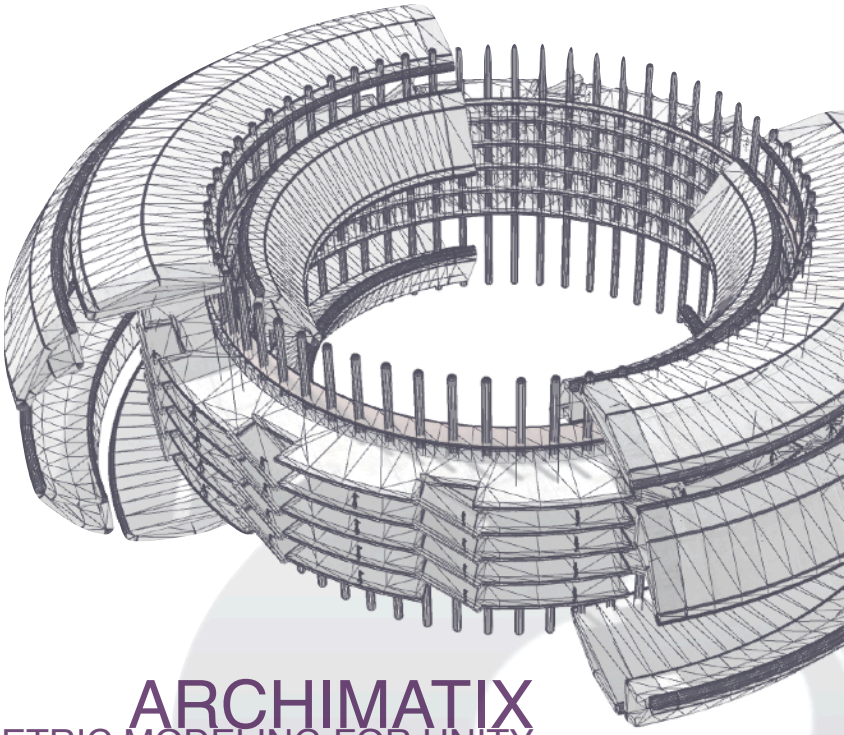
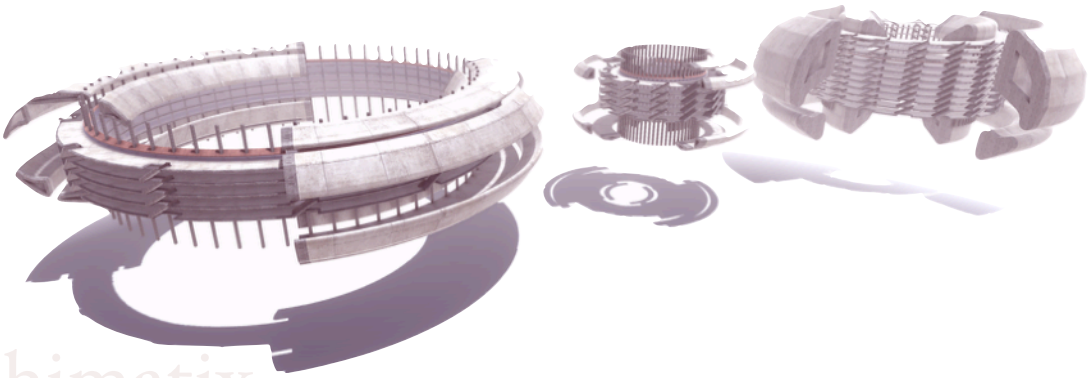


ARCHIMATIX

NODE-BASED PARAMETRIC MODELING FOR UNITY

Roaring Tide Productions

Generating Experiences



Archimatix

Contents

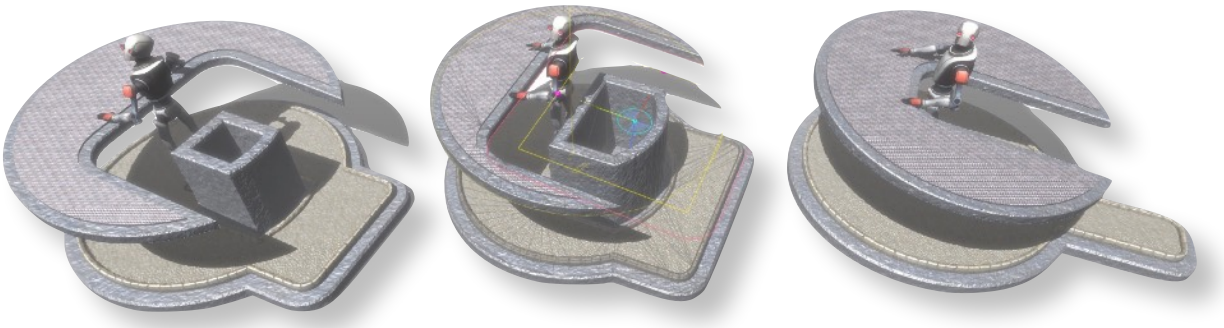
From the Desk of the Developer

Welcome to parametric modeling in Unity with Archimatix!

Archimatix (AX) is a powerful node-based parametric modeler extension for Unity. Archimatix's purpose is to enable artists and game developers to quickly create rich worlds using intuitive "smart models" capable of generating hundreds of unique architectural forms and typologies from simple geometric primitives.

As the developer of AX, I am excited to walk you through your first steps in what should prove to be an exciting in-editor modeling adventure!

If you have never tried parametric modeling, don't worry, it's not hard to pick up. While quite different from brush-based, or topological, modeling, where you directly manipulate vertices and faces, parametric modeling enables you to manipulate relatively few handles or gizmos that alter the entire mesh in smart ways. Archimatix features non-destructive mesh creation, allowing you to produce many variations of a form quickly, either as an iterative design process, or as a way to stamp out multiple versions of the model to create a rich and varied environment.



These three versions of Robot Kyle's reception deck are generated from the same parametric model. The variations are created by adjusting the handles visible in the center image.

As the result of several years of development and testing, Archimatix is a full-featured extension to Unity. I hope you find this User Guide to be a fun and useful introduction to parametric modeling in Unity with Archimatix!

Rory O'Neill, Ph.D.

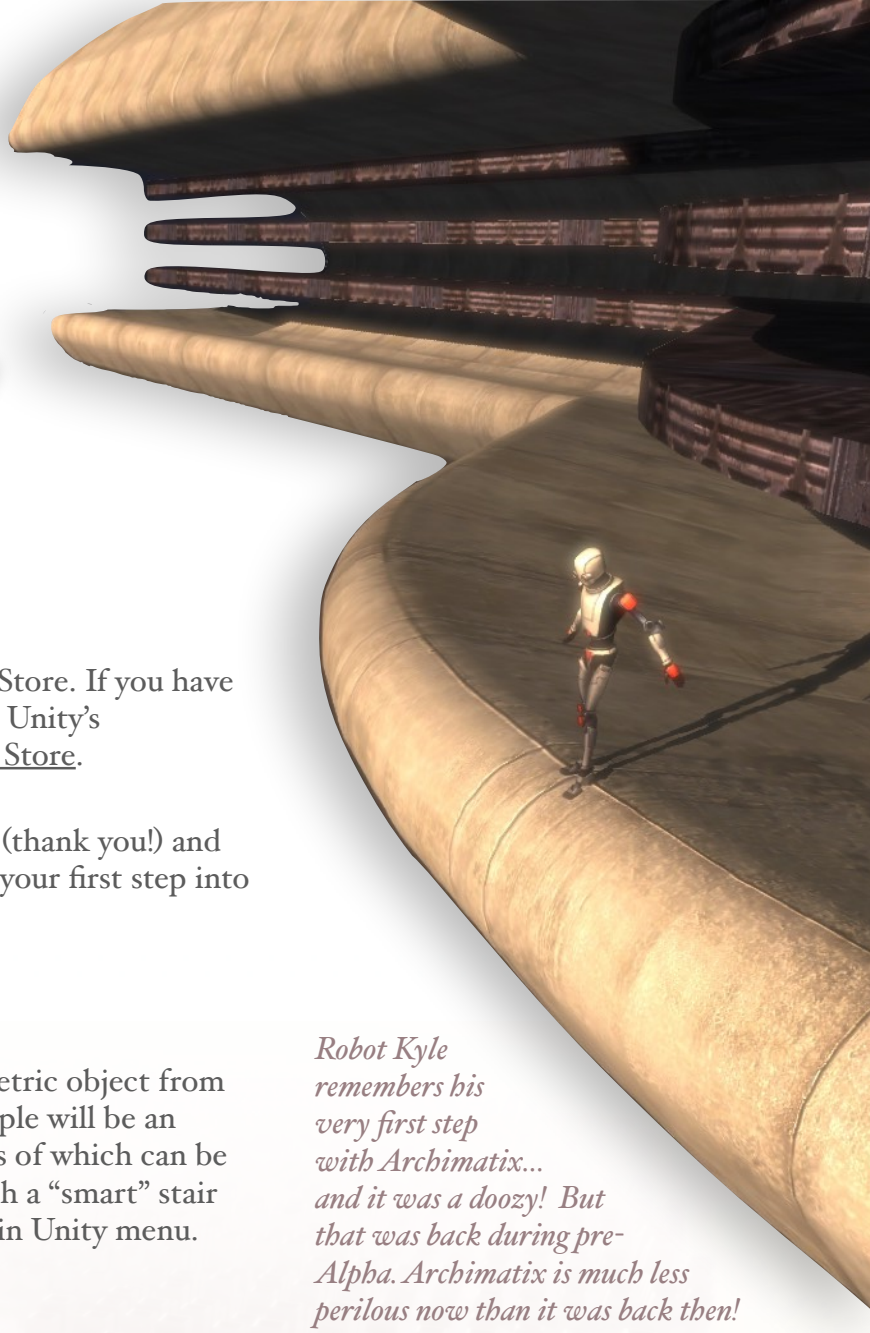
1. First Steps

Archimatix is available in the Unity Asset Store. If you have not installed Archimatix yet, please follow Unity's instructions for Importing from the Asset Store.

If you have already successfully purchased (thank you!) and installed Archimatix, then you have taken your first step into parametric modeling in Unity!

Congratulations! 🎉

For the next step, let's instantiate a parametric object from the Archimatix 3D Library. Our first example will be an interactive staircase?, the basic dimensions of which can be changed in realtime in the editor. To launch a "smart" stair from the Library, choose one from the main Unity menu.



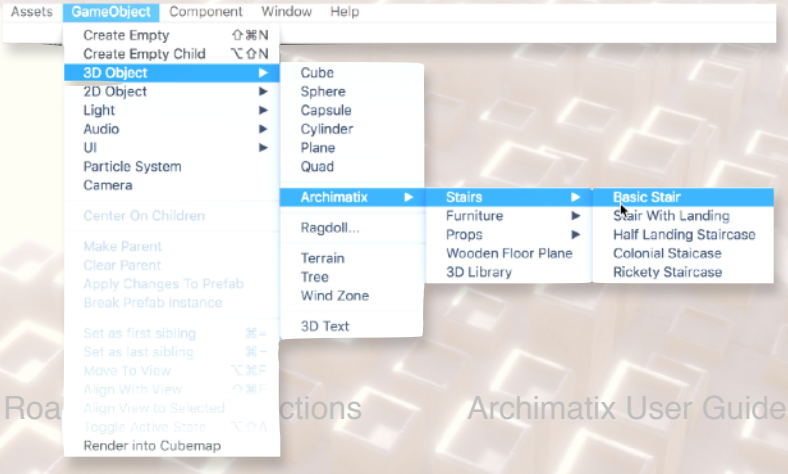
Robot Kyle remembers his very first step with Archimatix... and it was a doozy! But that was back during pre-Alpha. Archimatix is much less perilous now than it was back then!

Robot Kyle's Skyplatform, 5 nodes.

City (below), 7 nodes.

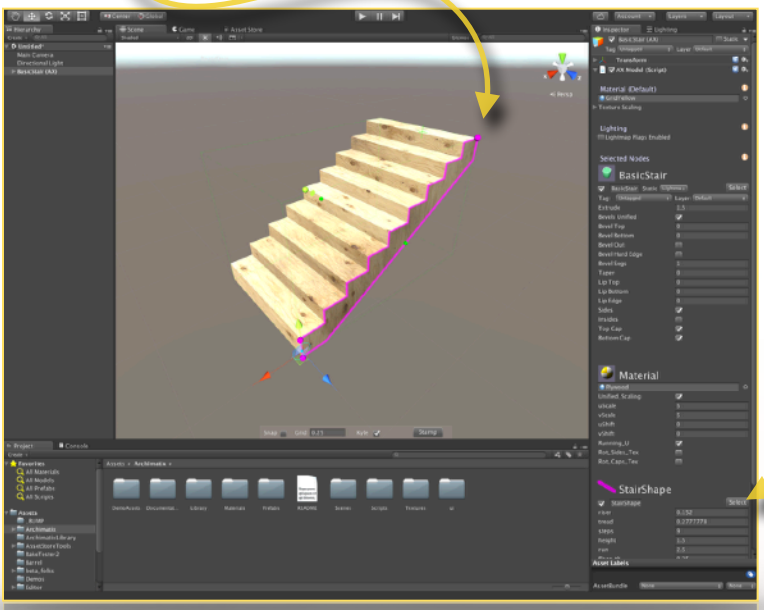
Instantiate a Parametric Object

Step 1.1: From the Unity menu, choose:
GameObject > 3D Object > Archimatix > Stairs > BasicStair



You should now be looking at a basic stair with a plywood material in your scene. You can frame it using the F-key. Since it is selected, it should also be displaying its relevant control handles.

Step 1.2: Click on the point handle at the top of the magenta stair *Shape* and begin to drag it. In the Inspector, you will notice the relevant parameters changing.



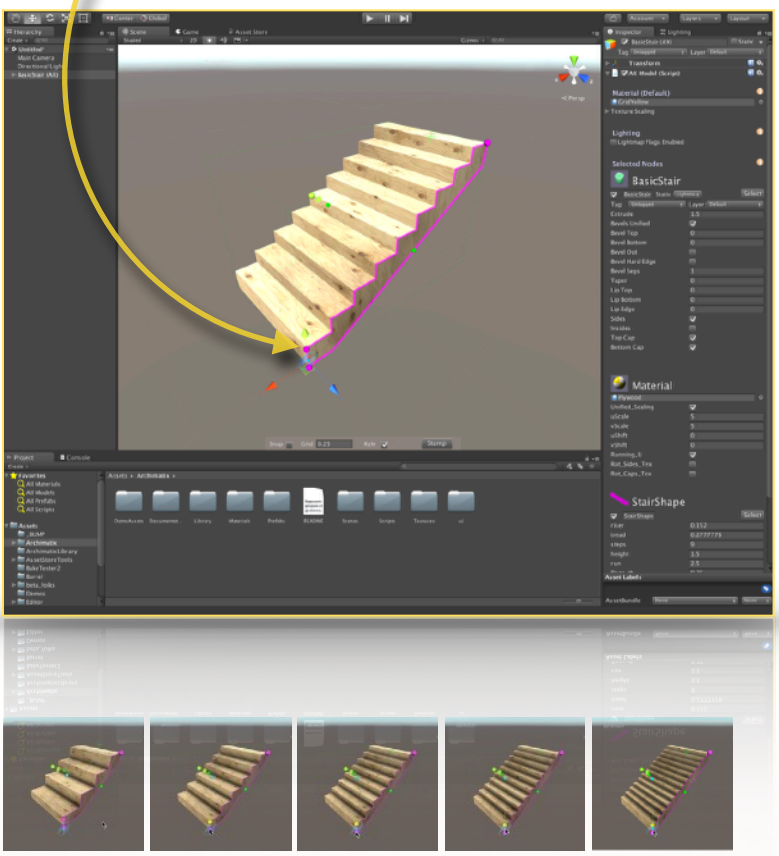
As you drag the handle around you should be able to create the following variations:

This handle is modifying two parameters at once: the stair length, or *run*, and the stair *height*. The logic of the parametric object dictates what other parameters should change in response to your modifications. For example, altering the run of the stairs re-calculates the tread size, making the steps deeper.

You may have noticed that this run-height handle modifies the stair as fast as you can drag. In the development of Archimatix, a key goal was to ensure that the modification of parameters is realtime within the editor.

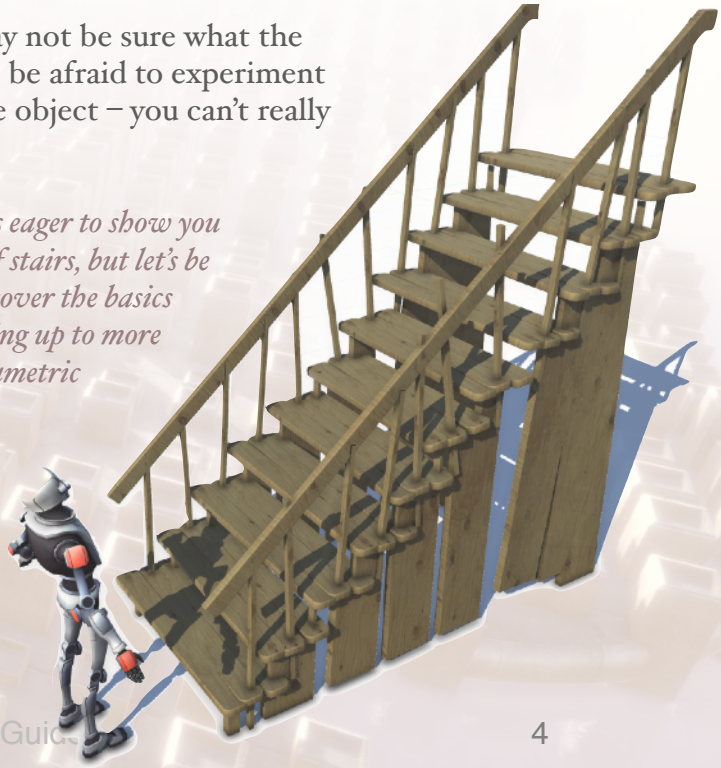
Let's go ahead and modify another parameter of the stair.

Step 1.3: Click on the point handle at the top of the lowest step of the magenta stair *Shape* and begin to drag it. Notice that this handle controls the *riser* height of the steps, with the number of steps adjusting to maintain the overall *height* value for the stair. Try to replicate the variations shown below.



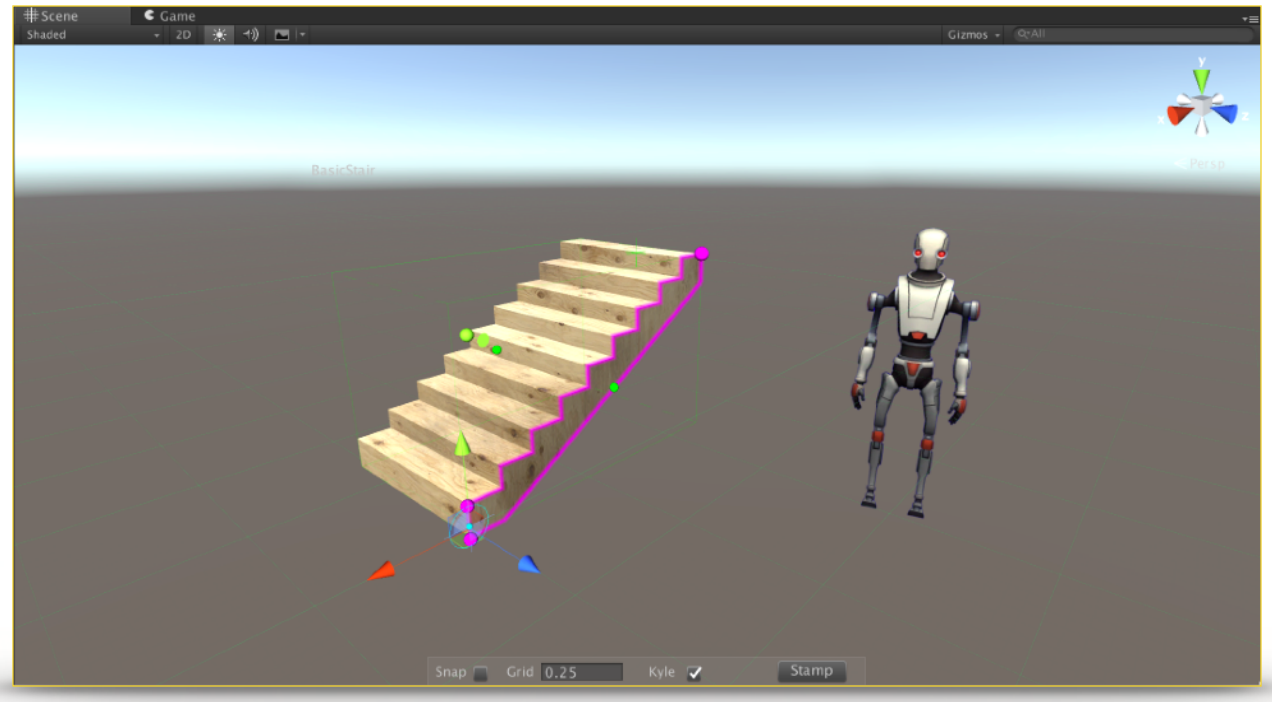
When first encountering new Library items, you may not be sure what the displayed handles are supposed to control, but don't be afraid to experiment with them to explore the parametric behavior of the object – you can't really break them, after all; and if you do, there's always undo!

Robot Kyle is eager to show you other types of stairs, but let's be patient and cover the basics before climbing up to more complex parametric models!

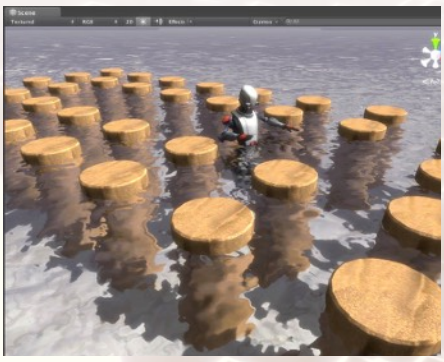


The Talented Mr. Kyle

You may have notice that, when you created the *BasicStair*, another object appeared in the scene: the one and only Robot Kyle!

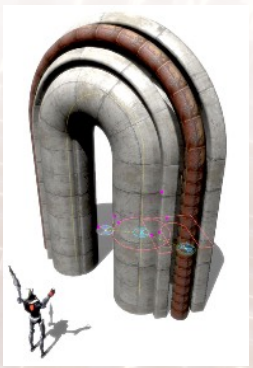


This is actually a flat stand-in for the real Robot Kyle – a free asset from Unity Technologies. While you can download Robot Kyle, or any other character, to help give your design space a sense of scale, we include this 2D billboard as a lightweight alternative, while paying homage to the character that has served as an Archimatix mascot throughout the development process. You can follow the trials and tribulations of Robot Kyle on the Archimatix community thread on the Unity Forums.

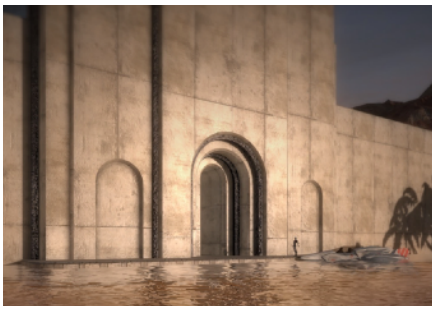


It was at that very moment Robot Kyle began to wonder if he was indeed waterproof...

Robot Kyle speaks in defense of the last human.



Every now and then, Robot Kyle stands in stupefied awe of his own creations!



With no welcoming party at the gate to greet him, Kyle's robot senses were on high alert.

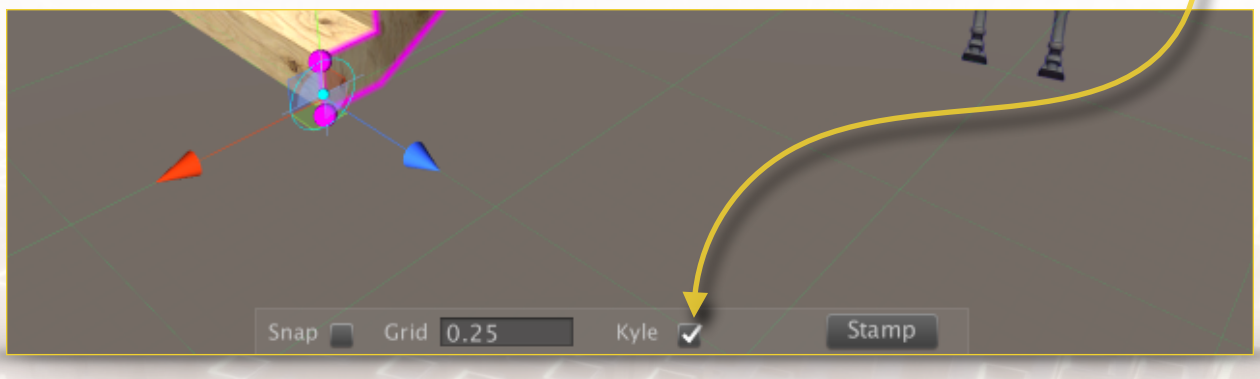
Robot Kyle hesitates before the time portal. He remembers last time!

Hold on, Kyle. This thing isn't even at cruising speed yet!

The Robot Kyle billboard, which is included with Archimatix as a Prefab, is automatically instantiated in the scene when you create a new model. Being a standard GameObject, the Kyle billboard can be translated as such. The billboard rotates automatically with the editor camera.

Creating parametric objects while maintaining a sense of scale is important when working with parametric modeling, since it is generally better to size your objects using their parameters rather than using the scale transforms.

If you would prefer to work without Kyle's omnipresence, you can click on the toggle at the bottom of the Scene View to hide him. Or you can simply delete him from the scene.



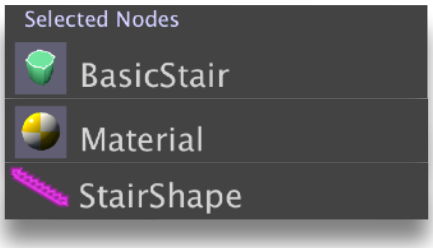
The Scene View footer menu appears whenever an Archimatix object is selected. The footer menu also includes grid snapping controls and the Stamp button.

Inspecting the Inspector

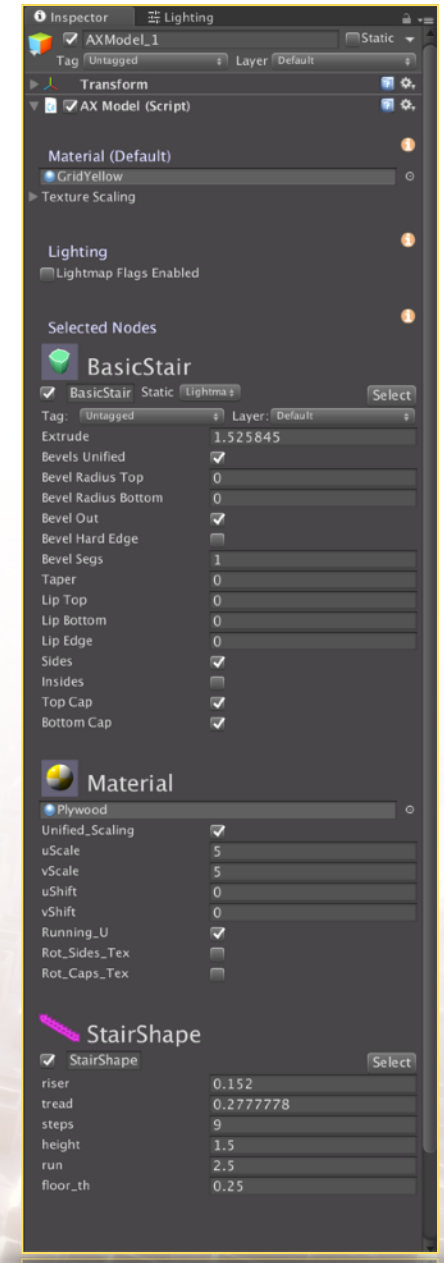
Not every parameter is represented by a handle in your scene. To see the full set of parameters available for a selected object, you can refer to the Inspector.

From the Inspector, we see the parameters for the *BasicStair* model are listed under certain headings such as *BasicStair, Material* and *StairShape*. This is because a parametric object in Archimatix is described by any number of nodes in a node graph.

When you select a parametric object in the scene, the Inspector displays the node that represents that object, as well as all of the "upstream" nodes" that feed into it.



These headings represent nodes in the Archimatix node graph.

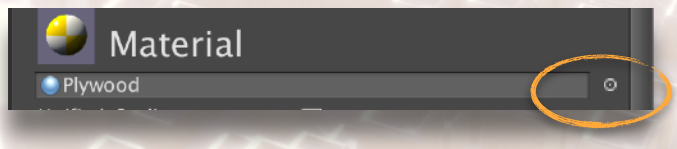


Step 1.4: With the *BasicStair* still selected, take a look at the Inspector and begin to manipulate some of its parameters.

The parameters we modified with the Scene View handles, *run*, *height* and *riser*, are under the heading of *StairShape*.

We can use the Inspector to select a material for the *BasicStair*. The default material for the model is listed near the top of the Inspector. This default material will be applied to any node that does not have its own material defined. In this case, the *BasicStair* object does have its own material node, listed below it in the Inspector.

