ALGORITHMIC JEWELRY APPLICATIONS IN GRASSHOPPER 3D

Scott Bradford
Technical Sales Engineer
Gesswein
Bridgeport, CT, USA

INTRODUCTION

Originally developed as a CAD program for boat design, Rhino has been adopted by numerous industries for general 3D design applications, including jewelry, architecture, and automotive design, due to the robust offering of tools and commands for creating these 3D models in real world dimensions. As each industry has adopted Rhino for their design applications, each has created their own ways to increase efficiency during the design process through the use of plugins that automate, streamline, or allow more customization for certain repetitive tasks. One such plugin has become an integral part of Rhino and is now part of the full installation: Grasshopper 3D.

Released in 2008, Grasshopper is a node-based visible programming language allowing for parametric or algorithmic modeling within the Rhino environment. Due to the nature of how algorithms are created within Grasshopper, we can create simple inputs that can accomplish complex tasks that would require multiple steps by a CAD designer on their own. This makes Grasshopper a powerful tool to automate certain tasks that are required with each CAD model and streamline the design process.

With the need for automating the creation of certain elements of jewelry design, such as prong creation or making ring shanks for example, developers have created entire suites filled with numerous commands to streamline these processes and make designing easier and more efficient. Plugins for Rhino like MatrixGold, CrossGems, and RhinoGold come with a variety of commands to increase speed, but this comes at a cost. For CAD designers new to the industry, they can be prohibitively so. While recreating the commands within these plugins would require extensive development, education, and time, there is the possibility of

creating some of these using Grasshopper 3D. Let's start with the fundamentals of how Grasshopper works.

HOW GRASSHOPPER 3D WORKS

On the Grasshopper canvas, it starts with a node. The nodes in Grasshopper perform a variety of different functions such as creating an input, converting inputs into something new, running a command similar to a Rhino command, organizing information, amongst many others. Each node has attachment points on both sides: ones on the left to receive information as an input, and ones on the right side to output information to somewhere else as seen in Figure 1. This is a node designed to contain a curve. This curve could have been created in Grasshopper through another node or imported from a Rhino document.

Figure 1: Curve Node

To string inputs and outputs together, we use wires to transfer the output data of a node to an input of another node. We can have multiple wires coming from another node into a single input, and multiple coming from the output. Here's an example of the creation of a Sphere within Grasshopper (Figure 1a). In this example we have a Sphere node on the Grasshopper canvas. It is seeking two inputs: the base plane to orient the sphere, such as the XY, YZ, or XZ axes (set to XY by default), and the radius of the sphere. We have a node called a Panel inputting the number "5" into the radius attachment of the sphere node using a wire. On the right side of the node, is the output which is a sphere with a 5 mm radius.

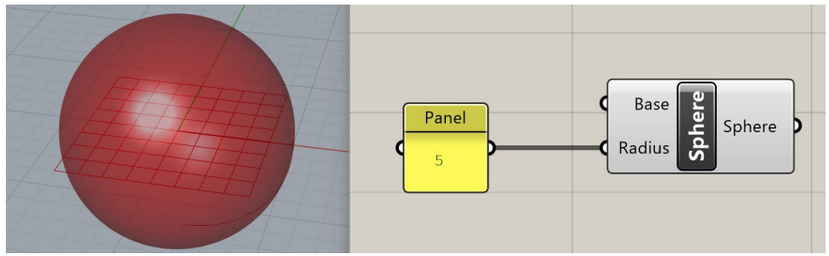

Figure 1a: Panel containing the number "5" attached to radius input of Sphere node

The resulting sphere is shown within our Rhino document in a translucent red color. Based on the input method we choose, we can create static elements like the Panel, simply containing a number "5", or we can choose an input method that allows for more customization. Here's an example (Figure 1b) of the same sphere with a Number Slider input that allows us to manually slide the cursor to adjust the radius value, or key in the specific number of our choosing.

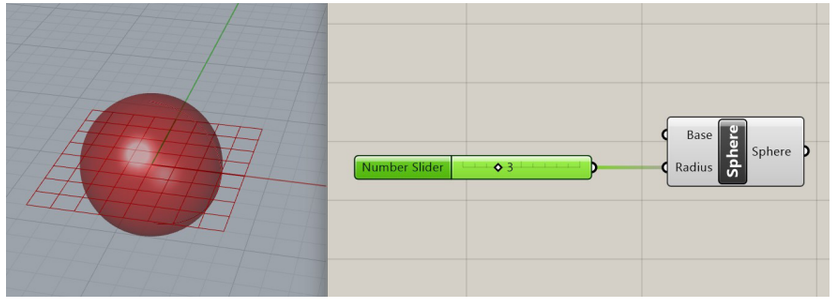

Figure 1b: Sphere node with a Number Slider

This Number Slider, as with other input elements available within Grasshopper, can be customized on their own. We can adjust the title of the slider to be more descriptive, the range of values available within the slider, and number of decimal points for each value. As we continue to create our algorithm, the output, in this case the sphere with a 3 mm radius, can be the input for another node. In Figure 1c, we are using the output of our sphere node as an input for another node through a wire, in this case, the Mesh Brep node. The third node will convert the sphere into a mesh.

BRADFORD

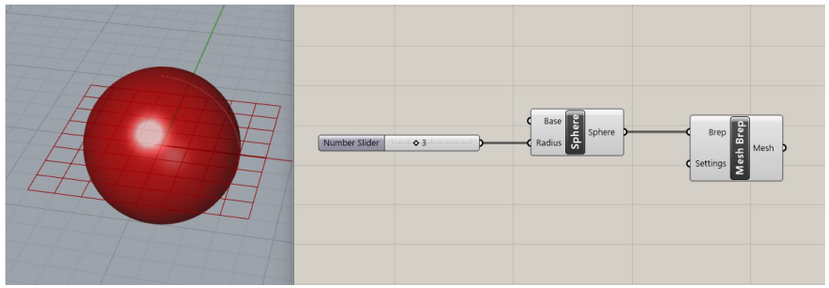

Figure 1c: Output of Sphere node connected to node that will convert to a Mesh

This is the essence of how Grasshopper works. We have a node with an input, or multiple inputs, that creates an output, or multiple outputs. The output of one node is connected to the input of another through a wire, and so forth down the chain. By controlling the types of nodes and the inputs we connect to it, we can automate long chains of commands that are shown in the Rhino document as a representation, with the intention of bringing it into the Rhino document for further manipulation. Grasshopper refers to the process of transferring an output to the Rhino document as "baking."

For a practical jewelry operation, we are going to create a Metal Weights Calculator for our CAD models. In the process, you will see some of the commands that Grasshopper has to offer, and how we can create our own customization within the calculator based on our needs.

CALCULATING METAL WEIGHTS IN RHINO AND GRASSHOPPER

In order to create our Metal Weights Calculator in Grasshopper, first we need to understand how metal weights are calculated within Rhino. This method is generally how jewelry-related plugins are calculating the necessary amount of metal you will need to produce the 3D model. In most cases, these plugins will show the resulting metal estimate in grams or pennyweights, or both, based on the metal you want to cast.

To calculate the estimated metal weight, we first need to calculate the overall volume of the 3D model by running the Volume command within Rhino. This will return the volume of the CAD model in cubic mm. For this example, let's say our model returns a value of 560.245 cubic mm. Next, we must convert that number to cubic cm. This is done by taking the value from our Volume command, and divide by 1000, or move the decimal point three spaces to the left. The resulting value would be 0.560245 cubic cm. Next, we multiply this value by the specific gravity of the metal we will be casting with. For this example, we will be using the specific gravity of 14K yellow gold: 13.07. The result will be in grams.

0.560245 * 13.07 = 7.32240215

The result is 7.32240215 grams of 14K yellow gold required for the entire model. This number will be rounded to the decimal point of your choosing when determining required metal amounts. Now we can break this down through inputs and outputs for Grasshopper. Our inputs will be the CAD model and the specific gravity of the metal we will be casting with. We will start with the Volume command.

