A Jewelry tool for Maya using the Houdini Engine

A Master's Project Thesis discussing the ideas and creation of a jewelry tool for maya

Matthew J Stanton

MSc Computer Animation and Visual effects student at Bournemouth University

1 ABSTRACT

This thesis intends to discuss the creation of a jewelry tool for Maya and how a tool can be taken from Houdini, brought into Maya, and then turned into an even more useful tool using the Maya API. Procedural generation of geometry is a powerful technique for 3D use as it allows the creation of a large variety of computer graphics (Miller, St'ava, Benes, & Mech, 2011) and can be used to create 3D objects that would usually take lots of time to replicate through regular modelling (Muller, Wonka, Haegler, Ulmer, & Gool, 2006). Although the creation of one piece of jewelry may not take a skilled 3D modeler much time to create but on a production where there could be hundreds of pieces of geometry, all needing to look somewhat different a tool that can create this variation with ease can come in very useful. The tool utilizes Houdini's excellent procedural systems (Okun & Zwerman, 2015) but is more accessible to 3D artists through its Maya Plugin. This tool allows control of the creation of one type of jewelry, a Jhumka, giving the user control over the shape, size and patterns within the piece of jewelry through the easy-to-use UI set up in Maya.

2 INTRODUCTION

In film computer generated imagery has become increasingly common appearing in many films but most commonly sci-fi films. (Li, Guan, & Lu, 2021). Just looking at the top grossing films of all time it can be seen that it is filled with CGI blockbusters such as Avatar, Avengers Endgame and Jurassic World but also computer-generated animations such as Frozen *II* and The Minions (IMDb, n.d.). These films needed 3D modelling and they needed a lot of it, time and cost are always extensively thought about when making these films, the ability to improve on both of these contingencies is largely sort after and parametric modeling can help save time, money and energy. (Still, 2021)

2.1 Procedural modelling

The procedural method represents creating a 3D model through the use of sections of code or algorithms that take an object in 3D space and do some adjustments to the object. Procedural modeling is extremely useful as it contains a feature called 'abstraction' this represents storing the procedures that create the final object not the details of the scene (Ebert, Musgrave, Peachey, Ken, & Steven, 2003), this allows for variation within the object based on either a random variable within the code segments that create the final object or user defined parameters which allows the user to change variables within the code segments which can change things like height, depth or width, this is called parametric control and is the main procedural aspect of the Jewelry tool (Ebert, Musgrave, Peachey, Ken, & Steven, 2003). Procedural modelling can save time and thus save money as it both allows for the creation of many types of one thing with variation between each thing with ease and the reuse of previous code segments can allow for future procedural geometry to be created even quicker. (Morkel & Vangay, 2006) In the terms of the jhumka generator a lot of the ideas, described later, can be reused on other pieces of jewelry such as the ability to lay out geometry in a circle as many pieces of jewelry are in a circle shape, this means that if new pieces of jewelry were to be added to the tool it would take less time as these procedures can be reused (Morkel & Vangay, 2006)

2.1.1 Houdini

A great way to see the way procedural modeling works is through Houdini:

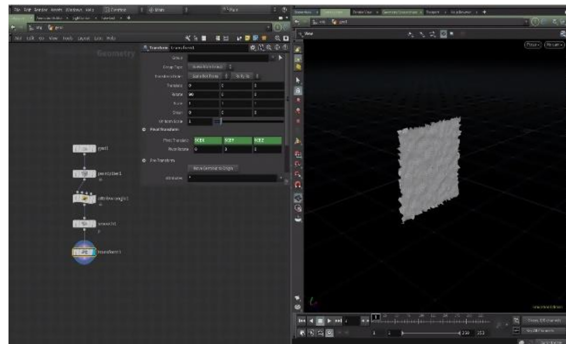

*Figure **Error! No text of specified style in document.**1 Image showing a piece of geometry (right) created using a set of nodes (left) within Houdini*

Figure 2.2 An image showing how the Houdini node graph can be split into procedures

Houdini represents the geometry through points and edits the points and attributes the points hold using nodes or procedures, making it a procedural way of creating geometry. In figure 2.2 there are 5 procedures in the scene, the first procedure being the grid which creates points in a grid pattern. The second procedure, the point jitter geometry node, is a procedure that randomizes point locations is all axis (SideFX, n.d.). The third procedure is the attribute wrangle, this allows custom VEX code, written by the user to adjust attributes on each point (SideFX, n.d.), in figure 2.2's case it multiplies the y position attribute by 2. Next is the smooth geometry node which is a procedure that smooths out the points reducing roughness whilst keeping the number of points the same (SideFX, n.d.). Finally, the transform geometry node transforms the points in the scene by multiplying them by a transform matrix (SideFX, n.d.), in this case it rotates in X by 90 degrees.

As can be seen this represents the procedural method, as it is just a set of procedures that adjusts the geometry in a specific way, the user can easily go back and change a variable, such as the point jitter amount or random seed resulting in a new variation on the outcome of the geometry.

2.2 Maya

Whilst Houdini is popular for procedural modelling (as well as animation, character rigging, lighting and dynamic simulations) Maya is popular with regular 3D modelers (as well as animation and rendering) (Okun & Zwerman, 2015) and is commonly used in an artistic sense (Steffen, Koning, Wenger, Morisset, & Magnor, 2011). The difference between Houdini and Maya is maya relies more on Direct modeling, meaning that it relies more on the creation and manipulation of the geometry through the viewport instead of through nodes (although Maya does contain a node graph most people don't use it). (Foundry, n.d.). The use of Direct modeling makes Maya appealing to artists as they prefer to touch and edit objects and work visually rather than apply technical procedures, this can be seen through the traditional methods that

an artist would work either through a painting or sculpting clay where the artist would physically paint on a canvas or mold clay. An artist working for the Foundry described the work they do as being direct modelling 90% of the time as it is a quick way to get the results they want. (Foundry, n.d.)
Many 3D artists do not use Houdini and a some don't even know how to use Houdini, this can be seen by the fact that on the MA 3D Computer animation course at Bournemouth University which is self-described as a 'perfect launch-pad for a career in a wide range of disciplines, including concept design, character and creature animation, modelling, lighting, rigging, texturing, compositing and simulation.' (Bournemouth University, n.d.) does not include Houdini as a main piece of software for the course instead focusing on Maya.
Talking to a student from the MA 3D Computer Animation course, Samruddhi Shinde, an aspiring 3D artist she states that she has never used Houdini nor been taught it therefore a procedural tool is useless to her however she does know Maya. The ability to bridge the gap between Houdini and Maya and do so in a way more usable for artists is something that can boost the workflow of geometric creation as it allows for Houdini's amazing procedural system to be used by artists making use of Maya's great direct modelling system.

2.3 The downsides of procedural

Although procedural geometry is efficient in improvement of time, money, and energy (Still, 2021) it does come with its own negatives. In the case of Muller et al when working on the procedural modeling of buildings they found 'the generation of smaller complex geometrical details is sometimes inefficient' another thing they stated in their report was ' it sometimes generates configurations of shapes that are not plausible' (Muller, Wonka, Haegler, Ulmer, & Gool, 2006) this brings up two important points of procedural modelling, efficiency and reliability. When using direct modelling efficiency does not need to be thought of in the same was as when procedurally modelling as in software such as Maya the user is often only ever adding stuff to the 3D model, never really going back to edit previous procedures done on the 3D object. In Maya it is so unusual for artists to go back and change something they've already done to the object that it's common to delete history in Maya just to speed up the software. They're also only working towards one 3D model, they're not planning on creating thousands of the same model and if they are it's not through repeating the procedures a thousand times but simply duplicating the explicitly specified model as the duplication tool in maya automatically deletes history. The extra thought process must go into procedural generation as if efficiency is not considered it can cause a slow and unusable tool. The next issue is a procedural tool not always showing 'plausible' geometry, as with a procedural tool every detail is not created by a 3D artist but instead a set of procedures. Procedural methods can sometimes create incorrect geometry. With procedural geometry it needs to be created so that every option the user inputs can be accounted for, and strict testing can search for any problematic geometry that the tool can output however there's always a chance that something will go wrong when relying on a computer to create the geometry.

3 INDIAN JEWLERY

This tool was made through the request of a 3D modeler who is sculpting an Indian dancer; therefore research was necessary for the completion of the tool.
The jewelry industry in India is massive, heavy jewelry on Indian women is a common sight (Bhushan, 1964). Dr K. Chitra Chellam states that 'every woman in India loves to war at least a small pieces of gold jewelry' including that of 'dangling earrings' and ' even men wear simple gold ornaments' (Chellam, 2018). Jewelry in India is no recent obsession either with

it dating back over 5000 years (Chellam, 2018) and staying popular until the modern day where the Indian Jewelry market is now worth an estimated $13 billion which is the second highest in the world next to America. (Indian Law Office)

As jewelry is common on Indian women for any CGI scene that would be required to be set in India would need many pieces of jewelry for the shot, especially if it was an animated film set in India where many characters would need to be modeled for the film. A tool to procedurally be able to rapidly produce variants of jewelry would reduce cost time and power usage.

3.1.1 The Jhumka

Jhumkas are a traditional Indian piece of jewelry in the shape of a bell and was described as 'the most common type spiral earrings' by Priya Thakur in her paper on jewelry in South Asia (Thakur, 2019).

The Jhumka can be seen being worn by the sculptures at Bharhut (Thakur, 2019) which date between 184BC and 82BC (Kumar, 2014) and can even be seen on ancient temple statues in India dating back to 300 BCE (Raniwala, 2021)showing the historical value for Jhumkas for Indians. They have stayed relevant to this day as can be seen by the fact that massive jewelry sellers in India such as Tanishq and Tribe Amrapali, both massive Indian jewelry sellers, have large sections of Jhumkas on sale.

For these reasons the Jewelry tool takes this importance of jhumkas in Indian Culture and makes it the first piece of jewelry to be added to the tool. Another reason this was the jewelry of choice was that someone interested in using this tool for their project is sculpting a South Indian dancer with this being the typical person who would wear Jhumkas (Raniwala, 2021) therefore this was a perfect piece of jewelry to start with.

Figure 3.1 An image showing a Jhumka split into sections. Jhumka image accessed from: https://www.tanishq.co.in/product/drop-earring-511920dezaba00

This figure shows how a common jhumka can be split into sections, this is a particularly useful way of looking at a jhumka as it allows for a mindset that helps with the setting up of the procedures to generate it as it allows for an understanding of how the tool will work and what parameter will need to be editable for it.

Figure 3.2 Jhumkas following the same pattern of rings and sections for the jhumkas accessed from: top left = https://www.pskstore.com/?product_id=261115030_62 , top right = https://www.tarinika.com/products/matar-antique-jhumka-earrings , bottom left = https://www.utsavfashion.com/product/stone-studded-oxidised-jhumka-style-earring-jpm6261 , bottom right = https://www.purplle.com/product/crunchy-fashion-traditional-gold-plated-white-pearls-jhumka-earrings

Creating a tool that is able to produce every variety of jhumka in a singular tool would not only be an incredibly difficult task, but it could also add far too many options taking away from that simple ability to generate many jhumkas. Taking the simple idea of rings and sections of detail is something the jewelry tool implements into the jhumka generator allowing for fairly accurate to the real object jhumka generation.

4 CURRENT METHODS

4.1 Houdini to Maya

Currently there are 2 main methods of taking object from Houdini to Maya and which method is chosen depends on what abilities the geometry needs to have. The two methods are exporting as a geometric file or using the Houdini engine plugin in Maya.

4.1.1 Geometry file

Sometimes all that is needed is the geometry in the Houdini scene and nothing more this is where just saving the geometry to a file such as a alembic file for the Houdini to Maya Alembic workflow (Chaos, n.d.) or simply by using one of Houdini's ways to export any file type such as right clicking and saving the geometry as shown in the Entagma tutorial (Moritz, 2016) or just using the file node built into Houdini and setting the file mode to write allowing the user to save the geometry as a choice of different 3D geometry files such as FBX or OBJ which can be imported into Maya. The issue with exporting geometric files is that it removes the procedural functionality as it saves the file explicitly instead of implicitly meaning it forgets how the object was made and just saves the end model shape. As for the tool the reason for Houdini's usage was to access the procedural benefits and with the geometric file losing that capability is simply something that would not work for this tool.