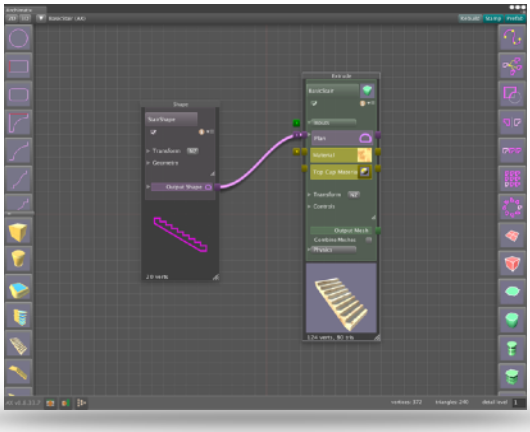
Step 1.5: Drag a material from your Project Window to the material ObjectField below the *BasicStair* section of the Inspector. Alternatively, you can click on the circle to the right of the ObjectField to bring up the Material chooser.



Note: Archimatix does not yet support dragging and dropping a material directly onto the object in the Scene View window, but this is an upcoming feature.

Node Graph Editor (Sneak Peek)

What makes the *BasicStair* so smart? Underneath the hood is a full-featured node graph that visually encodes the logic of how the model is generated. The image of the Node Graph Editor to the right reveals that the parametric object *BasicStair* is actually composed using only two nodes.



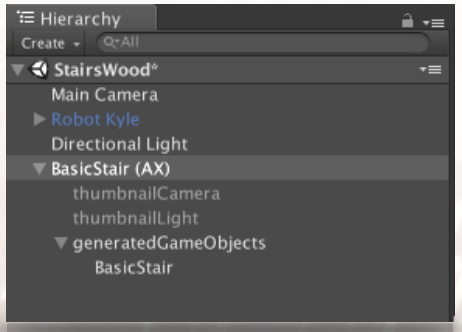
The BasicStair *is actually composed of two nodes: a parametric* Shape *named* StairShape *and an* Extrude *node named* BasicStair. *The output of the* Shape *is fed into the* Plan *input of the* Extrude *node.*

Node base classes can be *Shapes*, *Meshers*, *Repeaters*, *Tools* and other bits of functionality to help you describe the logic of a model. In the *BasicStair* parametric object, the *StairShape* (instantiated from the 2D Library) has its output fed into the "Plan Shape" input of an *Extrude* node. The *Extrude* node then takes the *StairShape* data and, based on the *Extrude* node's parameter values, generates a Unity mesh. This mesh output can then be fed into other nodes – such as a *Repeater* node – to duplicate it according to the *Repeater*'s parameters.

Not all that scary, right? We will revisit the node graph in Chapter 6, but for now, let's see what else is happening in the scene.

Hierarchy Window

You may have already noticed that a GameObject has appeared in the Hierarchy window named "BasicStair (AX)." This is the root GameObject that instantiates an AXModel Component, the manager of all Archimatix smart objects within that model. While you may have multiple AXModel GameObjects in a scene, it is quite common to have very few or even only one. When instantiating a new Library item, the default behavior is to add these to your scene as node sets within the existing AXModel's graph.



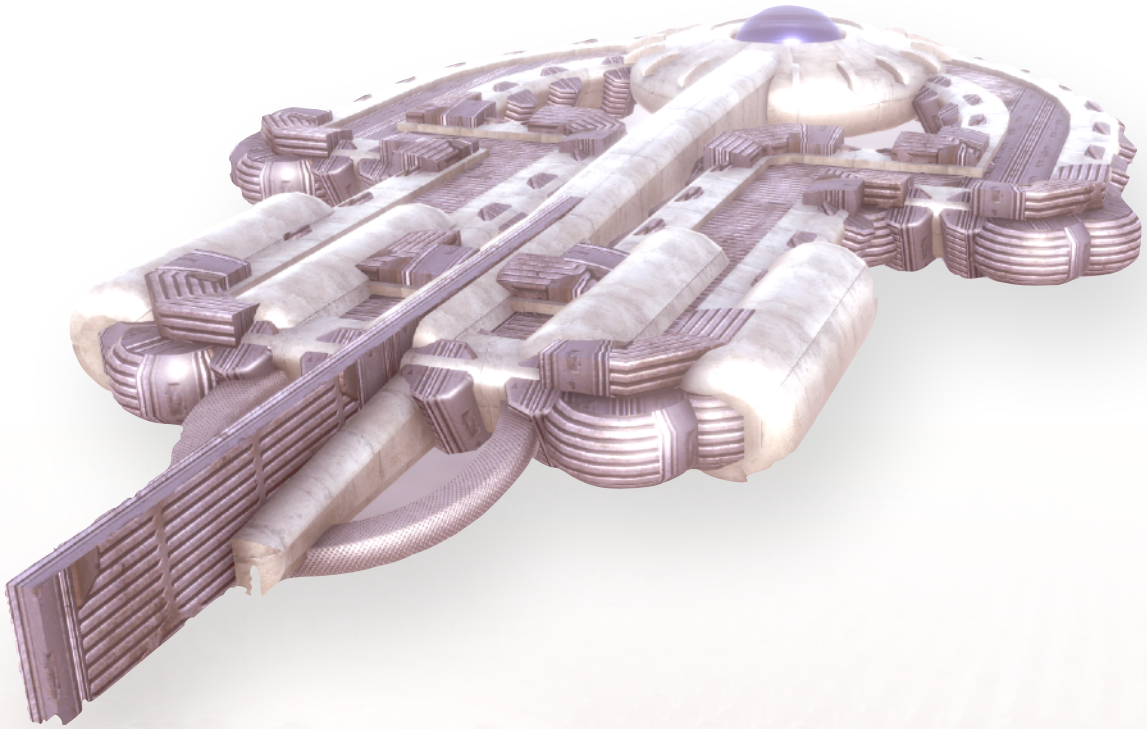
Hierarchy view of a typical AXModel GameObject. Generally, you will not be using the Hierarchy view to select the children of this GameObject, but instead you will be working in other views, such as the Scene View, the Inspector and the Node Graph Editor.

Two children of the AXModel GameObject, the *thumbnailCamera* and *thumbnailLight,* are inactive, being only used internally by the AXModel component to render thumbnail images. generatedGameObjects is a temporary container to hold GameObjects that are generated as the model and its parametric objects are manipulated.

You cannot actually select a child object of the AXModel

root in the Hierarchy, because with parametric modeling, we do not operate on GameObjects directly, but rather on nodes that generate GameObjects. Smart objects in your scene behave like separate GameObjects, but are, in fact, ParametricObject nodes within the model.

That's enough basic anatomy for now! Since you are probably itching to manipulate some more parametric objects, let's call up a more complex example and explore another way to instantiate Library objects.



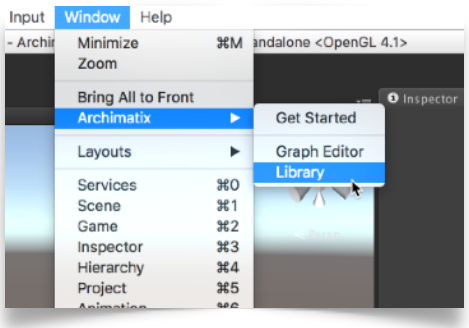
2. The Archimatix Library

Parametric modeling is made easier with the help of the Archimatix Library. With shelf space for a vast array of parametric props available for you to check out (and check in!), both 2D and 3D parametric objects archived in the Library are searchable.

While some Library items can be instantiated from the Unity menus, visiting the Archimatix Library via its own window is a great way to see the full holdings of the Library at any given time. Archimatix ships with some example Library items to help get you started, and your Library will grow over time as you add your own parametric props and share props with friends, and as packs of parametric props become available from the Asset Store.

To get a better sense of how this works, let's open a Library window.

Step 2.1: Create a new scene and then open the Library window via the main Unity menu: **Window > Archimatix > Library**



When the Library window opens, feel free to dock it

within the Unity Editor interface.

The Library Window

The Library window displays a grid of the items archived in the Library. Each item may be thought of as a Parametric Prop, a smart object that you can add to your model, complete with its own Scene View handles as parameters.

You can search and filter the items in the Library by keywords and other metadata tags associated with each item.

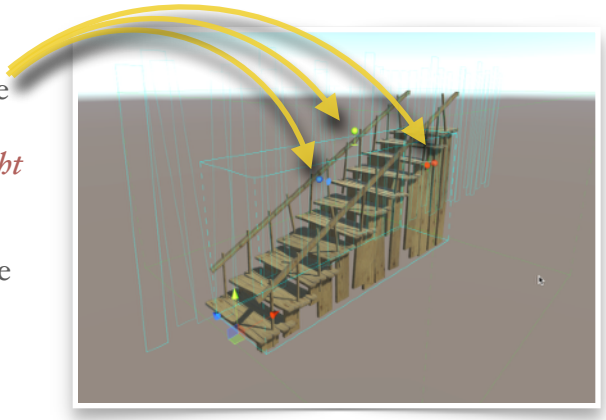
To try the Library out, let's appease Robot Kyle by checking out the *RicketyStaircase* item from the Library, adding it to our scene.

Step 2.2: At top of the Library window, click in the search field and type "stair."

Step 2.3: In the filtered results now displayed, locate the *RicketyStaircase* and click on it.

The *RicketyStaircase* should now be selected in your scene.

Unlike the *BasicStair*, which had only one *Mesher* node (the *Extrude*), this *RicketyStaircase* is more complex, composed of several *Shapes*, *Mesher* and *Repeater* nodes. These parts are all brought together in a *Grouper* node.

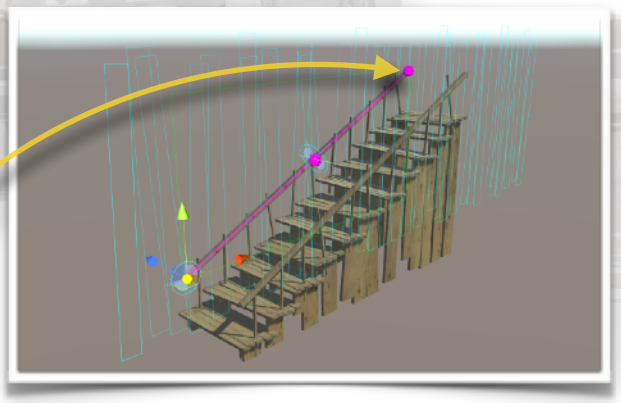


Step 2.4: Click and drag the red, green and blue handles visible around the *RicketyStaircase* to control its *run*, *height* and *width* parameters.

You should be creating different variations of the *RicketyStaircase*. Notice that the step treads, rail posts and side boards are being "jittered" with Perlin noise to give that rickety effect.

The boards are created by a repeating 2D *Rectangle* with Jitter, and then extruded.

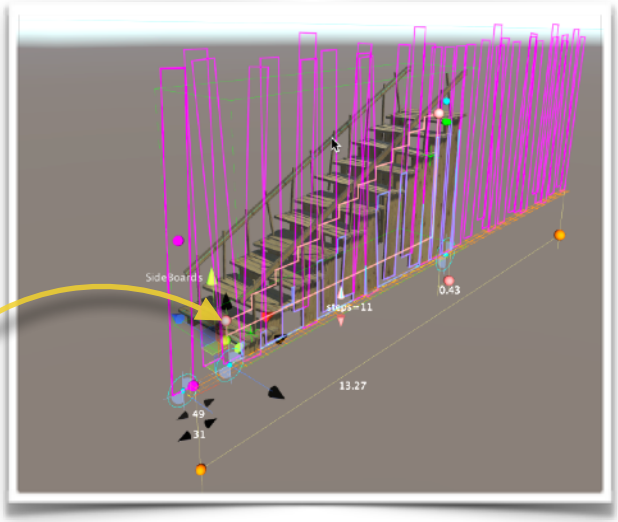
You can find more handles to control the *RicketyStaircase* by clicking on various parts of the model.



Step 2.5: Click on one of the stair rails. Click and drag the point handle at the top end of the rail to modify the *run* and *height* of the staircase.

It is fun to manipulate multiple parameters with one handle. Like with the *BasicStair*, this handle at the top of the rail is modifying both *run* and *height*.

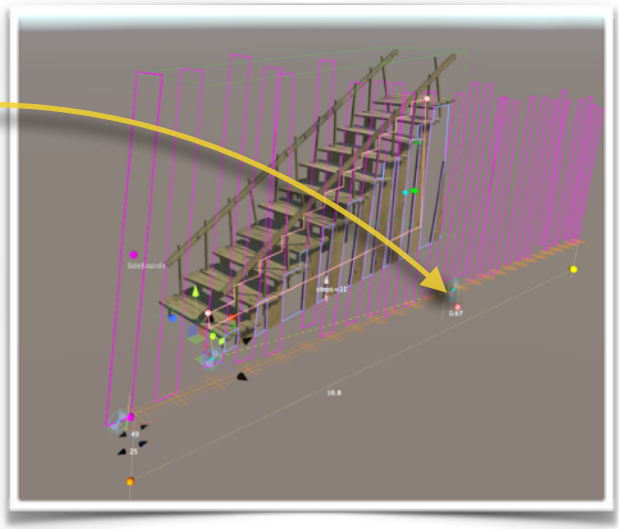
Step 2.6: Click on the sideboards of the staircase.



The handles that appear when you click on the sideboards are numerous because they are made of a number of subnodes. Note the stair profile *Shape* with top point handles at the top and bottom of the stair.

Step 2.7: Click on the control point on the first (lowest) step and drag to alter the *rise* parameter (and the number of steps).

Let's alter the spacing of the sideboards themselves.



Step 2.8: Drag the cell size handle of the *LinearRepeater2D* that spaces the sideboard *Rectangle Shape*. Also, try dragging the cyan circle to rotate the repeater.

You should see the boards changing in real time as you adjust the repeater handles. This *RicketyStaircase* prop is driven by two *Repeaters*, the *LinearRepeater2D* to space the sideboard *Shape* and a *StepRepeater* to distribute the tread boards. As you may have guessed, the node graph for the *RicketyStaircase* is a bit more elaborate than that for the *BasicStair*.

Before leaving the Library, let's check out one more feature of working with Library items: creating stamped or "frozen" versions of models.

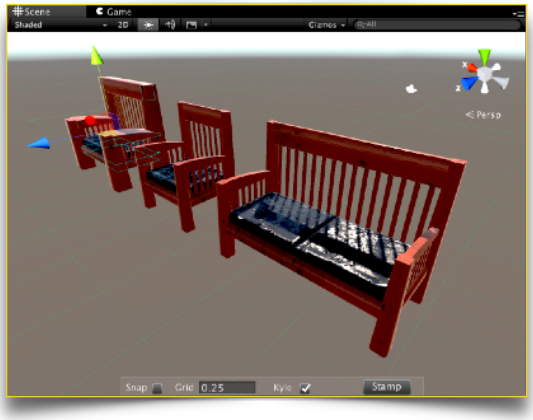
Step 2.11: Create a new scene. Find the *ArtsAndCraftsChair* in the Library and click on it to add it to your model.

Step 2.12: Modify the chair by dragging its handles.

Step 2.13: Click on the "Stamp" button in the footer control panel at the bottom of the Scene View.

At this point you have two chairs, one that is a "frozen" copy and the original parametric object, which remains selected. The frozen copy no longer needs Archimatix.

Step 2.14: Move the parametric object away using the Transform handle. Click anywhere on the object and manipulate the handles that are displayed.



Archimatix features "cycle selecting," whereby repeatedly clicking on an object selects "downstream" nodes until eventually the model itself is selected. If you would like to select the entire parametric object at anytime, you can do so in the Hierarchy window.

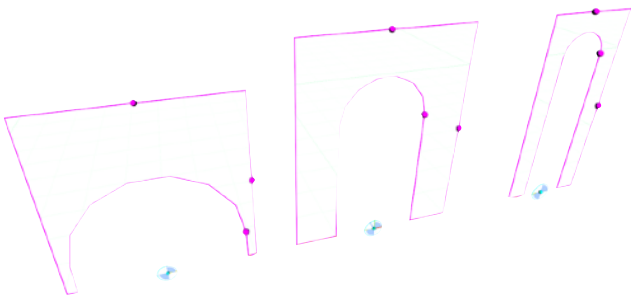
To see stamping in action, please try this tutorial.

The Last Human thinks that Robot Kyle has him under surveillance, when, in fact, Kyle is contemplating what merged Shapes *define the cross section of the skyplatform above.*

3. Getting into Shapes

Although Archimatix is an amazing tool for generating 3D meshes, one of its key strengths is its ability to manipulate 2D *Shapes*. With rich *Shape* modification features, including *thicken*, *offset*, *rounding* and boolean *merge*, *Shapes* provide a powerful basis for generating 3D forms.

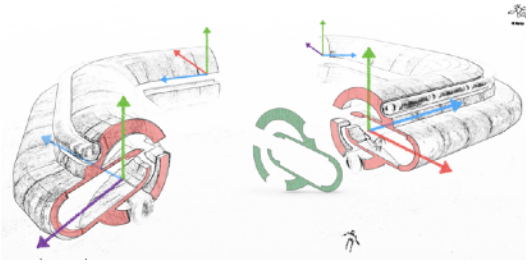
In Archimatix, *Shapes* too are parametric objects. As we already saw with the stair *Shape*, when you select a *Shape*, or an object that uses a *Shape*, you will see *Shape* handles.



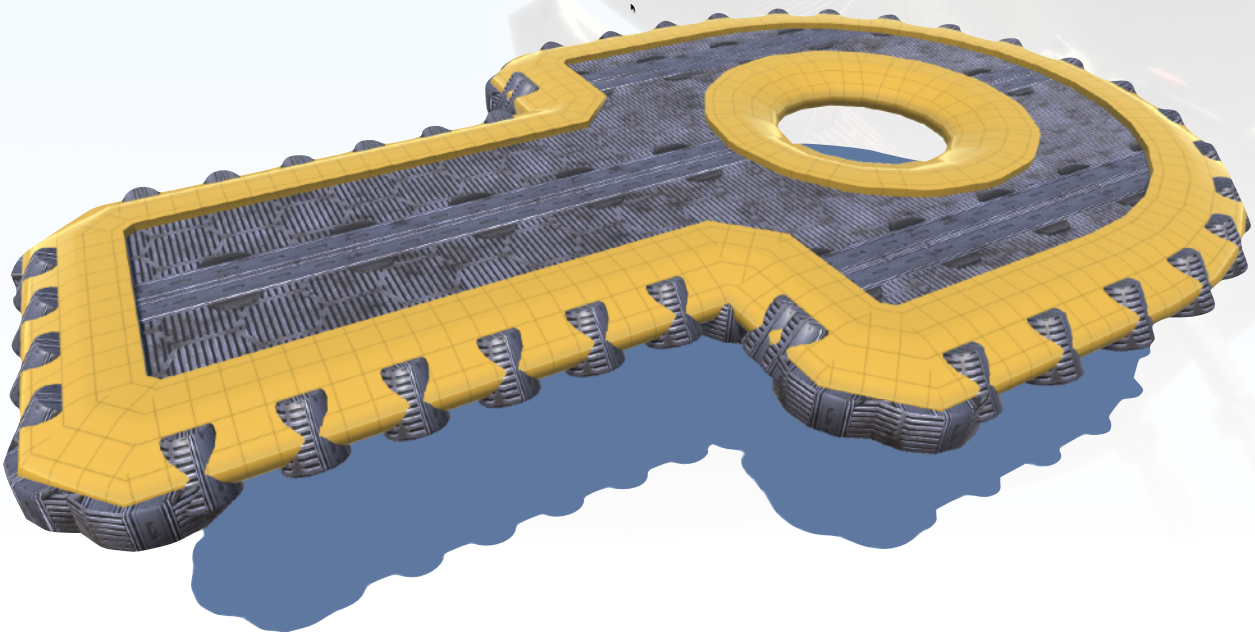
These three arch variants are really the same ArchShape checked out from the Library. They have been created by playing with the point handles.

Now let's explore the power of *Shapes*.

Shapers are nodes that operate on Shapes.



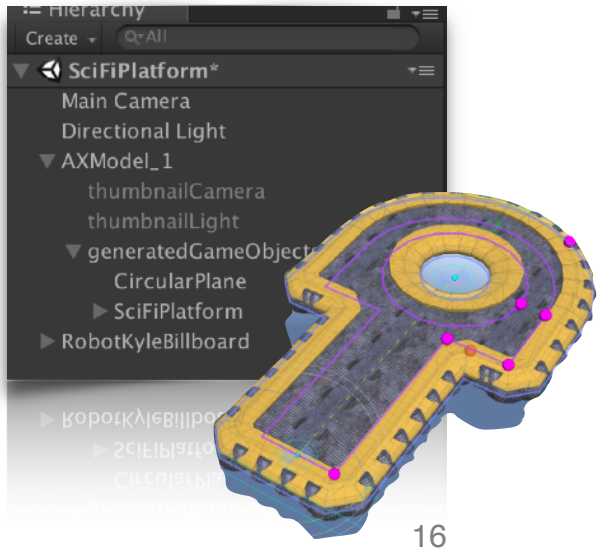
The skyplatform above is made with a composite Shape *(the product of a boolean addition and subtraction of several simple* Shapes*) as a section swept around a* Plan Shape.



This time, instead of selecting an object from the Archimatix Library, let's start with a demo scene. Smart Objects' states are saved in the scenes themselves. Loading a scene containing Smart Objects also loads their parametric relationships and values.

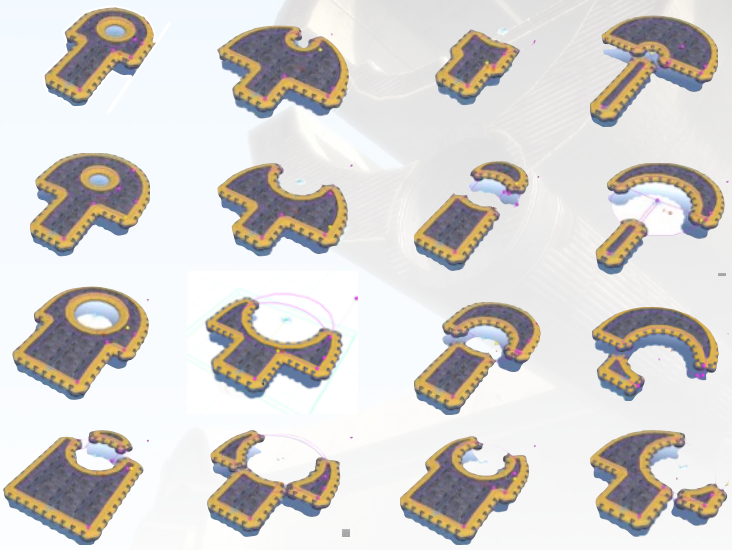
Step 3.1: Using Unity's Project panel, open the scene: **Archimatix/Scenes/SciFiPlatform**

This scene has one Archimatix models in the Hierarchy called AXModel. This is the main GameObject that manages the parametric model in the scene. If you expand the foldout, you will see a *thumbnailCamera* and a *thumbnailLight*, both of which are inactive and are used to render node thumbnails. The generatedGameObjects item holds all of the temporary GameObjects created by Archimatix. You can not select these temporary objects here will select the node that generated them.



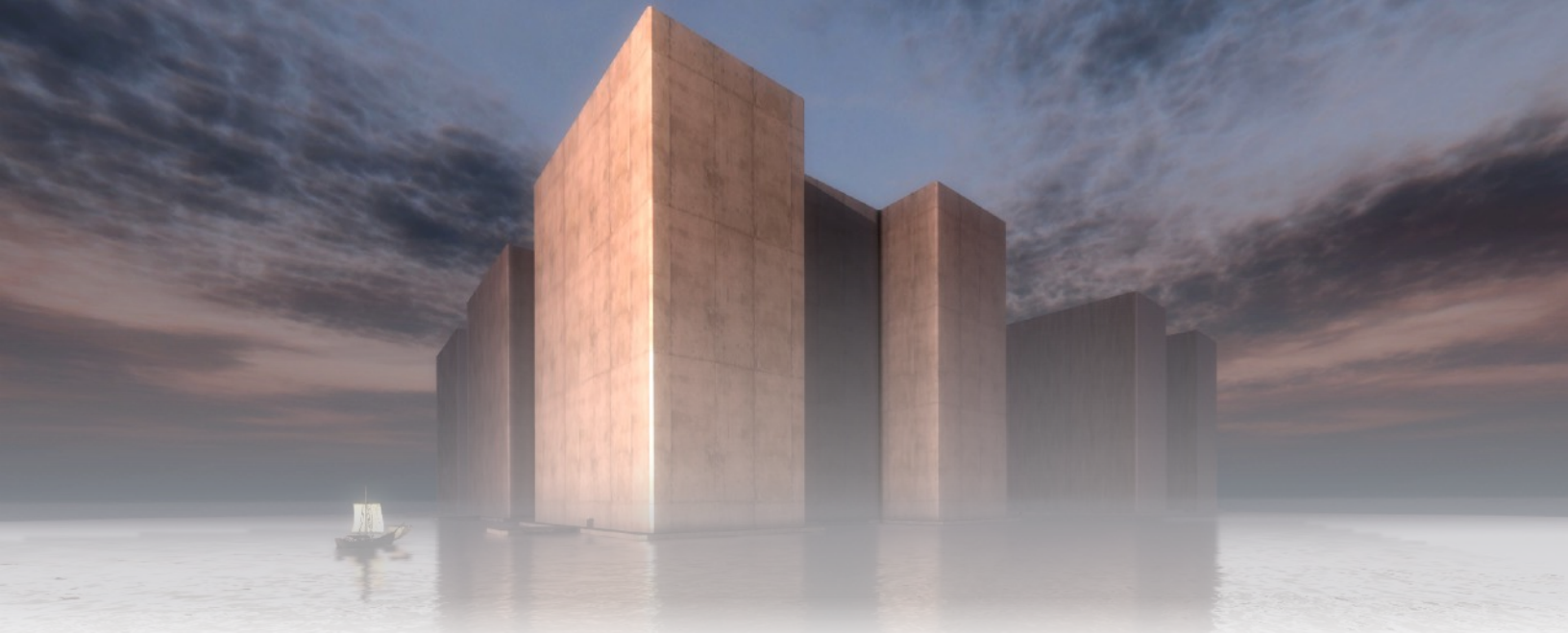
Step 3.2: Select the platform in Scene View to reveal its handles.

Step 3.3: Drag the magenta point handles to create variations of the *SciFiPlatform* like these:



Don't forget that you can grab the *Circle*'s centroid to displace it.

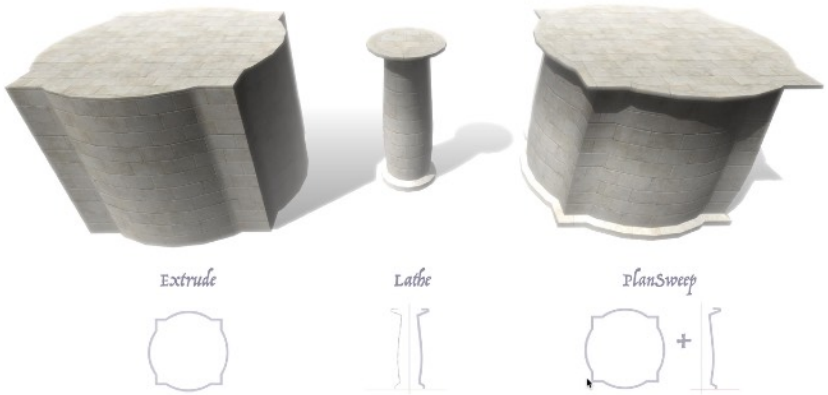
This isn't the first time Robot Kyle has needed to take a leap of faith... and for some reason, he thinks it won't be the last!

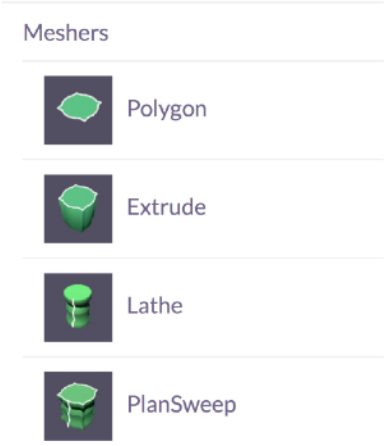


Fighting an overwhelming desire to turn back, Robot Kyle sails cautiously toward the Island of the Robot-With-No-Name. He knows in his mechanical heart that these towering edifices are merely ***I-Shapes*** *fed into an* **Extrude** Mesher*, but their sheer verticality and oppressive mass unnerve him as they pierce the horizon.*

4. Just Meshing Around

Archimatix has four main types of *Meshers* that transform 2D *Shapes* into 3D forms. The *Polygon* simply triangulates a 2D *Shape*, turning it into a flat mesh. The *Extrude Mesher* takes a *Plan Shape* and lofts it to a certain height. The *Lathe* takes a single *Section Shape* and spins it around an axis at a certain *radius*. Finally, the *PlanSweep Mesher* takes a *Section Shape* and sweeps it around a *Plan Shape*.