For demonstration purposes I have created a size 7 ring, with a half round shank that measures 1.5 mm x 1.5 mm in Rhino. We need to import the ring from the Rhino document into the Grasshopper environment, so we will start by inputting this ring into a node specifically for storing polysurfaces. In the Grasshopper environment, polysurfaces are referred to as a Brep. In Figure 2, you can see the Brep containing the ring polysurface being input into the Volume node, with the output demonstrated on a Panel. The output value is 117.872385 cubic mm. In this case, we are using the Panel node to display output information coming from the node for easier visualization.

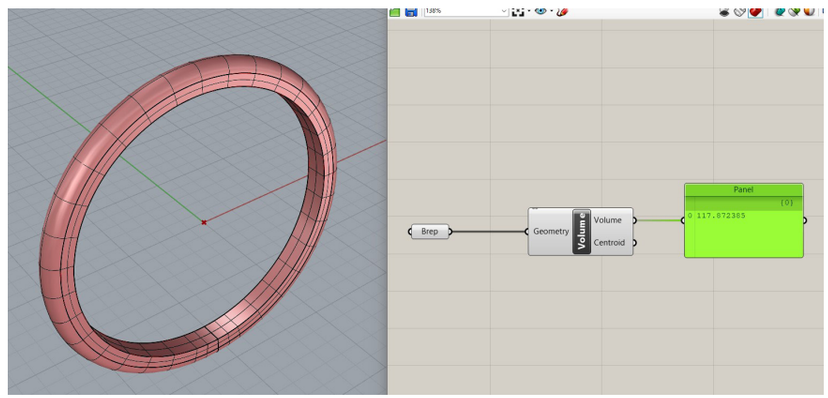

Figure 2: Ring inside Brep node input into Volume node (result shown in Panel)

BRADFORD

Next, we must convert this number from cubic mm to cubic cm. This is done by dividing the resulting number from the Volume command by 1000. We will be using a Division node to accomplish this math. The output of the Volume command is fed into the input of the Division command, along with a Panel containing the number “1000.” This is shown in Figure 2a. The resulting value is 0.117872.

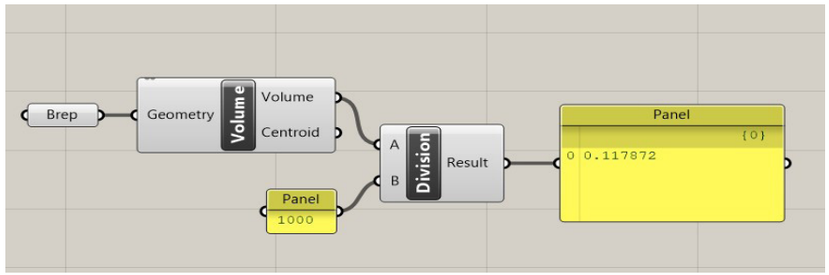

Figure 2a: Volume Command and Conversion.

Next, we must multiply this number by the specific gravity of the metal we will be casting with. In this example, we will be casting with 14K yellow gold which has a specific gravity of 13.07. We will accomplish this by using a Multiplication node so we can multiply the output of the division command by the specific gravity of our metal. This is shown in Figure 2b. The resulting value is 1.540592 grams of 14K yellow gold.

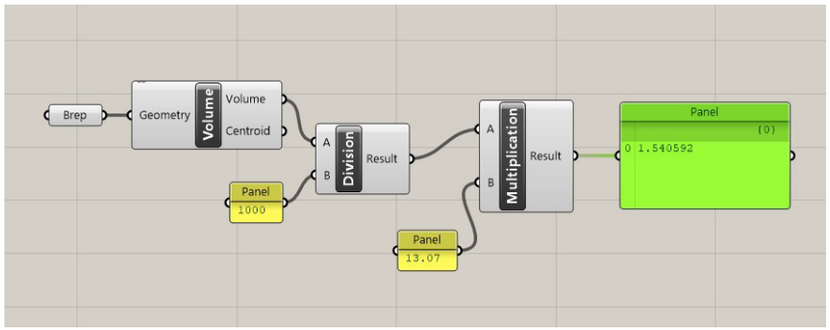

Figure 2b: Output of Division connected to input of Multiplication as well as the specific gravity of 14K yellow gold

This is our metal weights calculator in its simplest form. At this point, we have the capabilities to simply input our model into the Brep node in our algorithm, and the Panel at the end will display the required metal in grams for casting this model in 14K yellow gold. However, in its current form, this calculator is only capable of computing the metal weights for 14K yellow gold, unless we physically change the value inside the Panel connected to the multiplication node to the specific gravity we need. Let’s add some customization so we can choose any metal we wish with more ease.

In Figure 3, I have replaced the Panel containing the value for the specific gravity of 14K yellow gold for another type of input called a Value List. This is a dropdown box where we can customize what is displayed on the dropdown box, and what value that display will transfer through the wire. In this case, I have added a list of metals in the following format:

Plat/Ruth 950 = 21.49

Through this format, the “Plat/Ruth 950” will be displayed on the Value List, but it will transfer the value “21.49” to the next node. With this method, the Value List node can contain all of the precious metals I currently use, and the specific gravities specific to the alloys I prefer, rather than generic specific gravities found online for precious metals.

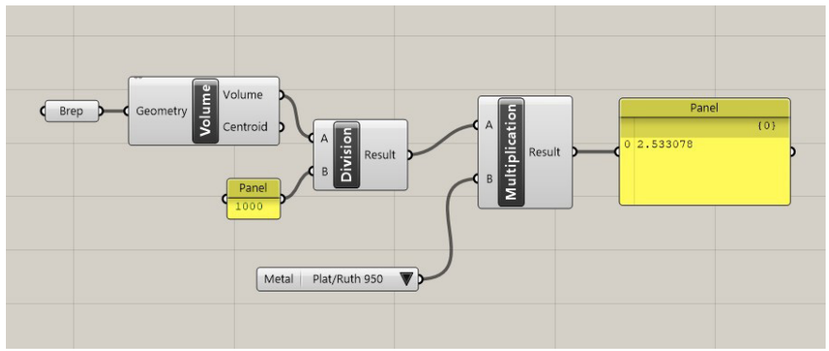

Figure 3: Value List input added to Multiplication node for more customization

The list of metals currently within my Value List component can be seen in Figure 3a.

BRADFORD

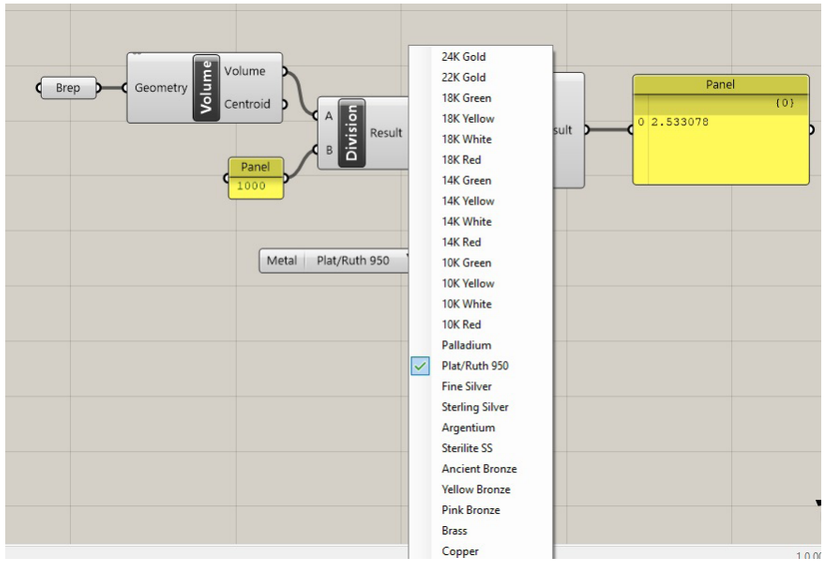

Figure 3a: Full list of specific gravities to choose from inside Value List

Already we have made a significant improvement over the standard industry plugins in that we can input specific gravity values that are specific to certain alloys to make our estimates more accurate. When looking to purchase precious metals like gold, the amounts you buy are typically sold in pennyweights (dwt). To make our metal estimates easier to visualize, let's add in a conversion that will take the gram value and convert it to pennyweights. We will still keep the Panel containing the gram output of the metal in case we need it. To convert grams to pennyweights, we must take the value in grams and divide by 1.555. In Figure 4, I have connected the output of the Multiplication node (our metal weight in grams) and connected it to the input of a new Division node. I have also added a Panel with the value "1.555", and the result is displayed on a second Panel. The two Panels have been labeled with "Metal Weight (g)" and "Metal Weight (dwt)" for easier identification. This is shown in Figure 4.