4.1.2 Houdini Engine Plugin

SideFX brought out the Houdini plugin for Maya that allowed for the Houdini procedural method within Maya (SideFX). This method of taking an object from Houdini to Maya works using Houdini Digital assets. Digital assets in Houdini allow for the user to take a network of nodes and condense it down into one node which can have a set of parameters defined by the user allowing for a parametric workflow within the digital asset. Digital assets therefore are useful for the jewelry generator as it allows for the tool to be neatly wrapped up into one node with the parametric controls then through the use of the Houdini engine plugin it can be exported to Maya, (SideFX)

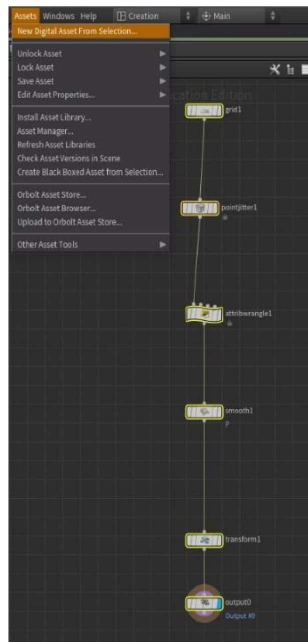

Figure 4.1 The user can select the nodes and use Asset > New Digital Asset From Selection to turn the nodes into a digital asset

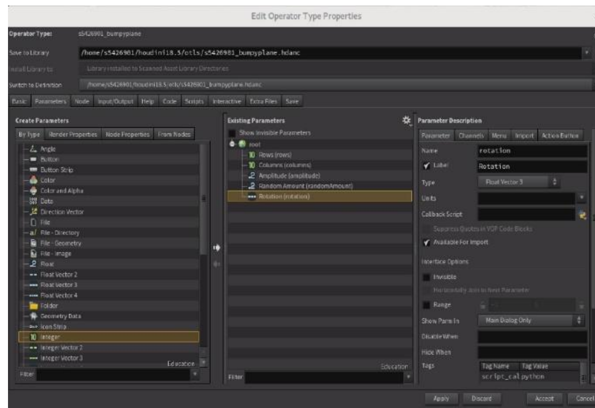

Figure 4.2 Selecting Asset > New Digital Asset opens the window allowing for the setting up of the digital asset which contains the ability to add custom parameters to use within the digital asset

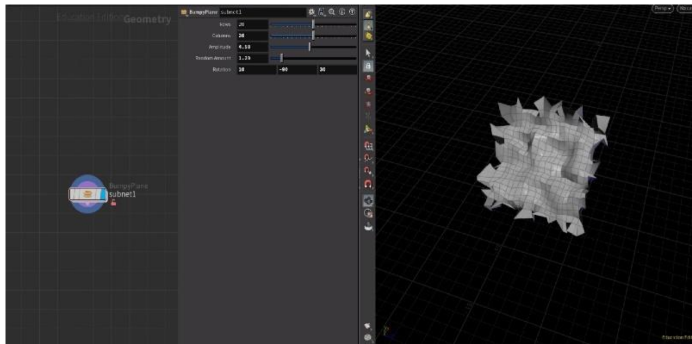

Figure 4.3 This image shows the Houdini Digital Asset set up with parameters set within Houdini

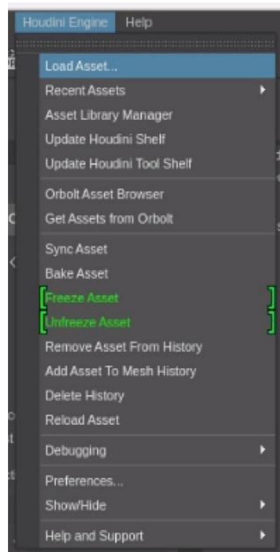

Figure 4.4 In Maya, with the Houdini engine loaded the user can load their digital asset using Houdini Engine > Load Asset

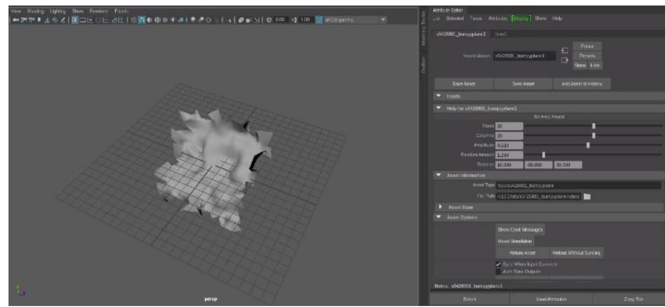

Figure 4.5 An image showing the Houdini Digital Asset loaded in Maya

The images figure 4.1 through 4.5 show the process for taking a Houdini digital asset and loading it into Maya.

4.1.3 Houdini Engine Plugin problem

The Houdini Engine Plugin works well for keeping the Houdini procedural approach and allowing for parametric control within Maya however it is not a perfect solution.
The main issues that face the plugin is UI based although in my personal experience the digital asset loaded into Maya is slower as well causing efficiency issues.

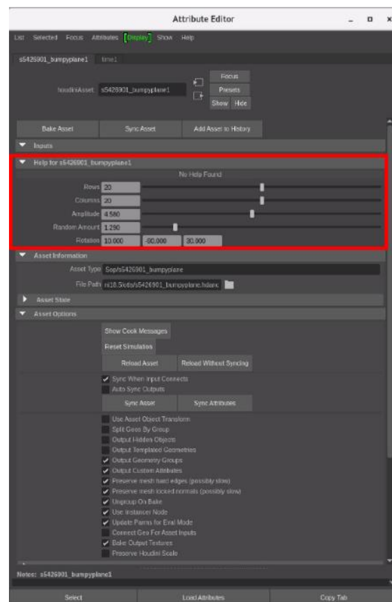

Figure 4.6 The controls for the Houdini Digital Asset with the parameter controls circled in red

As can be seen in figure 4.6 The default Houdini Digital Asset tool has a UI that overloads the user with controls that are not needed, in figure 4.6 the only controls that an artist who this tool has been made for would need to access is in the red box.

Another problem found with the Houdini Digital asset brought into Maya is the layout of the UI as it differs from the UI in Houdini.

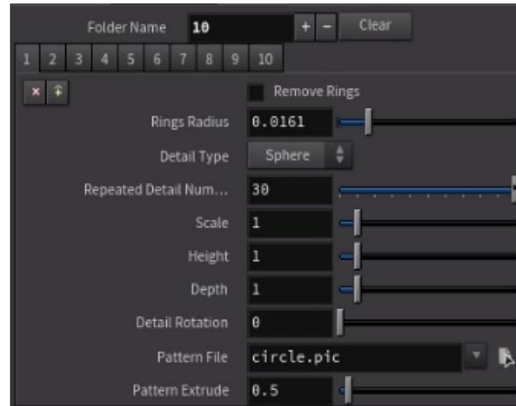

Figure 4.7 A multiparam block of tabs in Houdini

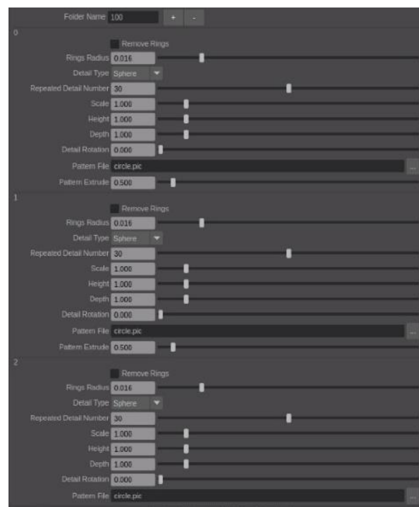

Figure 4.8 An image showing the same UI now in Maya with the multiparam block of tabs changed into a list view

As can be seen in figure 4.7 and 4.8 when the Houdini Digital asset is loaded into Maya the UI originally created for the digital asset in Houdini changes taking a tab layout and changing it into a list view, this lack of tabs makes for a poor layout for UI's with lots of sections, where a tab layout would be more suited. Another issue with this type of UI is the Folder Name attribute still being an editable option as in the final jewelry tool this is procedurally set based on other parameters meaning it's view in the Maya UI is unnecessary and potentially confusing.

Overall, the use of the Houdini Engine plugin in Maya is useful for the ability to access the powerful procedural tools of Houdini within Maya however the default UI isn't suitable for an artist who doesn't need the extra Houdini functionality, removing the Houdini aspect of the tool was the point of bringing it into Maya in the first place. The creation of a new UI for the Houdini Engine plugin would benefit the artist using the tool greatly, this is something that the jewelry tool manages to do. The jewelry tool creates a new UI with only the parameters that needs to be edited by the artist plus it has a working tab layout that changes based on the other parameters on the controls.

4.2 Other jewelry tools

There are not many procedural jewelry tools discoverable on the internet however one example stands out which is Alterity Design's procedural jewelry however this jewelry works in a different way procedurally to the jewelry tool as instead of the pieces of jewelry being mainly controlled by parameters this jewelry focuses on random pattern creation opting to create jewelry such as the ocean generative pendent which relies on random noise generation to create the bumps on it. (Alterity Design, 2020) This type of procedural modelling is a type of non-interactive procedural modeling. (Morkel & Vangay, 2006)

4.2.1 Types of procedural models

Procedural modelling can be one of three methods, (Morkel & Vangay, 2006) in the previous case it was the type of procedural modeling that creates the model without user interaction as the system is run and the jewelry is randomly generated this can be seen in the ocean generative pendent as the article written on the jewelry even shows the noise used to create the bumps on the jewelry (Alterity Design, 2020) showing the non-interactive state of the jewelry (Morkel & Vangay, 2006). The next type is a procedural method that works by being explicitly stated using a procedural modelling language (Morkel & Vangay, 2006), this can be seen as something created using a language such as the Open Shading Language which allows for procedural shaders to be created. The final type of procedural modeling and the type that the jewelry tool uses is the interactive modeling technique as it uses Maya to show the jewelry in the viewport and allow changes made by the user to be updated (almost) instantly. (Morkel & Vangay, 2006) This method of procedural generation is super useful for object creation as it allows for small tweaks to the object to be made without the need for a new piece of jewelry to be generated every time as the user can see the updates instantly on their screen.

4.3 UI elements

Simple widgets for control are commonplace for software such as Houdini and Maya these can easily be seen in the torus control for both:

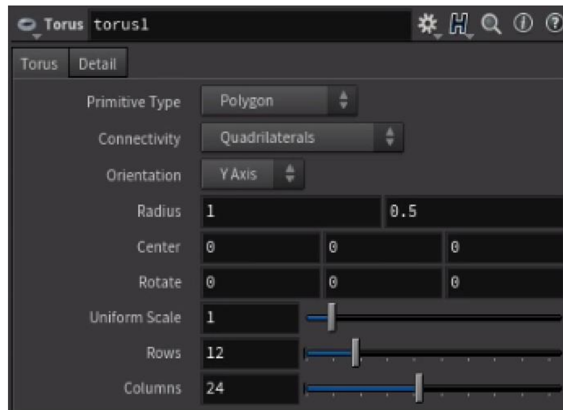

Figure 4.9 Houdini's parameter control for a torus

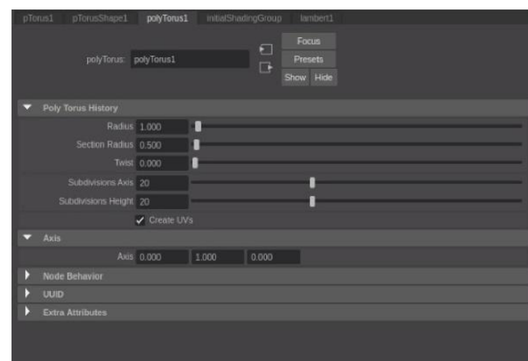

Figure 4.10 Maya's polyTorus attribute editor

Widgets such as sliders, checkboxes or combo boxes can be used for the control of parameters. In the jhumka creator there has to be a layout for the widgets that control the sections of the jhumka, this is why tabs were used to hold the widgets of each section. The tabs must change amount based on parameters previously set, in Houdini this can easily be seen through the multiparam block.