Meshers *are represented in the node graph by the above nodes.*

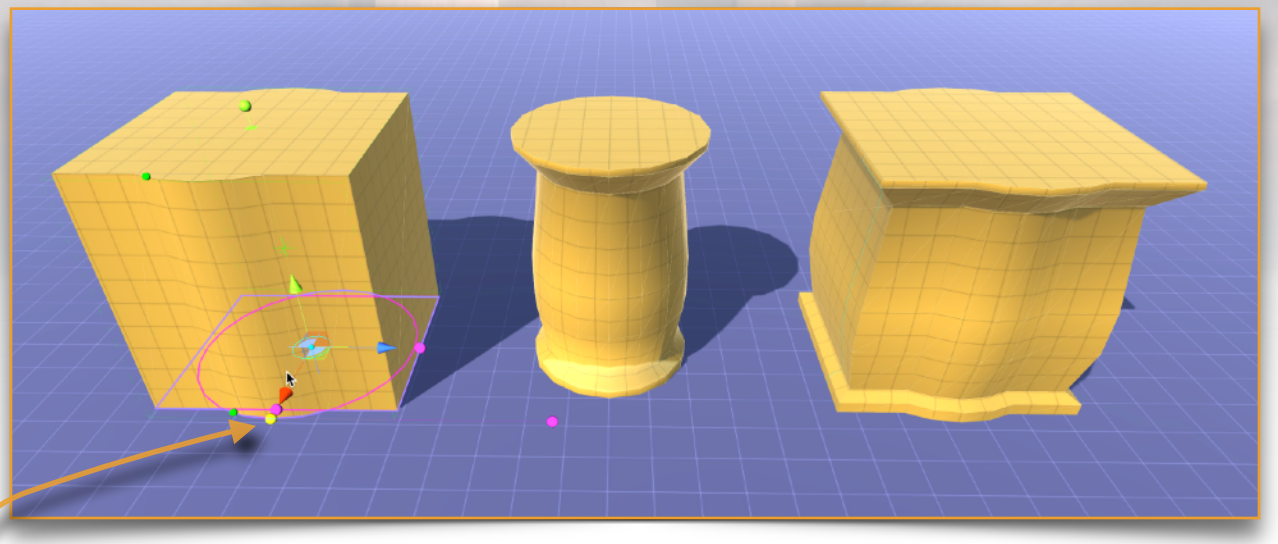
Archimatix *Meshers* use the logic of their geometry to automatically generate uv's, determining whether faces appear smooth or faceted.

You can control how the normals respond to the curvature of the input *Shapes*, as well as define the break angles separately, in *Plan* and *Section*. The image above shows identical columns with different break angle settings. The first column has high break angle values; the middle two have high break angle values on only one axis; and the last column has low break angle values on all axes. You can also scale and shift textures that you have applied to the sides and/or caps for these meshes.

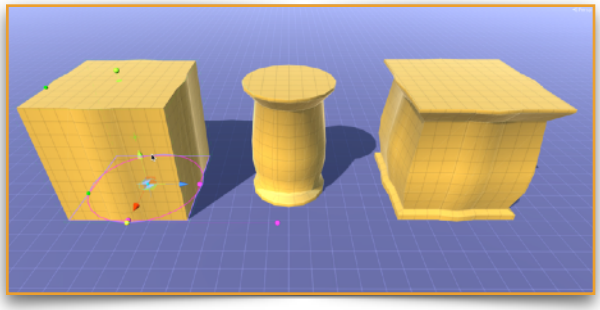
These controls are only accessible from the Node Graph Editor, which will be introduced shortly!

But first, let's play with these *Meshers* to see how they behave.

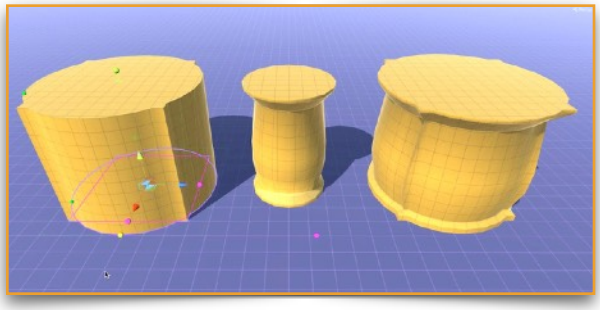
Step 4.1: Open the scene **Archimatix/Demo Scenes/MeshingAround**.



Step 4.2: Click on the *Extrude Mesher* and manipulate the point handle for the *radius* of the *Circle*.

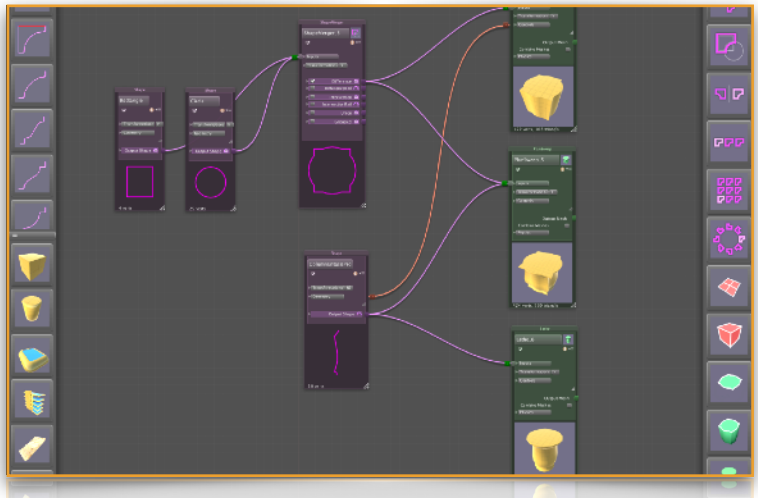


When the Circle *is smaller, the* Square *is dominant.*



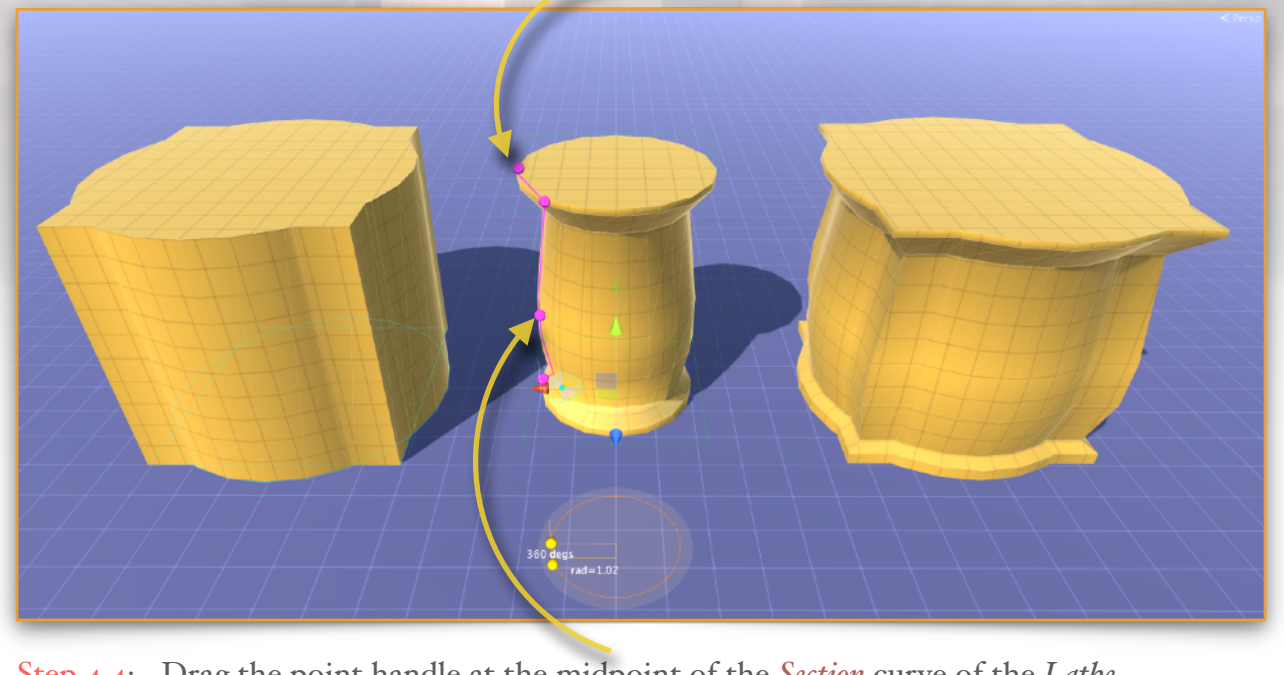
With a bigger radius, the Circle *is dominant.*

As you vary the *radius* of the *Circle*, the *PlanSweep* object changes as well. This is because the same *Plan Shape* is being fed into both the *Extrude* and *PlanSweep Meshers*.

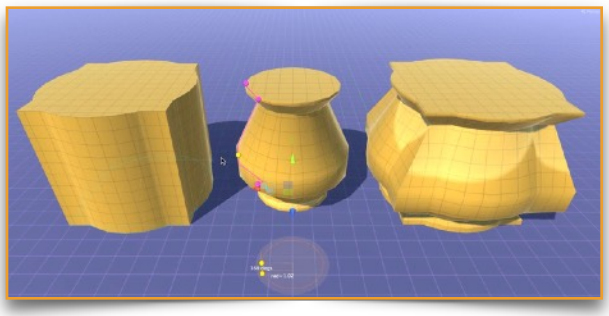


A sneak peek at the node graph for this model shows that, indeed, the Plan Shape *is shared by the* Extrude *and the* PlanSweep, *while the* Section Shape *is shared by the* PlanSweep *and the* Lathe. *The red connector is linking the height parameters of the* Section Shape *and the* Extrude *Height.*

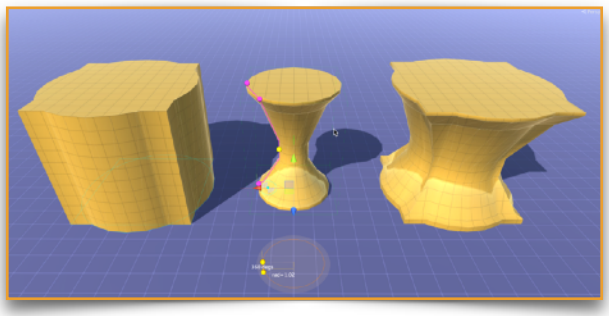
Step 4.3: Select the *Lathe* object and modify the point handle at the top of the *Section*.



Step 4.4: Drag the point handle at the midpoint of the *Section* curve of the *Lathe*.



The midpoint of the Section *is drawn outward.*



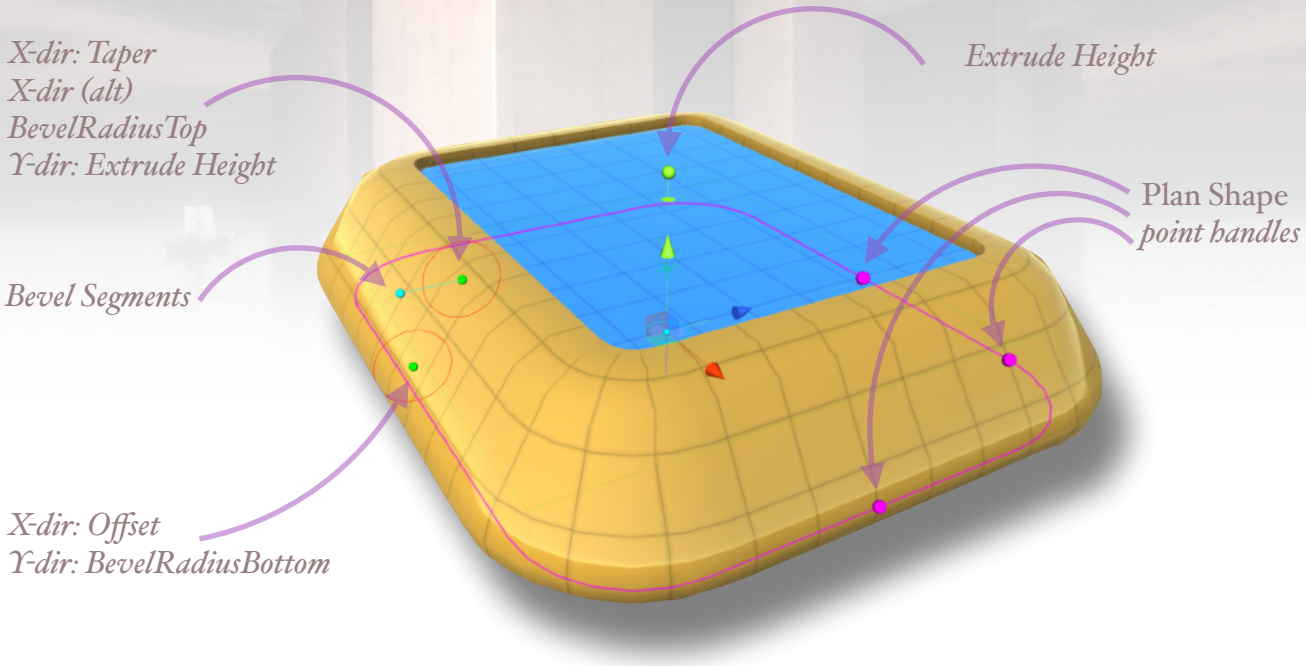
The midpoint of the Section *is drawn inward.*

The heights of all three objects adjust together. This is because the *Extrude Mesher*'s *Height* parameter is "related" to the *height* parameter of the *Section Shape*. The Relation between them (expressed by the red connector line in the Node Graph Editor image on the preceding page) has "equals" expressions. We'll learn more about Relations and their expressions in Chapter 8.

It's fun to play with the *Section* curves of the *Lathe* and the *PlanSweep* objects, but what about poor old *Extrude* – will it always be consigned to sheer walls? As it turns out, *Extrude* has some extra handles up its sleeve for tapering and beveling.

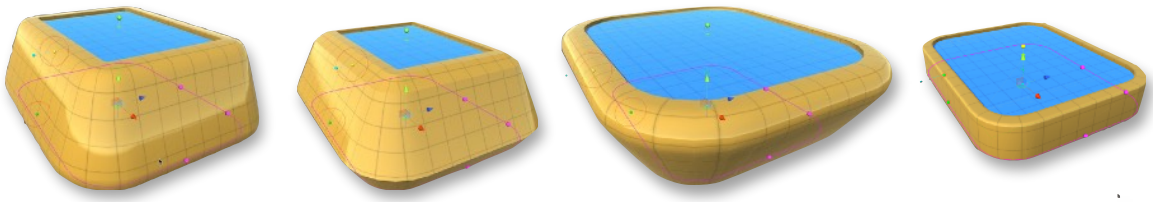
Let's take a quick look at these *Extrude* handles.

Step 4.5: Create a new scene and instantiate the ***BevelBox*** from the Library.

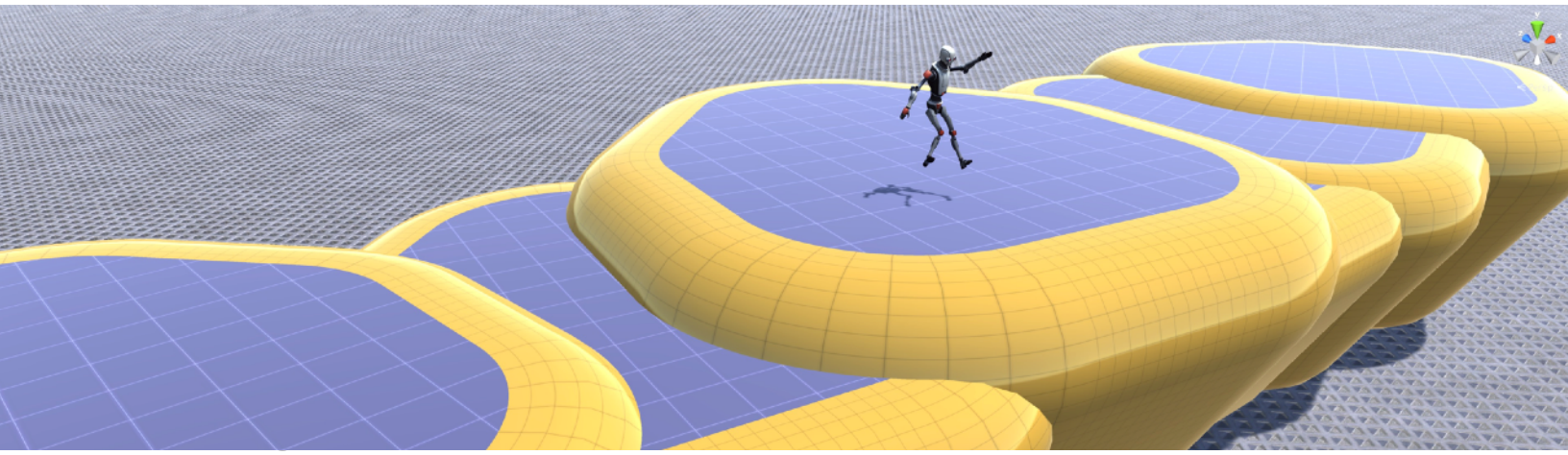


Although it may look like a more complex object, the *BevelBox* Library item is really just an *Extrude Mesher* with the *Bevel Top* radius and *Taper* adjusted. The depression in the Top Cap is set from the *Lip Top* parameter in conjunction with the *Lip Edge* parameter found in the Inspector.

Step 4.6: Vary the handles to make variations like these:



Robot Kyle is a little disappointed that these models are not as bouncy as they look!



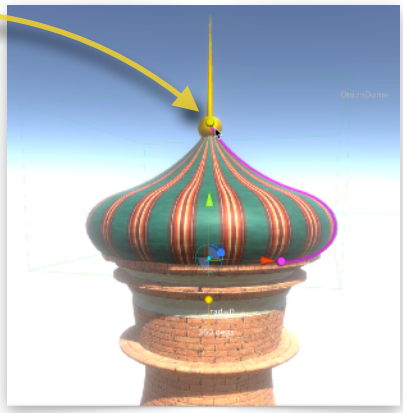
Taking Lathe for a Spin!

Circular forms are common in nature and in architecture. The *Lathe Mesher* is useful for creating round objects, such as barrels, tables and towers. It rotates a single *Shape* around a center axis at some radius.

A great example of a lathed object is a tower with a dome.

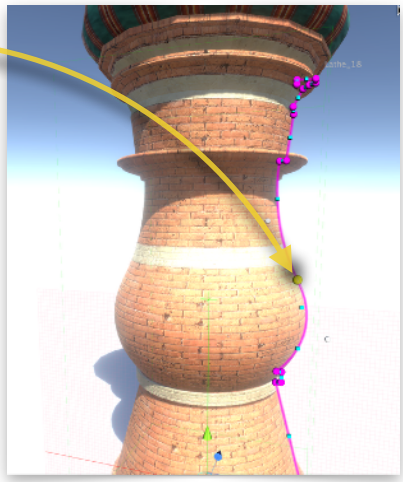
Step 4.7: Open the scene ***TowerWithOnionDome*** from the folder **Archimatix/Scenes**.

Step 4.8: Click on the dome of the tower and drag the top handle of the section curve to flatten the dome a bit.



This curve is a parametric *Shape* with a certain logic, namely, it offers only two parameter handles (at either end of the *Shape*) and makes assumptions about how the curve between the two points should be generated. Although this parametric curve is not as general or flexible as a *FreeCurve*, it is likely to always produce results that look like an "onion dome," such as those found in Eastern Europe or the Near East.

Step 4.9: Click on the middle of the brick shaft of the tower and drag the center handle.

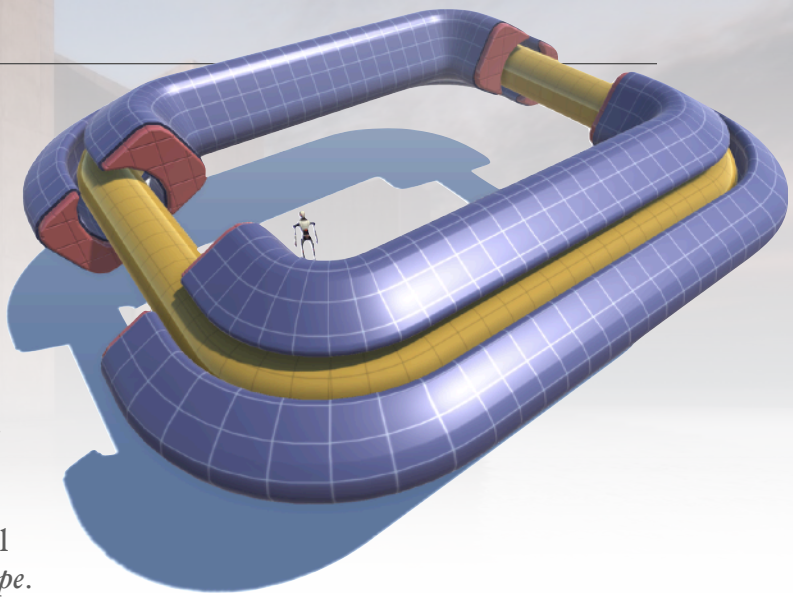


The center handle is a *Bezier* handle, so you can drag its center point or the points at the end of the tangent handles that extend from the central point.

This *Shape* is a *FreeCurve* that allows you to manipulate any vertex handle in a free-form way. While it is the most general type of *Shape* that Archimatix offers, there are no assumptions about the logic of the final outcome, meaning that the design is completely up to you. Over time, you will develop a sense of when to use a parametric *Shape* or a *FreeCurve Shape*.

The *Lathe* is a common and powerful tool for generating standard Unity meshes from *Shapes*. A more general and powerful *Mesher* is the *PlanSweep Mesher*.

Meshes with Sweeping Changes

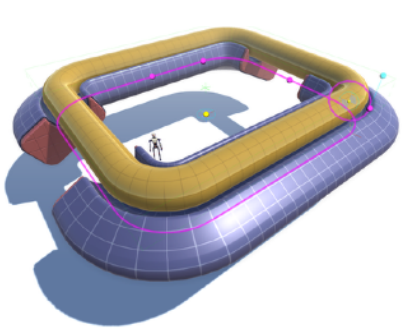


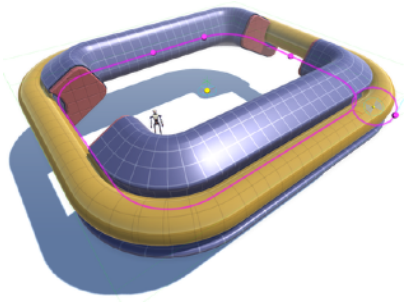
The *PlanSweep Mesher* is perhaps one of Archimatix's greatest features. It sweeps a *Section Shape* around a *Plan Shape*. Combined with *ShapeMerger*, the results can be particularly exciting!

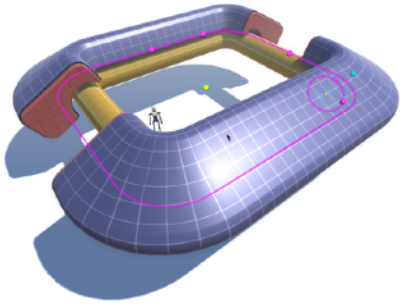
Step 4.10: Open the scene ***PlanSweepTube*** from the folder **Archimatix/Scenes**.

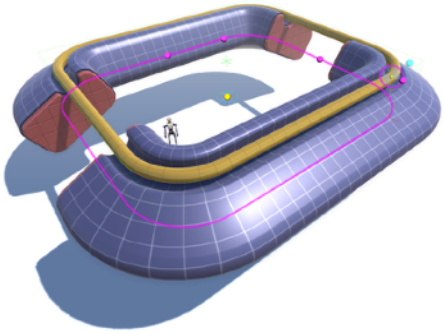
Step 4.11: Click on the yellow tube to reveal the tool for the tubes *Section Shape*.

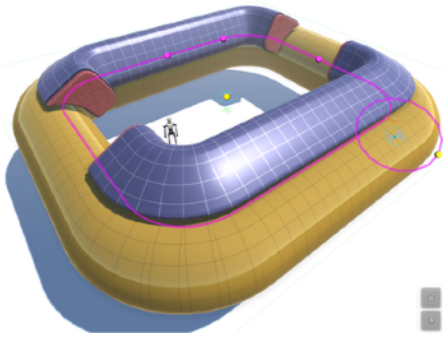
Step 4.12: Use the center point handle of the disc to drag the yellow tube in and out of the purple tube.







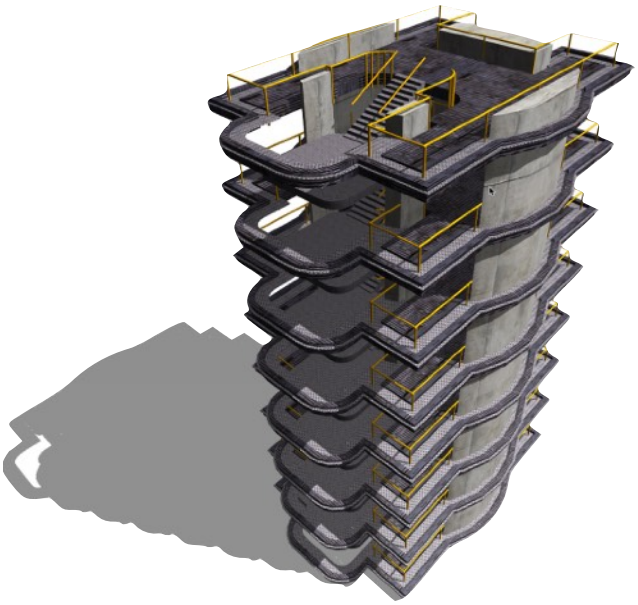


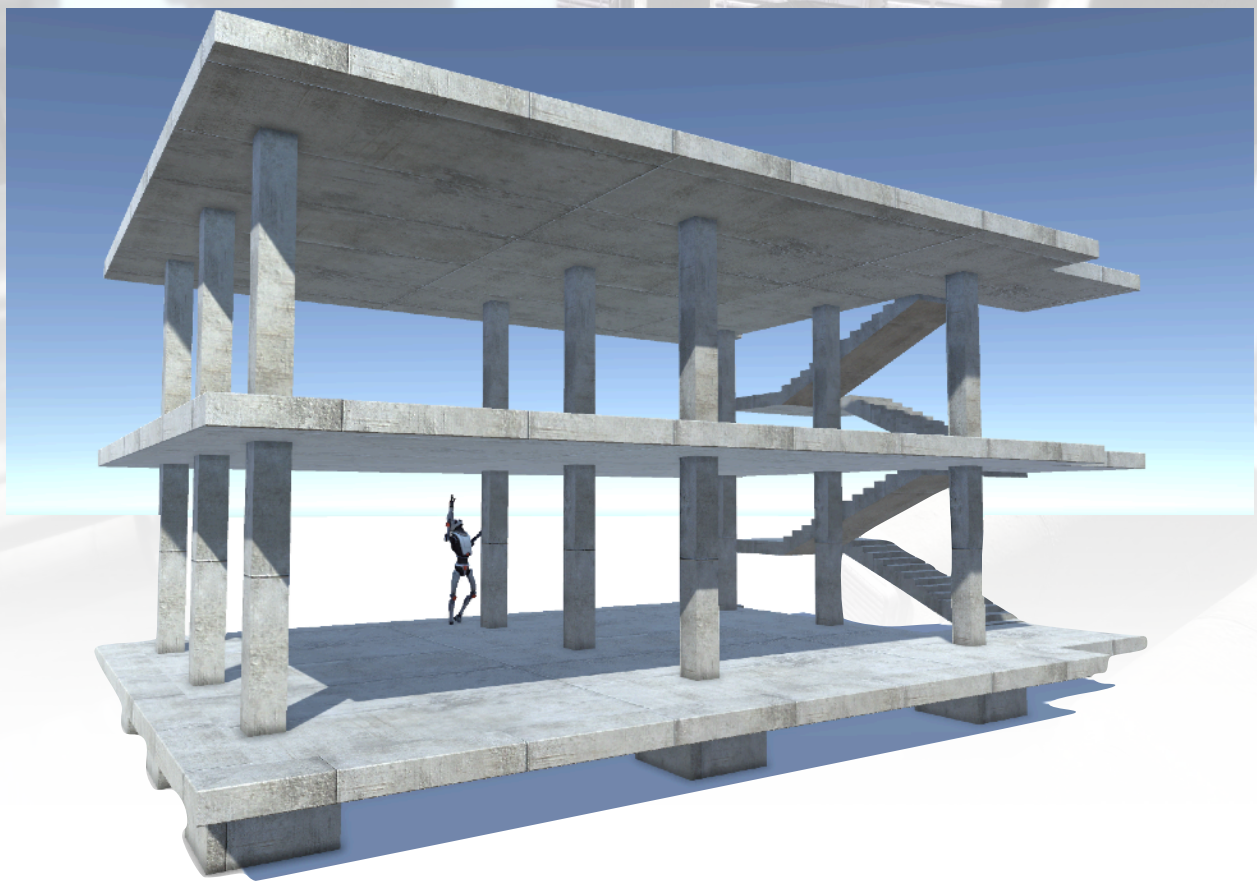


Ah, per chance to dream... of GridRepeaters!

