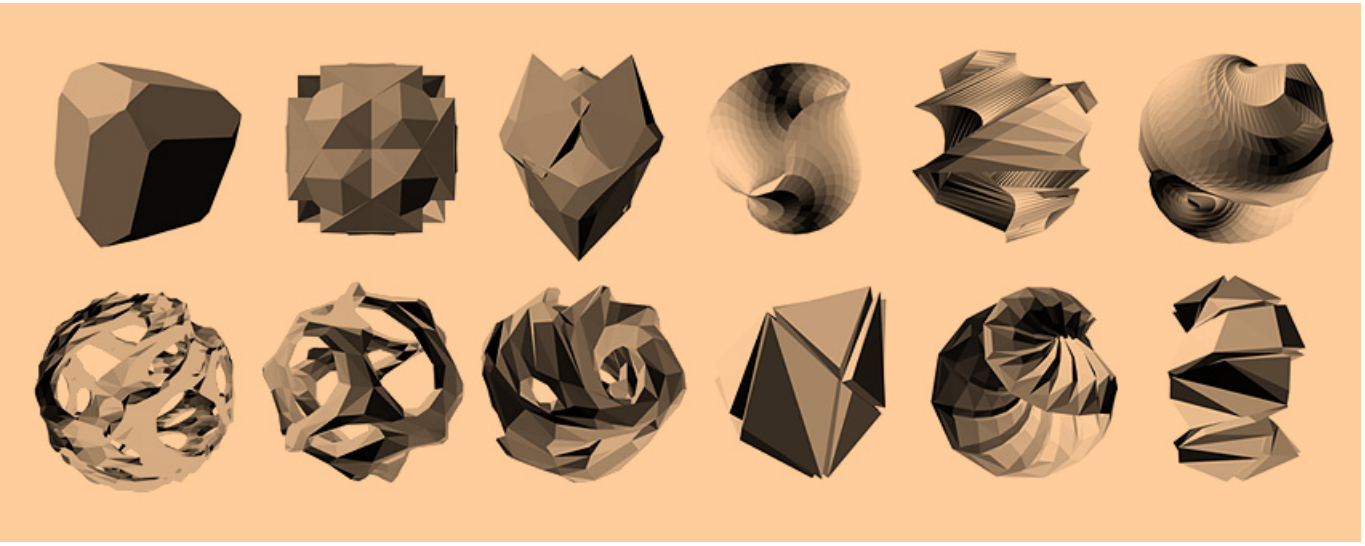
# Hemesh : 3D Printing Part 1

« Previous / Next »

By mark webster / January 4, 2015 / Learning / No Comments



Various evolutions of Hemesh shapes

////////////////////////////////////////////////////////////////////////
—— **HEMESH Library by Frederik Vanhoutte** ———-
////////////////////////////////////////////////////////////////////////
Frederik Vanhoutte's website

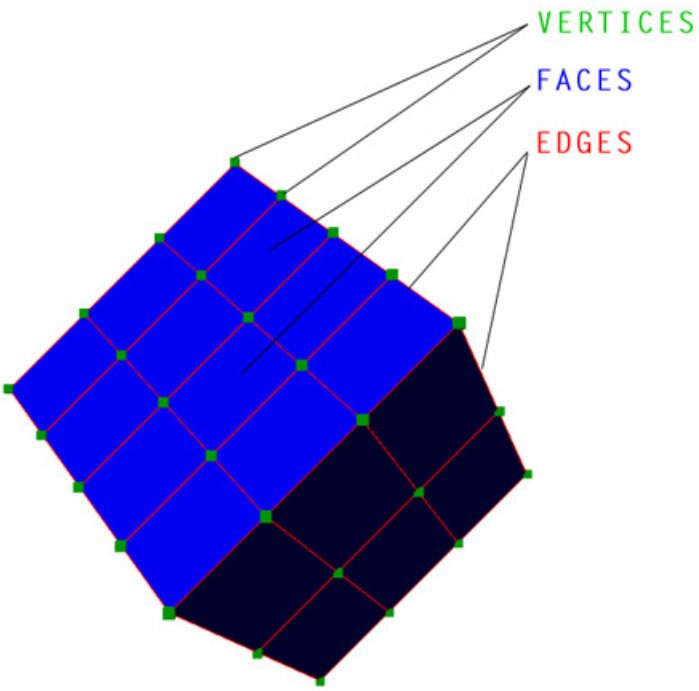
Source Code :
Sketches Processing
Hemesh Library