4.3.1 Multiparam block

In Houdini when setting up a Houdini Digital asset it allows for custom parameters and custom ways to lay out these parameters one of these methods is a multiparam block which comes with three options for the layout, multiparam block tabs, multiparam block list and multiparam block scrolling.

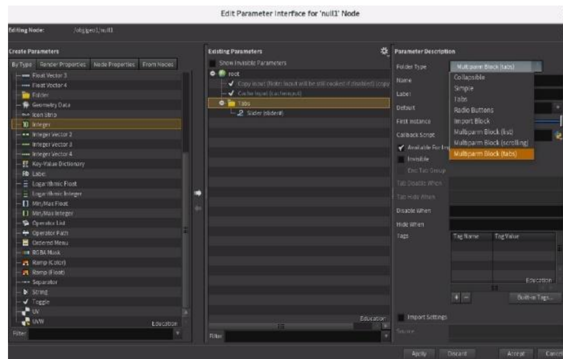

Figure 4.11 Houdini parameter interface showing the options for multiparameter blocks, tabs, list and scrolling

Multiparam block list and scrolling is similar opting to layout the widgets in the same way.

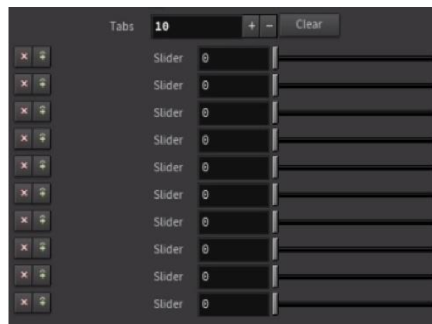

Figure 4.12 A multiparam block laid out as a list

The method seen in figure 4.12 is useful for laying out widgets such as single slider or situations where there should not be too many repetitions of this widget however for multiparam blocks that require lots of widgets per section or many sections this could cause a long complicated UI which would be difficult to use.
This is where a tab layout can be more efficient for use.
Tab widgets allow the user to view the content in individual tabs rather than having to scroll through lots of long text (Landau, 2018) for this reason tabs are a perfect way to layout lots of the controls into one organized set of tabs.
In Houdini this can be set through the folder parameter which can be set as tabs however for a set of tabs which change amount based on another parameter, multiparam tabs can be used.

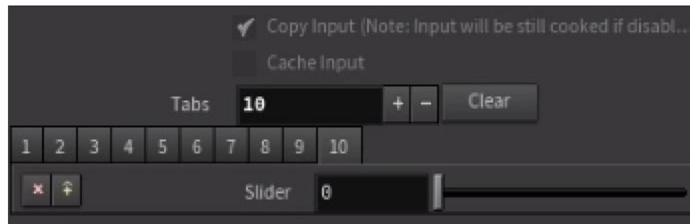

Figure 4.13 An image showing Houdini's multiparam block tabs

An example of where this is used within a node built into Houdini is the KineFX adapt to terrain node.

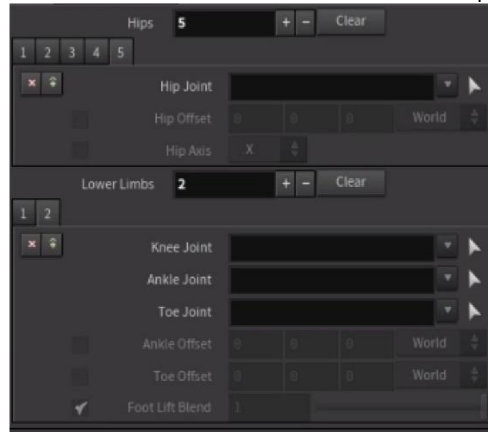

Figure 4.14 KineFX Adapt to terrain multiparam block UI

As can be seen in figure 4.14 the multiparam tab has helped lay out the UI in the adapt to terrain node in a less cluttered format.

Therefore, for the jhumka tool tabs are used to hold the control widgets for the sections in both the hook and main jhumka, both of which are also their own tabs.

5 IMPLIMENTATION

As described previously the two pieces of software that is used for this project are Maya and Houdini, Houdini for its procedural workflow allowing for the creation of procedural geometry through Houdini Digital assets and Maya for the popularity amongst character artists where this tool can be utilized the best.

5.1 Objectives

The objectives of this tool are clear:

- A working Houdini Digital asset for a jhumka
- That same digital asset being accessible in Maya
- A redesigned digital asset UI more suitable for the artist’s use

Something not talked about previously is the organization of the jewelry in the tool itself. To give the ability for the artist to close and reopen the UI a main window will display the geometry in the scene allowing the artist to click on it and reopen the controls for the jewelry they want. The controls won't be per jewelry but allow the creation of multiple pieces of jewelry from the same control this way if the user wants to create something like earrings instead of having to change the attributes on both earrings each time a change is made one set of controls can be tethered to two earrings, improving on time.

5.1.1 The objectives for the UI

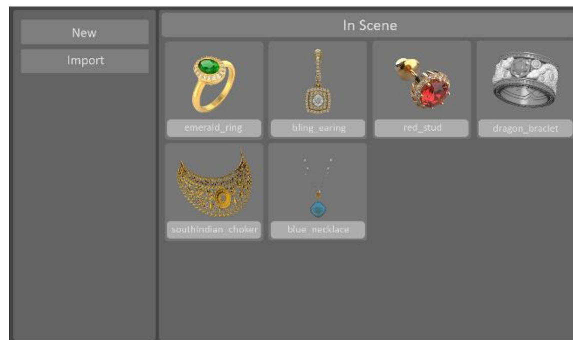

Figure 5.1 The UI plan for the main window

The main window allowed for the user to reopen controls of a previously created jewelry, add a new piece of jewelry or import jewelry; however the import feature was never added.

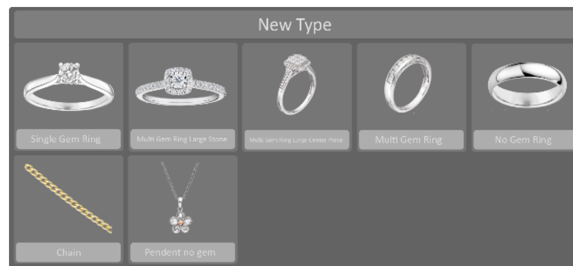

Figure 5.2 the UI plan for the new type window

The new type window, opened by the new button on the main window allows the user to select their new type of jewelry, this largely stayed the same in the final jewelry tool.

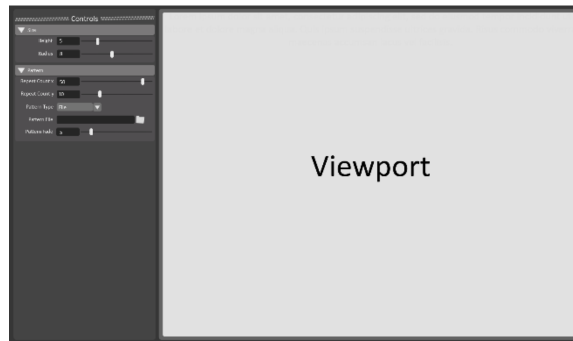

Figure 5.3 the UI plan for the control window

The controls originally featured its own viewport but instead ended up being a tab next to the attribute editor to allow the changing of parameters in a location familiar to the artist.
Overall, the goals for the UI were to allow for the creation of new jewelry, the editing of the parameters of said jewelry through the controls tab and finally the ability to reopen these controls whether that be after they were closed or after Maya was closed itself.

5.2 Things that had to be considered

The three main things that needed to be kept in mind when the tool was being created was efficiency, usability, and outcome.
Efficiency is important as described earlier the point of a procedural tool is to save time, a procedural tool that works slower than a 3D artist is useless. Another reason efficiency is so important is slow tools can be irritating to use and, in some situations, unusable all together. For small tweaks the quick visibility of that change on the screen makes the tool more usable as it allows the artist to mess around easier with the parameters to get the look they want rather than having to wait for an update each time they change something.
During the creation of the tool efficiency was mainly considered in Houdini as that was where the Jhumka was being created which took the most power. Things like VEX for loops where reduced as much as possible, such as in one case where three VEX loops were reduced into one. Efficiency in Houdini was largely aided using the performance monitor allowing for the locating of nodes taking up a lot of the cooking time and either the removal of that node or a way around the slow behavior of it.

Usability is a big part of this tool as making the Houdini Digital asset more usable by an artist is the entire reason the Houdini tool has to be usable in Maya. Usability must always be kept in mind as a tool that does not leave someone wondering how to use it saves time and therefore money. Another big part of the usability is allowing the controls that the artist would need as without it then the output of the tool could be something that does not satisfy them.
As described previously, the UI was a big part of this tool, the creation of a new UI for the better use of the tool by artists was a large part of the usability of the project. Another thing to increases usability was adding many parameters to control the look of the jhumka.
Finally the outcome is very important as the tool needs to produce something that is usable in a scene otherwise the tool is not completing it's whole purpose.

5.3 Houdini Digital Asset

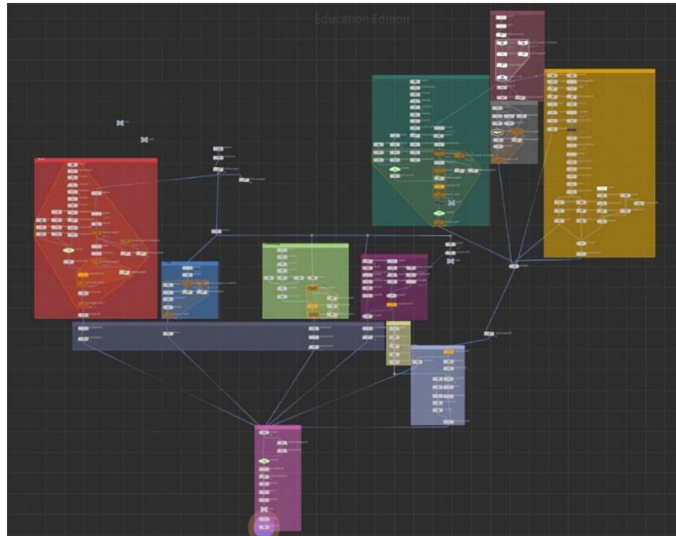

Figure 5.4 The whole Jhumka tool node graph

As the main shape that can be seen for the jhumka is a half sphere, the obvious first steps for the creation of a jhumka is a sphere.

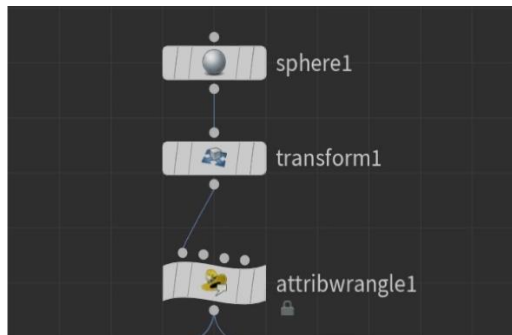

Figure 5.5 The first nodes for the jhumka creation

The first node, the sphere node simply just creates a sphere and the only digital asset parameters that affect this node is the number of rows and columns. The number of rows makes each section smaller, as can be seen in figure 5.5 and the columns add more of the dangling balls.