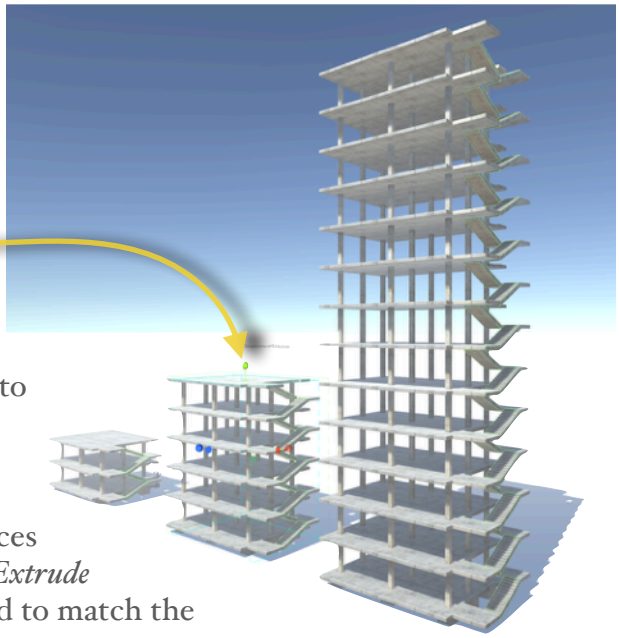
5. Repeating Yourself

Repeaters in Archimatix allow you to distribute instances of any AX-generated mesh or Unity Prefab in various patterns, including grids, radial arrays and symmetrical pairs. We have already seen an example of 2D *Repeaters* in the sideboard of the *RicketyStaircase*. Now, let's explore repetition in 3D.





The early 20th-century architect Le Corbusier explored the possibilities of new building materials such as concrete. Calling a house a "machine for living," he proposed the *Maison Dom-ino*, a structure that represented the essence of domestic space: floors, columns and stairs. While the Domino House looks a bit drafty, this essential structure provides a great example of repetition using a *GridRepeater* and a *FloorRepeater*.

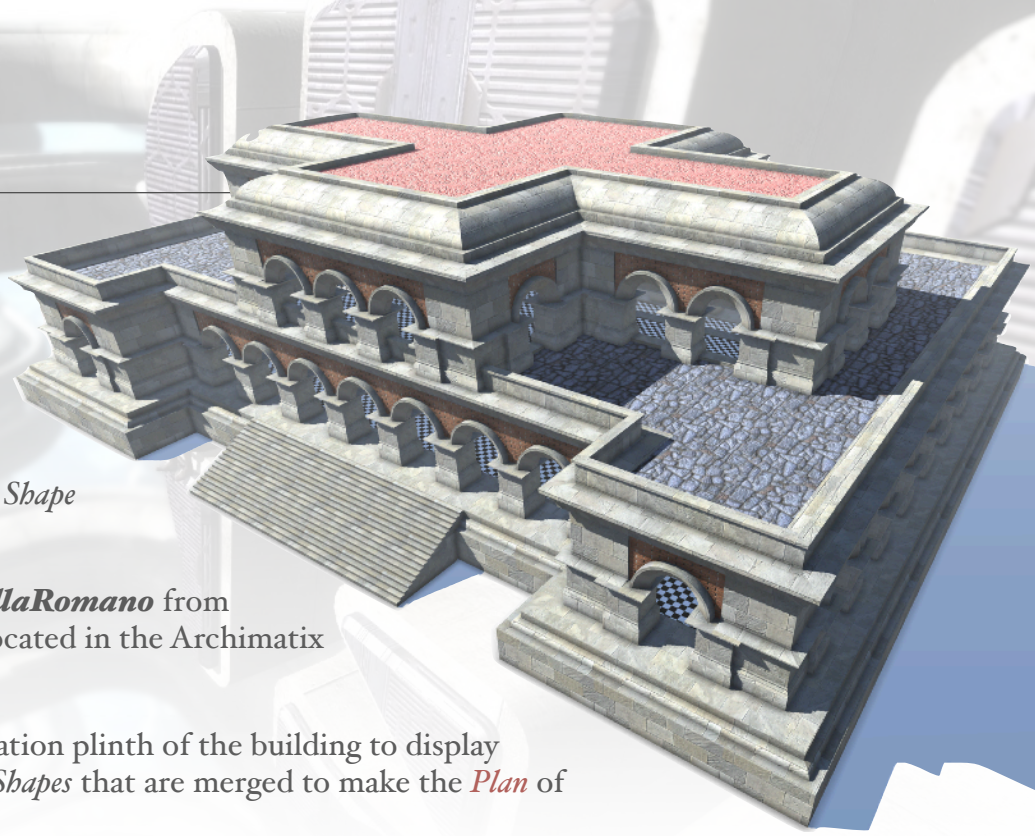


Step 5.1: Create a new scene and instantiate the ***DominoHouse*** from the Library.

Step 5.2: Click and drag on the green handle at the top of the structure.

As you drag the handle upwards, the FloorRepeater SizeY parameter increases and, to maintain a preferred story height, the FloorRepeater adds more floors. The actualStory heights vary slightly as the number of floors increases. Notice how the stair instances grow or shrink to match the story height.The *Extrude Height* of the GridRepeated column is increased to match the overall SizeY.

Repetition with a Plan

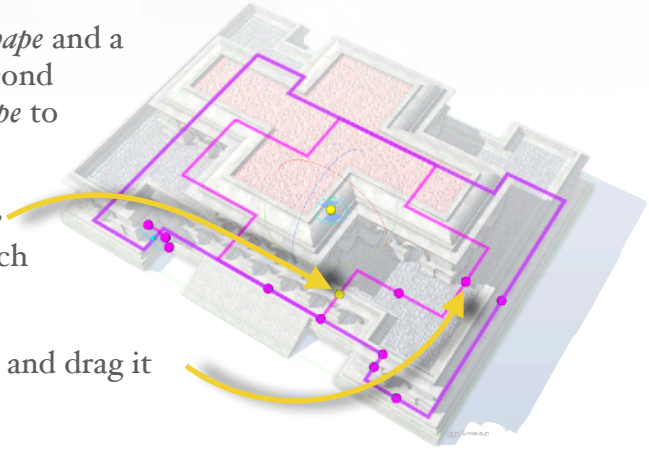


A powerful form of repetition in Archimatix uses a *Plan Shape* to distribute objects along its path. For example, a window unit can be placed around the *Shape* of a building.

Step 5.3: Open the scene ***VillaRomano*** from the Scenes folder located in the Archimatix folder.

Step 5.4: Click on the foundation plinth of the building to display the handles of the *Shapes* that are merged to make the *Plan* of the building.

The *Plan* of this villa is a composite of two *Shapes*, an *I-Shape* and a *Cross Shape*. The *Cross Shape* is used as the plan for the second story of the building, and it is also merged with the *I-Shape* to form the first story *Plan*.



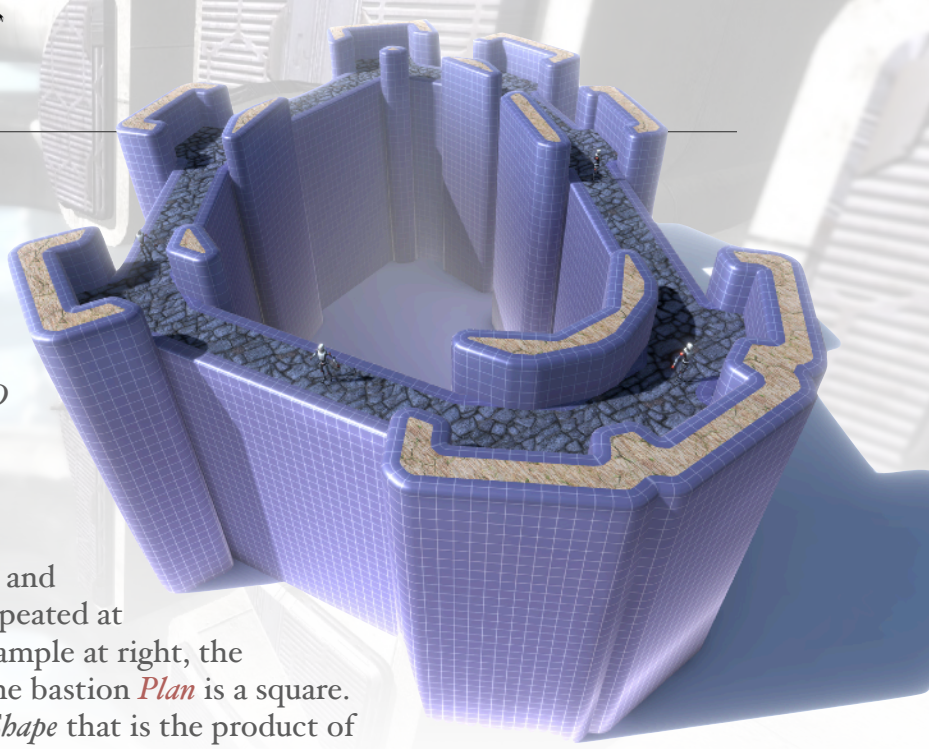
Step 5.5: Adjust the handle at the side of the *Cross Shape* and drag it toward the center until only one arch is on the front face of the upper story.

Step 5.6: Adjust the handle at the end of the *Cross Shape* and drag it outside of the perimeter of the *I-Shape*.

When you drag the end handle of the *Cross Shape*, you should notice that the first story *Plan* is now the combination of the two *Shapes*.

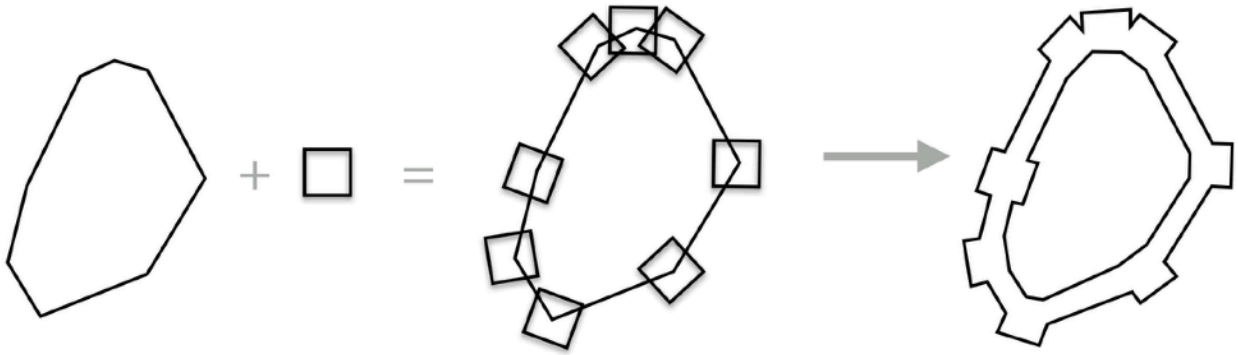
Step 5.7: Continue to adjust various handles to make alternative forms such as the ones depicted below.

Flat Out Repetition

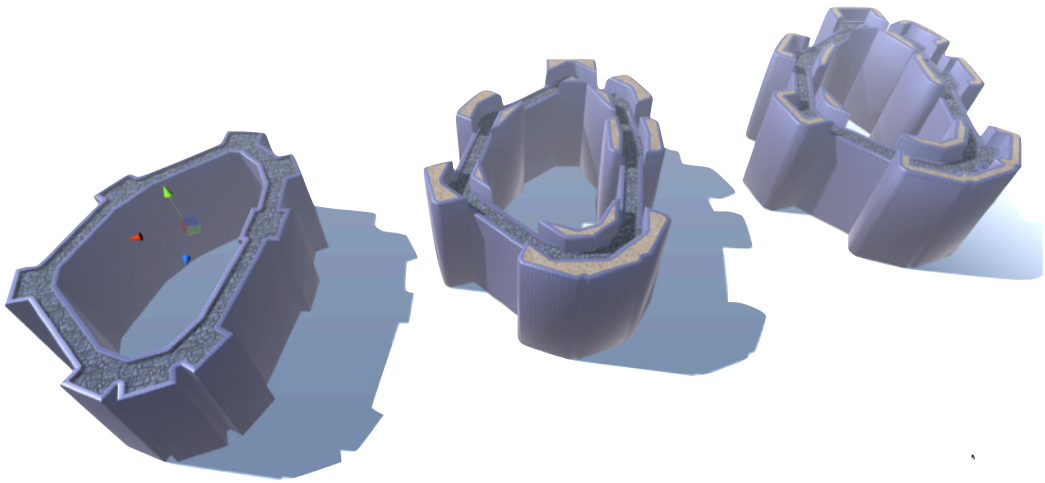


Most of the 3D *Repeaters* have 2D counterparts. These 2D *Repeaters* are particularly powerful for elaborate *Shape* merging. A good example of this using a *PlanRepeater2D* node included in the Scene folder is *BastionMerge*.

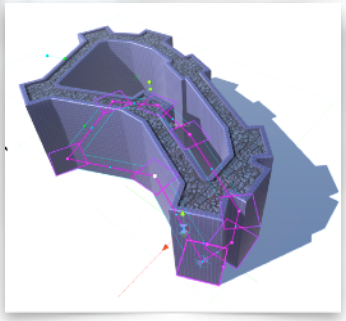
BastionMerge uses a *Plan Shape* that represents a sort of castle wall outline and another plan for a bastion that gets repeated at each node of the wall *Shape*. In the example at right, the fortress plan is a *FreeformCurve,* and the bastion *Plan* is a square. The *PlanRepeater2D* generates a new *Shape* that is the product of each *Rectangle Shape* copied and transformed to each vertex of the fortress *Plan*. Then the fortress *Plan* itself is "thickened," producing a composite *Shape* seen on the right in the diagram below. This composite plan is then extruded.



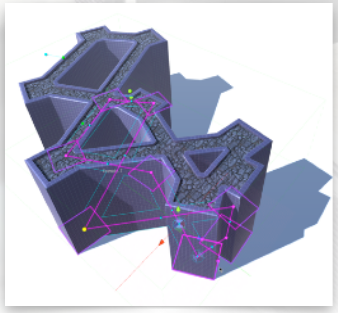
Step 5.8: Open the scene ***BastionMerge*** from the Scenes folder located in the Archimatix folder. In the scene, you will find three versions of the *BastionMerge* fortress.



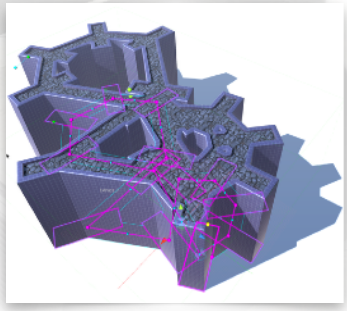
Step 5.9: Click on the stone pathway atop the main wall of the first (simplest) fortress. When the *Shape* handles appear, click and drag a node of the wall *Shape* to make variations similar to the ones depicted below. Clicking on purple point handles will move a vertex, while clicking and dragging a cyan handle will create a new vertex.



Moving vertices.

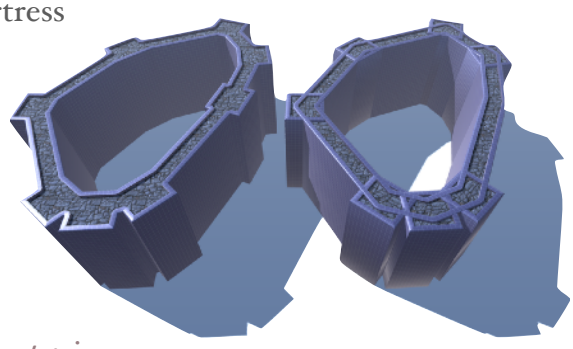


Crossing vertices.



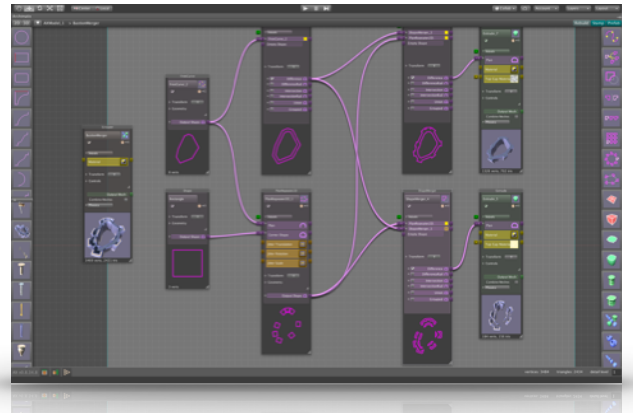
Adding vertices and adjusting the Rectangles' *handles.*

Notice that, as you modify the *Plan Shapes* of the fortress wall and the bastion *Rectangle*, the outline of the composite *Shape* is expressed in the parapet wall, which is parallel to the outline and forms a continuous path through the bastions. If this fortress had used a 3D *PlanRepeater*, each bastion would have its own parapet. While the latter may be preferred in some cases, the power of 2D *Repeaters* is in the generation of complex forms.

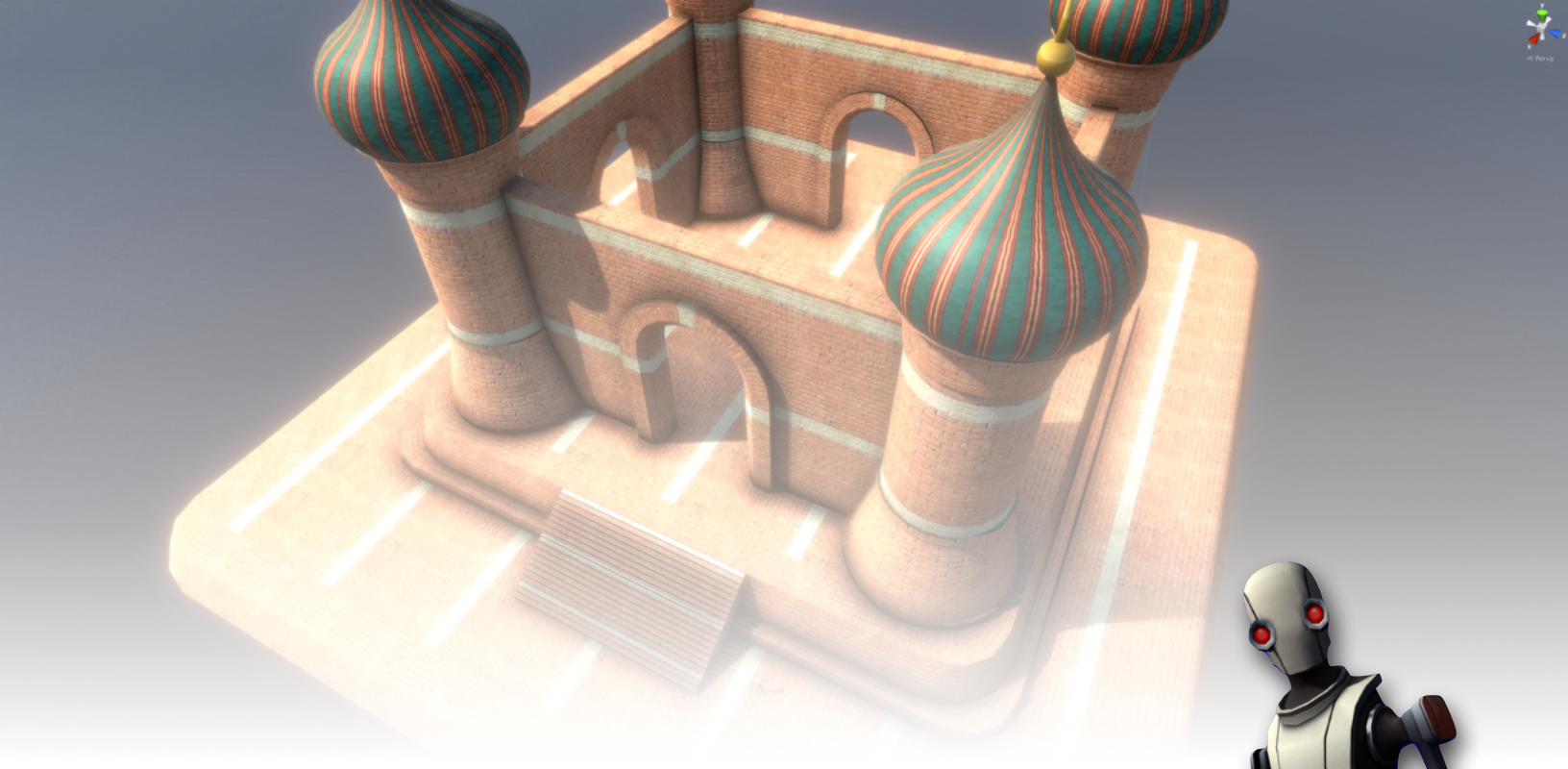


Comparison between fortresses using a PlanRepeater2D *(left) versus a 3D* PlanRepeater *(right).*

When the "thickened" wall *Shape* is subtracted from the repeated bastions, a more distinct pathway is made with openings in solid bastions that are aligned with the wall. In the other two fortress variations in the scene, more *Shape* merging has allowed for separate bastion walls that have openings that respond to the fortress wall *Shape*.



In the next chapter, you will be introduced to the Node Graph Editor. Once you are familiar with editing the logic of forms using nodes, it will be useful to return to the *BastionMerge* scene and see how it was organized.

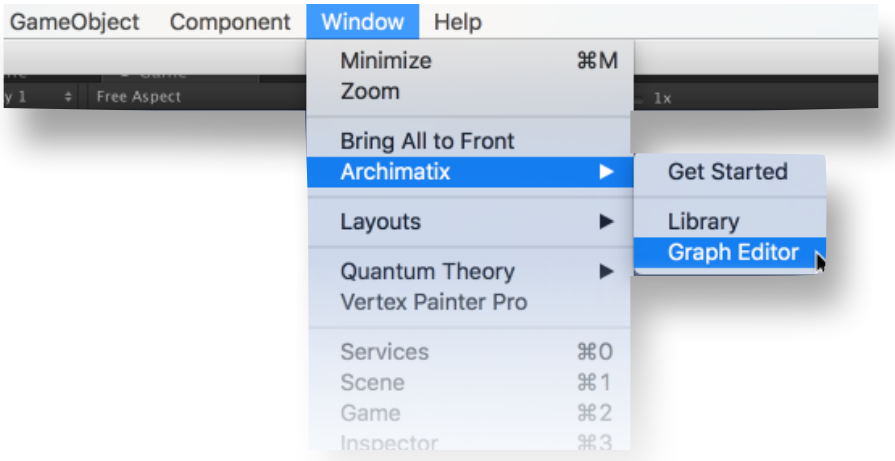


6. It's All What You Node

As we mentioned earlier, the parametric objects that you have been trying were all created with the Node Graph Editor. It's high time we pull back the curtain and see how a castle, such as the one depicted above, can be modeled with relatively few nodes.

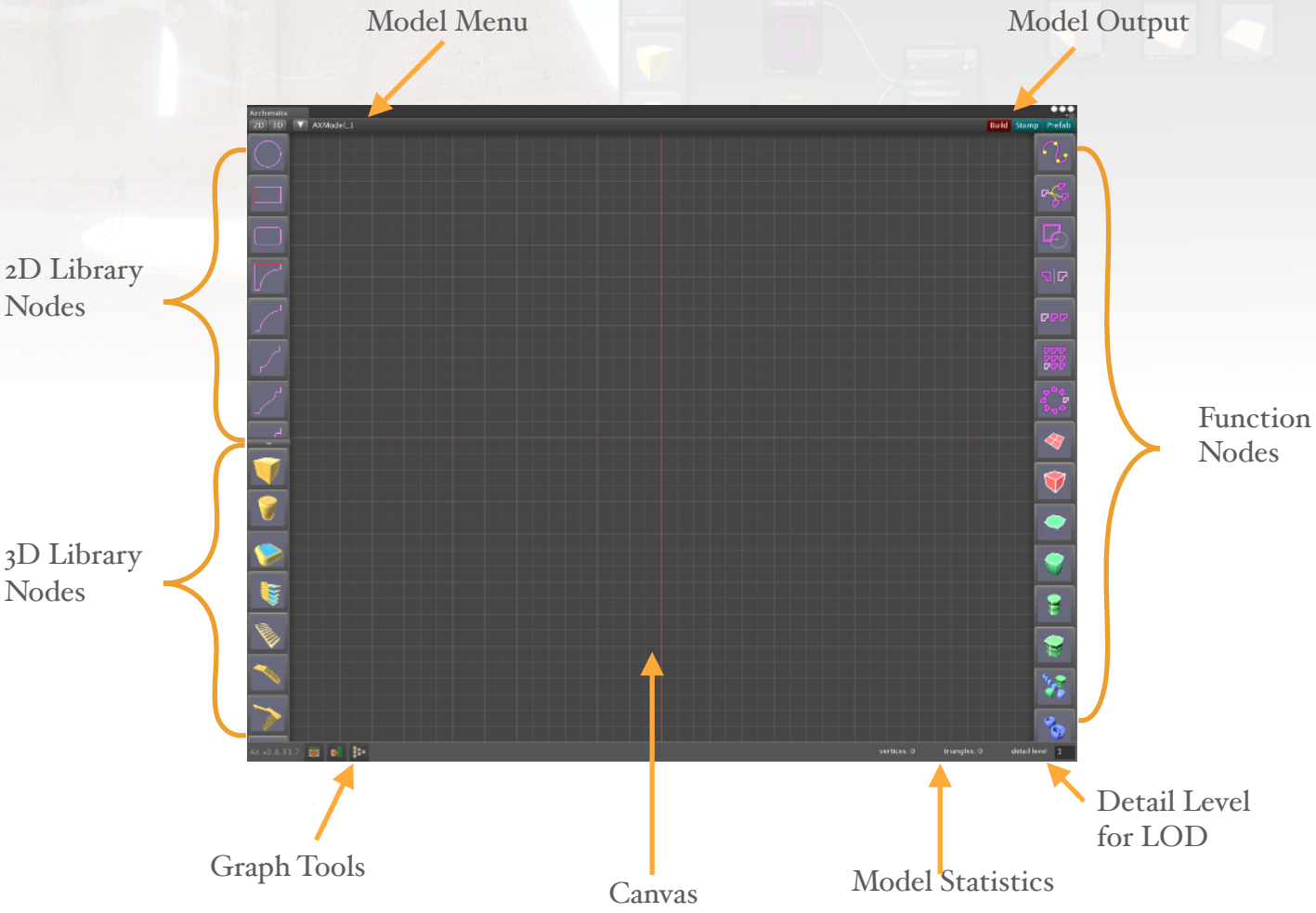
To get started, let's open up the Node Graph Editor window.

Step 6.1: Create a new scene and open the Node Graph Editor window with the Unity menu by choosing **Window** > **Archimatix** > **Graph Editor**.



Anatomy of the Node Graph Editor Window

Before jumping in with node creation, let's take a look at the main areas of the Node Graph Editor window.



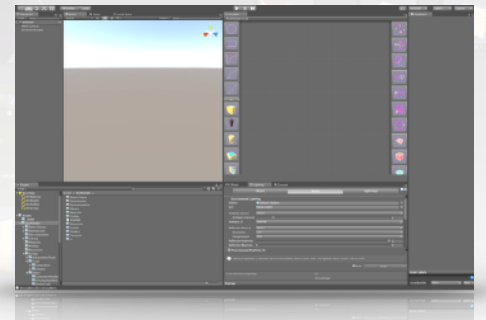
The Node Graph Editor window is organized as a canvas area surrounded by tool menus. To the left and right of the canvas are menus which are used to add nodes to the canvas. The left sidebar features items from the Archimatix Library. Theses items are divided into 2D and 3D items. The right sidebar displays functional nodes, nodes that take some input, process that input and then make the results available to other nodes through an output socket.

Along the top you will find model controls. The Group Address is a "bread crumb trail" that expands as you open subnode groups. The three buttons on the upper right give you options for model output. The "Build" of the model usually happens automatically when you complete a parameter change. The "Stamp" button creates a "frozen" version of the model that no longer needs Archimatix in the project. A stamped model can then be modified in other modeling packages such as *ProBuilder* or *Megafiers*. Finally, the Prefab button saves the

model's meshes to Unity's AssetDatabase and creates a standard Unity Prefab. This Prefab is also "frozen" and no longer parametric.

