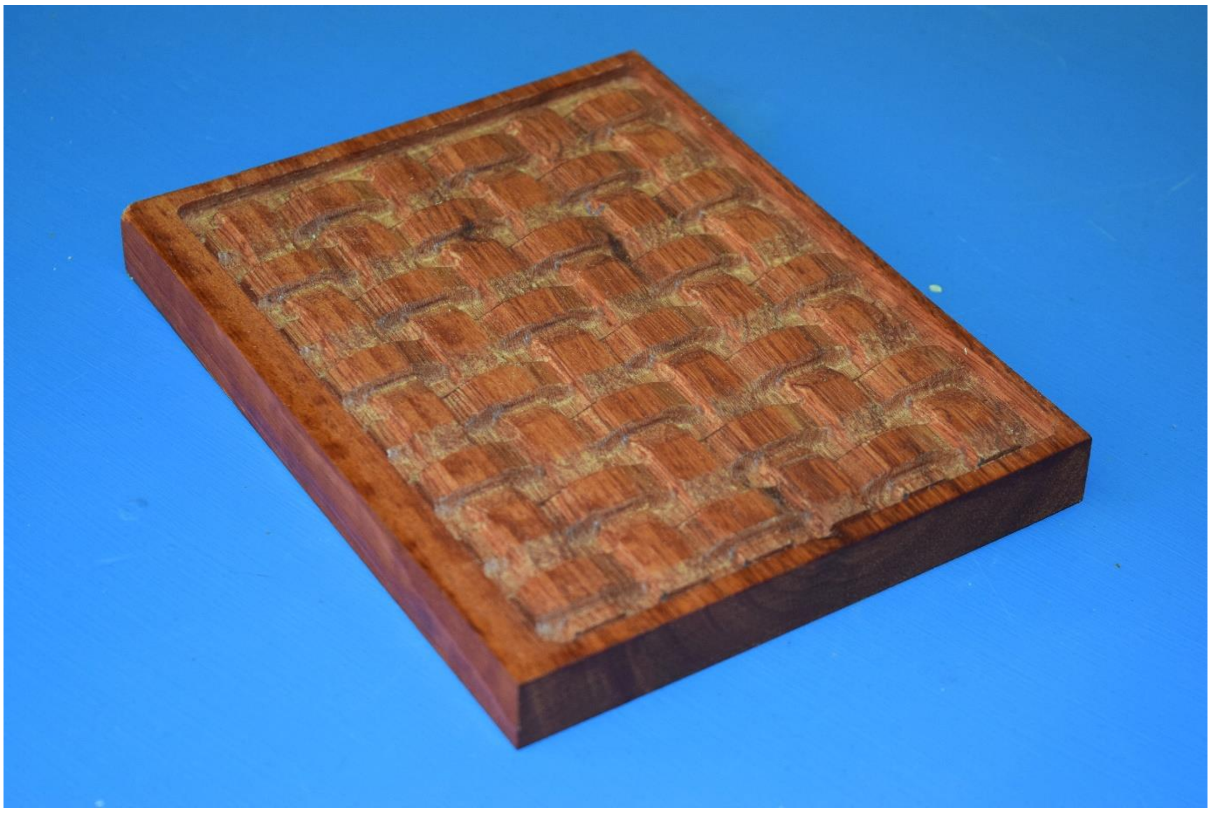
The steps in this script are:

- Downscales the image by a factor of 40 (so tile size is 25 x 25 mm here), with no simplification and sets Z-white to a Z value of 5 (mm), to create a tile named T1.
- Make a copy of T1 and name it T2, also at the origin.
- Rotate T2 by 90 deg (clockwise) about the origin.
- Translate T2 by 50 units in the X-direction.
- Make a copy of T2 and move it -25 units in the X-direction and 25 units in the Y-direction, and name it T3.
- Make a copy of T1 and move it 25 units in the X-direction and 25 units in the Y-direction and name it T4.
- Group the tiles T1, T2, T3 and T4 into a new tile named TC (T1, T2, T3 and T4 are now disabled).
- Repeat TC 4 times in the X-direction at steps of 50, and 3 times in the Y–direction at steps of 50.
- Saves the result (all enabled tiles) to a file named test3.stl.