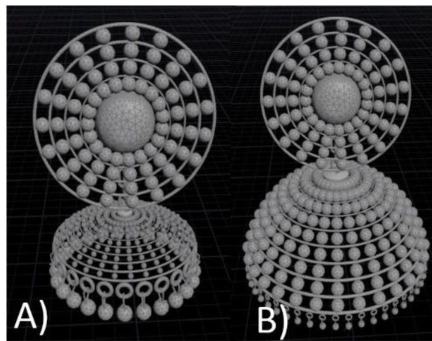

Figure5.6 A) rows 100, columns 25 B) rows 50, columns 50

The next node is a transform node which allows the parameter stretch to adjust the Y scale in this node and the jhumka size which affects the uniform scale of the jhumka.

Then next is an attribute wrangle, this is perhaps one of the most important nodes in the whole network. The VEX code can be seen in Appendix 1.

These lines of code simply go through each point splitting each Y value into their own group and setting these groups to one of two values, ring or detail which depends on whether the while loop counter can be divisible by two with no remainder. This while loop can be seen working through three steps.

Step 1: If the counter, for the number of loops, is above the digital asset parameter for the jhumka amount then the while loop breaks

This step is important to allow user control for the number of rows the jhumka has as once the while loops has created enough loops for the user input, no more rows need to be created which is why the loop breaks.

Step 2: Check each point in the scene, if the loop has been run before there will be a previous max Y, if the point is greater than or equal to this value, discard it as this Y value has already been put into a group and add one into number above variable. For the Y values not already in a group the loop checks each of these values checking the Y value of each to find the largest Y value adding each point with the largest Y value into a list and then puts this into either a detail group or ring group based on whether the counter is divisible by two, this basically just creates alternating rows of detail then ring to follow the jhumka designs analyzed earlier.

Step 3: Once the while loop breaks, either because it has reached the jhumka amount parameter or because all Y values have been added to the group, it takes the final Y value and adds it to a last loop group. This step is to locate the position for the dangling parts on the jhumka.

What comes next is another attribute wrangle

Figure 5.7 The node which comes after the initial Y group setup

The code in this attribute wrangle can be seen in Appendix B.

This code works out the angle and size for the detail in each section of the main jhumka.
This attribute wrangle takes each of the points going down one edge of the sphere using an if statement to see if the point number is divisible by the number of columns. It only works on those points that represent a ring loop by checking if the for-loop counter is divisible by 2. It then adds each of these points to a list.

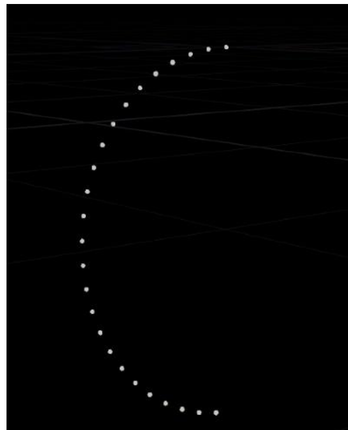

Figure 5.8 The points in endPosList

The list is used in a for loop to then work out the angle between each of the rings and Y, this is done using the cross product:

The cross product states that to find the angle θ between two vectors A and B the equation:

$$\theta = \frac{A \cdot B}{|A||B|}$$

Where:

$$A \cdot B = a_x b_x + a_y b_y + a_z b_z$$

And:

$$|A| = \sqrt{(a_x)^2 + (a_y)^2 + (a_z)^2}$$

Therefore when finding the angle between vector A, which in this case is the vector between the two points, and Y axis (0, 1, 0) the equation looks like:

$$\frac{a_y}{\sqrt{(a_x - b_x)^2 + (a_y - b_y)^2 + (a_z - b_z)^2}}$$

Which is what can be seen in the code.

The way to work out the distance between the two rings, to work out the size of the details, is through finding the magnitude of the vector between the two points which can be found using the equation:

$$\sqrt{(a_x - b_x)^2 + (a_y - b_y)^2 + (a_z - b_z)^2}$$

Which can also be seen in the code.

These values are then added to a list for use later.

Figure 5.9 The nodes that add the detail to the main jhumka

This section of the digital asset deletes all the points that are not in the detail group leaving only points that will be used for detail. It also creates a name attributed based on the Y value of each point using an attribute set up in the first attribute wrangle, this way it can go through each of the Y values separately as they all represent separate sections that need to be worked on individually.

The two attribute wrangles coming off of the for each begin 3 node import all the tab values from parameters using a piece of code which can be seen in Appendix C.

Houdini lays out its multiparam parameter's names as the parameter name then the tab num such as:
scale1 is the scale slider in the second tab therefore using the iteration number when refencing the parameter allows for each tab to edit a different section of the jhumka.
The add node then turns the points into a curve to then allow the resample node to change the number of points to be the repeat amount external parameter in the sections tab. These points are then used to copy the jhumka detail to them meaning the more points, the more detail. Connected to the for each begin 4 is an attribute wrangle that finds the angle of the points position using the cross product, this is exactly the same as before however this time it works it out in regards to the x axis:

$$\frac{a_x}{\sqrt{(a_x)^2 + (a_z)^2}}$$

Ignoring the Y axis as this doesn't matter in a 2D circle.
This value is used to rotate the detail to face outward on the jhumka by making sure that the detail is facing down the z axis to begin with then the angle between the point it's being copied to, and the x axis should be the rotation it undergoes to make sure its facing outward.
Next the detail is copied to the points positioning and sizing them right based on the parameters calculated previously, such as the distance between rings, and the user inputted parameters.

There are three options for detail, sphere, line or file, all controlled by a combo box in the UI which connects to the switch seen in figure 5.9. The setup of the sphere, as can be seen in figure 5.9 on the far left, just rotates the sphere based on the angle around the circle it is and the angle between the rings and scales it based on the distance between the rings. This is the same as the line which also just adds a bend node to create a bump in the middle of a tube.
The file needs a few more procedures to work.

Figure 5.10 The procedures to take a file and make it into a piece of detail

To take a file and turn it into a piece of detail the trace node is extremely important. The trace node allows the user to input a file which then will take the white parts of the image and turn it into a polygon and the black parts will be discarded. This polygon can then be extruded and smoothed to remove any sharp edges. The same transformations to rotate it correctly and size it can then be done like on the sphere and line. All of these types of detail also undergoes the transformation for the user inputted parameters like height, scale in the Y axis, and depth, scale in the z axis.

These nodes allow for the creation of the detail for the main jhumka as can be seen in figure 5.11

Figure 5.11 The output from the detail section of the digital asset

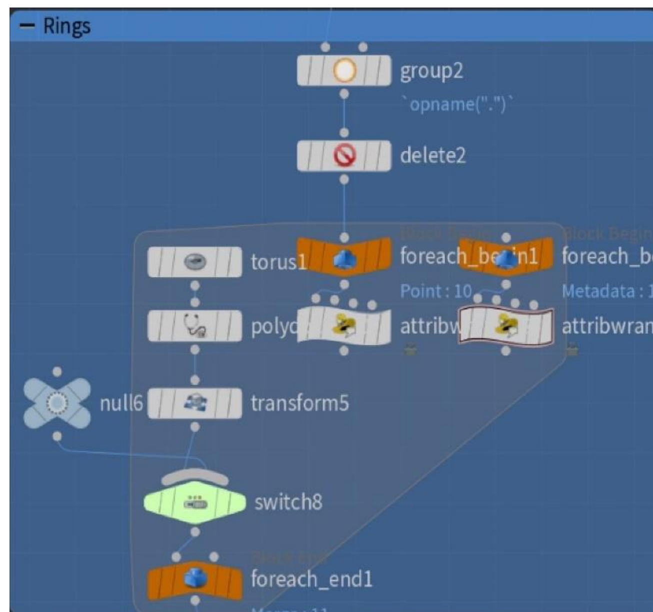

Figure 5.12 The nodes for the creation of the rings for the main jhumka

For the rings it's a simple procedure, every point that isn't part of the main rings is deleted and a for loop goes through each of the rings (separated by the name node). The attribute wrangle connected to the foreach begin contains the code seen in Appendix D.

Which just gets the max and min BBOX values and the center of the object, which is then used in the translate node connected to torus to scale and position the torus to fit to the ring.

The attribute wrangle connected to the metadata node gets the ring radius from the digital asset tabs utilizing the same technique seen in Appendix C

Which is then used in the torus node for the min radius, allowing for the user to be able to change the radius of the ring. The remove rings checkbox for the current section is then accessed in the same way as the ring radius and this is used in the switch node to decide between using the ring geometry and no geometry if the remove ring checkbox is checked.

As can be seen in figure 5.13 this works well in creating the rings.

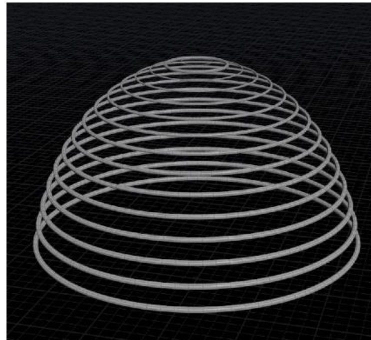

Figure 5.13 The rings created using the main ring procedures

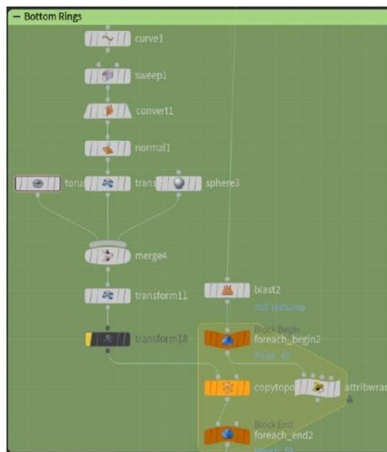

Figure 5.14 The nodes used to create the dangling balls for the jhumka

The dangling parts only have to be created on the bottom row of the jhumka therefore, using the last row group from the first attribute wrangle, all but the bottom row is deleted.
The curve, torus and sphere which are merged is used to create the dangling geometry. The translate below the geometry for the dangling parts both rotates the geometry so that they are always facing outward and scales the geometry to the size set by the user in the digital asset UI.
The angle for the dangle parts is found in the attribute wrangle using the equation:

$$\frac{a_x}{\sqrt{(a_x)^2 + (a_z)^2}}$$

The same as the detail rotation discussed earlier.

Figure 5.15 The dangling balls created through the jewelry tool

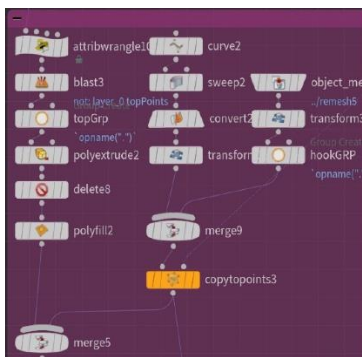

Figure 5.16 The node layout for the hook creation

For the top of the main jhumka three things are needed, the top of the jhumka that a hook attaches to, the hook at the top of the jhumka and the first chain link attached to that hook. The node stream on the far left makes the top of the jhumka, the middle stream makes the hook and the right stream makes the first chain.
For the actual top of the jhumka the first node is an attribute wrangle coming from the sphere discussed earlier.
This attribute wrangle contains the code seen in Appendix E.
This gets the first loop of points from the sphere, essentially getting the top of the sphere by getting every point from one loop around the columns. Then the blast node deletes every point not in the group leaving just the top of the sphere, as can be seen in figure 5.17.

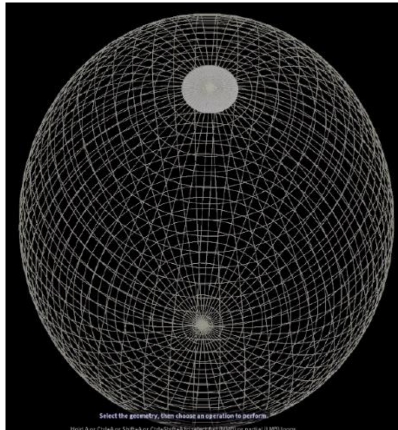

Figure 5.17 only the top of the sphere remaining after attribute wrangle 10 and blast 3

This is then extruded to add depth and the face on the end of the cylinder is deleted so it can be rebuilt by the fill node with simpler geometry.
The hook on the middle node stream is just a curve with geometry swept across it.
The first chain starts with an object merge as the chain link creation is stored in another section of the tool. The first chain is then placed in the correct position in relation to the hook.
The hook and chain are then copied to a point which is found through blasting all points on the original sphere except the 0 point which will always be at the top of the sphere, this then copies the geometry into place at this point.
As can be seen by figure 5.18 this creates a well-built main jhumka hook.