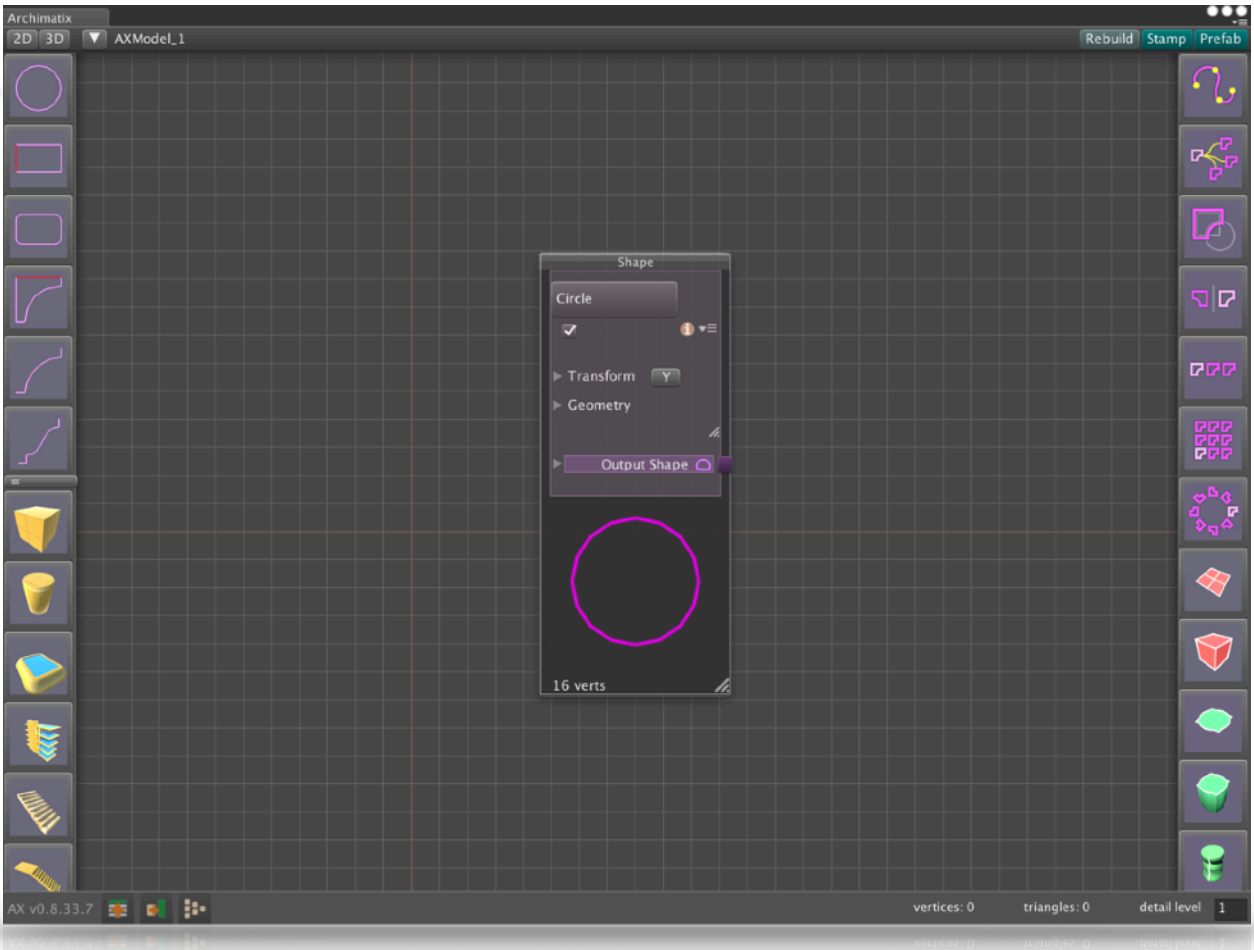
The bottom bar under the canvas features tools for cleaning up the graph in the canvas and provides quantitative feedback about your model, such as vertices and triangle counts.

You will be using the Node Graph Editor window a lot as you create smart models with Archimatix. Take a moment to find a place to dock it. It works well to have the Scene View window and the Node Graph Editor window the same size and side by side. The image on the right provides an example layout suited to modeling with Archimatix.



Now that you have an overview of the Node Graph Editor, it's time to jump in!

Step 6.2: Start a new scene and click on the *Circle* node icon in the left sidebar menu.



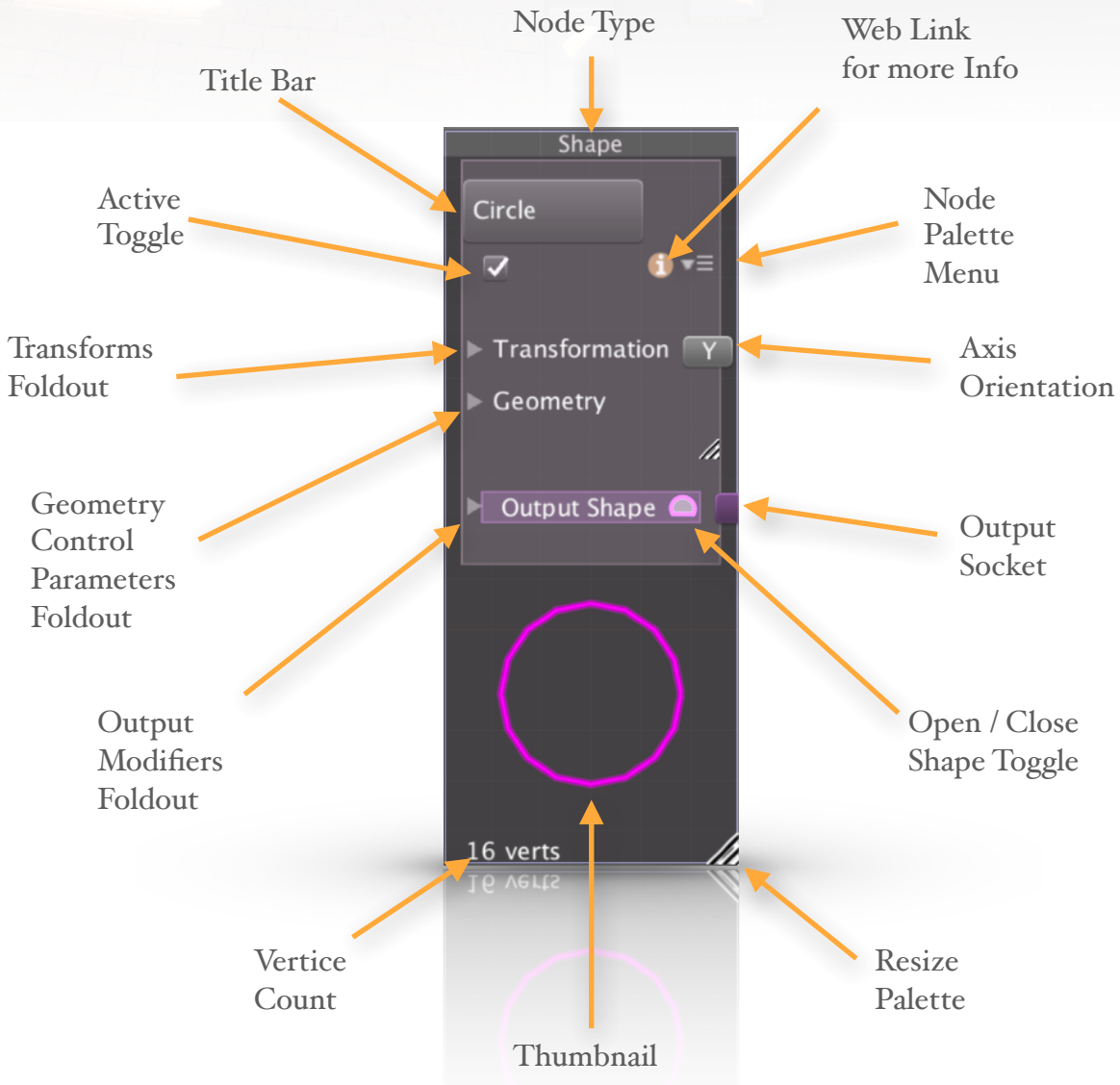
A *Circle* node will appear on the canvas and be framed in the Node Graph Editor window. The 2D circle it produces will be framed in your scene.

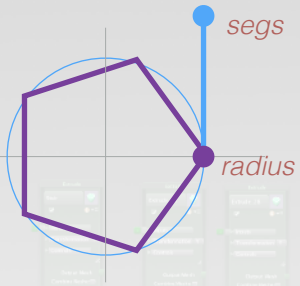
You can pan and zoom the canvas to get a better view of the node palette for the *Circle* node you instantiated.

Step 6.3: Use the scroll wheel on your mouse to zoom the canvas in and out. On a touch trackpad, you can use two-finger scrolling to zoom. Try RMB-drag to pan the canvas around.

Now that we can zoom into the Node Graph Editor, let's take a closer look at the Node Palette.

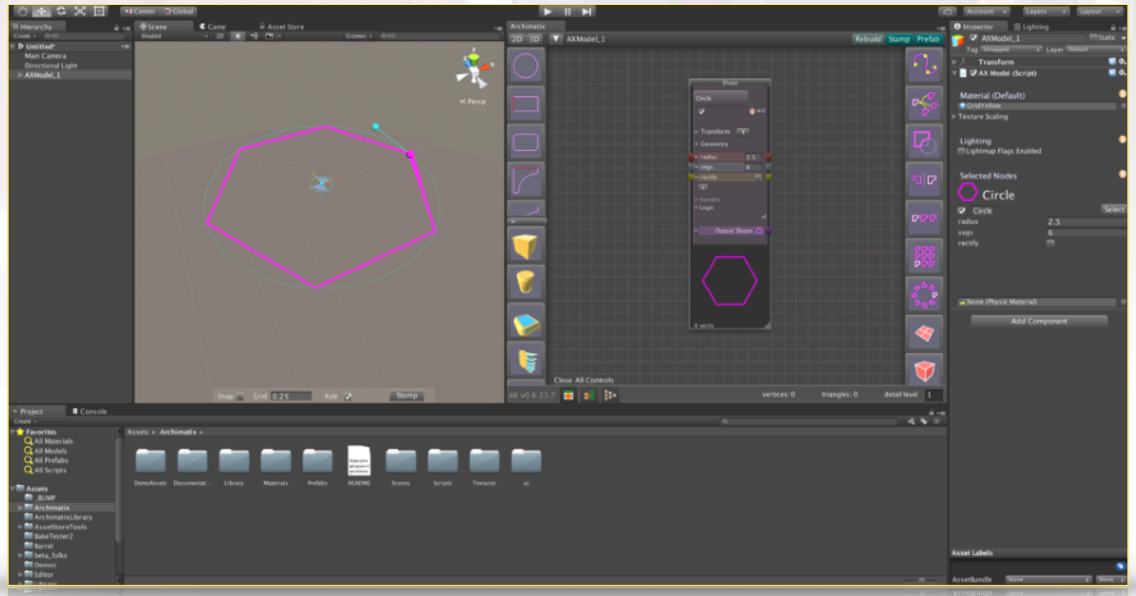
Anatomy of a 2D Node Palette





The *Circle* node has 16 sides, or segments, and a radius of 2 units, by default. Let's adjust these values.

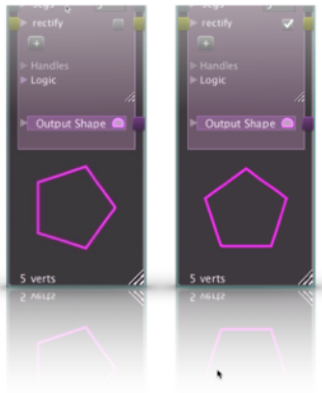
Step 6.4: Open the Geometry Control Parameters Foldout and modify the *radius* to 2.5 and *segs* to 6 to make the *Circle* a hexagon.



You will notice that some parameter fields in the node palette are redundant with the fields listed in the Inspector, with an important difference. Each parameter in the Node Palette has a socket on either side of it. These sockets, which are color-coded based on the type of parameter data (red: *float;* blue: *int;* yellow: *bool*) allow you to link parameters from this node to parameters from other nodes.

The *radius* and *segs* parameters are the two basic values needed to generate vertices for the *Shape*. These vertices constitute the final output of the *Shape*.

The *rectify* parameter comes in handy when dealing with an odd number of *segs*. If the *Shape* is rectified, the *Circle* is turned so that one facet is parallel to the X-axis, rather than starting with a vertex falling on the X-axis. This comes into play for *Shapes* with small segment counts, such as a pentagon.

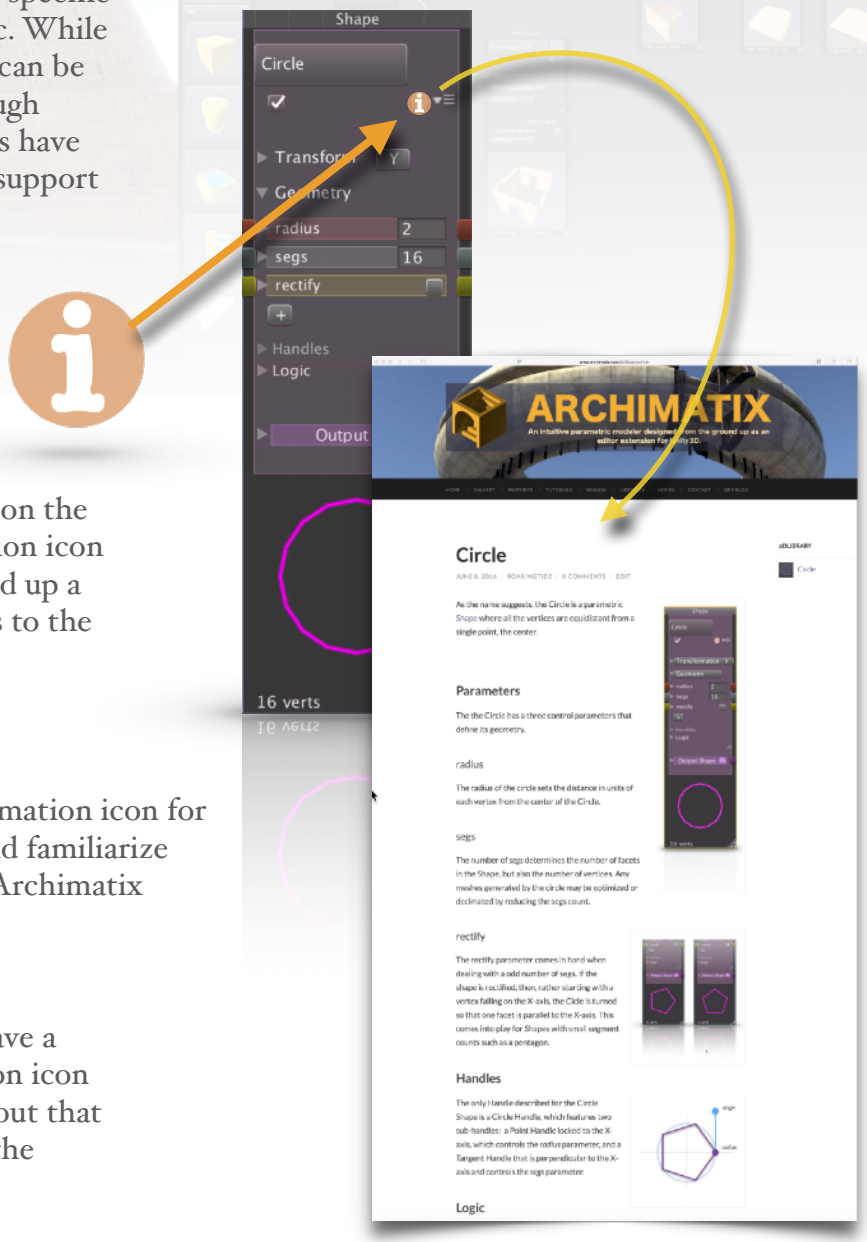


The handles and Logic foldouts, which enable you to customize or create your own *Shape* behaviors and control handles, are covered in the advanced topic of coding parametric *Shapes* using AX Script. More information on this topic can be found here.

The *Circle Shape* has a *Circle* handle, which features two sub-handles: a point handle locked to the X-axis, which controls the *radius* parameter, and a Tangent handle that is perpendicular to the X-axis and controls the *segs* parameter. The rectify parameter does not have a scene handle, and can only be checked in the Inspector or on the node palette.

Each node in Archimatix has specific parameters, handles and logic. While often the specifics of a node can be learned quickly enough through experimentation, many nodes have documentation pages at the support site, archimatix.com.

To quickly get to the documentation for a node you are working with, you can click on the information icon just below the title of the node. In the example shown in the image on the right, a click to the information icon on the *Circle* node has popped up a Web browser and directed us to the URL for the *Circle Shape*'s documentation.



Step 6.5: Click on the information icon for the *Circle Shape* and familiarize yourself with the Archimatix support site.

If a specific node does not have a page yet, then the information icon will link to a general page about that class of node – for example, the *Circle* is a *Shape* node.

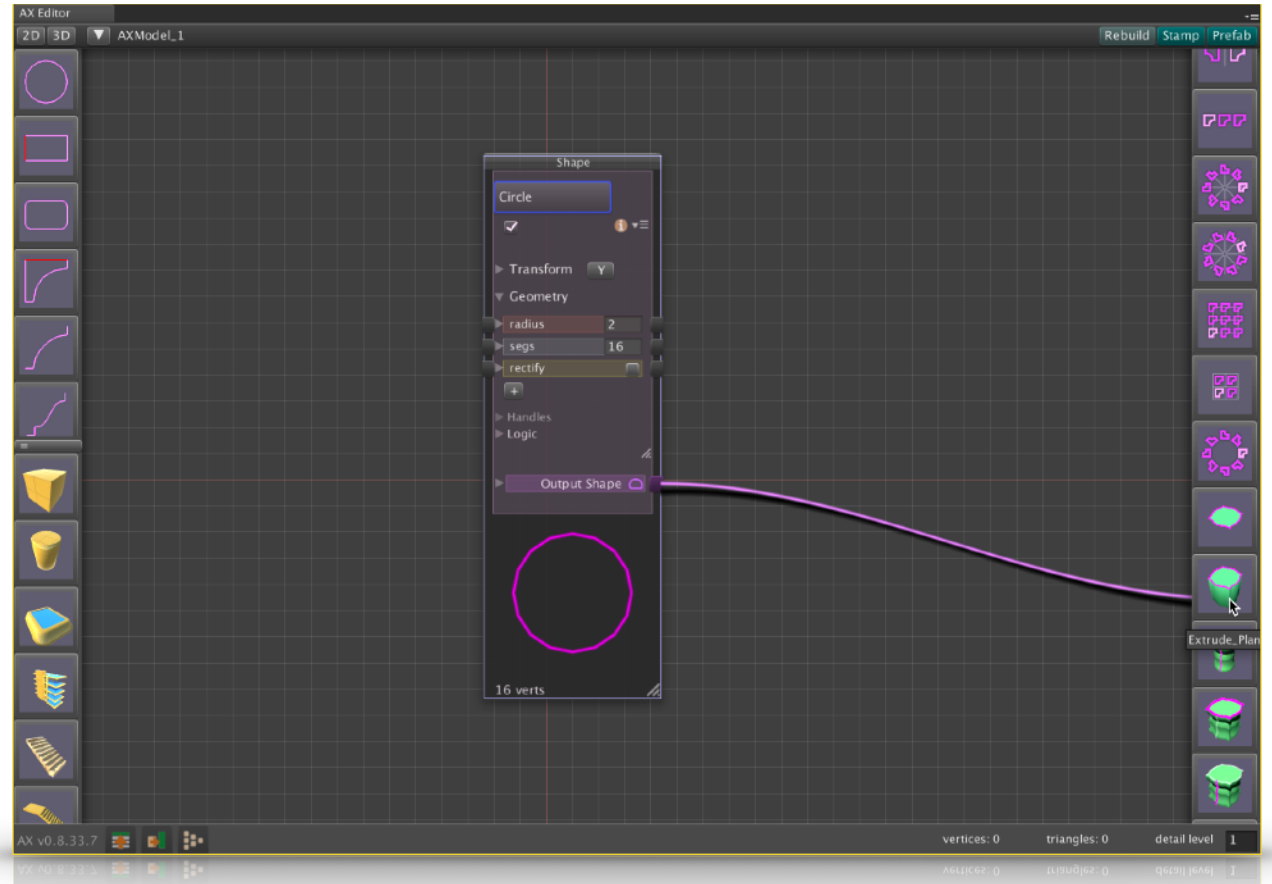
Next, let's make a mesh of this thing!

Meshing with the Node Graph

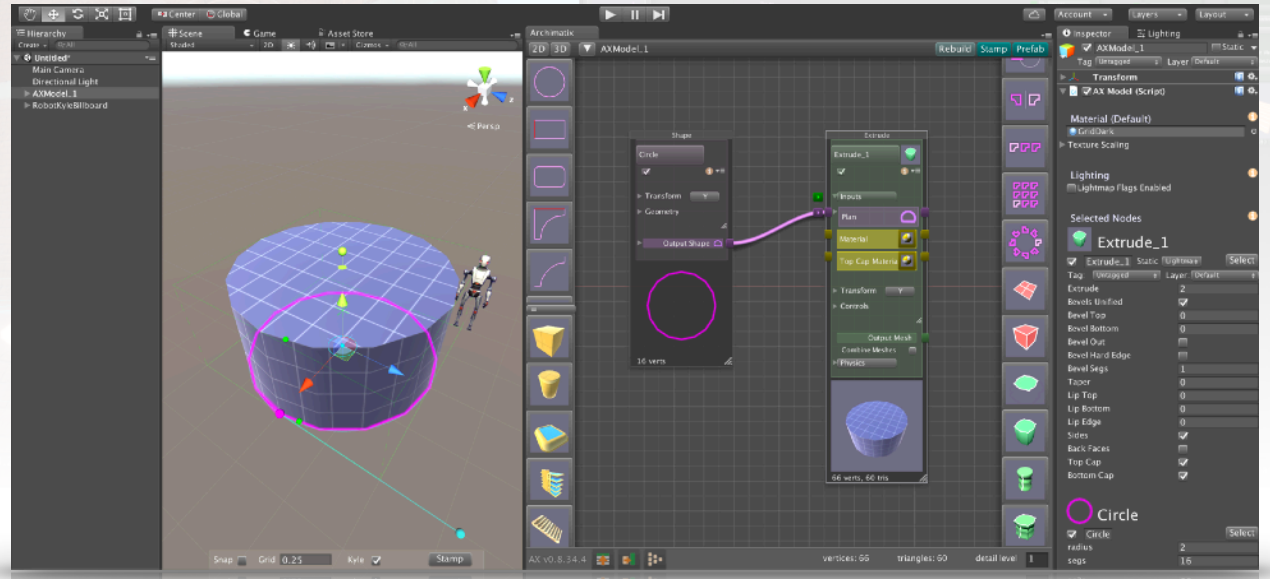
Having a 2D *Shape* node in the scene is a great start, but nodes don't do much in isolation. Generally two or more nodes work together to generate meshes. As it is with any team organization, communication is important. Nodes communicate with each other in two ways: 1. Connections between inputs and outputs; and 2. Relations that are defined by mathematical expressions. As nodes are connected to each other in non-mutually exclusive ways, a network of logical relations forms, making a *graph* data structure. While a graph may seem like a challenging thing to master, it is really not that difficult. To start, we can just keep creating nodes and connecting their outputs and inputs.

Since we have the *Circle* in the scene already, let's use it to generate a mesh by feeding its output to the input of an *Extrude* node, a *Mesher* node that takes an input *Shape*, doubles the *Shape* at some extrude distance or height, and creates a mesh between the two. As we shall see, the product of this nodal teamwork will be a *Cylinder*.

Step 6.6: Click on the purple button, or *socket,* on the lower right side of the *Circle* node that is labeled "*Output Shape*." Release the mouse button, and move the mouse to the right sidebar. Use your scroll wheel to scroll through the list of nodes in the sidebar until you see the *Extrude* node, and click on it.



A new *Extrude* node will appear in your graph with a connection line automatically rigged up between the *Circle* node's output and the *Extrude* node's "*Plan*" *Shape* input socket.



With the addition of the *Extrude* node and the input-output connection between them, you have officially created a graph!

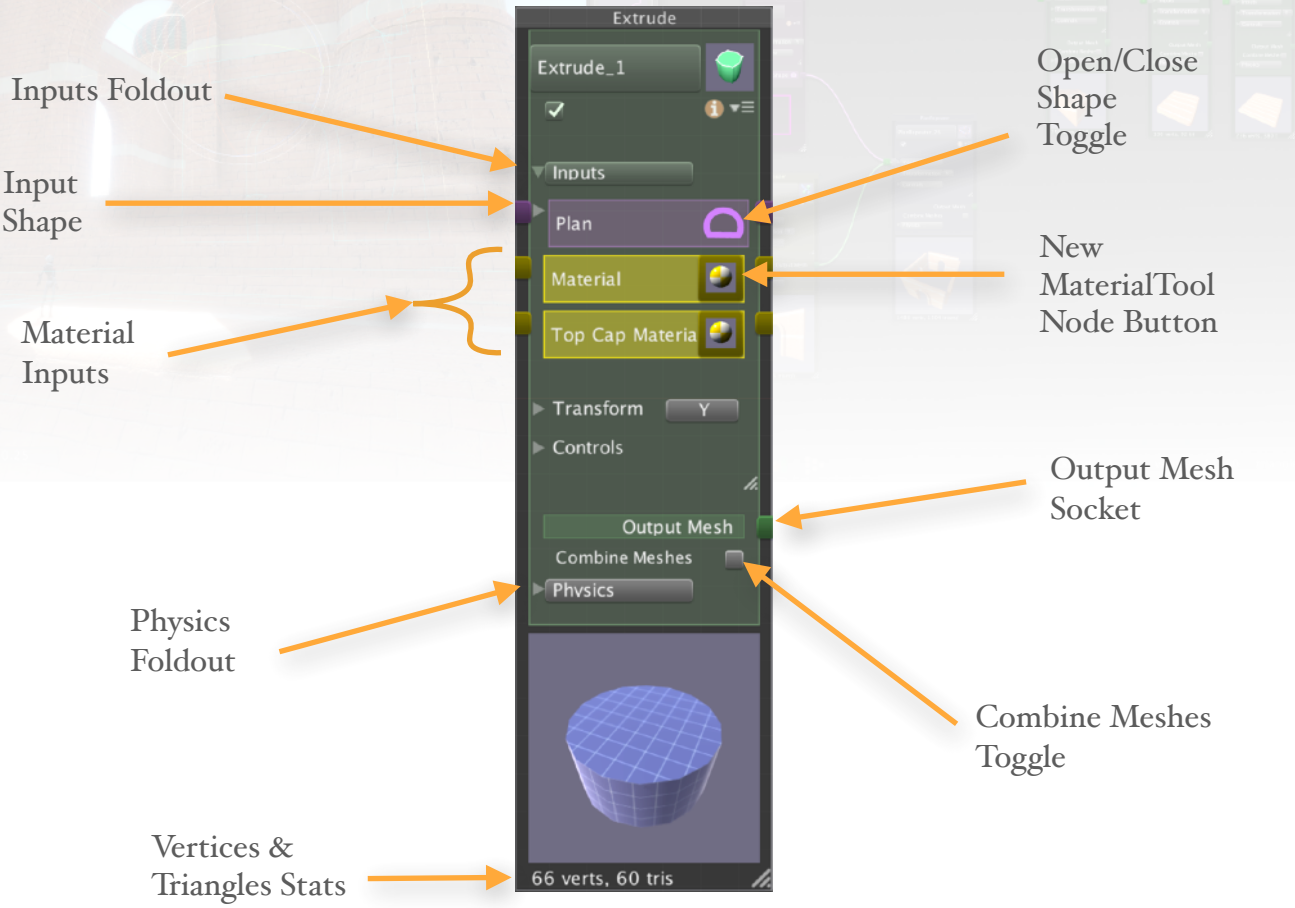
Now we see that the *Cylinder* Library Item is really just a two-node graph created in the Node Graph Editor.

You may click and drag on the window title bar or thumbnail of either node to rearrange them on the canvas. As you drag, the node moves freely. When you let go, it will snap to the underlying canvas grid.

In the Scene View, you will notice that the *Extrude* is displaying its handles and the handles of the upstream *Circle* node (as we saw in the *SciFi Platform* and the *BevelBox* examples already). But now that we have the Node Graph Editor open, we have more controls available in the node palette. For example, we can see the *Material* inputs and the mesh output parameters.

Let's zoom into the *Extrude* node to take a closer look at it.

Anatomy of an Extrude Node



With the Inputs foldout expanded, we can see various options for inputs. In the case of the *Extrude* node, we have a *Plan Shape* and two *Material* inputs. The first *Material* input will be the main material for all the surfaces of the *Extrude*. A *MaterialTool* node connected to the *Top Cap Material* input will be the material for the *Top Cap* and *Bottom Cap* of the *Extrude*.

When clicked, the new *MaterialTool* node button will create a new *MaterialTool* node and connect it to the *Extrude* node. Alternatively, you can connect a *MaterialTool* node already in the graph to either of the *MaterialTool* node inputs.