Step 4: <Run> this script and import this file into CamBam to give the weave texture:



Here is a sample machined result:



Example using random pattern from OpenJSCAD

OpenSCAD is a standalone package for building 3D models parametrically using CSG. It uses its own scripting language which, while quite compact, does require some commitment to learn and apply. OpenJSCAD is a Javascript version of a similar modelling process. After trying both, I think the OpenJSCAD version is probably a better starting point as it provides:

- Access to a well-defined programming language.
- An inherent O-O approach that can be usefully developed for complex tasks.
- OpenJSCAD seems to run faster on more complex models that OpenSCAD.

OpenJSCAD can be run inside a browser (and online from the openJSCAD.org web site), or locally on a client computer using a command line interface (CLI). The CLI provides more flexibility for use for anything but simple models, but does not include a GUI for viewing the models. With the help of a good code editor (e.g. NotePad++) it is possible to setup a working environment to support coding and debugging. Models created by the CLI can be conveniently viewed in the browser version, or other STL file viewer (e.g. MeshLab)

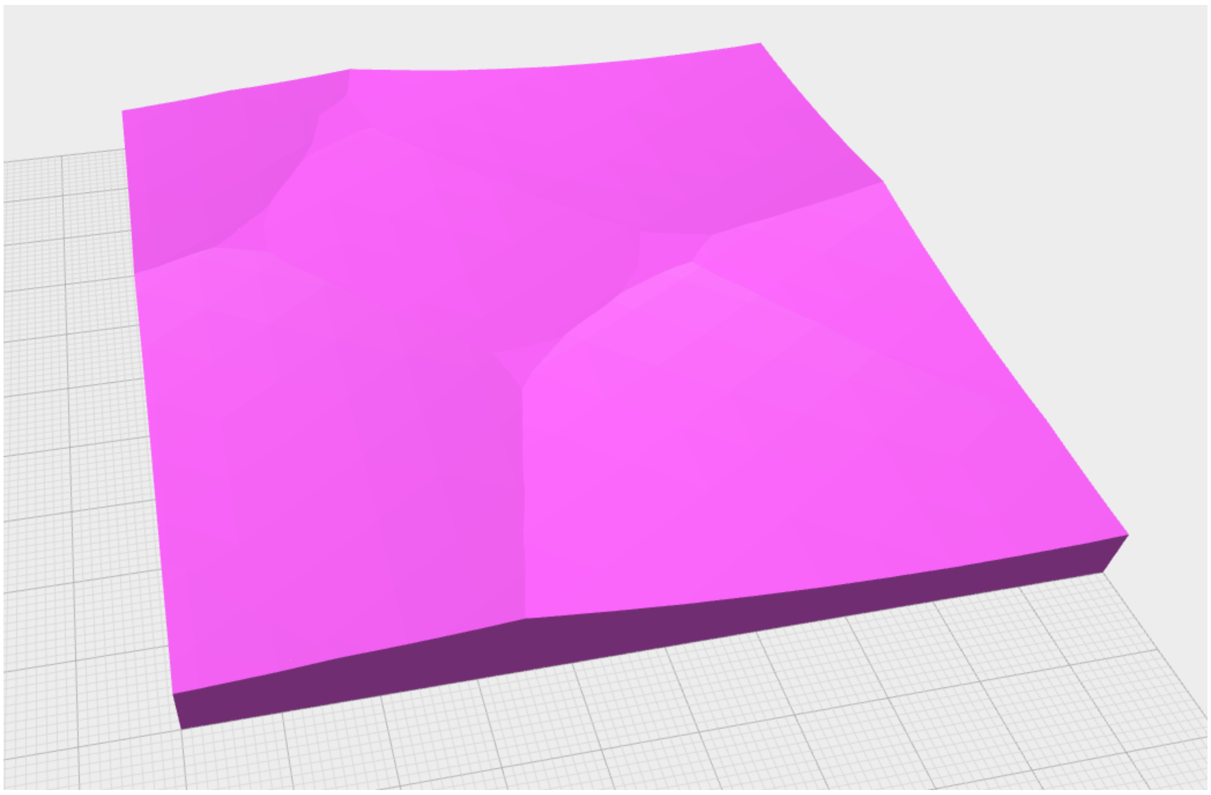
The goal for this example is to create a scalloped surface created by removing spherical patches from a flat surface so that the whole surface is scalloped. Three spheres of the same radius are used they but are positioned at varying X,Y positions and varying heights using random values for their centre coordinates. Several trials are necessary to get a “complete” coverage of the surface of the square flat surface. Special care is required to ensure that the edges align when the tile is assembled in the tessellation.