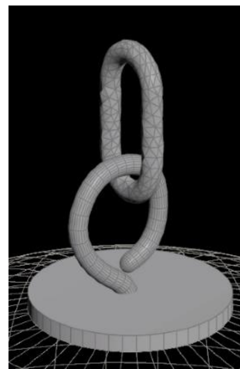

Figure 5.18 The final top of jhumka hook

For the ear pin section, in the tools naming, called the hook the creation on it starts off with:

Figure 5.19 the nodes to set up the hook

A circle is created, and the ends node just removes the polygon face on the circle. Then a for loop is used, not to loop through the geometry but instead to repeat the, the number of sections (multiplied by two) times.
In the attribute wrangle connected to the meta data node it gets tab attributes like previous examples utilizing a similar method to that seen in Appendix C.

However it also works out the size of the current loop using the code seen in Appendix F.

This code gets the ratio of difference in size between each loop which is worked out by using the number of iterations divided by one then it multiplies this ratio by the iteration number plus one (as the iterations start from 0) to work out the size of the circle in that current iteration. This size is then used in the transform node, within the loop, to set the circle size.
Next the attribute wrangle after this is used to work out the groups for the rings and detail like with the main jhumka which can be read in an earlier section however because with the hook there is a jewel at the center the if statement:

```
if( iteration > `chs("../jewelSize")`*2 )
```

is used to make sure that none of the groups, which are used for the creation of rings and detail, are created for where the jewel is going to be.

As can be seen in figure 5.20 this works well for the creation of rows within a circle.

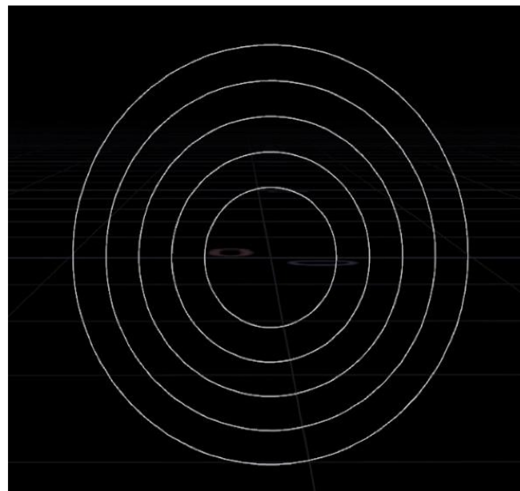

Figure 5.20 The circles created by the nodes in figure 5.19

The attribute wrangle coming out of the loop has no downstream node, this is used to setup the size of the detail like before with the main jhumka however because there's no variation in the distance between rings this value is one distance variable which is worked out by dividing the radius of the circle by the number of sections.
The creation of the detail for the hook is largely the same as the main jhumka as can be seen from figure 5.21

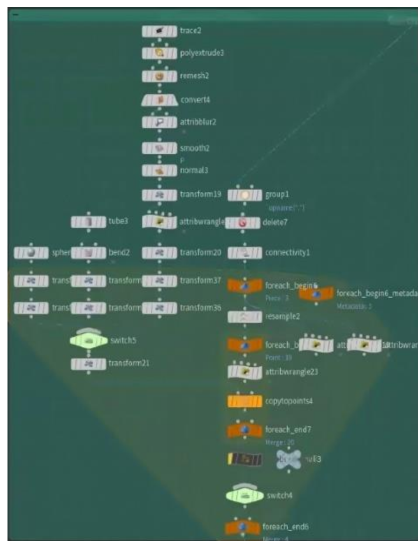

Figure 5.21 The procedures used to create the detail on the hook

The main difference is the attribute wrangle that is in the for each point loop before the copy to points, this works out the angle for the detail as this is a 2D circle in the YX plane, so the attribute wrangle the code seen in Appendix G:

To work out the angle between the current point in the loop and the Y axis. As can be seen in figure 5.22 this works to create the detail in the hook.

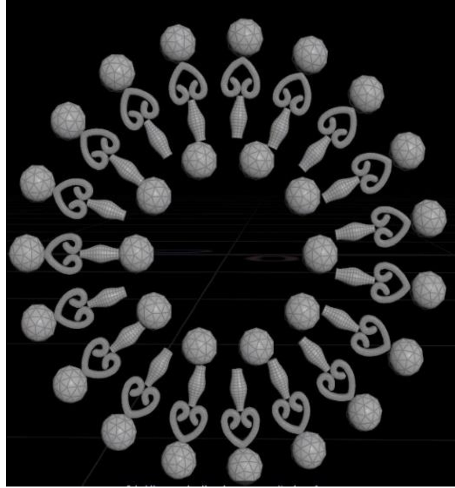

Figure 5.22 The hook detail created using the nodes in figure 5.21

The rings for the hook are created in largely the same way as the main Jhumka as well.

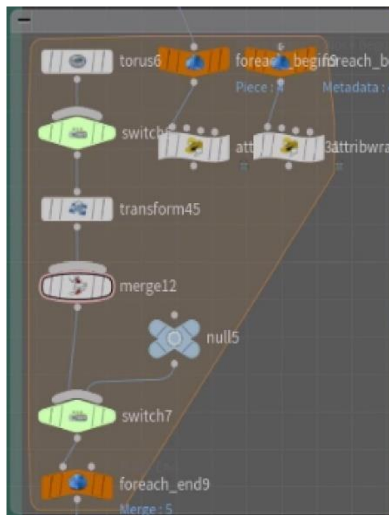

Figure 5.23 These procedures create the rings for the hook

This is created through getting the torus radius and remove ring checkbox value of the current section using similar code to that in Appendix C.

Which is then used to set the torus radius and switch value and then the size of the ring is fitted to the current loop circle's BBOX like with the main jhumka rings. The result of these procedures can be seen in figure 5.24.

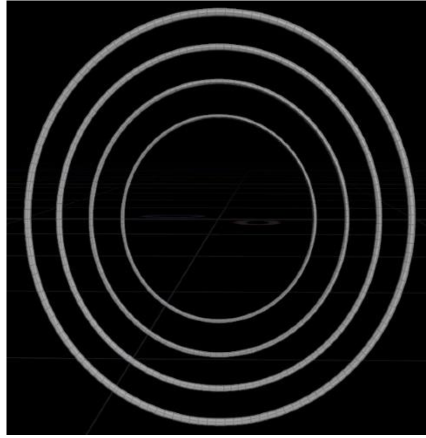

Figure 5.24 the rings created using the nodes in figure 5.23

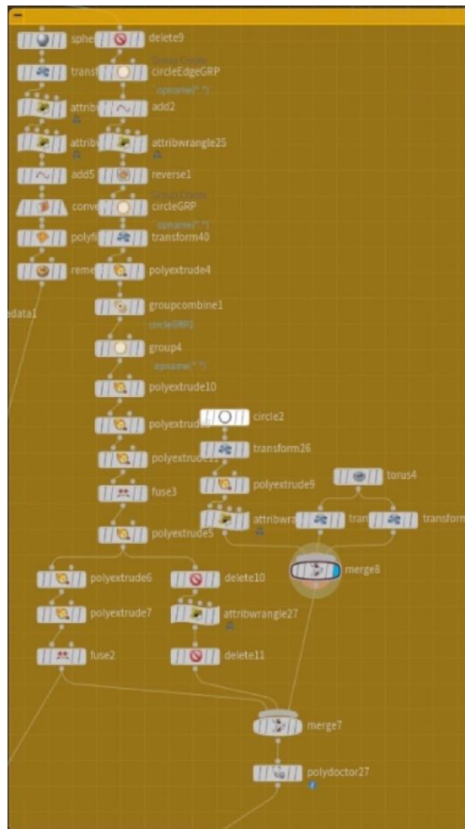

Figure 5.25 The nodes used to create the pin

Figure 5.25 shows how the pin is created. The middle row, which starts with the delete node begins the creation of the pin. The delete node is attached to the circles seen in figure 5.24 and is used to delete all circles but the one which will hold the gem. This gem circle is worked out using the user input for jewel size where a group is set in attributewrangle15 in the loop using the code seen within Appendix H.

This curve circle then uses an add node to make it into a closed circle, to help give an idea of what is going on figure 5.26 shows what has been created up until this point (the circle) compared to the circles inputted into this node from figure 5.24.

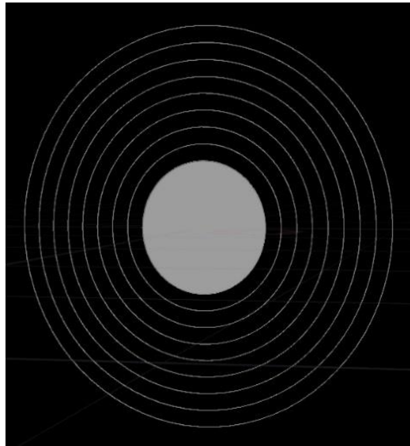

Figure 5.26 The beginning of the pin and jewel creation

Next up this the width and height is found for this circle, using the bounding box code seen in Appendix D and subtracting the min x from max x and min y from max y.

This is used to set the size for the jewel creation, discussed later.
This circle is then reversed, so it's extruded backwards, and extruded out into a pin shape, This shape can be seen in figure 5.27

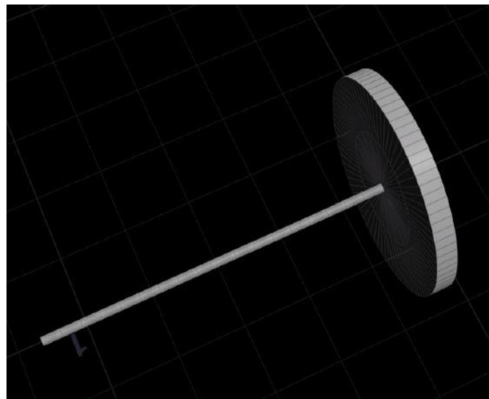

Figure 5.27 The view from node fuse 2, this is the ear pin for the jhumka

Torus 4 and circle 2 on the right side of the pin node graph is used to create the butterfly backing of the jhumka earing which can be seen in figure 5.28

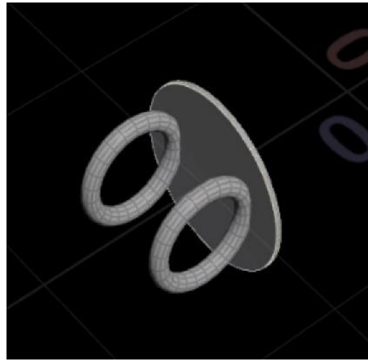

Figure 5.28 The butterfly backing of the jhumka jewelry

For the jewel which is created using the nodes seen on the left of the node network in figure 5.25 a sphere is used. The sphere columns are set to match that of the pin so that it fits perfectly to it. The sphere is then scaled down to be the same size as the pin's circle using the width variable calculated in attributewrangle25 discussed earlier. The negative z points on the sphere are deleted using vex code to create an if statement to check if a point has a negative Z value then the point is removed using the removepoint vex function.

And the single point on the max z is deleted using the VEX code to check which point has the same Z value as the max bounding box then, the same as before, removepoint is used to delete the point.