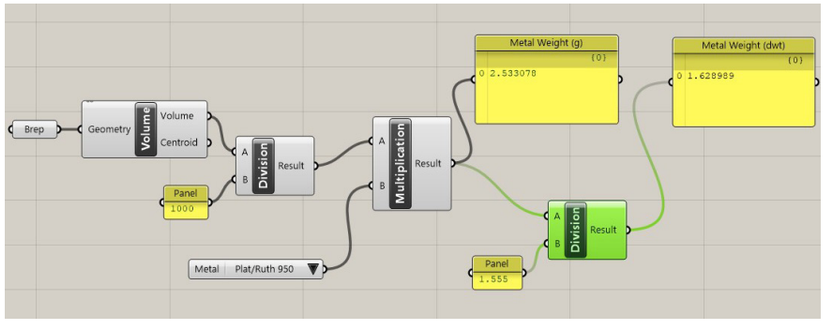

Figure 4: Conversion from grams to pennyweights using Division node (panels have been labeled)

Let's keep going. At this point we have a fairly accurate estimate of the CAD model based on the alloys we specifically use. However, when casting we also know that extra metal must be added to account for the button. When casting a single item or tree, it is generally accepted that adding around 10% extra metal will allow the casting to fill properly. To calculate this, we must take the value in grams or pennyweights, multiply by 10%, and add that amount to the overall estimate. In Figure 4a, we run a wire from the output of the Multiplication node producing our metal weight in grams to another Multiplication node that will multiply that value by 0.1 (10%). This result is connected to an Addition node. The other input of the Addition node is connected to the same output that provided our metal weight in grams. The resulting metal weight in grams, including the 10% for the button, has been displayed on a Panel that is labeled. This process will be repeated for the weight in pennyweights.

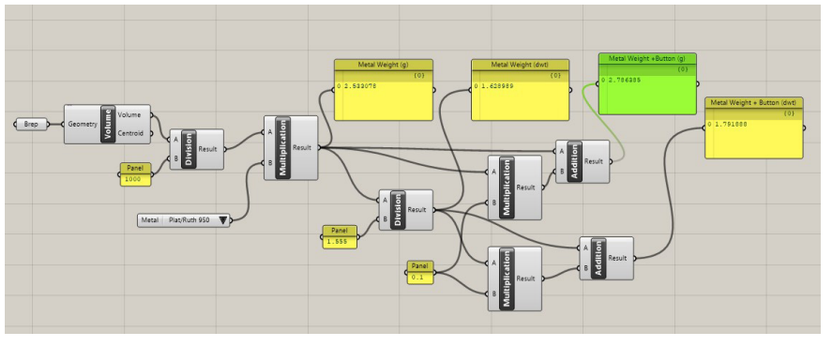

Figure 4a: Metal Weights plus Button

BRADFORD

At this point, we have a metal weights calculator that will provide estimates based on our own specific alloys, as well as providing estimates in grams and pennyweights for both the CAD model alone, and the extra we will need for casting. Because we are automating the math involved with calculating these values, already we have surpassed what is widely available with the plugins available for jewelry making. However, we have already made a mistake.

When calculating metal weights, we start by determining the volume based on a known quantity: the specific gravity of our starting material. In the case of lost wax casting, this can be a carving wax, injection wax, or with 3D printers being ubiquitous in our modern age, a castable resin. Specific gravity is essentially its relative density compared to another known quantity against which all materials are compared: the density of water. The density of water equals 1, and all other densities of materials are represented as a comparison. So, the mistake we made was assuming that our starting material is equal to 1, or the specific gravity of water.

In lost wax casting, we know about this discrepancy and largely ignore it since the specific gravity of our injection wax or castable resin is simply "close enough" and we are adding extra to account for that discrepancy, as well as having an adequate amount of metal for the button which the casting can draw from during cooling and prevent defects. What if we could calculate the specific gravity of our starting material and input that into our calculator for even more accurate estimates? To calculate the specific gravity of a material, we need to perform the following calculation:

Weight in Air/(Weight in Air – Weight in Water) = Specific Gravity

By weighing the starting material in air, as well as weighing in water with a scale sensitive enough to provide measurements to at least two decimal points, and performing the above calculation, we can determine the specific gravity of the starting material more precisely. For the following example, we have determined that the specific gravity of our castable resin is 1.17. Now let's add this into our calculator. To add this value into our calculator, the 1.17 specific gravity of our castable resin must be multiplied by the output from our Volume command.

In Figure 5, a multiplication node was added to multiply the Volume output by 1.17, and a panel was added after converting the volume to cubic cm to visualize the resulting weight in wax or 3D printed resin.

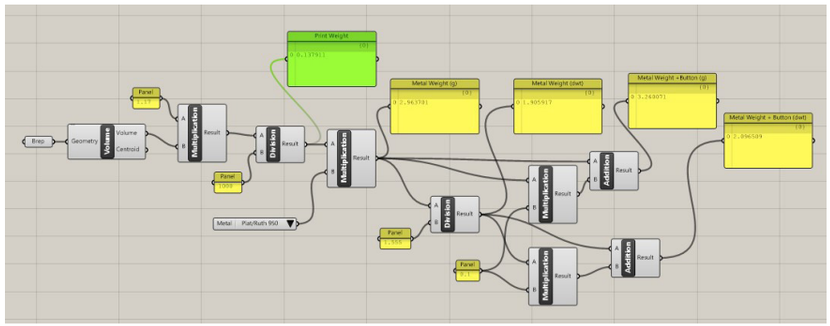

Figure 5: Print Weight panel added after converting multiplying volume output by 1.17

My personal motto is, "anything worth doing is worth overdoing", so let's take this one step further and clean the algorithm up. In Grasshopper, you have the option to create what is called a cluster. A cluster is a full algorithm contained within a node. We can create the inputs and outputs of the node, and give them the names we want, so rather than having all of the individual elements of the algorithm visible, they can be contained in one, easy-to-use node. First step is to define our inputs and outputs. We know that the CAD model from the Rhino document, and the specific gravities of our metals will be our inputs.

To define our inputs and outputs, we will bring in the node "Cluster Input" for our inputs, and "Cluster Output" for our outputs. Within this node, we can give the inputs/outputs a title, nickname, and description as seen in Figure 5a.

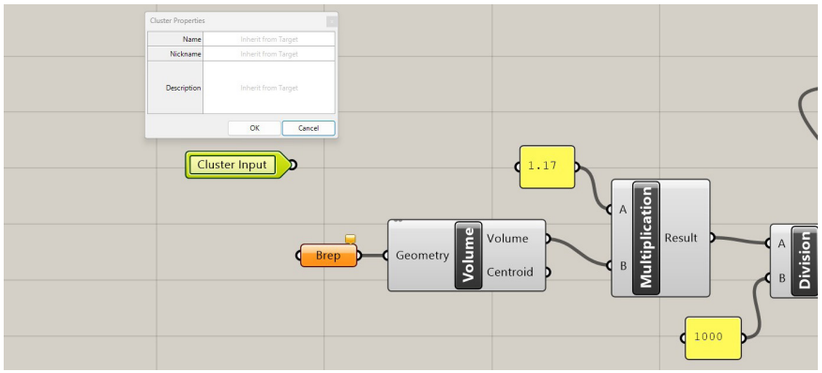

Figure 5a: Customization within Cluster Input node

BRADFORD

The above Cluster Input node will be used to upload our CAD model into the algorithm. The title will be called “CAD Model” with a description of how to upload the model into the input. The Brep node we originally used will be deleted, and the Cluster Input installed in its place. The process will be repeated with the Value List input we used to select the metal we are casting with.