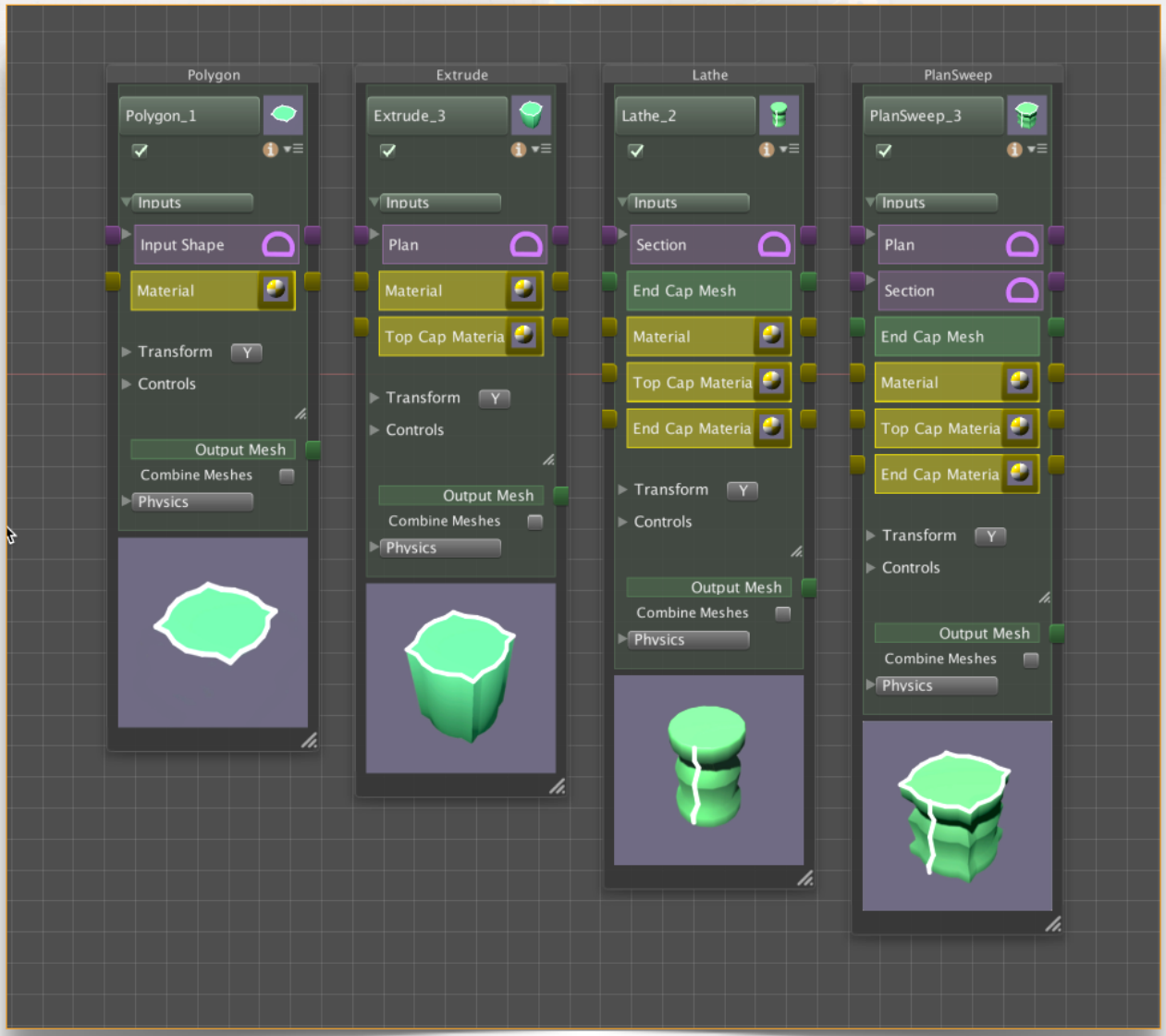
The "*Combine Meshes*" toggle signals to the node that it should combine any meshes that it creates into a single mesh per material. In the future, there will be an option to have one mesh with submeshes for each material.

The Physics modifiers found in the Physics foldout allow you to specify the type of collider you would like the output of the *Mesher* to have. There is also an option to add a RigidBody and make it kinematic.

Comparing Input Foldouts

Unlike the *Shape* node, *Mesher* nodes like *Extrude* have multiple inputs. The node's inputs help to define its particular mesh-generating functionality. For example, the *Extrude Mesher* needs a *Plan Shape* on which to operate, and it can optionally take a main *MaterialTool* and a *Top Cap Material* as inputs.

We can quickly get a sense of the types of *Meshers* we have in Archimatix by comparing their inputs.

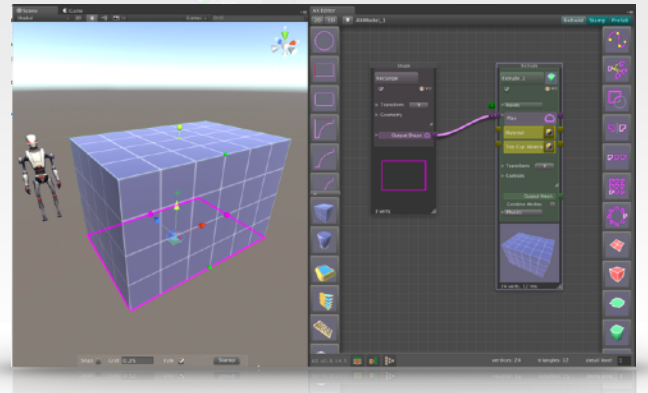


With additional complexity, we need more *Shape* and material input options. Otherwise, looking at the lower half of the four *Meshers*, we see that they all share an *Output Mesh*, *Combine Mesh* toggles and a Physics controls foldout.

Backfaces

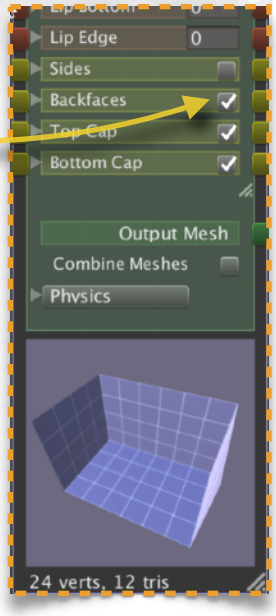
Meshers allow you to choose which surface you would like to be visible. For example, in an *Extrude Mesher*, you can disable the *Top Cap* and *Bottom Cap*. You can also choose to turn on the *Backfaces* if you want a thin-shelled form or a thin railing. Turning *Backfaces* on is also convenient for inverting meshes to use as interior environments.

Step 6.7: In a new scene, instantiate a *Rectangle* from the 2D Library. Connect the *Rectangle*'s output to an *Extrude* node item in the right sidebar menu.



You can have the *Sides* and the *Backfaces* visible together, but the caps are only either normal or inverted. They are inverted only when the *Backfaces* are on.

Step 6.8: Expand the Controls foldout on the *Extrude* node, and turn the *Sides* off and the *Backfaces* on.

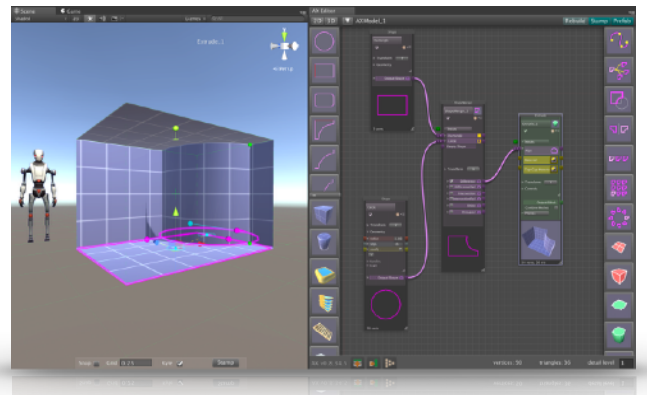


You now have the beginnings of an interior model. If you orbit the model in the scene, you will be looking into the space, but without seeing any of the exterior surfaces of the walls or floors.

Step 6.9: Instantiate a *Circle Shape* and connect its output to the *Plan* input of the *Extrude* node with a RMB-double-click (Windows) or command-click (OS X), automatically creating a *ShapeMerger* node in between. Move the *Circle* and resize it to build out the interior space.

You can continue to add *Shapes*, making the interior space larger and larger.

You cannot currently cut holes in the sides of an *Extrude*. Instead, Archimatix works with a philosophy of 2D Boolean operations and an additive process of meshes. This is closer to real-world building practices, where a window is not really cut out of a wall, but, rather, the mass of the wall is built up and the window opening an area where no material was added.

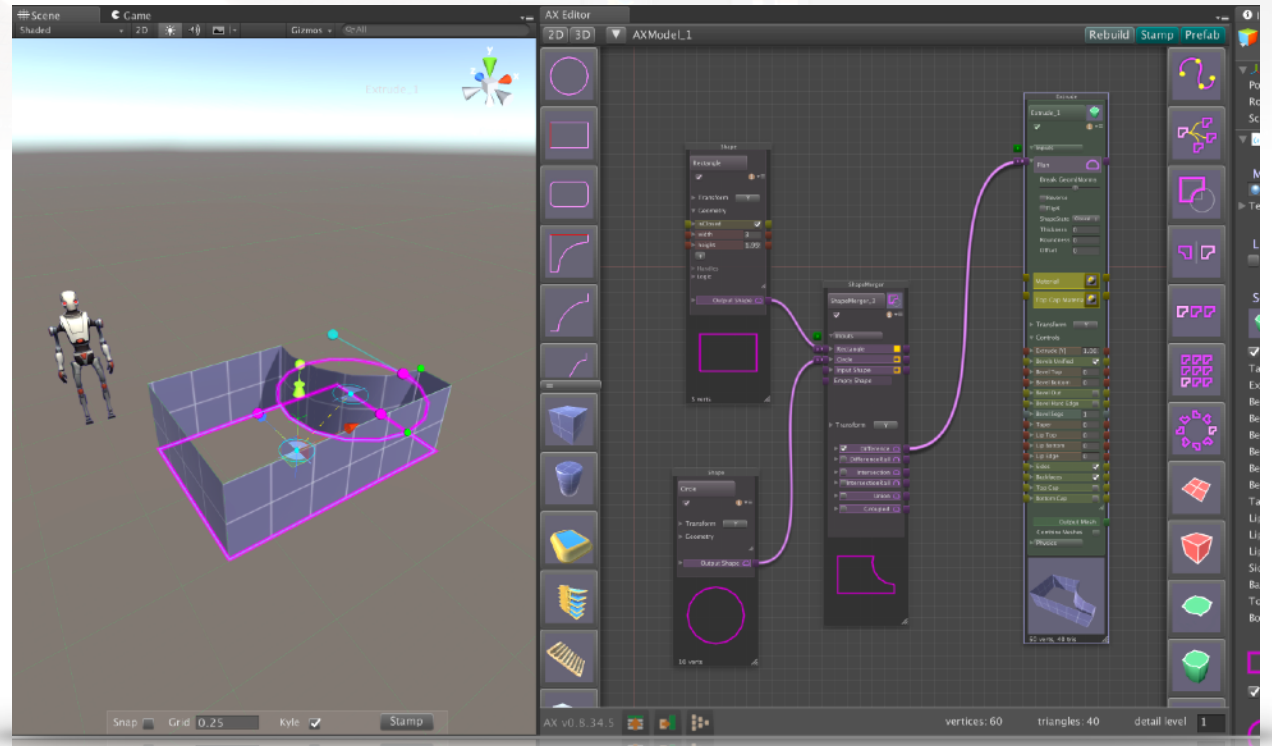


Let's take a look at an opening made through 2D *Shape* merging.

Meshing Off the Rails

In addition to exterior and interior models, a common case that uses both at the same time is a thin railing. Archimatix is well-suited to making thick railings, but for buildings with a low level of detail, a thin railing can save triangles.

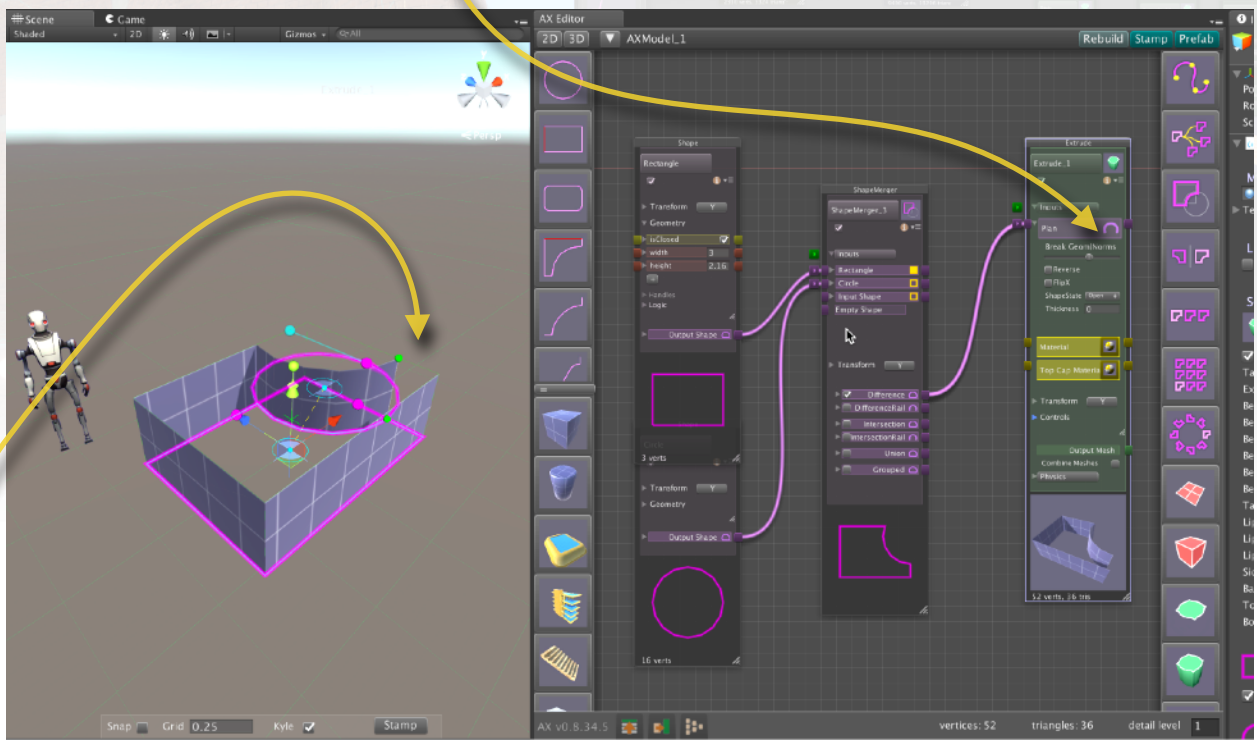
Step 6.10: From the scene on the previous page, turn the *Top Cap* and *Bottom Cap* off and turn the *Sides* on, leaving *Backfaces* on as well. Adjust the height of the *Extrude* to be about waist-high on Robot Kyle.



You can continue to add *Shapes* to the *ShapeMerger* and move them around or modify their parameters. But, at the moment, we see that there is no way for Robot Kyle to get inside of this fenced area.

The simplest way to open the fenced area is to open the *Plan Shape*. This can be done in one of two places, either in the *Difference* output of the *ShapeMerger* or in the *Plan* input of the *Extrude*. It makes most sense to leave the *ShapeMerger* output alone, since that *Shape* could be reused to generate a floor under the fenced area. This leaves the *Plan* input as the best option. Altering a *Shape* at an input point is essentially a just-in-time alteration that has no effect on the upstream source *Shapes*.

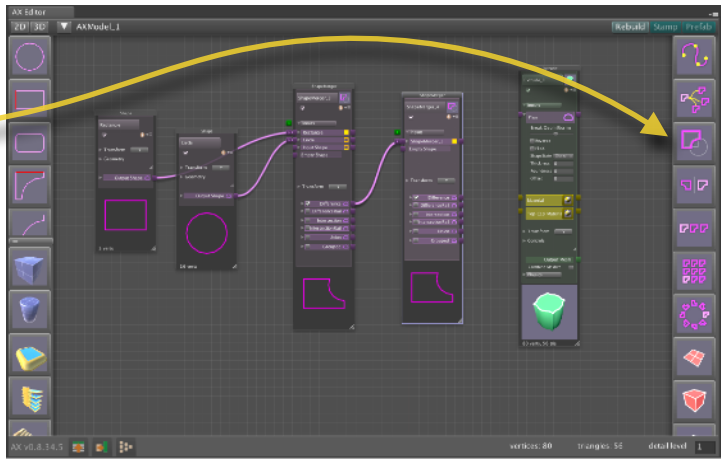
Step 6.11: Click on the *Open/Close Shape* toggle in the *Extrude*'s *Plan* input parameter.



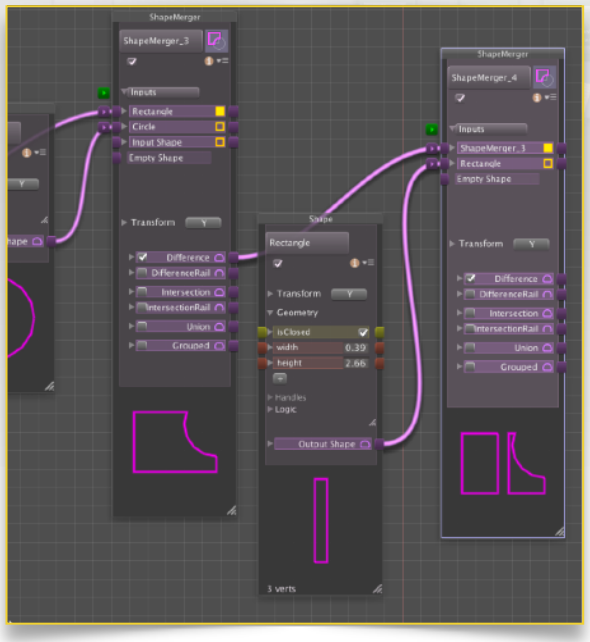
We see that, by opening the *Plan Shape*, we now have a part of the rail missing, in this case, right next to the quarter-circle. The problem with simply opening the *Shape* is that we cannot easily control where the opening is or allow for multiple openings. To remedy this, we can use the *DifferenceRail* of a new *ShapeMerger* to intentionally cut out parts of the rail. The *DifferenceRail* is a *Shape* composed of the lines of the solid *Shapes* in the *ShapeMerger* inputs without any lines contributed by the void *Shapes*.

Step 6.12: Click on the connector cable from the *ShapeMerger* output to the *Extrude*. While it is selected, hit the Delete key to remove it (Cmd-Delete on OS X).

Step 6.13: Click on the *ShapeMerger* to select it, and then click on the *ShapeMerger* icon in the right-hand sidebar menu. A new *ShapeMerger* will be created and automatically connected to the *Difference* output of the first *ShapeMerger*.



Now you should have the original *ShapeMerger* feeding into a new *ShapeMerger*. It is with this new one that we will trim out openings using a new void *Shape*.

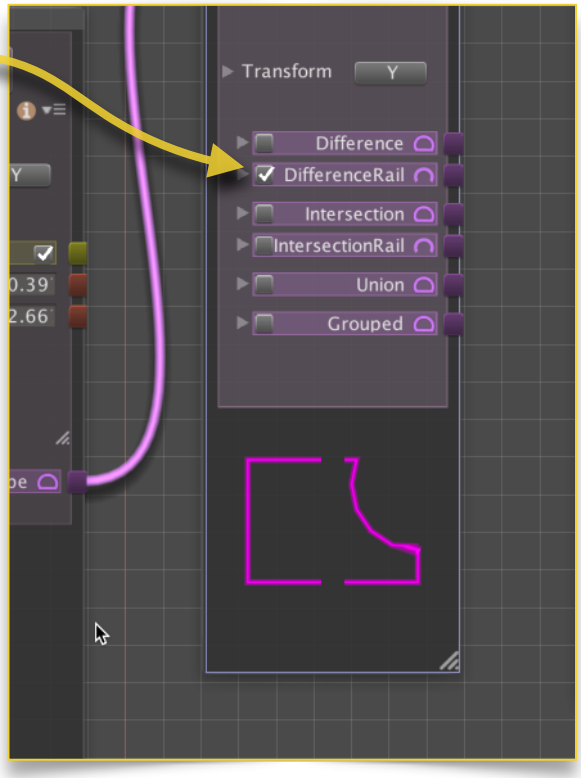


Step 6.14: With the new *ShapeMerger* selected, instantiate a *Rectangle* from the 2D Library items in the left-hand sidebar. The *Rectangle* will be automatically fed into the new *ShapeMerger*.

At first, the *Rectangle* will probably be too large and leave little of your *Shape* left.

Step 6.15: Resize the *Rectangle* so that it is removing a sliver of your *Plan Shape*.

Step 6.16: Click on the checkbox to the left of the *DifferenceRail* output to display the result of that Boolean operation.

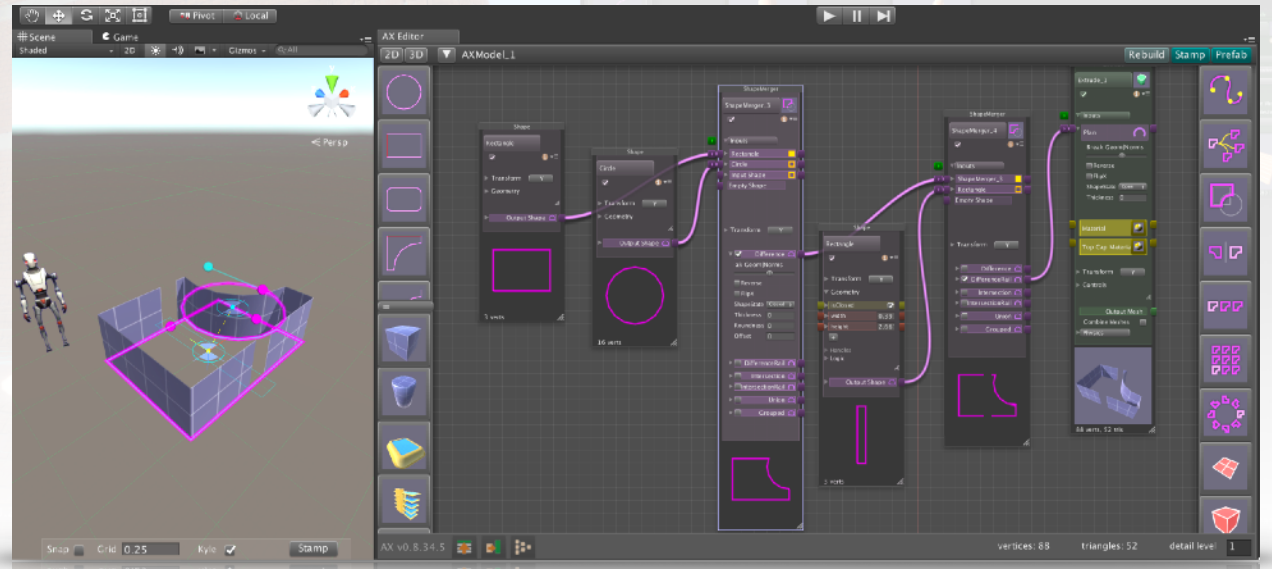


Using the original merged *Shape*, we now see the *DifferenceRail*, an "Open" *Shape* that is essentially a "rail." Such a rail *Shape* can be used to create thin and thick railings and walls with openings.

Note that the checkbox that you clicked to display the *DifferenceRail* output of the merge is only for displaying that output parameter's *Shape* in the node's thumbnail. You can hook different nodes simultaneously to any and all of the outputs of the *ShapeMerger*.

We can use the rail *Shape* we created in various *Meshers,* as we shall see in some examples presently.

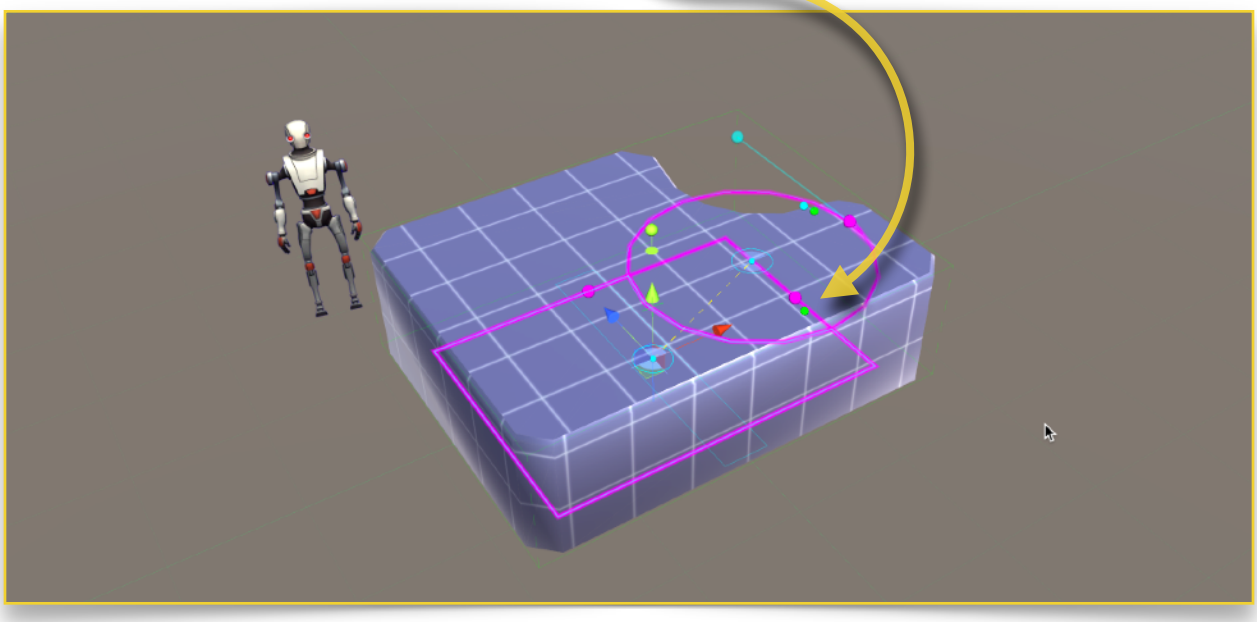
Step 6.17: Connect the *DifferenceRail* output of the new *ShapeMerger* to the *Extrude* node.



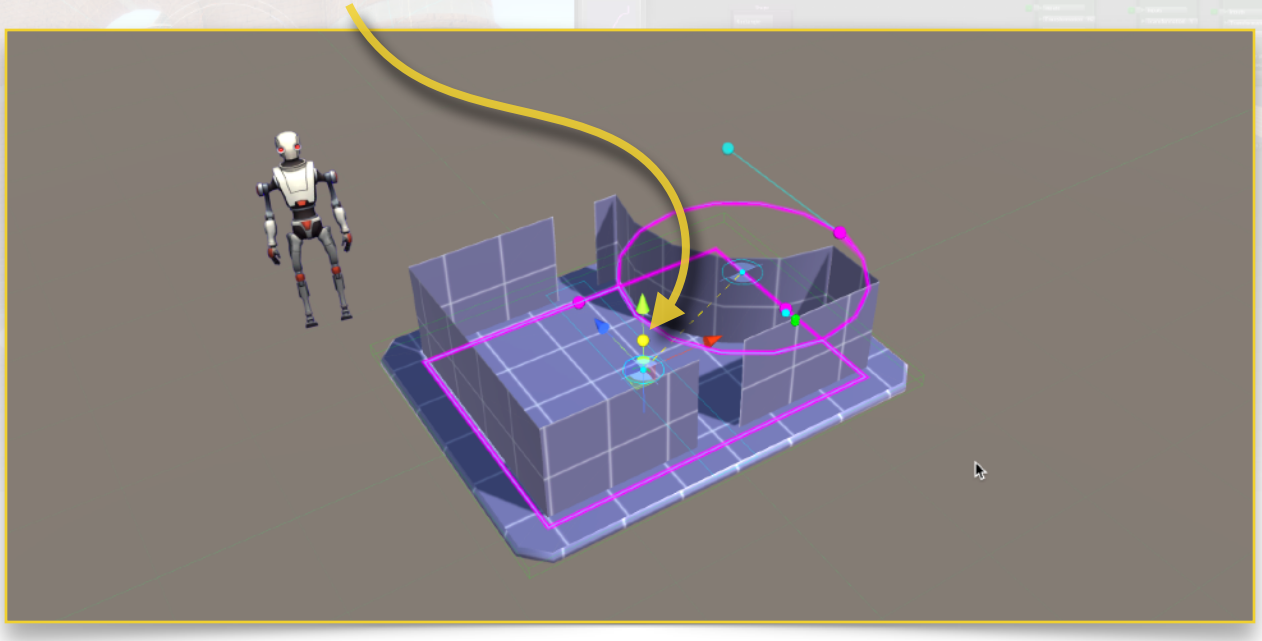
Now Robot Kyle will be able to enter the railed area, but we had better add a floor to provide some footing. This is where the original *ShapeMerger* output can come into play and demonstrate how powerful the non-mutually exclusive aspect of the node graph can be.

Step 6.18: Click on the *Difference* output of the first *ShapeMerger,* and then click on the *Extrude* icon in the right-hand sidebar menu.

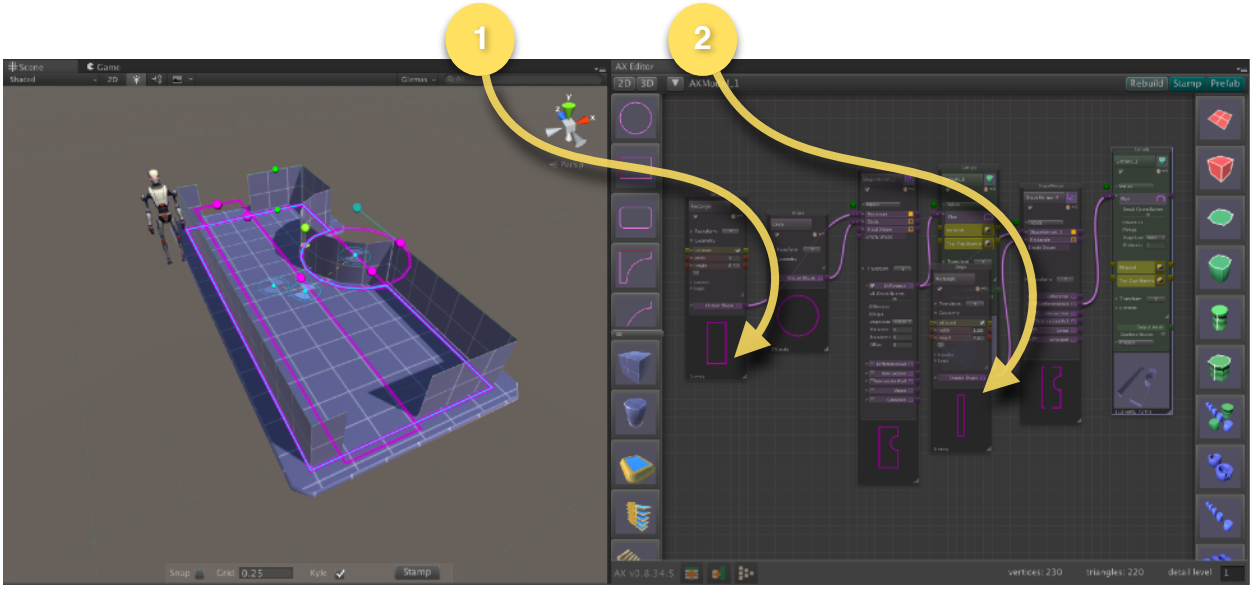
Step 6.19: Click and drag the lower green point handle on the *Extrude* to offset the *Plan* so that it is larger than the railing.