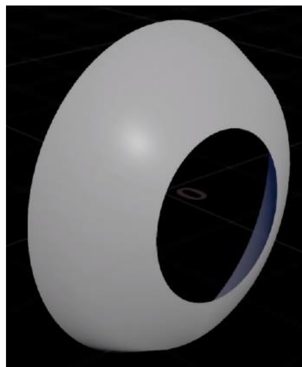

Figure 5.29 The jewel mid way though creation

By getting the max Z value again, this time it's the circle surrounding the hole and assigning these to a group which is then used in an add node to close the circle.
The back hole is then filled in using a fill node and the jewel is re-meshed to create a better geometric representation of the jewel. The final jewel can be seen in figure 5.30

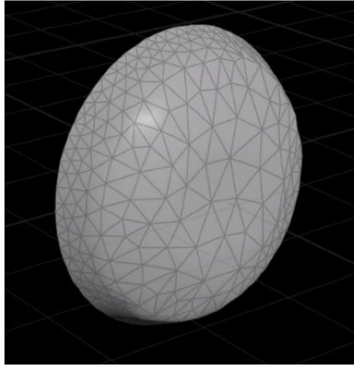

Figure 5.30 The final jewel created by the jhumka creator in the jewelry tool

Once all the sections of the hook are completed they are then merged together to leave the geometry seen in figure 5.31.

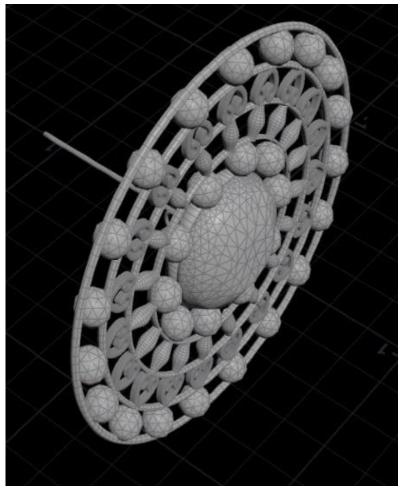

Figure 5.31 The final hook geometry created using the nodes seen in figure 5.25

For the positioning of the jhumka hook and the chain, points were extracted from both the main jhumka and the hook with copy to points nodes used to position the geometry correctly.
For the positioning of the hook, the 0 point from the sphere, seen in figure 5.17, for the creation the main jhumka is used, the same as what was used to position the attachment at the top of the jhumka which can be seen in figure 5.18 .

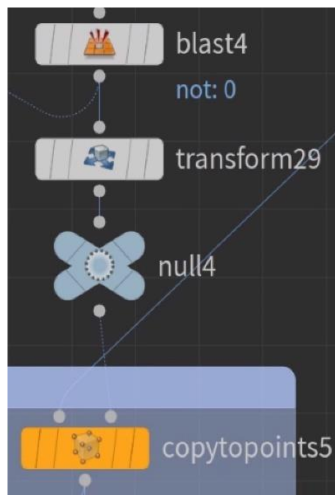

Figure 5.32 The nodes positioning the point for the Jhumka hook

This point then goes through a transform node which moves it upwards a distance equal to the radius of the circle in figure 5.20 so that the main jhumka does not intersect with the big circle on the hook, which can be seen in figure 5.31. It is then copied to this point positioning it correctly in terms of the main jhumka, this mix of the hook and jhumka can be seen in figure 5.33

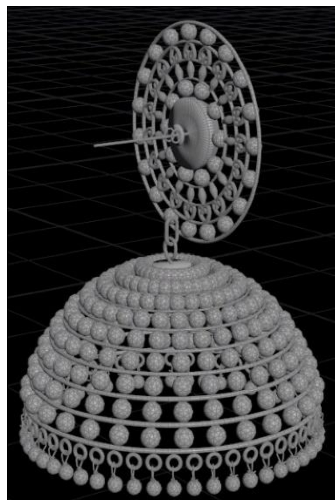

Figure 5.33 The Jhumka and the hook merged together

The final thing to for the tool to do is create the chain that connects the hook to the main jhumka, this is done using two points, one created during the creation of the pin, and one created using the following node graph:

Figure 5.34 the nodes used to get the starting point for the chain

This node graph takes in the geometry seen in figure 5.34 which is the hook at the top of the jhumka and start of chain. The hook is then deleted leaving only the start of the chain. The center of this chain is then found using the attribute wrangle and VEX code which finds the BBOX center position parameters and then a new point is created using these position attributes.
This will be used as the point for the start of the chain.

The point for the end of the chain is created during the pin creation and happens at

Figure 5.35 Image highlighting where the end point for the chain is created.

These nodes take the partly created pin deletes everything but the end cap of the last extrude, finds the center point of that face then deletes the face leaving just the point which can be seen in figure 5.36.

Figure 5.36 The point created by nodes highlighted in figure 5.35

The start point and end point are then merged in merge10 in figure 5.37. An add node then creates a curve between the points and the resample node adds more points for the chain links to be copied to.

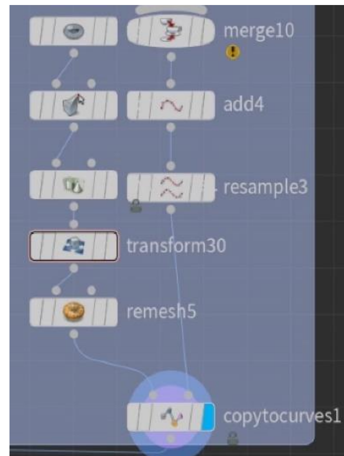

Figure 5.37 The nodes that create the chain

The chain links are then copied to the curve using the copytocurve node which also allows for the varying rotation of the chain links using the roll parameter in this node. As this is a chain created between two points, one on the top of the main jhumka and one on the hook this allows for the procedural creation of a chain between the two objects which works well as can be seen in figure 5.38

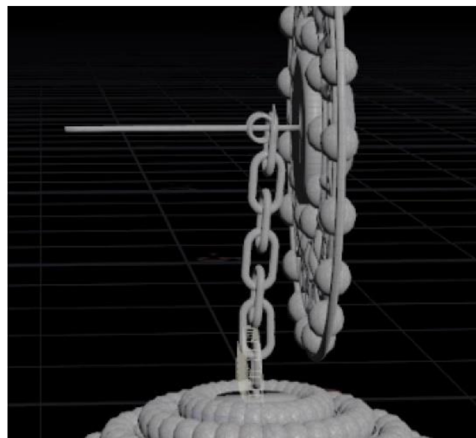

Figure 5.38 The chain between the main jhumka and the hook

With the chain created that is the entire node network of the jhumka creation allowing for the user to have control to create a jhumka with parameters allowing them to setup the jhumka to look how they want.

5.4 Maya

From the Maya API this tool utilises 2 important features, the create command and create node features.

5.4.1 MPxNode

As this Tool needs the ability to reopen controls after Maya has closed plus have one place to save the parameters of multiple pieces of jewellery, an MPxNode was used to contain this information,
A function called initializePlugin is used to tell Maya how to initialise the node plugin, this can be seen in Appendix I.

This relies on the line of code, plugin.registerNode("AttributeNode", AttributeNode.nodeId, AttributeNode.createNode, AttributeNode.nodeInitializer) to set up the AttributeNode plugin where AttributeNode.nodeId is a variable containing a hexadecimal number giving the node a unique ID which Maya needs for the attribute node to work. Next is the create node function AttributeNode.createNode which can be seen in Appendix J.

The create node function just returns the MPxNode of AttributeNode. Finally, is the node initializer function AttributeNode.nodeInitialize which tells Maya how to initialise the function in the case of the jewellery tool this is a long function initialising each of the parameters for the node for example jhumka rows is initialised using the code seen in Appendix K

This is just the inRows parameter for the values coming in from the controls and a outRows parameter for the values going into the jhumka digital asset.

When it comes to defining an attribute for a node in the Maya API it can easily be done by setting up an attribute with the functions relating to that attribute, in this case a numeric attribute, using a create function to set up a specific function with a name, even more detailed function set, in this case a kInt and a default value. Next, Boolean variables need to be set to say what can be done with this attribute such as whether it’s readable and writable. Finally, this new variable can be added to the node using the MPxNode method addAttribute.
The final thing to do is to tell Maya what in parameters can affect what out parameters using the MPxNode method attributeAffects.
This is done for every variable in the jewellery tool, even the tabs however these are set up in a for loop that just creates 50 of every tab variable as the user will never exceed 50 tabs.

Another very important function that lives within the attributeNode is the compute function. What the compute function does is tell Maya how to work out the out values for each of the out attributes.
An example if outRows was the only attribute can be seen in Appendix L

The way the compute function works is by first checking if the outputting parameter that needs to be set clean is the outRows as if it’s not there's no point wasting computational power on working it out. It knows which plug needs to be cleaned based on what in parameter has changed as during initialization of the node it had already been told through the attributeAffects method which in parameters will affect which out parameters. The dataBlock works as storage for the node and therefore it’s used to store the value of the inputted data and the outputted data.

The inRows data is retrieved through the dataBlock using the inputValue method, then this data is converted into a float using the asFloat method.
The variable outRows is then set as the float value for the inRows, this will be used to set the value of the data for the nodes outRows. Next the dataBlock is used to access the data for the output value of AttributeNode.outRows which is set as the outRows value which holds the inRows float value using the setFloat method. Finally to tell Maya that this plug, AttributeNode.outRows, has been updated to reflect the inRows change the setClean method is used.

Usually this section of the code in the node attribute is used to setup the calculations happening within the node however since this is just being used to hold information within the scene no calculations are done and the input value is the same as the output value.

5.4.2 MPx.Command

The other Maya API ability this tool takes advantage of is the MPx.Command. The command is set up through an initializePlugin function, like the nodes initialize plugin as seen in Appendix I.

The main line for the addition of the command is the line:
plugin_fn.registerCommand(JewleryTool.CMD_NAME, JewleryTool.creator)
Which tells Maya the command name through the variable JewleryTool.CMD_NAME which is a variable parented to the jewleryTool MPx.Command which is just a string name for the command. It also tells maya the creator function for the command which in the case of the tool just returns the jewellery tool command class.
As a command will not have any parameters that need to be set up there is no initialise function neither is a compute function instead there is a doIt function, this is a function that runs when the command is run, it can often be accompanied with a undoIt function that tells Maya what to do if the undo is pressed after the command has been run however for the opening of a window it did not seem necessary.
In the case of this tool the doIt just sets up and opens the MainWindow.
This command was created to simplify the opening of the tool down to one line of code, cmds.JewleryTool() and allow for the tool to be loaded as a plugin.

In the initialisePlugin function there is a class that is called shelf(), this class is used to setup a shelf with a Jewellery Tool button to allow the user to press the button to open the tool.
This class looks like can be seen in Appendix N.

In the initialisation of this tool it deletes any previous version of the shelf, if one exists then sets up a new Jewellery shelf using the line cmds.shelfLayout("JewllerySheIf", parent="ShelfLayout") parenting it to the ShelfLayout in Maya. Next a button is added using the maya command function for a shelf button, which just sets up the size of the button, the image the button shows, the colour of the button, the label of the button and most importantly, the command of the button.
The command is linked to doubleun which just runs the command cmds.JewleryTool().

5.4.3 The Maya UI

For many parts of the UI QT Creator was utilised, either to create the UI like the main window or the select type window or just to gain ideas on how to code certain aspects of the UI such as custom QWidget classes like the file selector.

To turn a .ui file into a Pyside2 (which Maya 2020 uses) compatible code, the Pyside 2 uic method was utilised by, in the Linux terminal, entering the code:

```
pyside2-uic fileName.ui
```

Which displays, in the terminal, the pyside 2 compatible code.

5.4.3.1 The Main Window

The Main Window is simple, it's just a window that has a button for new jewellery and a QListWidget showing all the jewellery already set up in the scene.

All that pressing the new button does is open the new type window.
Double clicking on one of the pieces of jewellery in the list widget either reshows the controls widget or if the controls is already closed it rebuilds the controls by creating a new controls widget and setting all the sliders to match that of the attribute node using the cmds.settAttr function.

To get the list of every piece of jewellery in the scene all the MainWindow does is go through every piece of geometry and look for attributeNodes and lists all the ones it finds.