We also repeat this process with our Cluster Output nodes. We will give them a name, nickname, and description of our choosing, and replace the Panels we originally used to display our outputs with the Cluster Output nodes. After completing the creation and labeling of our Cluster Inputs and Outputs, this is how our algorithm looks currently. See Figure 5b.

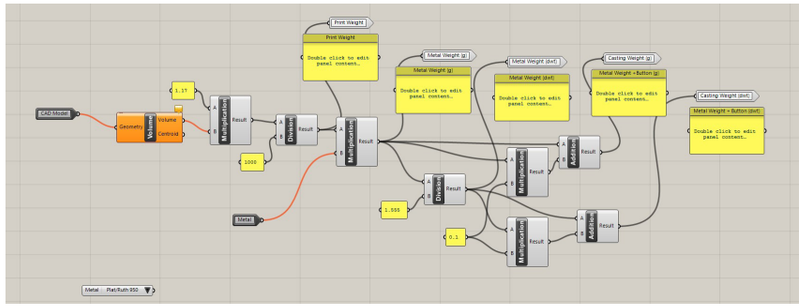

Figure 5b: Adding Cluster Inputs and Outputs to Algorithm

Lastly is the creation of the cluster. This is done by selecting everything in the algorithm and selecting Create Cluster. I have kept the Value List Input for easier selection of metals and connected Panels to each output of my cluster for better visualization. As you can see, the resulting cluster is much cleaner to look at and easier to understand, and the Panels can be organized as you see fit. Figure 5c shows the creation of the Cluster with the Value List input connected, and Figure 5d shows the finished cluster.

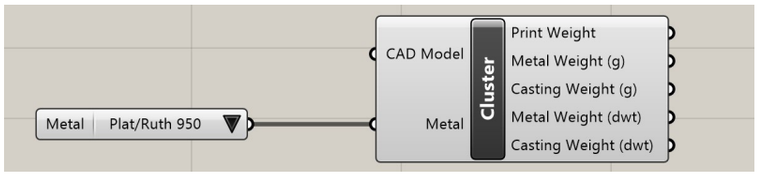

Figure 5c: Creation of the Cluster

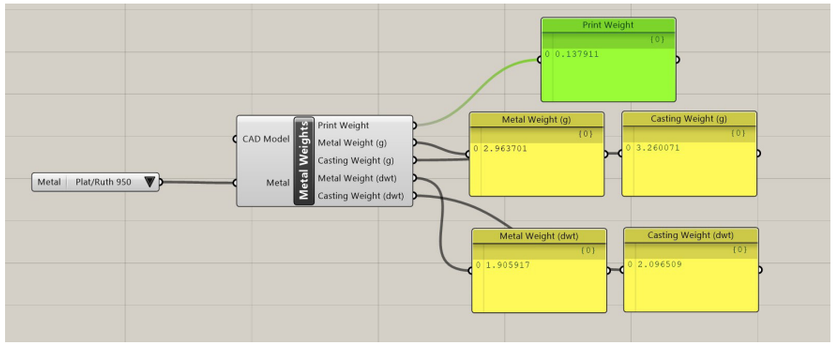

Figure 5d: Finished Metal Weights Calculator Cluster

Whether to create clusters for your algorithms are up to the end user and how they prefer to operate it. The algorithm within the cluster can be opened at any time to make any required adjustments, and further inputs and outputs can be added as well. Descriptions can be added to the clusters themselves, and passwords can be created to protect proprietary algorithms in the case of distribution.

SUMMARY

The benefit of this is that by dialing in a more accurate metal estimate of our CAD model based on the specific gravities of our starting material and our preferred alloys, we can more accurately provide cost estimates over the long term for both metal and starting material. This can result in less metal tied up in buttons and scrap and more left in usable grain. There is the possibility to experiment with the extra 10% of metal as well. While the extra metal is designed to provide additional functions beyond accounting for the discrepancy in our wax to metal calculations, I believe there is room for adjustment with that number based on the casting method you are performing, the metal alloy you are using, and the design you are casting. This level of testing would be another paper for another day.

Without getting into a debate about whether this level of granularity is necessary or not, it serves as a useful demonstration of the customization within Grasshopper to construct algorithms with the parameters we specify, however granular or general, and was created at no cost other than an investment of time. Could we go even further and calculate the actual metal cost with some multiplication nodes and a number slider representing current

BRADFORD

metal prices? Yes, we could. Could we apply this same algorithm and substitute the specific gravities of metal for the specific gravity of diamonds and set the outputs to display in carats through conversion factors? Yes, we could. This is the power of customization within Grasshopper.

PLUGINS FOR THE PLUGIN

As stated previously, Grasshopper started as a plugin for Rhino back in 2008. Since then, a vast number of plugins have been created specifically for Grasshopper to grant additional functionality to the base install. Some are more general, offering commands that can be applied to an array of algorithms, others being highly specialized for a certain industry. These can be utilized within the existing Grasshopper environment to expand functionality and provide more options for customization. These can range from an enhanced UI for your Grasshopper algorithms for better visualization, pattern generation that is fully parametric, engineering resources providing collision detection, or cell generation, amongst many others. For jewelry, plugins like Grasshopper Gold and even CrossGems are built with user interfaces directly in the Rhino environment, using Grasshopper as the backbone. Let's examine some special use cases created using the nodes available in Grasshopper, as well as various plugins.

RING RAIL GENERATOR

Every ring design with Rhino must start with a circle in the required ring size for sweeping a ring shank. In Figure 6, a ring rail generator was produced that creates the circle using a Value List as an input for ring size. The Value List contains a display of the ring size which outputs the exact diameter of the ring size into the circle command.

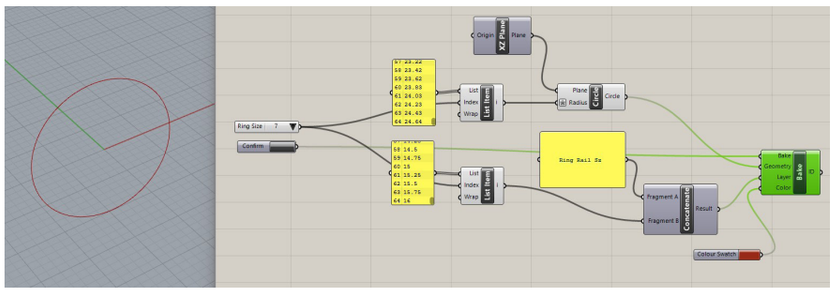

Figure 6: Ring Rail Generator

In the interest of customization, this generator has been constructed in a way that will "bake" the circle into our Rhino environment and carry additional information with it. The "Confirm" button will bake the circle into Rhino, create a new layer with a title identifying it as a ring rail, includes the ring size for better visualization, and assigns the layer color of our choosing for organization. See Figure 6a.

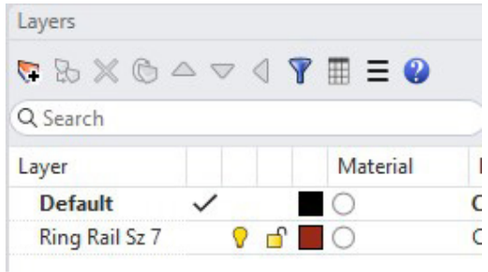

Figure 6a: Layer identifying Ring Rail with size and assigned color

GEM GENERATOR

This example shows a Gem Generator using a plugin known as Peacock. Peacock is a plugin for Grasshopper specifically for jewelry. This algorithm has been customized to produce standard gemstone sizes, as seen under the section "Uniform Stones", as well as gems with custom dimensions under the section "Custom Dimensions." This Gem Generator was also constructed to produce a curve around the gemstone that aids in creating settings. See Figure 7. The entirety of the algorithm is shown in Figure 7a.