Step 1: Create model in OpenJSCAD using a Javasscript program. In OpenJSCAD the scene is defined parametrically, some points to note:

- The base shape is a rectangular block.
- A number of spheres (3) of constant radius are generated at varying X,Y,Z positions based on a random number generator and subtracted progressively from the top of the base block.
- To ensure edge consistency, if a sphere overlaps an edge of the rectangular block, another sphere is added outside the opposite edge.
- The seed for the random numbers can be changed to produce different and, if required, predictable results.

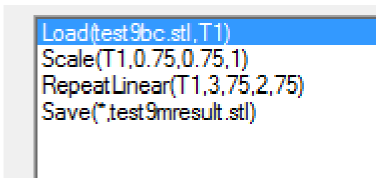
The program is not trivial, and requires some care to get it all working. On the upside OpenJSCAD offers considerable flexibility to create a wide range of patters using a parametric CSG approach.

Step 2: Run the program to produce a rendered model (takes several minutes) to give the result as viewed in the OpenJSCAD browser:



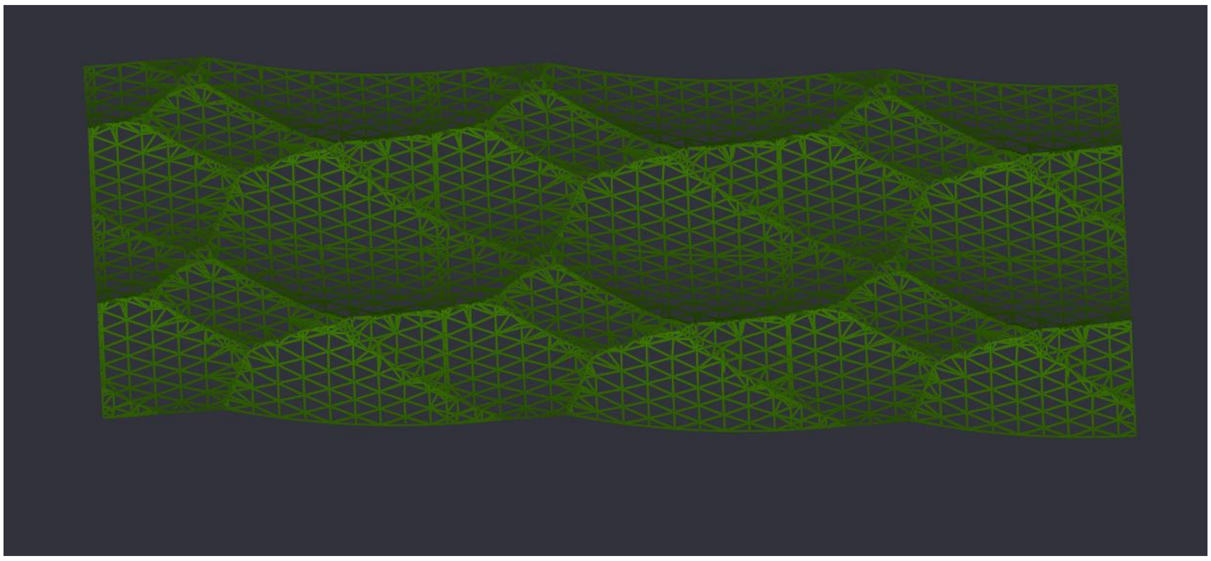
Step 3: Use Meshlab to clean up the model: to remove back and side faces.

Step 4: Create a Texture Builder script:



```
Load(test9bc.stl,T1)
Scale(T1,0.75,0.75,1)
RepeatLinear(T1,3,75,2,75)
Save(*,test9mresult.stl)
```

Step 5: Run this script and import the result into CamBam:



Here is a sample machined result (simular to, but not identical to the above model):