5.4.3.2 newTypeWindow

The newTypeWindow is a window that allows the user to select the new type of jewellery they want to create, currently there is only one type of jewellery in the tool, a jhumka. The jhumka is represented by a checkable button in the newTypeWindow, when this is checked the create button is activated. When the create button is pressed it pops up a QDialog window which lets the user enter a name for the new piece of jewellery, this name is then checked for errors.
This checks for 4 different name errors. The first error is if the name is just spaces which is checked for through the use of the .isspace() function which is a string method that returns true if the string is just spaces. The next error is if the name is an empty string which is checked for by seeing if the length of the name is 0. The next error is if the name has already been used before which it checks for by seeing if the name is in a list of previously used jewellery names in the scene. Finally, the last error is if the name contains any spaces as this is not accepted, this is done by using the line of code ' ' in Dialog.name as this will check every letter in the string and see if any of them are spaces. This error variable is then checked back in the new type window and if there is no error it minimises the main window, closes itself and then creates a new control widget for the piece of jewellery.

5.4.3.3 Control widget

The control widget works using two custom QWidget classes, the variable slider class and the tab class. The tab class also contains the variable slider class and another custom QWidget class the file selector class.

Variable slider class

As every time a slider is used in the Jewellery tool it comes with a QSpinbox linked to it. To simplify the creation of these sliders a class of base type QWidget was created containing a label, a spin box and a slider. The creation of the slider looks like:

```
createVariableSlider(self, name, value=50, max=100, int=1, parent = None, single_step =
0.05)
```

This allows for the name of the slider, which is what the label displays, the default value of the slide, the max value of the slider, whether the slider is an int value or a float, the parent of the slider (if there is one) and the single step value.
An important feature of the variable slider is the linking of the spin box to the slider through signals and slots.

The slider value is divided by 1000 before it sets the spin box value and the spin box value is multiplied by 1000 as because the slider only accepts integers so to get a slider value which can go to 0.001 it works through dividing the slider value by 1000 so 1 on the slider is 0.001 value.

The signals and slots work by changing the value of the other widget(spin box if the slider is changed and slider if the spin box is changed) so that they are always both linked.
These sliders are used throughout the entire control widget to control the parameters including in the tab widget.

The tab widget

The tab widget is used to easily set up a tab. Because the tabs need to be created many times wrapping it up into a class simplifies the code down.
The tabs are mostly just the custom variable sliders and other QWidgets for the selecting of the parameters however a notable QWidget is the custom file selector QWidget class.
The file selector allows for the selection of files for the custom patterns in the main jhumka and hook tabs, it works by having a line edit where the user can put in a file path or a button which opens a file view window to select a file. When the button is pressed the tool opens a QDialog using the code that can be seen in Appendix O which uses the QFileDialog.getOpenFileName method to get a file dialog Allowing the user to select a file, then the file path will be converted into a string and the line edit’s text will be set to match the file path using the QLineEdit’s set text method

For the controls widgets this is all that is needed.

To set up the dock for the controls to go on required a function to create a dock, this can be seen in Appendix P

This code first checks if the controls are being rebuilt or not. If the controls are being rebuilt it tries to reopen the tab, if it was able to reopen the controls isReopened becomes true and the controls class is done however if the controls are for a new piece of jewellery or if the controls are being rebuilt and they couldn't be reopened (if Maya has closed) then a new controls tab needs to be built.
If this is a new set of controls the node, which will hold the parameter values discussed earlier, linked to this set of controls is created.
The pymel function workspaceControl is then used to setup the dock control with the docking location to be with the Attribute editor.
This dock can then be used as the parent, once a suitable pointer to it has been created, in the controls widget.

If the controls are getting rebuilt every parameter will be set as the attribute node's parameter values using the cmds.setAttr function.

To link the controls and the node all that happens is that for every parameter a signals and slots is set up so that when it is changed it sets the node parameter to the same value. For example for the rows node can use:

```
nodeAttr = "{}.rows".format(self.nodeName)
```

To get the node name then set up signals and slots:

```
self.jhumkaRowsSlider.spinBox].valueChanged.connect(lambda n: self.slot(n, nodeAttr))
```

Where slot is just a set attribute function containing:

```
cmds.setAttr(nodeAttr, n)
```

Where n represents the slider or spin box value.
This ensures every time a value is changed in the controls it is updated on the node.

When the create jhumka button is clicked on the controls a Jhumka class is run this is used to create the Houdini Digital asset within the Maya scene.

5.4.3.4 The Jhumka

To load in the Digital asset Maya provides a list of commands that can be used in regards to Houdini Digital assets, the code for loading one into the scene can be seen in Appendix Q.

This line of code will load the jhumka tool created in Houdini, discussed earlier, into our scene.
The node related to this jhumka can then be discovered using the line of code:

```
self.nodeName = "{}_JhumkaAttributesNode".format(name)
```

To then allow for the connecting of attributes from the node to the jhumka.
To connect one parameter to another a maya command can be used:

```
cmds.connectAttr( nodeOut, JhumkaParameter)
```

This means that the jhumka parameter will now always be whatever the node Out parameter is essentially connecting the jhumka to the controls but in a way where the controls can be reopened even when Maya closes.

5.4.4 Installing the tool

To install the tool into Maya on other computers a install python file is used, the main part of this file looks can be seen in Appendix R.
This code creates a .mod file containing the path to the plugins folder which allows the loading of the MPxCommand and the MPxNode and it also adds the path to the jewellery tool folder to allow for accessing of all the files in the folders later. It locates the Maya directory and then saves this .mod file in the modulus folder

All the files that are used within the tool are laid out in folders within the jewellery tool directory so that all the files can be found relative to the path of the Jewellery tool directory.

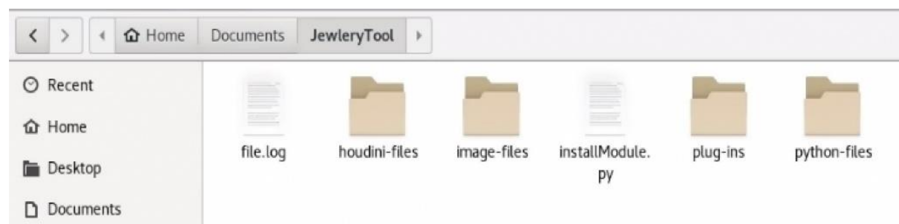

Figure 5.39 The layout of the jewelry tool files

Within the code then anyone of the files can be found easily, an example of this is the Houdini Digital asset which is inside the Houdini-files folder.
In the jhumka.py file the code:

```
plugin_root = os.environ.get("JEWLERYTOOLPLUGIN_ROOT")
houdiniFilesPath = plugin_root + "/houdini-files/"
```

can be used to find the file path to the houdini-files folder then to load the file

```
cmds.houdiniAsset(loadAsset=[houdiniFilesPath+"Jhumka_tool.hdanc",
```

This file path can be calculated on any computer therefore allowing for the jewellery tool to be portable between computers.

6 CONCLUSION

Looking at the tool's objectives, to create an artist friendly jewelry generator in Maya in an efficient way, the tool somewhat achieves this.

The reflection of the tool can be split up into two sections, The tool itself, made in Houdini and the usability of it within Maya.

6.1 Houdini

The Houdini side of the tool is perhaps the weaker part of the tool as it is inefficient, outputs poor geometry and doesn't allow for much control.

The tool is slow, far slower than such a simple tool should be. The power usage had not been thought about when it came to the creation of each aspect of the tool, instead opting to pick the easiest option instead of really thinking about the speed of the tool. In the end the lack of consideration and knowledge of the efficiency caused a slowness in Houdini which only grows when the tool is taken to Maya.

The geometry outputted, whilst replicating the reference material partly was also particularly bad, it not only had weird sharp edges, especially noticeable on the spheres but it also had a sheer lack of detail. Perhaps an artist could get around a lack of detail using shaders but some displacement to add realistic damaging or natural variation in the object, especially for metallic objects, would make the tool better.

Finally, the choices the Artist has just isn't enough, this tool can create one style of jhumka in a scene where there could possibly be hundreds of them, they could all look noticeably similar to an audience.

6.2 Maya

As for the Maya side of the tool the aspects here work better, the UI is good, the API usage helps run the tool and the organization for the tool works well.

The UI components all work well with the code for the UI efficiently making use of custom QWidget classes to represent repeated UI elements such as the tabs. The docking ability of the controls helps make the tool much easier to use for the artist as they can dock them wherever they want and start view there changes in the Maya viewport as they change parameter values, however this is made difficult due to the inefficiency of the tool.

The tool is slow and causes quite a few issues within Maya, including saving and loading files using the tool takes way longer than they should.

The use of the MPxNode works exactly as it should and makes sure that the parameters always stay within a scene. The nodes also allow for that separation between the jhumka and the controls as the controls aren't always going to exist but the nodes are so they can always stay connected to one another.

6.3 Final thoughts

Overall, I think the tool itself is not that well built but the ideas for the UI and the artist control have been well thought about and if the procedural jhumka was better than this could be a promising Maya plugin for artists to use however since the important part of the geometric generation falls short it's practically unusable.

Another key aspect to mention is the fact there's only one piece of jewelry, this was mainly because of time constraints so It's not a massive problem as it can easily be expanded but for the time being the tool is restricted having only one choice of jewelry.

To make the addition of more jewelry easier, instead of having to create a custom Maya UI every time an interesting way this project could move forward is with procedural generation of Maya UI's for the Houdini Digital Assets.

7 REFERENCES

Alterity Design. (2020, December 13). *Generative Jewelry*. Retrieved from https://www.alterity.design/procedural-jewelry

Bhushan, J. B. (1964). *Indian Jewellery, Ornaments and Decorative Design.* D.B. Taraporevala.

Bournemouth University. (n.d.). *MA 3D Computer Animation*. Retrieved from Bournemouth University: https://www.bournemouth.ac.uk/study/courses/ma-3d-computer-animation

Chaos. (n.d.). *Houdini to Maya Alembic Workflow*. Retrieved from Chaos Docs: https://docs.chaos.com/display/VMAYA/Houdini+to+Maya+Alembic+Workflow

Chellam, C. (2018, Febuary 2). *Jewellery Industry in India.* Retrieved from https://www.researchgate.net/publication/337273909_Jewellery_Industry_in_India

Ebert, S. D., Musgrave, K., Peachey, D., Ken, P., & Steven, W. (2003). *Texturing & Modeling A Procedural Approach.* Morgan Kaufmann Publishers.

Foundry. (n.d.). *Direct vs. Procedural Modeling*. Retrieved from https://learn.foundry.com/modo/essentials/content/essentials/modeling/direct-procedural.html

IMDb. (n.d.). *Top Lifetime Grosses*. Retrieved from Box Office Mojo: https://www.boxofficemojo.com/chart/top_lifetime_gross/?area=XWW

Indian Law Office. (n.d.). *Indian Jewellery Market.* Retrieved from Indian Law Office: http://indialawoffices.com/ilo_pdf/industry-reports/jewellerymarket.pdf

Kumar, A. (2014, September 15). *Bharhut Sculptures and Their Untenable Sunga.* Retrieved from https://d1wqtxts1xzle7.cloudfront.net/36329924/223-241-with-cover-page-v2.pdf?Expires=1660843432&Signature=SZBJ7mmsm-Diqc88xX~zok5WzHmIUHIL6WRjN-1WygndTIm1tg-SDaDdmRdE~7nr-8CrNmErNeGMd2XdYI0ctxZZxhml~fLLEhLgKsnVc9YGd7sae6l1zVy2n9wQNr70~VM-fQr8DGrqAACKLetn

Landau, R. (2018, December 11). *Tabs Widget - A New Way to Organize Content.* Retrieved from Duda: https://resources.duda.co/tabs-widget#:~:text=The%20new%20Tabs%20widget%20is,members%20in%20compact%2C%20connected%20sections.