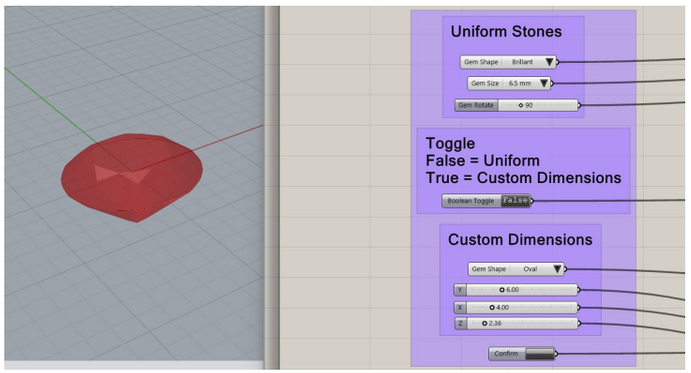

Figure 7: Gem Generator in Uniform and Non-Uniform sizes with accompanying curve

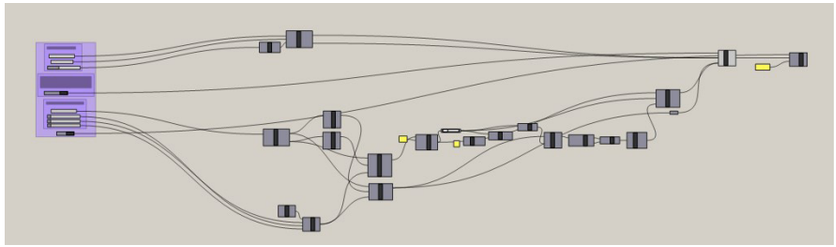

Figure 7a: Entirety of Gem Generator algorithm

RING RAIL & CENTER STONE

This example demonstrates combining two different algorithms, the Ring Rail Generator and Gem Generator, together with extra functionality to create a ring rail and center stone in one algorithm. A ring rail is generated with a layer showing the ring size in the Layer name for easy identification, a stone is generated in either Round or Fancy shape, a gem curve for creating a setting, as well as a separate layer showing the Gem Size in the Layer name. Extra functionality has been added to control the distance from the top of the ring rail to the bottom of the culet. See Figure 7b. The entirety of the algorithm can be seen in Figure 7c.

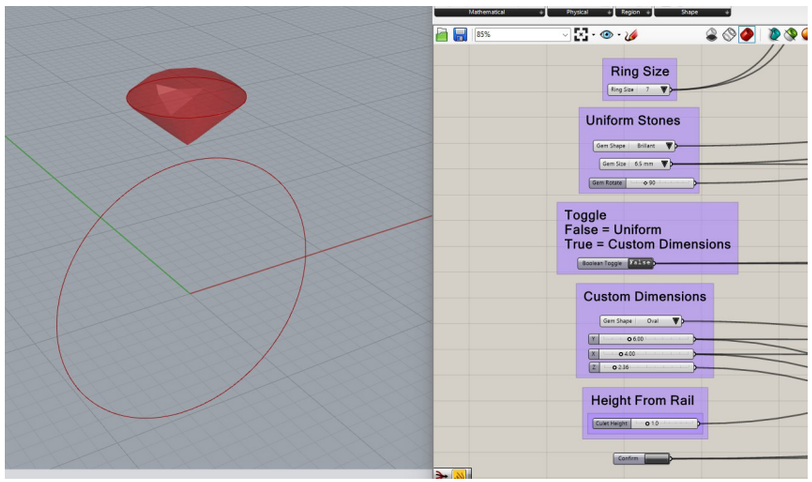

Figure 7b: Combined Ring Rail and Gem Generator with extra functionality

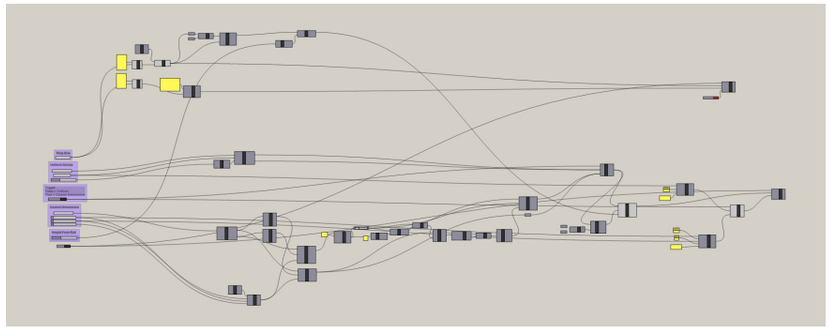

Figure 7c: Entirety of the Ring Rail and Center Stone Generator

MILLGRAIN GENERATOR

This example demonstrates a simple millgrain generator from a curve generated from a 3D model in Rhino. The curve from the model is input into the Curve node, then the distance between the spheres, dimensions of the spheres, and the start and end points can be modified. The result can be automatically baked into the Rhino document with the Confirm button. This algorithm has also been modified to assign the resulting millgrain to its own Layer within Rhino, along with its own color, for easier identification and organization. See Figure 8.

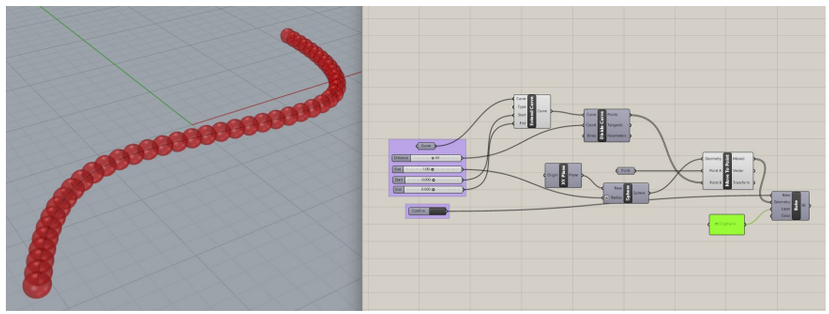

Figure 8: Millgrain Generator with full customization

RING BAND GENERATOR

This example was created to generate ring bands with full customization of the ring size, choice of profile shape, width, thickness, and fillet (rounding of sharp edges). There are three profile shapes to choose from: Rectangle, Half Round, and Half Round with Flat sides. This algorithm was created using a plugin

BRADFORD

to provide an easy-to-use interface within the Rhino environment to adjust these parameters, and the Confirm button will bake the ring into Rhino. The UI plugin is managed within Grasshopper. See Figure 9 for the custom UI and the resulting band represented in Rhino. See Figure 9a for the algorithm within Grasshopper.