Step 6.20: Adjust the height of the *Extrude* until it appears to be a thin floor slab.



Step 6.21: Select the original *Rectangle* in the Node Graph Editor and make the fenced-in area larger, then select the new cutting *Rectangle* and make the opening larger.

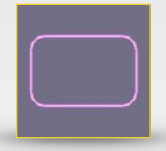


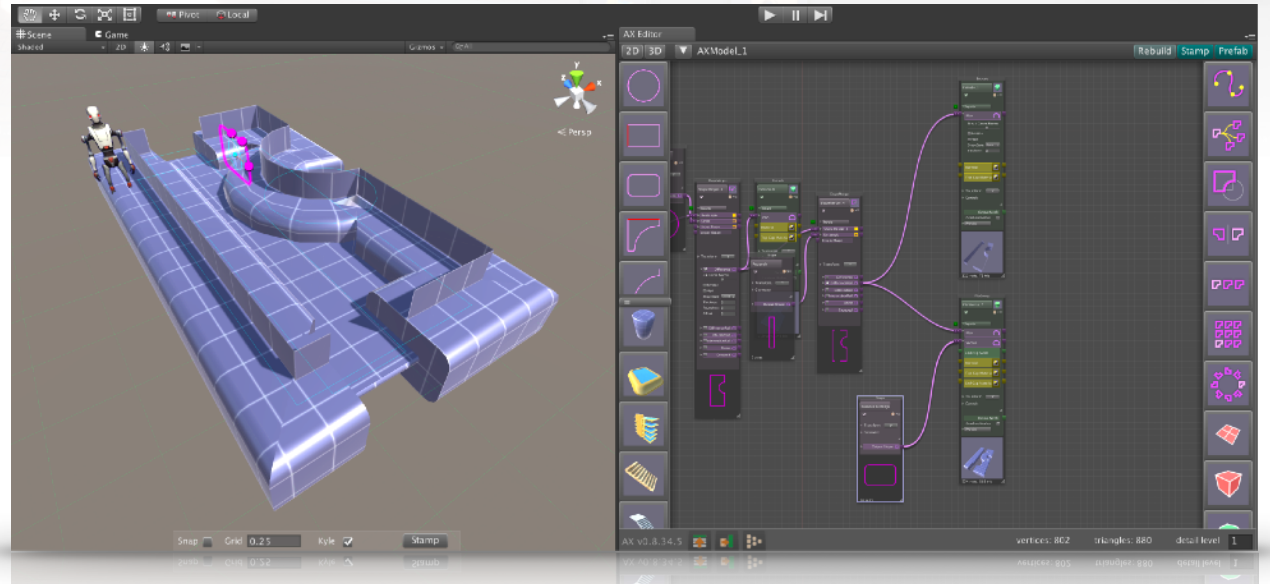
We can see that, as the node graph becomes more developed, we can modify various handles in the scene with a ripple effect. For example, at this point, modifying the first *Rectangle*'s handles alters both the railing and the floor.

Let's use the *DifferenceRail Shape* to create a thick handrail atop the current surface.

Step 6.22: Click on the *DifferenceRail* to start a connection and then scroll the right sidebar down until you see the *PlanSweep_Plan Mesher* icon, and click on it. A connection will automatically be created. A mesh will not appear in the scene until we provide the *PlanSweep* node with a *Section Shape*.

Step 6.23: Instantiate a *RoundedRectangle* from the 2D Library sidebar and connect the output of this node to the *Section* input of the *PlanSweep*.

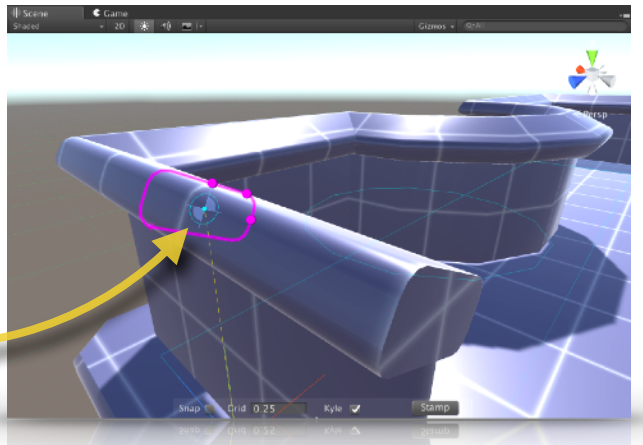




The *DifferenceRail* output is now used to generate two different meshes. The *PlanSweep Mesher* is currently at the base of the railing and is too large to be a handrail. To make the *RoundedRectangle Shape* smaller, it is far better to adjust its parameters than to scale it using the *Shape*'s transform controls. This is because, in parametric modeling, the values of parameters are important in building logical relationships. If we simply scale the *Shape*, then the parameters, which retain their value, will not reflect the size.

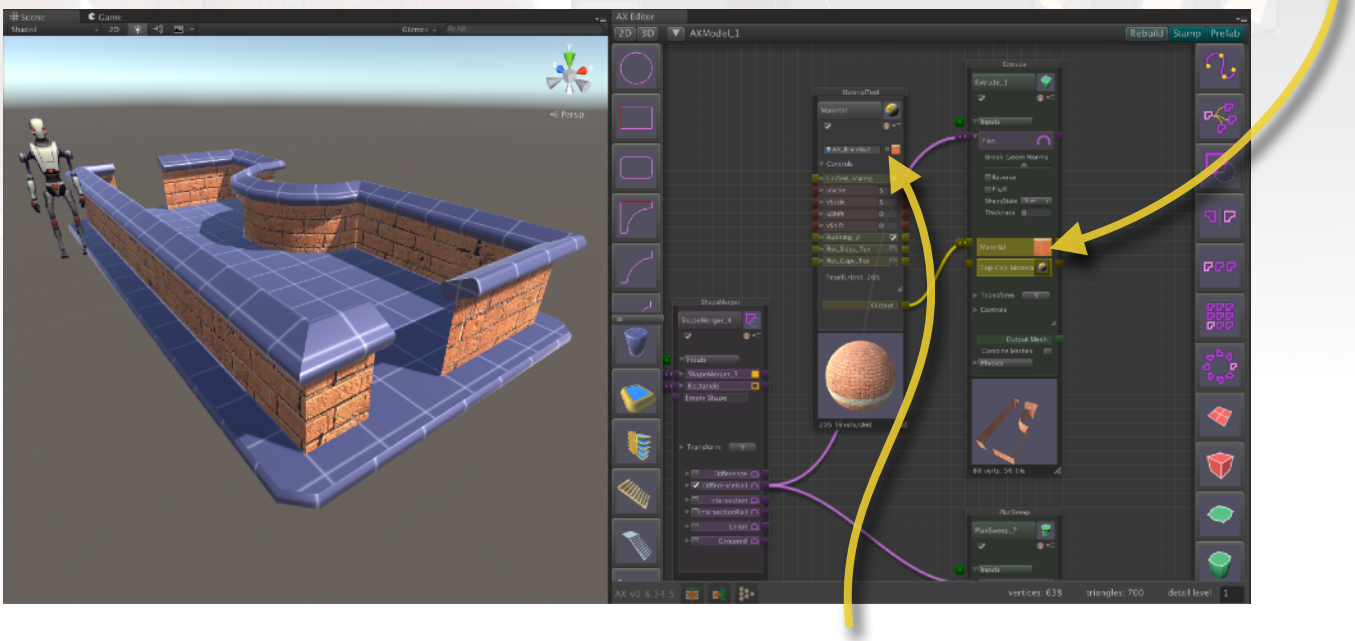
Step 6.24: Select the *RoundedRectangle* node in the Node Graph Editor window and hit the "f" key to frame the node in both the scene and in the Node Graph Editor.

Step 6.25: Adjust the *RoundedRectangle*'s handles in the scene to make the *Shape* smaller, and move and rotate the centroid to lift the *Shape* up to the top of the rail.



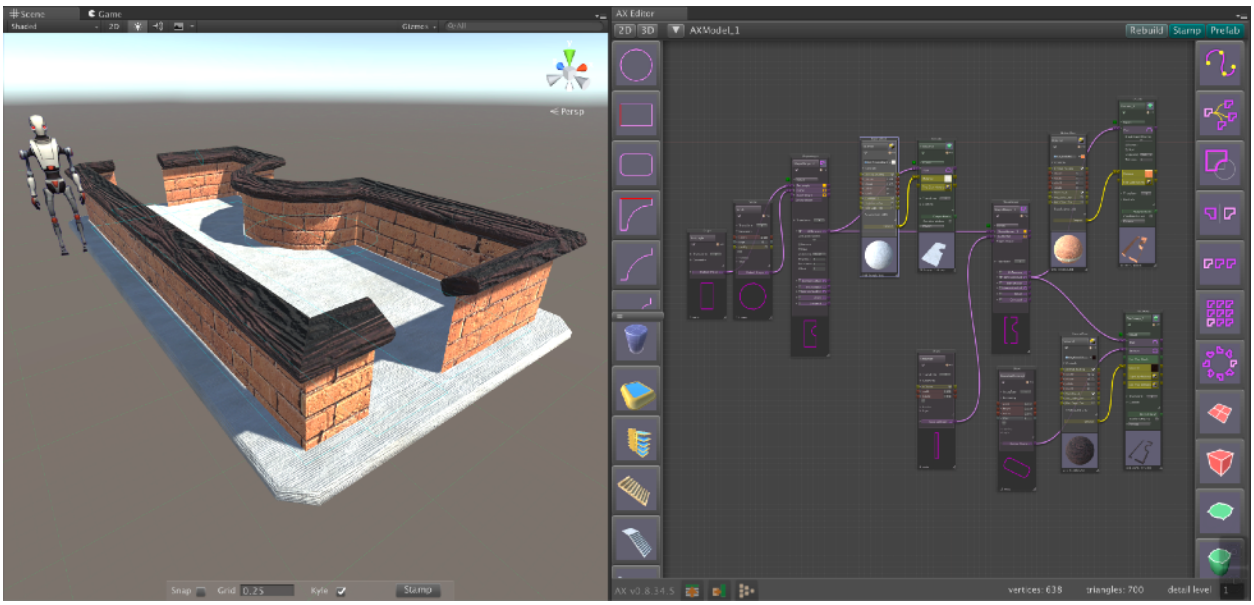
Now let's change the material of the rail sides.

Step 6.26: On the *Extrude* node palette for the rail sides, click on the icon button to the right of the *Material* input parameter. This will automatically add a *MaterialTool* node to the graph and connect it to the *Material* input.



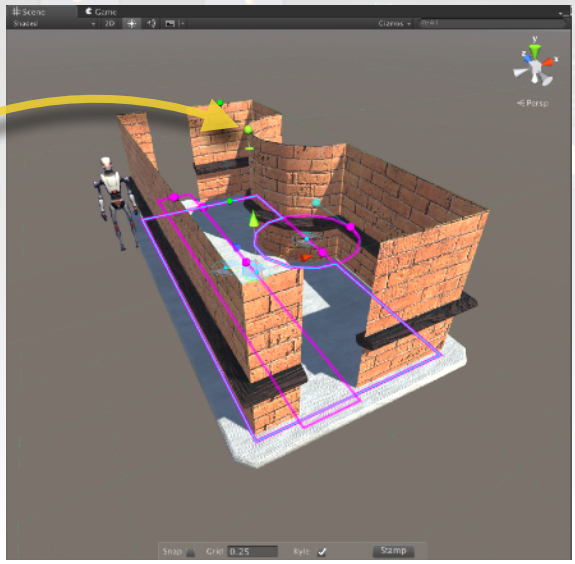
Step 6.27: On the *MaterialTool* node palette for the rail sides, click on the button to open the Unity material browser window. Choose *AX_BrickWall*. You can use the scale and shift parameters in the *MaterialTool* node Controls foldout to adjust the look.

Step 6.28: In a similar way, add materials to the *PlanSweep* node for the handrail and to the *Extrude* node for the floor slab, perhaps *AX_Wood01* and *AX_StriatedMarble*, respectively.



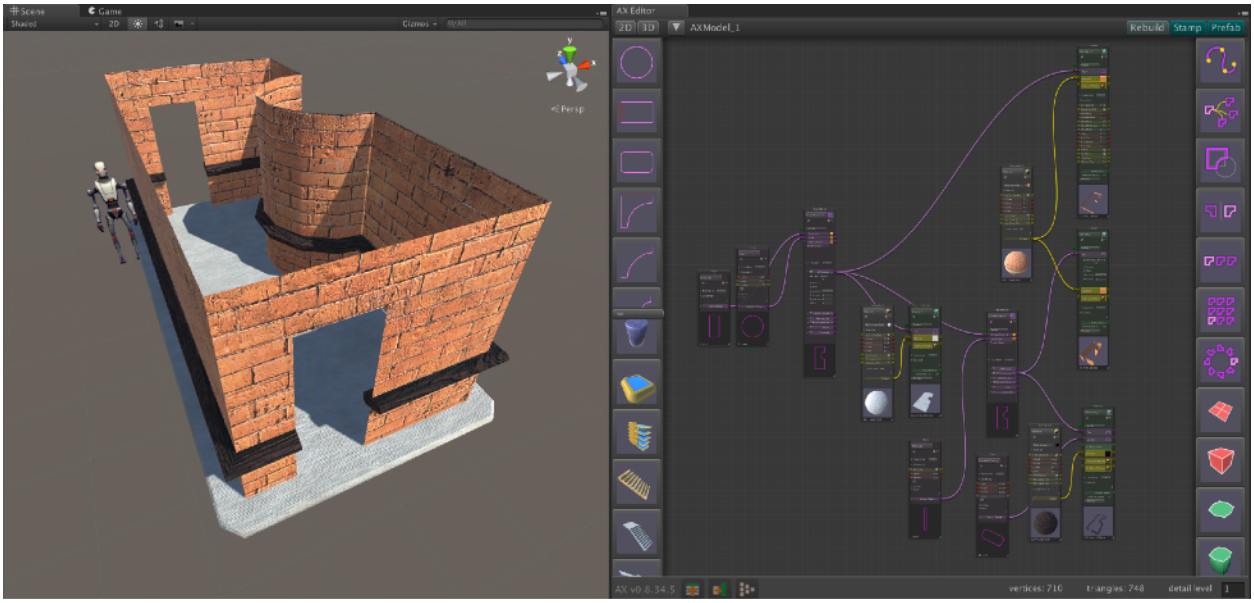
If we want to change this model to be more like a room with door openings, we can extrude the rail walls up.

Step 6.29: Click on the rail walls in the scene or on the *Extrude* node in the Node Graph Editor. Click and drag the green knob in the scene to increase the height of the wall.

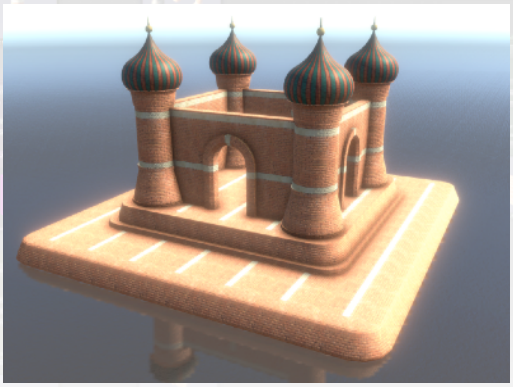


Step 6.30: Click on the *Difference* output of the first *ShapeMerger* and connect it to a new *Extrude* node from the right sidebar menu. Move the new *Extrude* node over near the rail walls' *Extrude*. Connect the output of the brick material to the new *Extrude*.

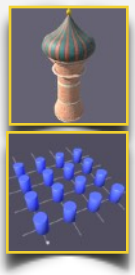
Step 6.31: Turn the *Top Cap* and *Bottom Cap* of the new *Extrude* off and the *Backfaces* on. In the scene, lift the new *Extrude* up until it is atop the rail walls. Adjust the height of the new *Extrude*.



Repeating Repeating

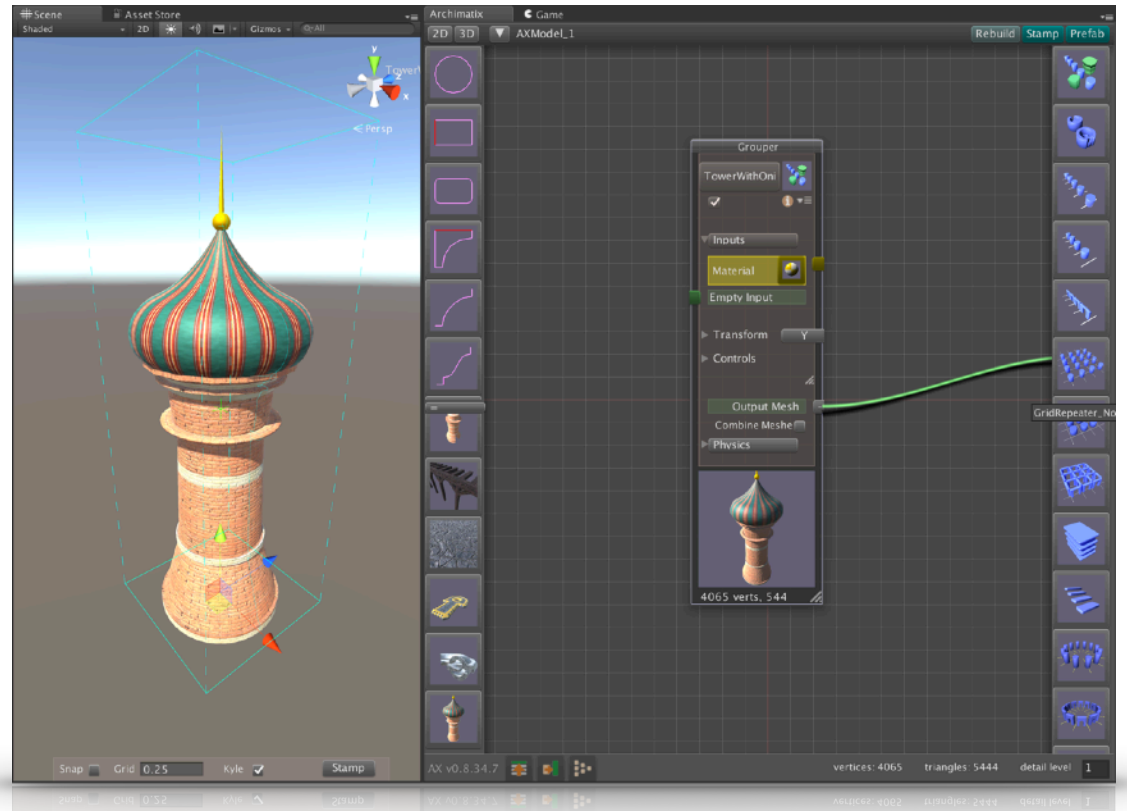


In Chapter 4, we looked at the Domino House, which used *Repeaters* to allow us to expand the building, adding floors and columns as we dragged the *Size* handles. Now, let's take a quick look at how to use *Repeaters* in the Node Graph Editor to make our own parametric building. To do this, we will use parametric props from the Library and connect them into a *GridRepeater* to make a sort of castle with arched openings.



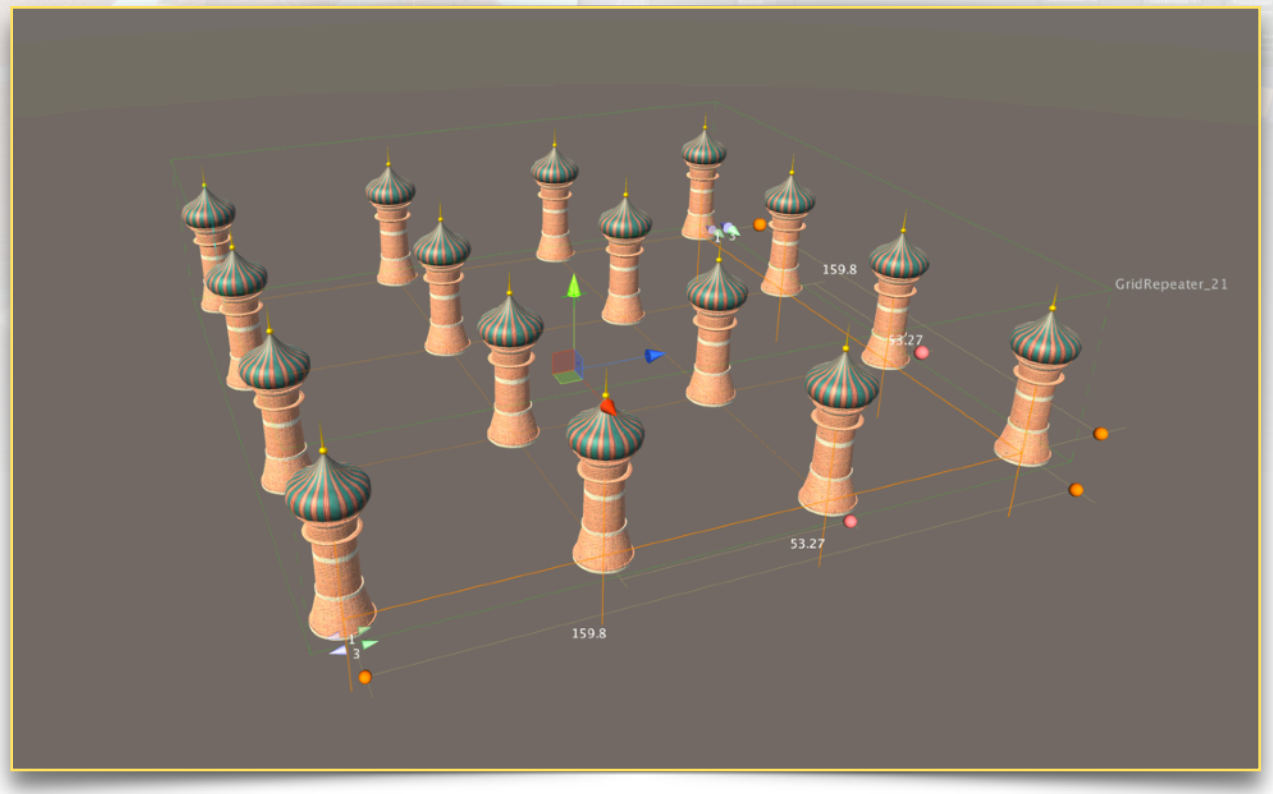
Step 6.32: In a new scene, instantiate a *TowerWithOnionDome* from the 3D Library left-hand sidebar.

Step 6.33: Connect the output of the tower to a *GridRepeater_Node* in the right-hand sidebar node menu.



Feel free to open the *TowerWithOnionDome Grouper* and see how it was assembled with its own set of nodes, including *FreeCurve*, *Lathe* and molding profiles. However, part of the beauty of *Grouper* and Library items is the ability to work with pre-formed parametric objects to give you a head start on environment building.

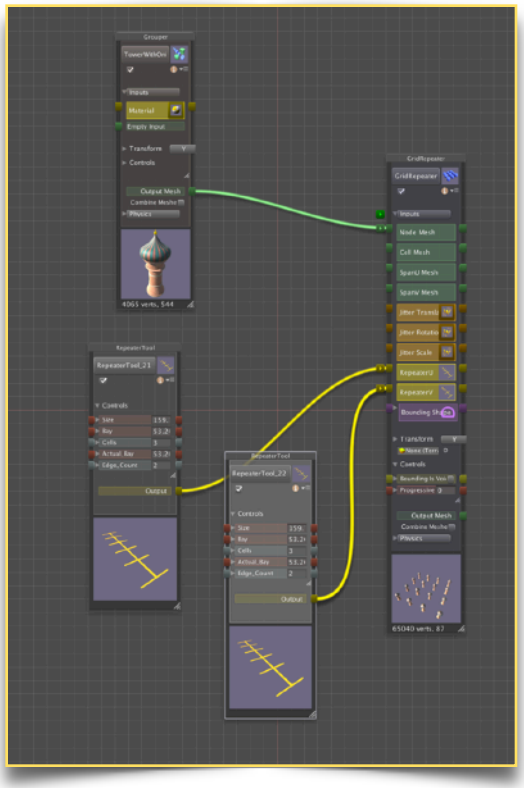
Step 6.34: Hit the F key to frame the grid of towers in the scene.



The default grid sizing is based on the size of the object that is initially connected to it. You can modify this sizing in the scene by dragging the orange point handles. The key parameters of the *GridRepeater* are the overall *Size* and the size of a typical *Bay*, or cell of the grid. Sizes in the X and Z axes are controlled independently.

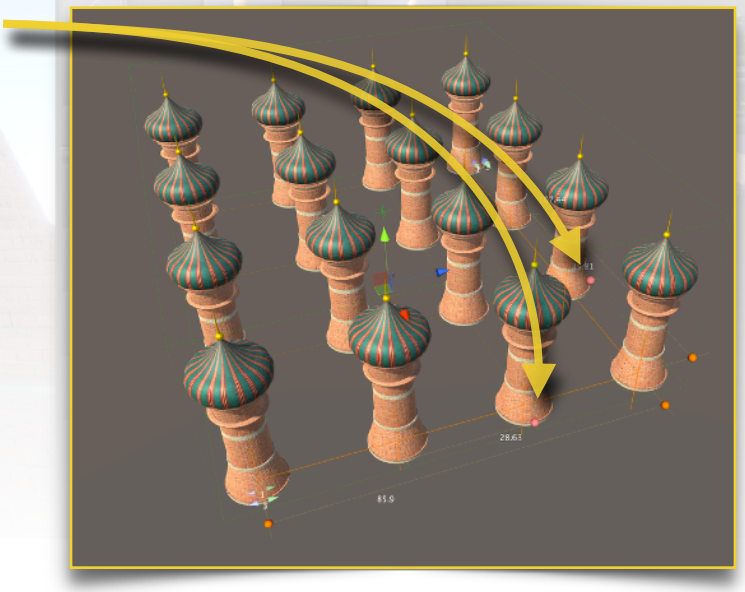
When you adjust the *Bay* size, the *Size* increases. When you adjust the *Size*, the additional bays are added, while the *Actual_Bay* size remains as close as possible to the *Bay* that had been set.

The parameters are not found in the *GridRepeater* node, but in *RepeaterTool* nodes that feed into it. For the *GridRepeater*, there are two *RepeaterTool* nodes for the two axes. These tool nodes were added to your graph automatically when you created the *GridRepeater*.



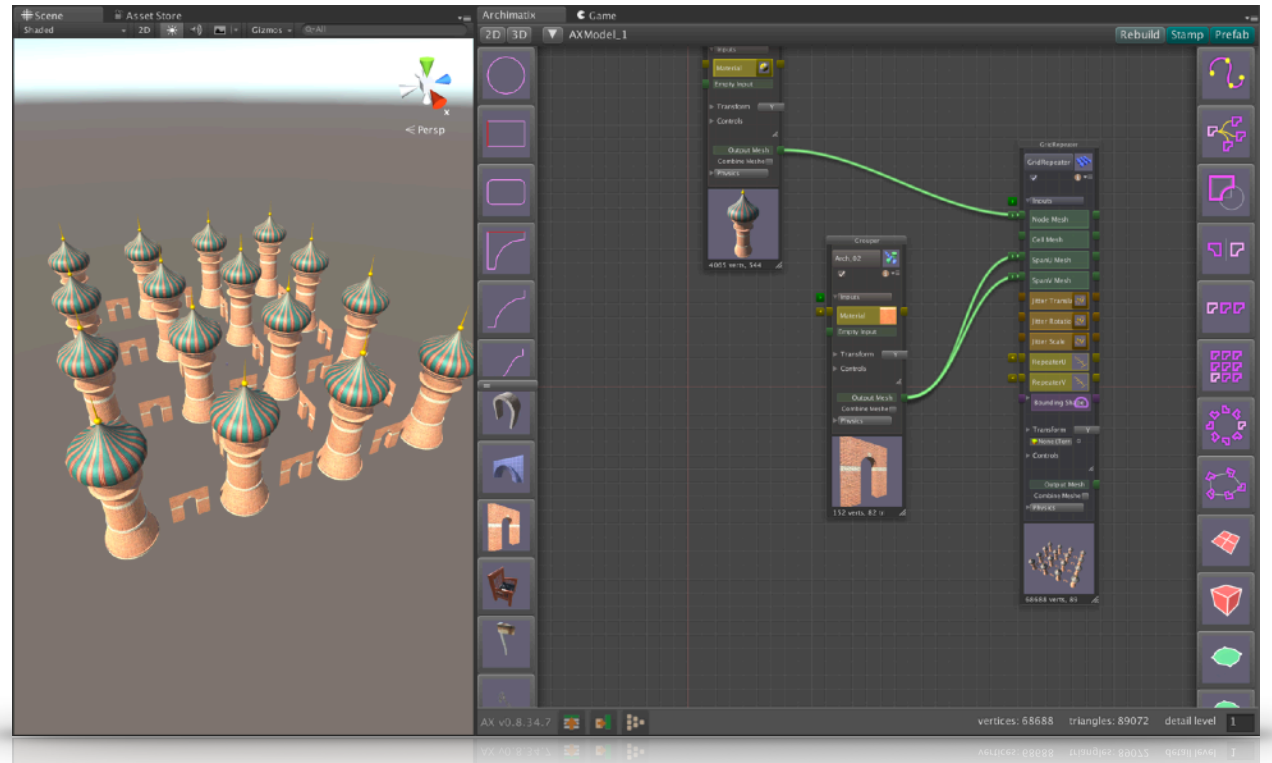
Let's add some walls to our castle.