////////////////////////////////////////////////////////////////////////

**INTRODUCTION**

HE_MESH is a powerful library for creating 3 dimensional geometry in Processing and is a great tool for exploring the possibilities of 3D printing. The basic logic in creating shapes with HE_MESH relies on some understanding of the fundamentals of 3D geometry and more specifically on what we call polygon geometry. In this entry level tutorial, we'll be looking at how Hemesh is configured, how to make basic shapes and finally how to export these for printing. All source code can be found at the following address on GitHub.

In rendering 3D graphics, everything begins with the mesh. What is a mesh? A mesh in polygon geometry is a collection of vertices, edges and faces each of which defines a polyhedral shape. Cubes, pyramids and prisms are all examples of polyhedral shapes as well as the five Platonic solids. In 3 dimensional geometry, these shapes often consist of triangular or quadrilateral faces.




A vertex is a position in space and can contain
other information such as colour or texture coordinates.
An edge is a connection between two vertices
A face is a closed surface of vertices and edges.

The mesh is therefore the most basic principal behind creating 3D shapes. However, there exists a number of ways to create meshes in the field of graphics. HE_MESH uses what is called half-edged mesh geometry to create all it's shapes, hence the name; HE signifying half-edged and mesh is, well, the mesh. The main building block behind HE_MESH is based on this principal. Objects created with polygon meshes all store different types of data : vertices, edges, faces, polygons and surfaces. On an expert level, this is important information because half-edged mesh geometry is an efficient means for storing vertices and face data. Perhaps more importantly this data contains information about how each of these are inter-connected and therefore gives the more advanced user the possibility to access it. If you want to read more on this subject, please check Frederik's link to an article written by Max McGuire.

HE_MESH therefore has a certain logic based on a specific data structure but we don't need to go too deep into that at this point to get some graphics up and running on the screen. Let's glide in at ease with the fundamentals. There are four main basic classes (non-exhaustive) that make up HE_MESH. We are going to look at two of these in particular but all four will be necessary for creating our graphics.

- All classes prefixed by HE_ specify core classes for accessing specific data concerning our mesh.
- All classes prefixed by HEC_ specify creators for creating & configuring the basic geometric shapes.
- All classes prefixed by HEM_ specify modifiers for applying a number of modifications to our shapes.
- All classes prefixed by WB specify a multitude of math and rendering possibilities.

All shapes are therefore created using the HE_Mesh class and the HEC_ classes. HEM_ classes are used for modifying our shapes and we'll see how to implement these a little later but let's have a look at one of the most basic shapes, the cube, and walk through the steps for creating this shape.

**THE BASIC LOGIC FOR CREATING A MESH WITH HE_MESH**

How do we create a mesh first of all? HE_MESH has both a class for initialising a mesh - HE_Mesh - and a class for creating the mesh - HEC_. This keeps the workflow a lot easier to handle because we have the HE_Mesh class that is dedicated to giving us access to the core data elements of a mesh whilst the HEC_ class is a family of classes that allows us to choose different shapes, each of which have their own unique methods necessary for

configuring them. So, one class to initialise, and one to create with a set of methods. For example, the HEC_Cube class has a number of methods for setting the edge, segment height, width, depth as well as the inner, mid and outer radius.