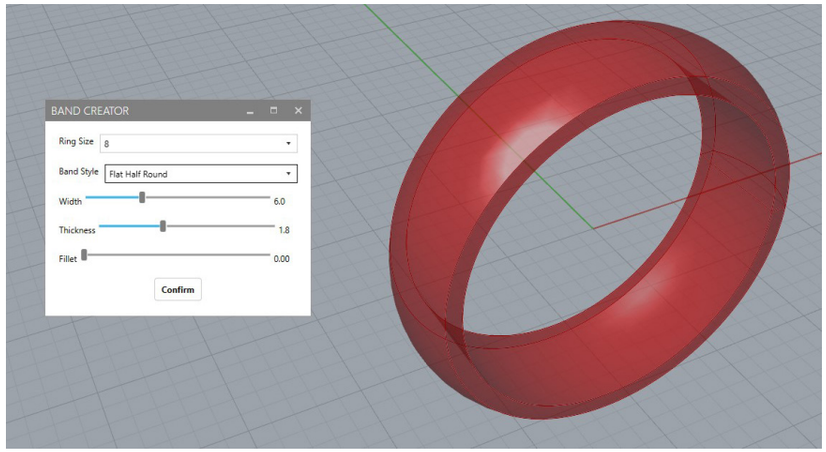

Figure 9: Band Generator with UI plugin

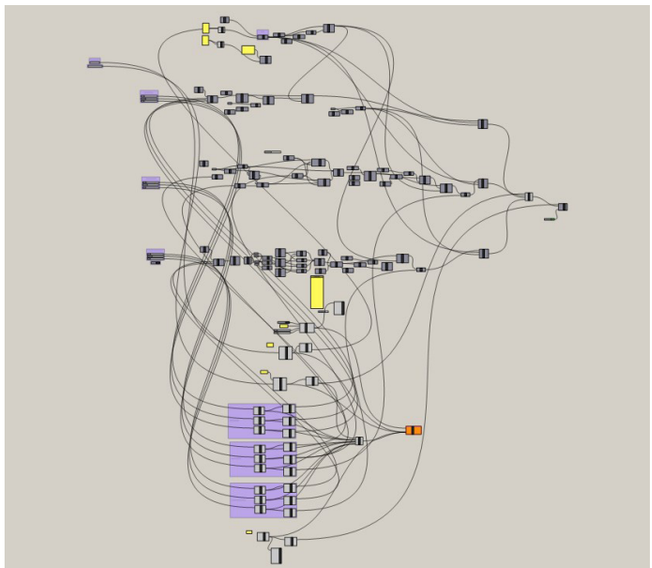

Figure 9a: Band Generator algorithm in Grasshopper

GEM SEAT CUTTERS

This example was created to quickly add cutters for gemstone seats when setting melee stones. Functionality has been added to control offset of the cutter at the girdle in the event of stone size variation, ability to control the height at the girdle, and overall depth to minimize culet chipping during setting. See Figure 10. The entirety of the algorithm can be seen in Figure 10a.

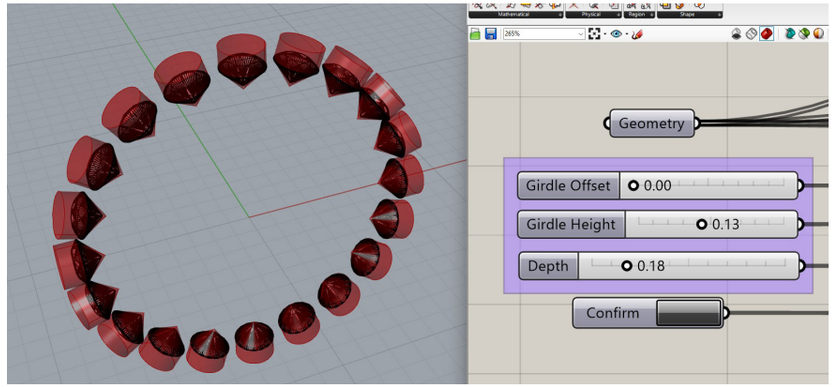

Figure 10: Gem Cutters for Melee Setting

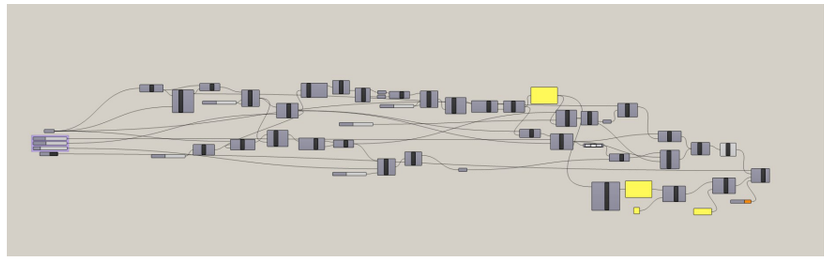

Figure 10a: Entire Gem Cutters algorithm

PRE-NOTCH MELEE CUTTERS

This example was developed for the creation of pre-notched settings, whether setting a center stone or melee. Functionality has been added to adjust the crown angle of the cutter, girdle offset to account for any size variations in the stones, and adjustment of girdle thickness. See Figure 10b. The entirety of the algorithm can be seen in Figure 10c.

BRADFORD

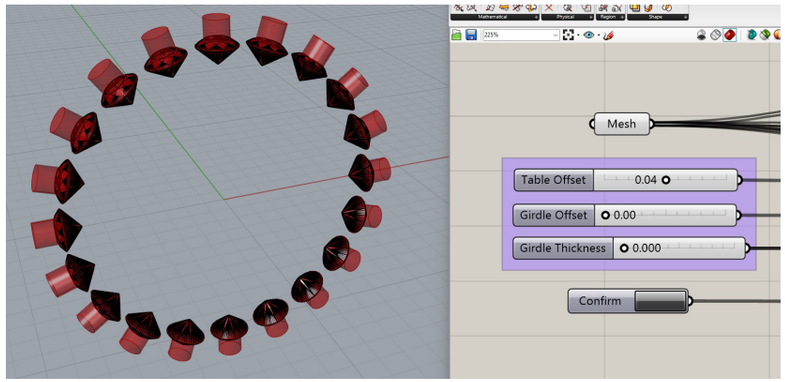

Figure 10b: Pre-Notched Cutters for Center Stone or Melee Setting

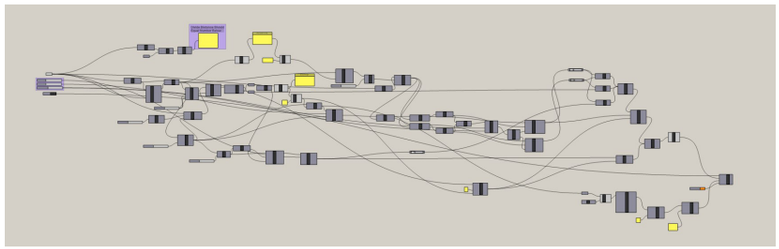

Figure 10c: Pre-Notched Cutters Algorithm

CATHEDRAL RING BUILDER

Extensive customization has been added to this Cathedral Ring Builder. By inputting a Ring Rail created from our Ring Rail Generator, we are able to control the angles of the shoulder at the start and mid-point where it angles upwards, the overall height of the cathedral, as well as the distance of the opening at the top. Further customization has been added to switch between three profile shapes for the outer ring: Rectangle, Half Round, and Flat Half Round. Control of Height, Width, and amount of Fillet has been added to all three profiles. A Bridge section has been added with control over Height, Width, and amount of Fillet, as well as the ability to toggle a bulge in the center of the bridge. The purpose of the bulge is to create more room if the cathedral will be used to set a Peg Head Setting into the bridge, or can be an overall design feature. See Figure 11 showing the Cathedral Builder with a rectangle profile and no bulge on the bridge, and Figure 11a showing the Cathedral builder with a Flat Half Round Profile with Bulge added. Figure 11b shows the entirety of the Cathedral Ring Builder algorithm.