Li, Q., Guan, Y., & Lu, H. (2021). *Development of the Global Film Industry.* Retrieved from https://library.oapen.org/bitstream/handle/20.500.12657/41531/9780367508234_text.pdf?sequence=1&isAllowed=y

Miller, G., St'ava, O., Benes, B., & Mech, R. (2011). *Guided Procedural Modeling.* Retrieved from https://onlinelibrary.wiley.com/doi/epdf/10.1111/j.1467-8659.2011.01886.x

Moritz. (2016, June 21). *Quickie: Exporting Geo From Houdini*. Retrieved from https://entagma.com/quickie-exporting-geo-from-houdini/

Morkel, C., & Vangay, S. (2006). *Procediral Modeling Facilities for Hierarchical Object Generation.* Retrieved from https://dl.acm.org/doi/pdf/10.1145/1108590.1108614

Muller, P., Wonka, P., Haegler, S., Ulmer, A., & Gool, L. V. (2006, 07). *Procedural Modeling of Buildings.* Retrieved from https://dl.acm.org/doi/pdf/10.1145/1179352.1141931

Okun, J. A., & Zwerman, S. (2015). *The VES Handbook of Visual Effects.* Focal Press.

Raniwala, P. (2021, May 27). *Tracing the Evolution of the Jhumka.* Retrieved from https://www.naturaldiamonds.com/in/style-innovation/history-evolution/history-of-jhumka-earrings/

SideFX. (n.d.). *Attribute Wrangle geometry node*. Retrieved from https://www.sidefx.com/docs/houdini/nodes/sop/attribwrangle

SideFX. (n.d.). *Grid geometry node*. Retrieved from https://www.sidefx.com/docs/houdini/nodes/sop/grid

SideFX. (n.d.). *Houdini Engine for film and TV.* Retrieved from https://www.sidefx.com/media/uploads/products/engine/hengine_filmtv.pdf

SideFX. (n.d.). *Point Jitter geometry node*. Retrieved from https://www.sidefx.com/docs/houdini/nodes/sop/pointjitter.html

SideFX. (n.d.). *Smooth geometry node*. Retrieved from https://www.sidefx.com/docs/houdini/nodes/sop/smooth-.html

SideFX. (n.d.). *Transform geometry node*. Retrieved from https://www.sidefx.com/docs/houdini/nodes/sop/xform.html

Steffen, W., Koning, N., Wenger, S., Morisset, C., & Magnor, M. (2011, April). *Shape: A 3D Modeling Tool for Astrophysics.* Retrieved from https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5453364&tag=1

Still, T. (2021, November 2). *Parametric Modeling vs. Direct Modeling: What's the Difference?* Retrieved from Autodesk: https://www.autodesk.com/products/fusion-360/blog/parametric-modeling-versus-direct-modeling/

Thakur, P. (2019, June). *Jewelllery a Treasure of Symbolism in South Asia.* Retrieved from https://www.researchgate.net/publication/333973775_Jewellery_A_Treasure_of_Symbolism_in_South_Asia

8 APPENDIXES

8.1 Appendix A Splitting Y values into groups

ALGORITHM 1: Splitting Y values into groups

```
int numPoints = npoints(0);
f@maxY = getbbox_max(0)[1];
f[]@height_list;
f[]@height_list_detail;
int counter = 0;
while(@numAbove < numPoints )
{
    f@maxYTest = -100;
    i@numAbove = 0;
    i[]@localList;
    if(counter >= `chs("../jhumkaDeleteAmount")` *2 + 1)
    {
        break;
    }
    resize(@localList, 0);
    for(int p=0; p<numPoints; p++)
    {
        vector Pos = point(0,'P',p);
        float PosY = Pos[1];
        if( PosY >= @maxY )
        {
            int uselessVariable = 0;
            @numAbove += 1;
        }
        else
        {
            if(PosY > @maxYTest)
            {
                @maxYTest = PosY;
            }
        }
        if( PosY == @maxY)
        {
            append(@localList, p);
            if( counter % 2)
            {
                string groupname2 =sprintf("layerDetail_%d", counter);
                setpointgroup(0, groupname2, p, 1, “set”);
             }
             else
```

```
            {
                string groupname =sprintf("layer_%d", counter);
                setpointgroup(0, groupname, p, 1, "set");
            }
        }
    }
@maxY = @maxYTest;
counter += 1;
}
for(int point : @localList)
{
    setpointgroup(0, "lastLoop", point, 1, "set");
}
f@lowestVal = @maxY;
s@lastPiece = sprintf("layer_%d", counter);
i@numOfPieces = counter + 1;
```

8.2 Appendix B: getting angle and distance values

ALGORITHM 2: Getting Angle and Distance value

```
i@numPoints = npoints(0);
v[]@endPosList;
f[]@distanceList;
f[]@angleList;

int counter_other = 0;
for(int p=0; p<@numPoints; p++)
{
    if(p % `chs("../sphere1/cols")`)
    {
        setpointgroup(0, "endPoints", p, 0, "set");
    }
    else
    {
        if(counter_other % 2)
        {
            int useless = 1;
        }
        else
        {
            setpointgroup(0, "endPoints", p, 1, "set");
            vector pointPos = point(0, 'P', p);
```

```
            //vector pointPos = {1,1,1};
            append(@endPosList, pointPos);
        }
        counter_other += 1;
    }
}
int counter = 0;
for( vector pos: @endPosList)
{
    vector nextPos = @endPosList[counter+1];
    vector subtract = pos - nextPos;
    float                                     distance                                   =
sqrt((subtract[0]*subtract[0])+(subtract[1]*subtract[1])+(subtract[2]*subtract[2]));
    append(@distanceList, distance);
    float  angle_rad  =   subtract[1]/(sqrt(pow(subtract[0],2)  +  pow(subtract[1],2)  +
pow(subtract[2],2)));
    float angle = acos(angle_rad);
    float angle_degree = angle * (180/3.14);
    append(@angleList, angle_degree);
    counter += 1;
}
```

8.3 Appendix C: getting tab parameters

```
string scaleSlider = sprintf("../scale%i",@iteration);
f@scale = ch(scaleSlider);
```

8.4 Appendix D: getting the bounding box values

```
v@bbox_centre = getbbox_center(0);
v@bbox_min = getbbox_min(0);
v@bbox_max = getbbox_max(0);
```

8.5 Appendix E: Getting the top of the sphere

```
int columns = `chs("../sphere1/cols")`;
for( int i = 0; i <= columns+1; i++ )
{
    setpointgroup(0, "topPoints", i, 1, "set");
```

```
}
```

8.6 Appendix F: Getting the size of the current loop

```
float sizeDif = 1/@numiterations;

f@currentSize = sizeDif * (@iteration+1);
```

8.7 Appendix G: finding the angle between a vector and the Y axis

```
f@angle_rad =  (py)/(sqrt(pow(px,2) + pow(py,2)));
```

8.8 Appendix H: getting the Jewel circle

```
if( iteration == 1 + (`chs("../jewelSize")`*2))
    {
        setpointgroup(0, "forHookGRP", @ptnum, 1, "set");
    }
```

8.9 Appendix I: Initializing a Maya Plugin

```
def initializePlugin(obj):
    vendor = "Matthew Stanton"
    version = "1.0.0"

    plugin = om.MFnPlugin(obj)
    try:
       plugin.registerNode("AttributeNode", AttributeNode.nodeId,
       AttributeNode.createNode, AttributeNode.nodeInitializer)
    except Exception as e:
        print("exception - {0}".format(e))
        om.MGlobal.displayError(
            "Failed to register command: {0}".format(AttributeNode.nodeName)
        )
```

8.10 Appendix J: Node create function

```
def createNode():

        return AttributeNode()
```

8.11 Appendix K: Attribute initialization

```
attr = om.MFnNumericAttribute()

AttributeNode.inRows = attr.create("rows", "r", om.MFnNumericData.kInt, 50)
```

```
        attr.readable = True
        attr.writable = True
        attr.keyable = True
        attr.storable = True

om.MPxNode.addAttribute(AttributeNode.inRows)

outAttr = om.MFnNumericAttribute()
AttributeNode.outRows = outAttr.create("outRows", "or", om.MFnNumericData.kInt)
        outAttr.readable = True
        outAttr.writable = False
        outAttr.keyable = False
        outAttr.storable = False

om.MPxNode.addAttribute(AttributeNode.outRows)
om.MPxNode.attributeAffects(AttributeNode.inRows, AttributeNode.outRows)
```

8.12 Appendix L: MPxNode compute function

```
def compute( self, plug, dataBlock ):
        if(plug == self.outRows):
                dataHandleRows = dataBlock.inputValue(AttributeNode.inRows)
                 inRowsVal = dataHandleRows.asFloat()

                outRows = inRowsVal
                 dataHandleOutRows = dataBlock.outputValue(AttributeNode.outRows)
                 dataHandleOutRows.setFloat(outRows)
                 dataBlock.setClean(plug)
        else:
                Return
```

8.13 Appendix M: doIt function:

```
def doIt(self, args):

        if not QApplication.instance():
```

```
            app = QApplication(sys.argv)
        else:
            app = QApplication.instance()


        window = MainWindow()
        window.show()

        sys.exit(app.exec_())
```

8.14 Appendix N the shelf function:

```
class shelf():

    def doublun(self):
        if not(cmds.pluginInfo( "MastersProject", query=True, loaded=True)):
            cmds.loadPlugin("MastersProject")
        cmds.JewleryTool()

    def __init__(self):

        if cmds.shelfLayout("JewlleryShelf", exists=1):
            pm.deleteUI("JewlleryShelf")


        cmds.shelfLayout("JewlleryShelf", parent="ShelfLayout")

        cmds.setParent("JewlleryShelf")
        cmds.shelfButton(width=100, height=100, image=(imageFilesPath+"jhumkaIcon.png"),
l="JewlleryTool", olb=(0,0,0,0), olc=(0.9,0.9,0.9), command = self.doublun,
imageOverlayLabel="Jewellery Tool")
```

8.15 Appendix O: Setting up a file dialog

```
name,_ = QFileDialog.getOpenFileName(self, directory = "~/", filter = ( "PNG (*.png);;
JPEG(*.jpg, *.jpeg);;
```

8.16 Appendix P: Setting up the controls dock

```
def createDock(name, rebuild, dialog_class):
    isReopened = 0
      if( rebuild ):
```

```
                isReopened = reopenDock(name)
       if not isReopened:
        if not rebuild:
attrNode = cmds.createNode("AttributeNode")
 pm.rename( attrNode, name+"_JhumkaAttributesNode" )
tabname = name + " controls"
        log.debug("created tabname as {}".format(tabname))
        log.debug("setting up dock control for {}".format(tabname))
        main_control = pm.workspaceControl(name, ttc=["AttributeEditor", -1],
label=tabname)
        control_widget = omui.MQtUtil.findControl(name)
        control_wrap = wrapInstance(long(control_widget), QWidget)
        log.debug("created dock control for {}".format(tabname)
        win = dialog_class(control_wrap, name, rebuild)
pm.evalDeferred(lambda       *args:cmds.workspaceControl(main_control, iw = 325, mw = 1,
wp = 'fixed',e=True, rs=True, rt=0))

  log.info("dock {} set up".format(tabname))

        return win
```

8.17 Appendix Q: Loading an Houdini Asset

```
self.asset = cmds.houdiniAsset(loadAsset=[houdiniFilesPath+"Jhumka_tool.hdanc",
"Sop/Jhumka_tool"])
```

8.18 Appendix R: Installing the tool into Maya

```
if not Path(location + "modules/JewleryTool.mod").is_file():
        print("writing module file")
        with open(location + "modules/JewleryTool.mod", "w") as file:
            file.write(f"+ JewleryToolMaya 1.0 {current_dir}\n")
            file.write("MAYA_PLUG_IN_PATH +:= plug-ins\n")
            file.write(f"JEWLERYTOOLPLUGIN_ROOT={current_dir}\n\n")
```