**BASIC SETUP**

Open sketch A_cube_basic_setup. The first thing to notice is the HE_Mesh MESH object and our WB_Render RENDER object declared as globals. Our MESH object contains nothing for the moment but as explained above we need to configure a creator using one of HE_MESH's HEC_ classes and add this to our MESH. This is exactly what can be seen in setup() using HE_MESH's HEC_Cube class and then calling this within the new HE_Mesh() instance.

```
1  HE_Mesh MESH; // mesh object
2
3  …then in setup();
4  HEC_Cube creator; // mesh creator object
5
6  //The actual mesh is created by calling the mesh creator in the HE_Mesh constructor
7  MESH = new HE_Mesh(creator);
```

The WB_Render class provides functions for all our objects within the library and is necessary to initialise at the beginning of our program. We use this class to render our form to the display too.

```
1  WB_Render RENDER;
2
3  …then in setup();
4   RENDER = new WB_Render(this);
5
6  …then in draw();
7   RENDER.drawFaces( MESH ); // Notice we add the mesh object within the parentheses.
```

Something important is also going on in this first sketch which has nothing in particular to do with HE_MESH as such. To view our 3D shape, I've opted for PeasyCam because this simplifies the process a little, letting us concentrate on the various steps of construction with HE_MESH without having to bother with translations and mouse interaction. If you don't already have the PeasyCam library installed, you can download it here: http://mrfeinberg.com/peasycam/

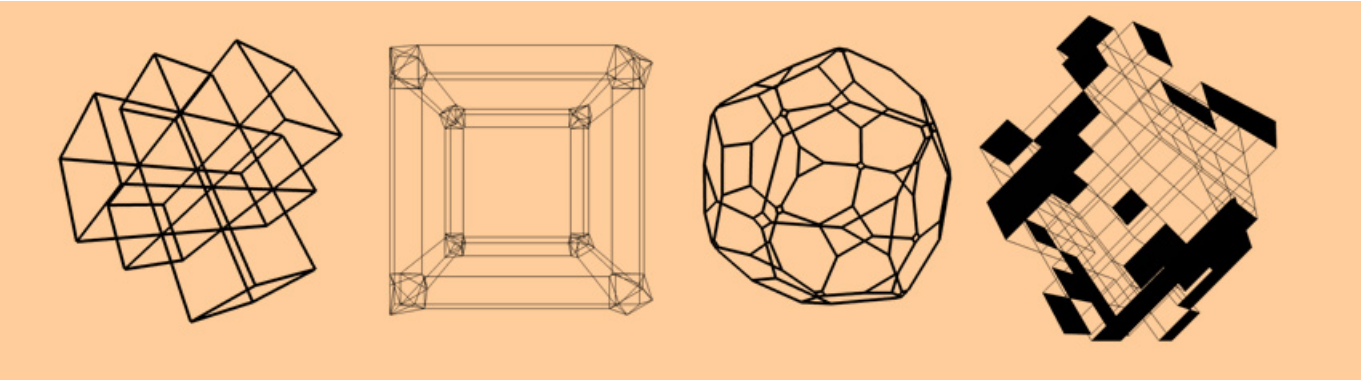


Illustration of various modifiers

**RUNNING THROUGH SOME BASIC METHODS**

Now open our B_cube_basic_setup_02. Notice the various methods for setting up our creator. Each creator in HE_MESH has a number of methods and you will need to read the documentation for these if you want to learn more. In this basic example, we are specifically looking at the methods for our HEC_Cube creator. So, edge length, segment height and width etc. We can also specify the position and rotation of our shape which is helpful when we want to render to screen from a precise view.

In draw(), I've also added various methods for viewing our three main actors in all this : faces, edges, vertices and normals. These methods are part of the WB_Render class and can be useful for visualising the construction of our shapes or debugging. For example, if you activate in setup() the triangulation method (which is part of the HE_Mesh class), you'll see that our set of cubes are not drawn in the same manner. Instead of quadrilateral faces, we now have triangular faces. However, if we had not drawn our edges, this change in construction would not have been apparent. The WB_Render class contains many other methods for viewing important and helpful data

about how our shapes are created : drawVertex, drawVertexNormals, drawFaceNormals being some of the most useful. For those of you who would like to know a little about normals then have a read of Max McGuire's basic intro.