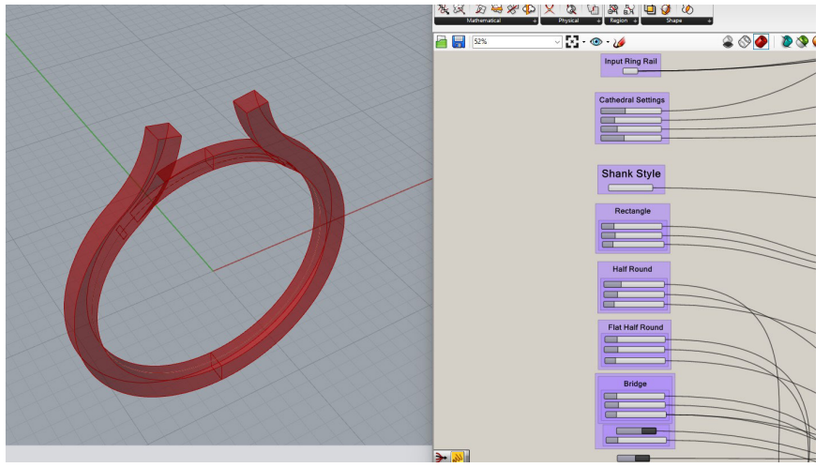

Figure 11: Cathedral Builder with Rectangle Profile and no Bulge

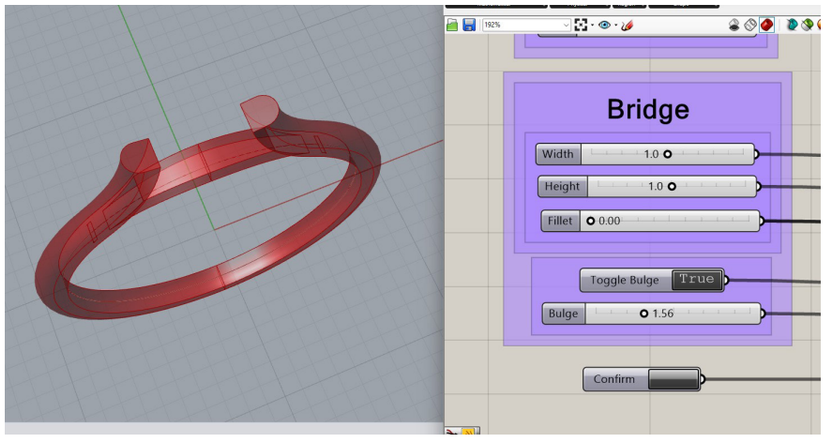

Figure 11a: Cathedral Builder with Flat Half Round Profile and added Bulge

BRADFORD

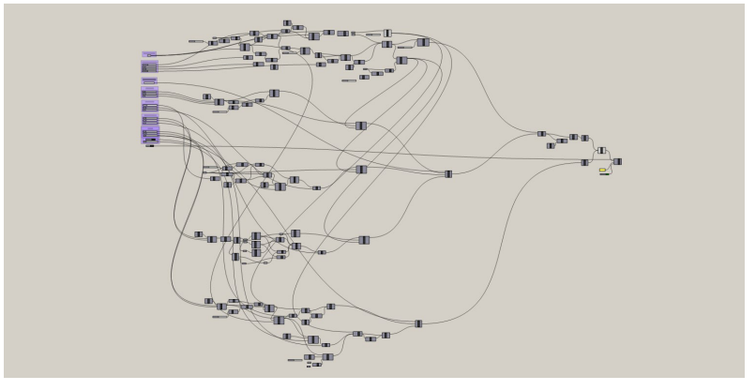

Figure 11b: Cathedral Ring Builder Algorithm

FOUR-PRONG HEAD BUILDER

This example was created to generate a basic four-prong head for a center stone. While somewhat basic in its current form, the algorithm offers the ability to customize upper and lower gallery rails by adjusting height and horizontal offset of both, profile height, width, internal chamfer on the upper gallery rail, and outside angle to create a taper in line with the prongs. Prong customization includes prong thickness, width, height above stone, and the ability to nudge the prongs inward or outward without changing dimensions. See Figure 12. Additional functionality has been created to allow for upload of a ring rail, and a toggle where the lower gallery rail will conform to the ring rail. Toggle is labeled, “Fit to Rail”. Can be used effectively with the Ring Rail and Center Stone Generator. See Figure 12a. Algorithm shown in its entirety in Figure 12b.

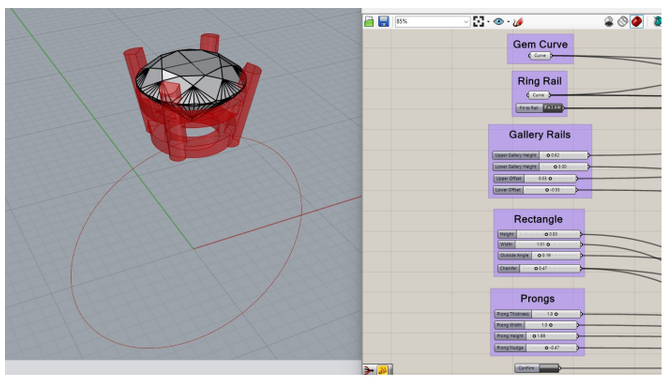

Figure 12: Four-Prong Head Builder

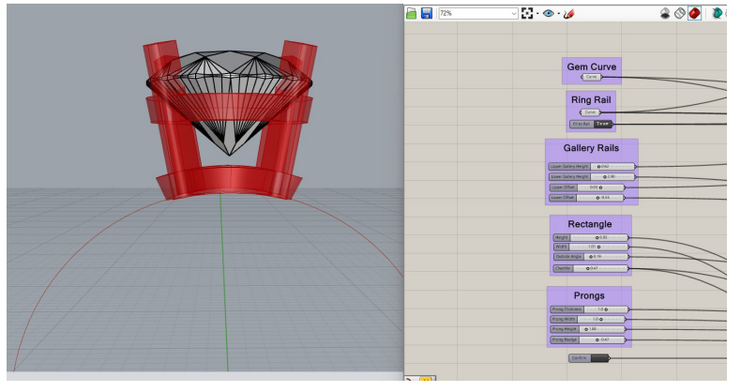

Figure 12a: Four-Prong Head Builder with “Fit to Rail” toggle enabled

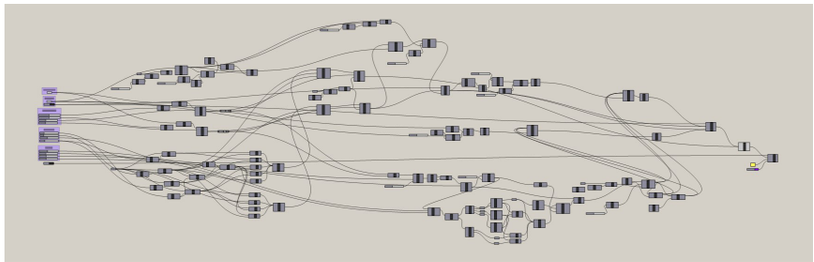

Figure 12b: Four-Prong Head Builder Algorithm in Grasshopper

Another addition was made to the Four-Prong Head Builder allowing for extra features to be generated when making heads for post earrings. Due to the need to have a bar on the bottom of the setting to connect the post, functionality has been added to generate the bar with control over the width, and the height is designed to match the height of the bottom gallery. The bar can be toggled on and off using the drop-down box. See Figure 12c.

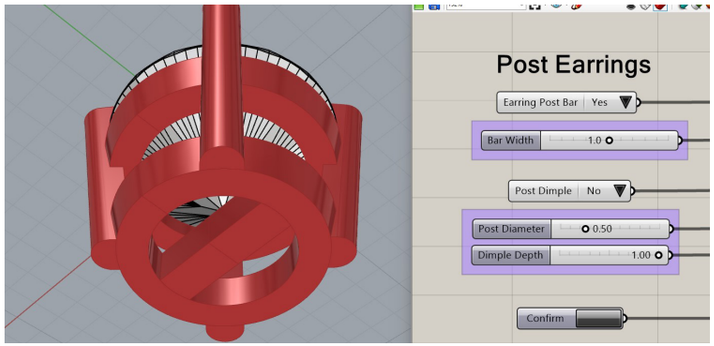

Figure 12c: Four-Prong Head Generator with Bar for Post Earring

In addition, the option to create a locating dimple to the bar itself has been added. The function of the locating dimple is so the operator that will be soldering the posts to the head will have an exact location where the post needs to be soldered on, and the post will lock into that spot. Control over the dimple diameter and depth has been added for extra customization. This feature can be toggled on and off with a drop-down box. This is shown in Figure 12d.

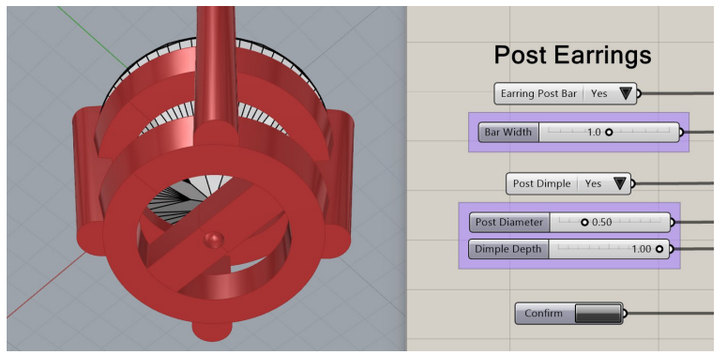

Figure 12d: Locating Dimple added to Bar on Four-prong for Post Soldering

This use case demonstrates how existing algorithms can have extra functionality added after the fact with features that can be toggled on and off depending on the design needed.

SUPPORT BASE GENERATOR FOR 3D PRINTING

This example was created to add a support base to an existing ring design. Full customization has been added for the depth, radius in X, and radius in Y for both the top and bottom of the support base. Depth at the top will increase the overlap with the ring geometry, and bottom depth will increase overall height of the support based on personal preference. The algorithm has been duplicated and adjusted so it can accept polysurfaces or meshes. Support base is automatically Boolean unioned to the polysurface or mesh. See Figure 13. Both algorithms for polysurfaces and meshes shown in Figure 13a.

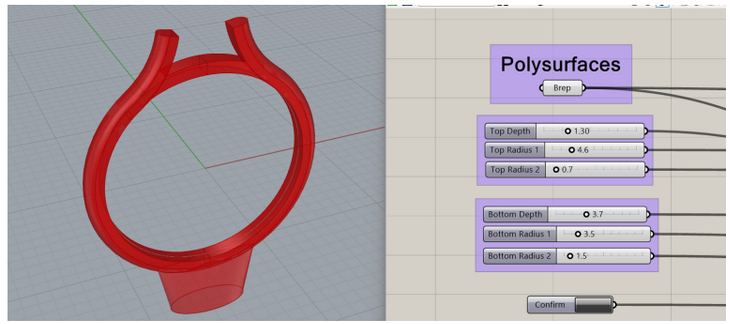

Figure 13: Support Base Generator for Polysurfaces or Meshes

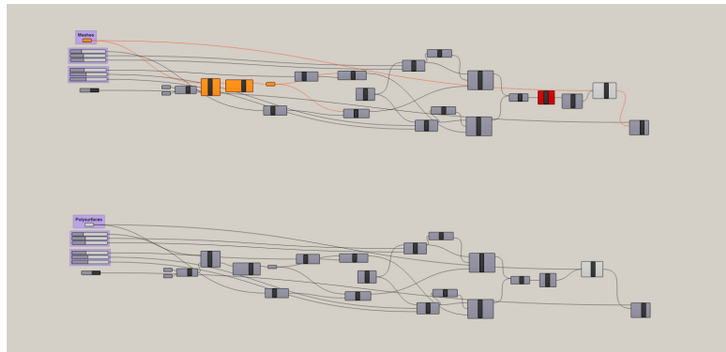

Figure 13a: Support Base Generator in Grasshopper

CUFF BRACELET GENERATOR

This example was created to quickly generate cuff bracelets based on the customer's wrist dimensions. The first part of the algorithm allows you to input the width and height of the customer's wrist, as

well as the amount of opening at the bottom they prefer. The actual opening size is demonstrated as an annotation inside the Rhino environment. Further customization has been added to provide three different profile shapes through the choice of a drop-down box: Rectangle, Half Round, and Flat Half Round. With all three profiles, options have been added to control the width, thickness, and amount of fillet for each profile. Furthermore, the algorithm was constructed in a way where the inner and outer surfaces can be extracted as one surface, allowing for designs to easily be added to the entire inner or outer surfaces of the bracelet. Figure 14 shows the interface and resulting bracelet. Figure 14a shows the algorithm in its entirety.

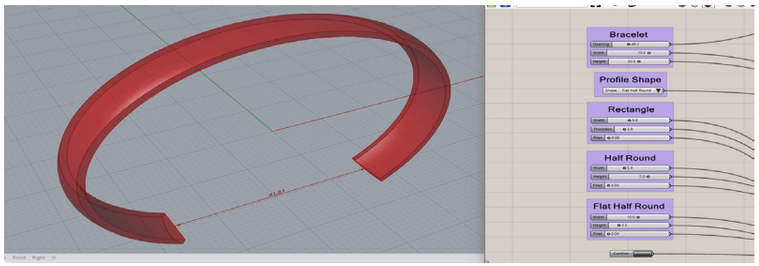

Figure 14: Cuff Bracelet Generator

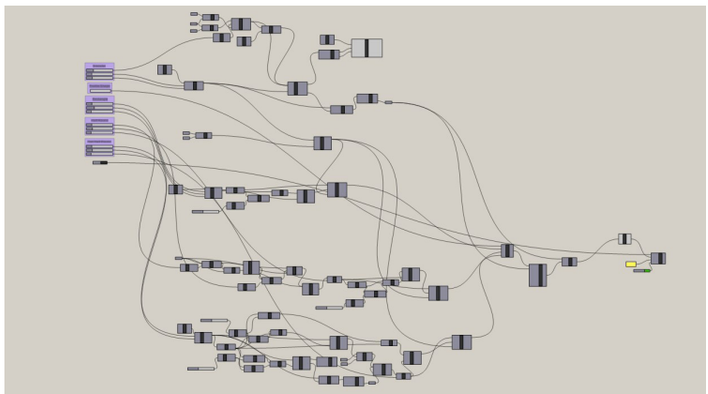

Figure 14a: Cuff Bracelet Generator Algorithm

The benefit of these types of algorithms is that they can automate the more repetitive operations with most jewelry designs, such as building settings, millgrain, or preparing for 3D printing. In addition to creating elements or serving as calculators alone, as we have already seen, these algorithms can also be strung together to create more complex algorithms that can, for example, input a model into an algorithm that will generate the metal weight estimates, automatically scale for casting shrinkage, and apply a support base for 3D printing in one go, or however the user wants to organize and combine based on their needs. The use cases above are meant to serve as a demonstration of what is possible within Grasshopper for creating design elements, calculating needed values, or adding our own personal customization, but this is only the tip of the iceberg.

CONCLUSION

The purpose of this paper was to introduce the reader to one of the largest underutilized resources within Rhino from a jewelry perspective, while giving insight into how Grasshopper works, providing a lesson on something the reader can make for themselves, and further use cases for how it can be utilized with continued practice. While Rhino has a significant upfront cost on its own, the most competent jewelry plugins on the market can be 5-8X the cost of Rhino. With an investment of time and persistence, some of the resources found in those expensive plugins can be made within Grasshopper and made to the specifications of the end user to be more precise or tailored to the desired end result. All of the Grasshopper plugins used in the demonstrated use cases are all free of charge and available for instant download. This presents an interesting opportunity for users working within Rhino currently or are getting into Rhino and cannot afford the professional level plugins. Grasshopper requires a different type of thinking to be successful, but then again, so does creating a jewelry design in Rhino.

REFERENCES

- Rhino 3D CAD Software: https://www.rhino3d.com/
- Grasshopper 3D: https://www.grasshopper3d.com/
- Matrix Gold CAD Software: https://gemvision.com/
- CrossGems Jewelry Plugin for Rhino: https://www.crossbytes3d.com/
- Peacock Plugin for Grasshopper 3D: https://www.food4rhino.com/en/app/peacock
- Rhino Gold: No longer available

TRAINING

Introduction to Generative Jewelry Design: https://www.akiyomatsuoka.com/grasshopper-training

BRADFORD