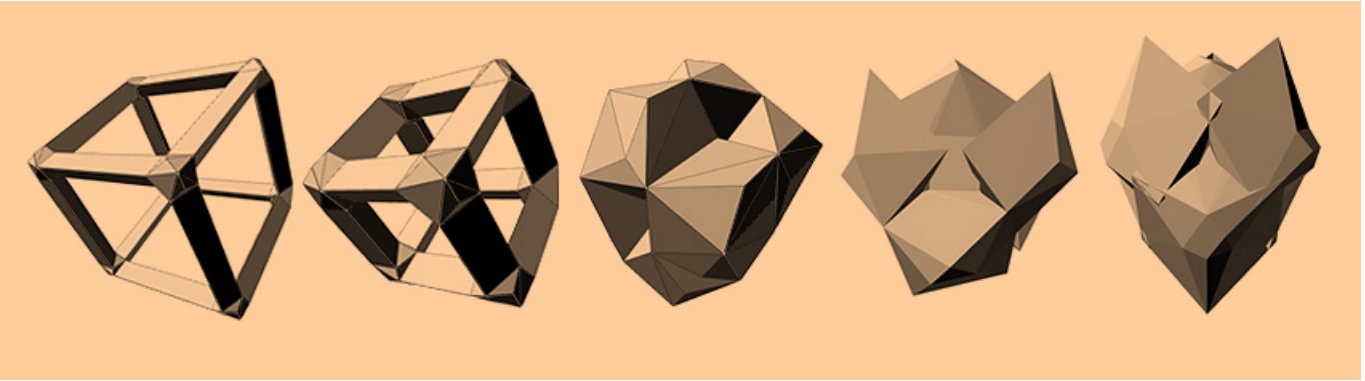
**MODIFIERS**

Lets move on to modifiers. These are also a separate class in HE_MESH and are similar to creators. They do exactly what they say they do i.e. they modify the mesh form. To do this, we first choose a HEM_ modifier class, set some parameters then use the modify() method to apply the modifier to our mesh. Open up the C_cube_modif_extrude_01 sketch.

- The HEM_Extrude class does exactly what it says on the box. It will take a face of a shape and extrude it by a certain distance in pixels.

```
//
HEM_Extrude extrude = new HEM_Extrude().setDistance(70);
MESH.modify( extrude ); // ADD OUR MODIFIER TO THE MESH
```

*Note that in this example we have simply created a cube and not set any parameters for segment width, height or depth. In doing this, HE_MESH creates a cube with default settings of 1 for each of these methods. Try changing these and seeing what happens.



Evolution of a cube

**COMBINING MODIFIERS**

Open sketch D_cube_modif_chamfer_01. In this next sketch we apply two modifiers at the same time showing the possibilities of mixing different modifiers together. Some surprising graphic results can be created here, especially when abused. Go easy on values though as some modifiers demand a lot of memory resources and can seriously slow down or even crash your program. So experiment with small values to begin with. Remember that some modifiers' parameters are float and therefore small values make for big changes. The ChamferEdges & ChamferCorners methods apply beveled edges to our shape. Experimenting with different values can lead to some very different shapes.

A note about stl export format. In the above example, I've added the HE_MESH function to export our shape as an stl file (Stereolithography).
Read more about this format here. Once you have stl files, you're all set for printing on a 3D machine.

In the next tutorial, we'll be looking at how we can add some simple keyboard interactions as well as using ControlP5 as a means for interacting with shapes and making modifications in real-time. This should help in exploring a variety of forms that can be eventually fine tuned for exporting and printing.

**Tags:** 3D, 3Dprinting, geometry, Hemesh, learning, printing, Processing, tutorial