Step 6.35: Adjust the *Bay* handles to bring the towers closer together so that the arrangement looks like the image to the right.

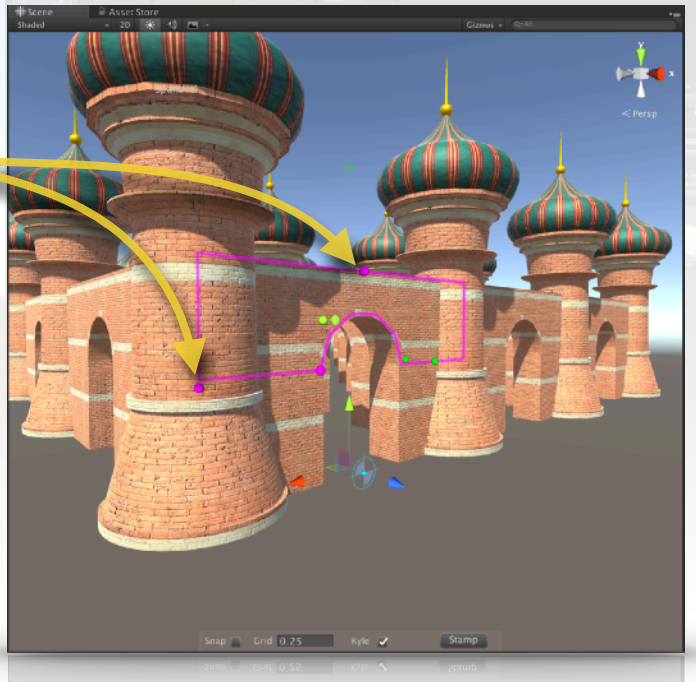


Step 6.36: Instantiate an *Arch_02* object from the 3D Library.

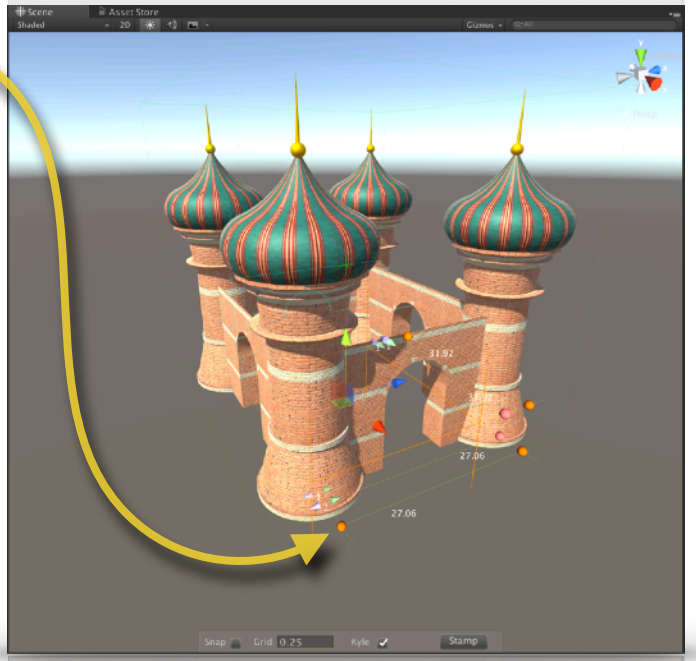
Step 6.37: Connect its output to both the *SpanU Mesh* and the *SpanV Mesh* inputs of the *GridRepeater* node.



Step 6.38: Click on any one of the arches in the scene and adjust its size to fill the spaces between the towers.

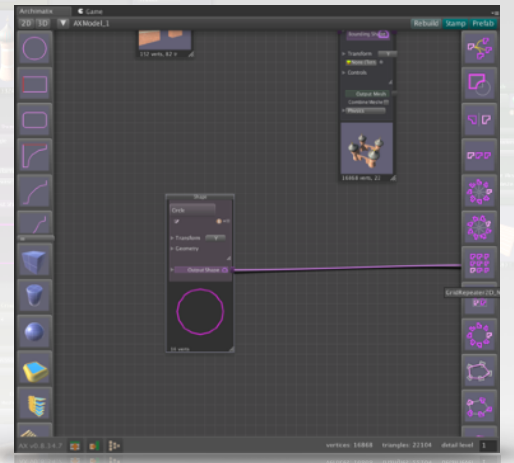


Step 6.39: Reduce the *Size* of the *GridRepeater* in both directions until there are only four towers remaining.



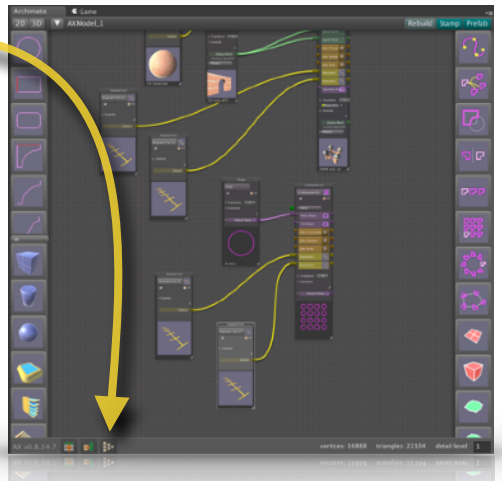
Arch_02 does not come with a top piece, since it is intended to be an infill object. Using 2D *Shapes* and the same *RepeaterTool* nodes, we can make a top piece with a molding profile, as well as a base.

Step 6.40: Instantiate a *Circle* object from the 2D Library.

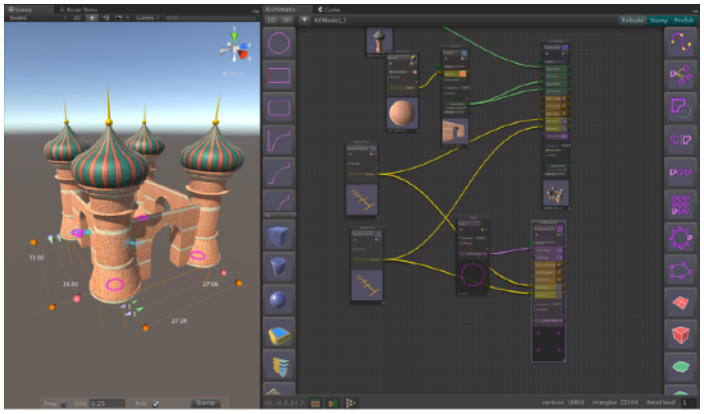


Step 6.41: Connect the output of the *Circle* to a *GridRepeater2D* from the right-hand sidebar node menu.

Step 6.42: Click on the "Show All Nodes" button in the lower left-hand corner of the Node Graph Editor window. Arrange the *RepeaterTool* nodes to look like the image to the right.



Step 6.43: Delete the two *RepeaterTool* nodes that are feeding into the *GridRepeater2D* and connect the ones that are feeding into the *GridRepeater* into the *GridRepeater2D*.

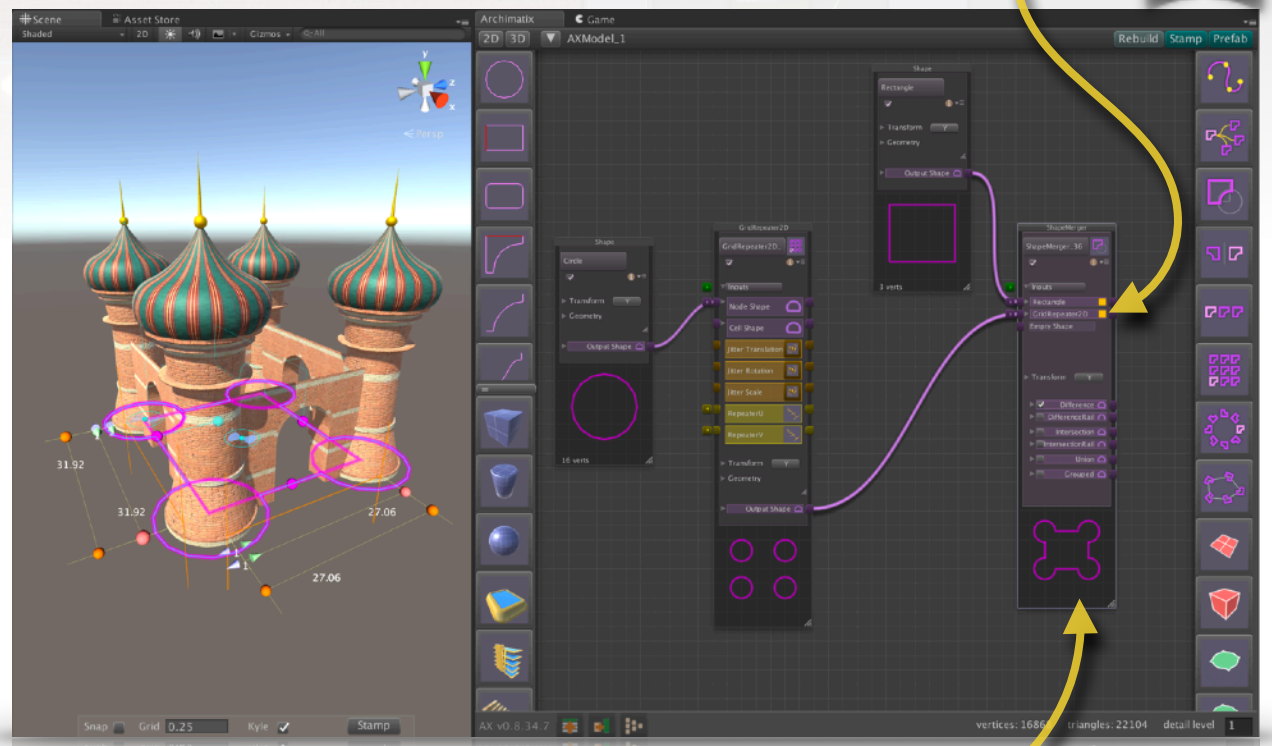


Now the circles will always be positioned where the towers are, regardless of how you modify the grid.

Step 6.44: Resize the *Circle* that is feeding into the *GridRepeater2D* so that it is slightly bigger than the base of the tower.

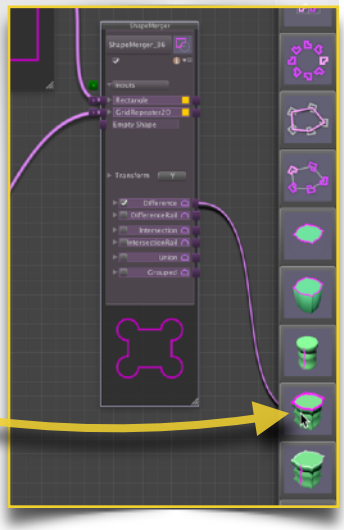
Step 6.45: Instantiate a *Rectangle* from the 2D Library left-hand sidebar and size it so that its corners are under the towers.

Step 6.46: Multi-select the *GridRepeater2D* and *Rectangle* nodes and then click on the *ShapeMerger* node in the right-hand sidebar. Click on the *Solid/Void* toggle button to make the circles solid.

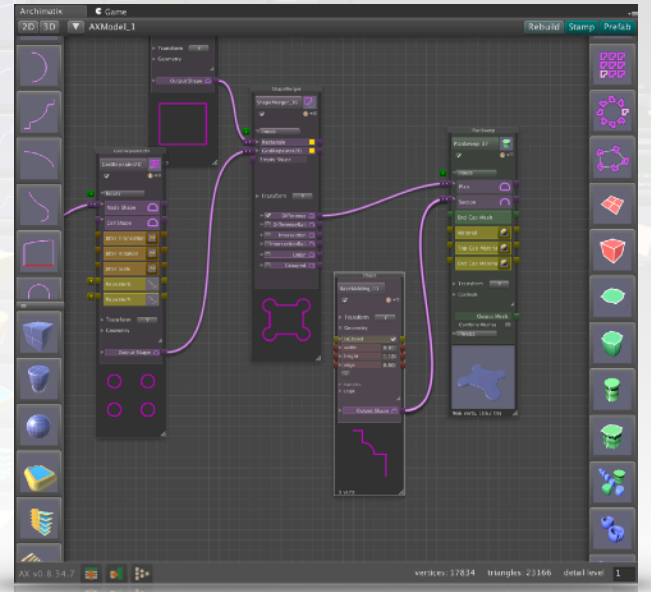


We now have a composite *Shape* that is the outline of the castle in plan. This *Plan Shape* will generate the base and top walkway for the castle.

Step 6.47: Connect the output of the *ShapeMerger* to a *PlanSweep_Plan* node from the right-hand sidebar menu.

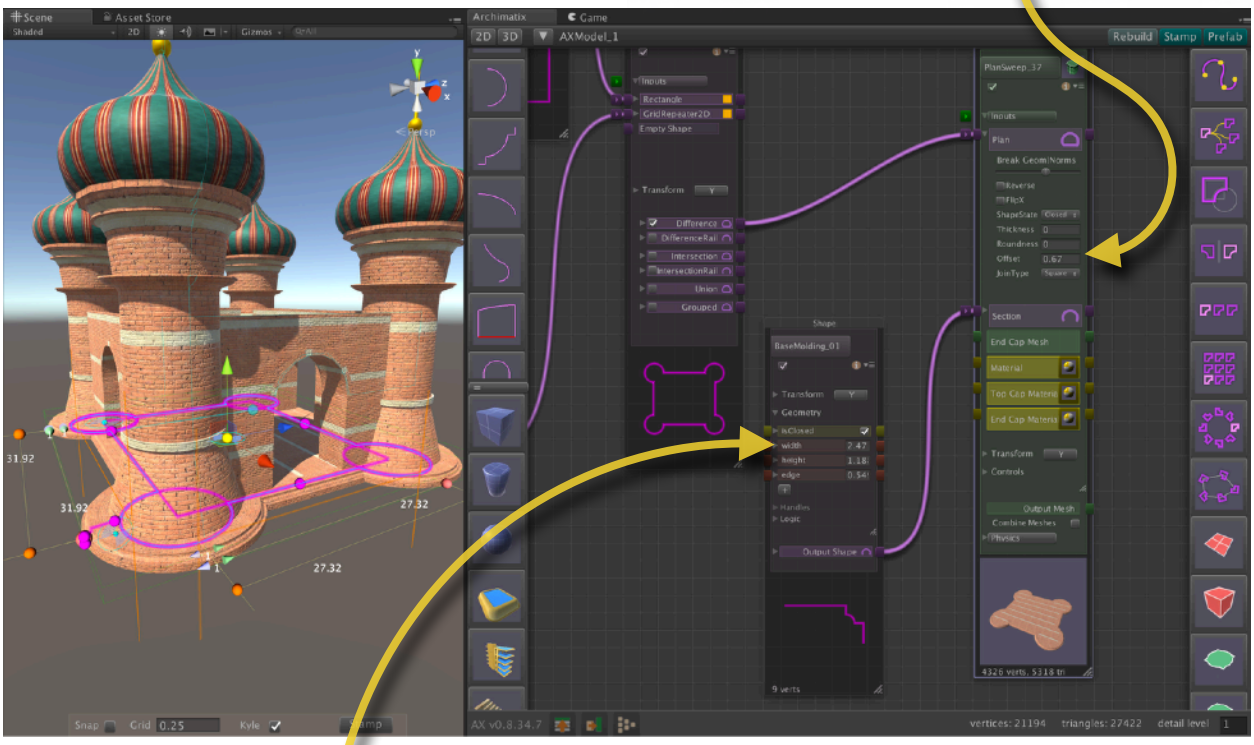


Step 6.48: Create an instance of *BaseMolding_02*. Connect it to the *Section* input of the *PlanSweep* node.

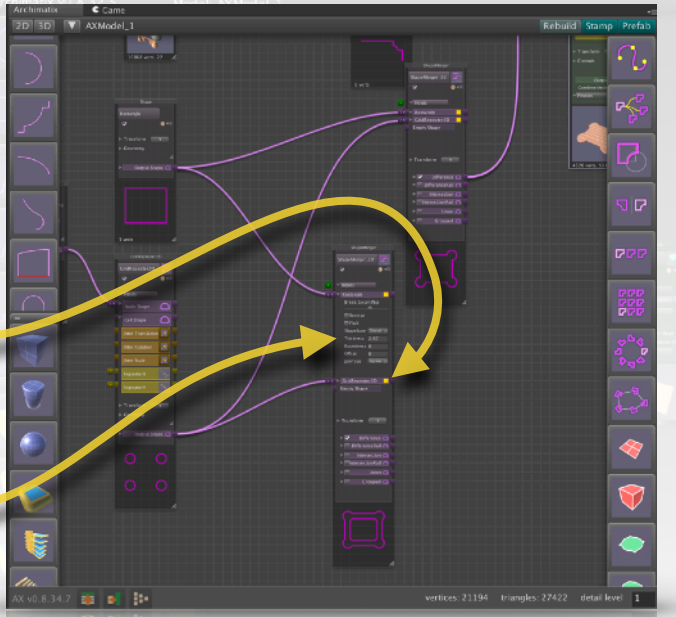


Step 6.49: In the Inspector, set the *Material (Default)* to *AX_BrickWall*.

Step 6.50: Open the *Plan* input foldout of the *PlanSweep* and adjust the *Offset* until the *PlanSweep* object looks like the image below.



Step 6.51: Adjust the *width* of the *BaseMolding_02* node to add to the parallel edge.



Step 6.52: Multi-select the *Rectangle* and the *ShapeMerger* with the four *Circles* again, and click on the *ShapeMerger* icon in the right-hand sidebar menu. Click on the *Solid/Void* toggle button to make the circles solid.

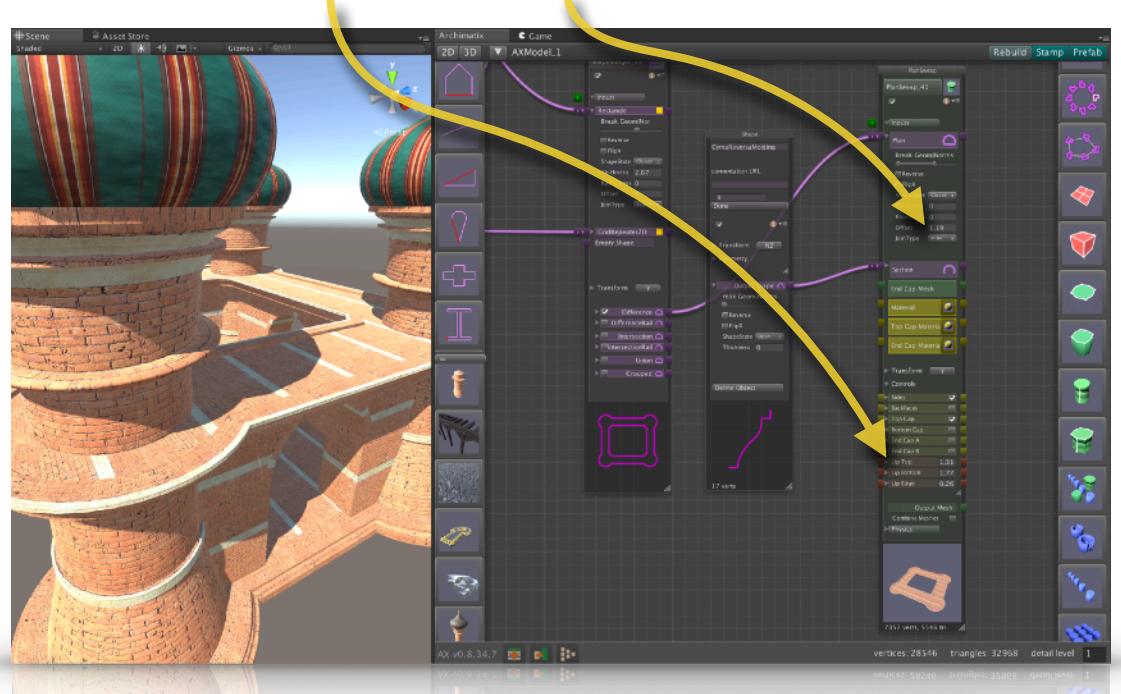
Step 6.53: Open the *Plan* input foldout and set a thickness for the incoming *Rectangle*.

We now have a *Shape* to use for the upper walkway and parapet. For this we will use another *PlanSweep*.

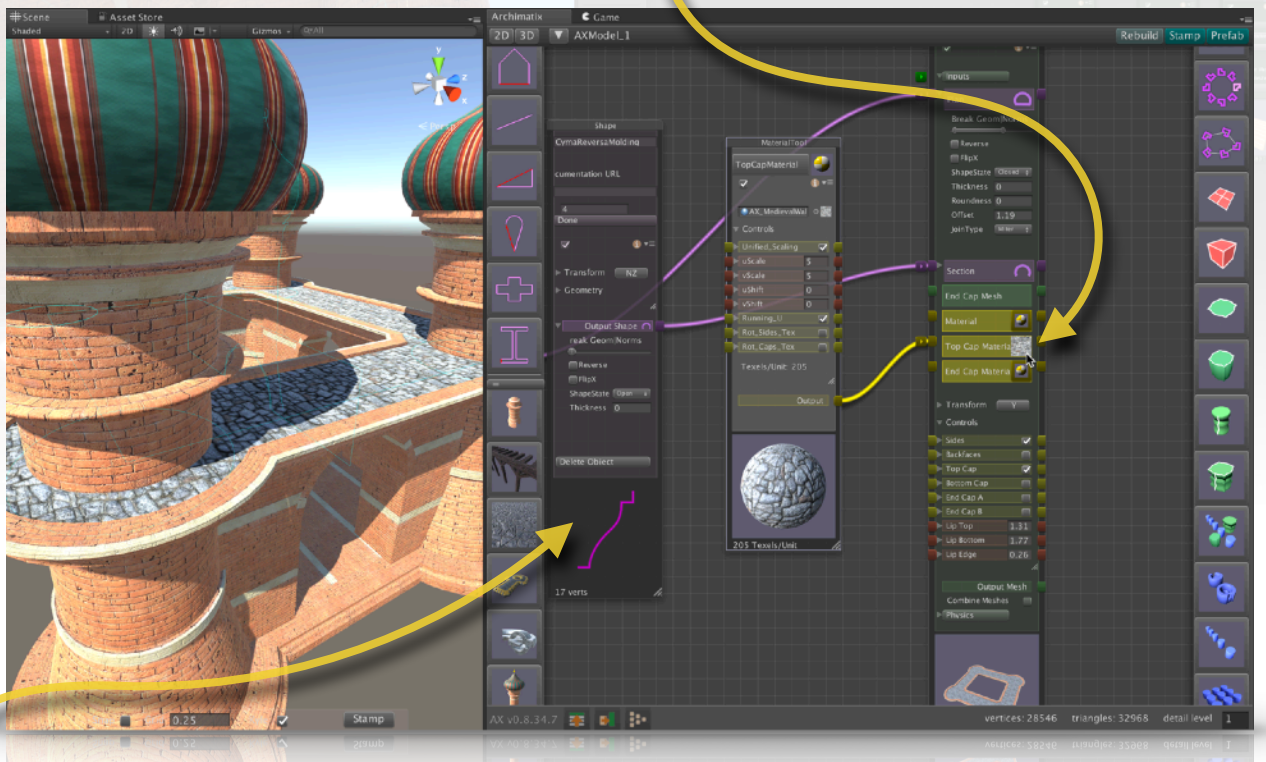
Step 6.54: Connect the new *ShapeMerger*'s *Difference* output to a *PlanSweep_Plan* from the right-hand sidebar node menu.

Step 6.55: From the left-hand sidebar, select the *CymaReversaMolding* profile *Shape* and connect its output to the *Section* input of the new *PlanSweep*.

Step 6.56: Select the *PlanSweep* object in the scene and move it up above the arch walls. Adjust the *Lip Top*, *Lip Edge* and *Offset* of the *PlanSweep*.

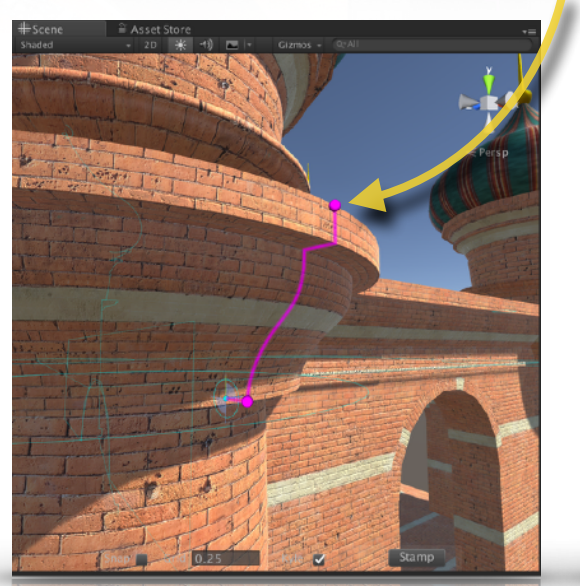


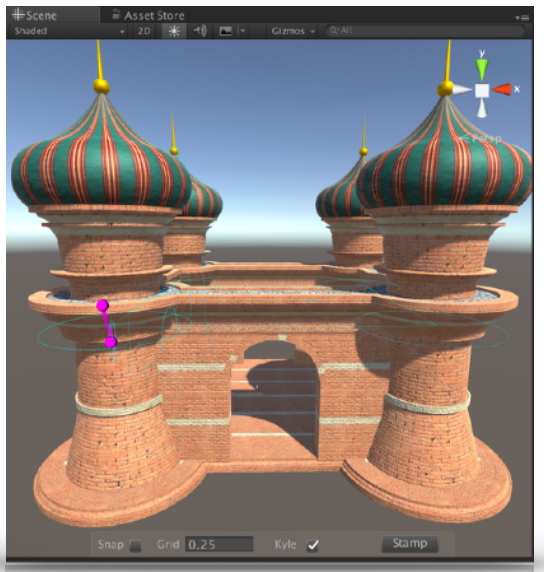
Step 6.57: Add a new *MaterialTool* to the *Top Cap Material* of the new *PlanSweep*. Set the material to *AX_MedievalWall*.



The *CymaReversaMolding* is a little small for the massing of this castle. Let's make it a bit more muscular so that it can hold up some sizable battlements.

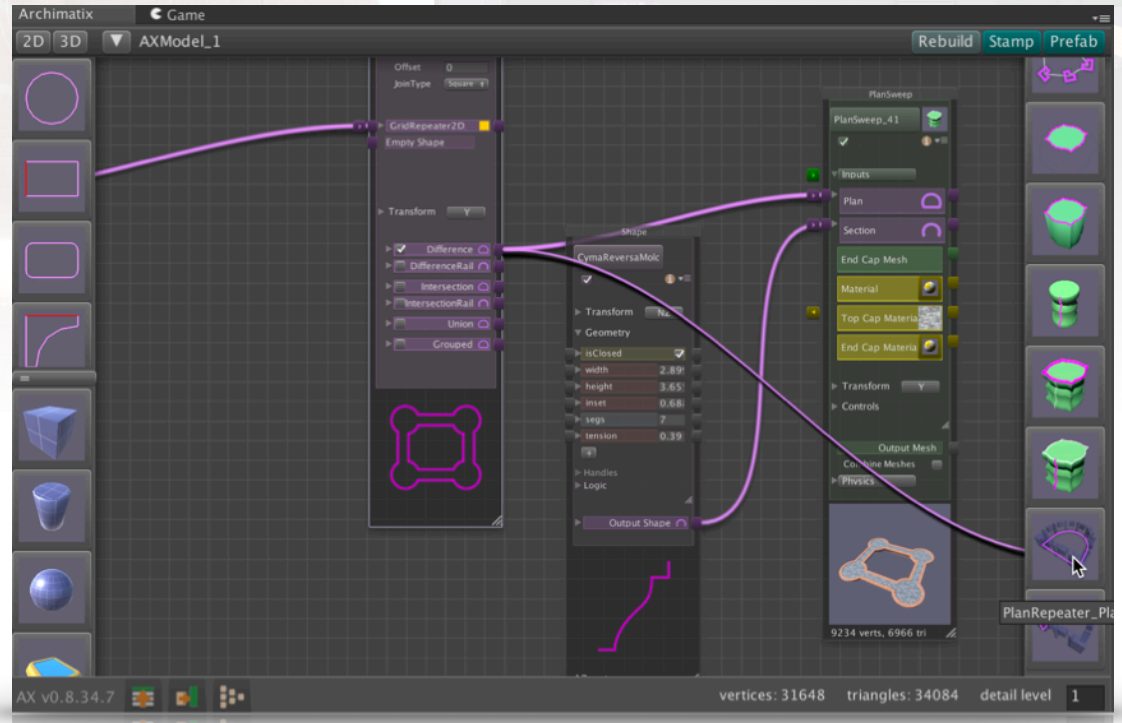
Step 6.58: Select the *CymaReversaMolding* node in the Node Graph Editor and hit the F key to frame it in the scene. Pull the upper handle upwards and outwards.



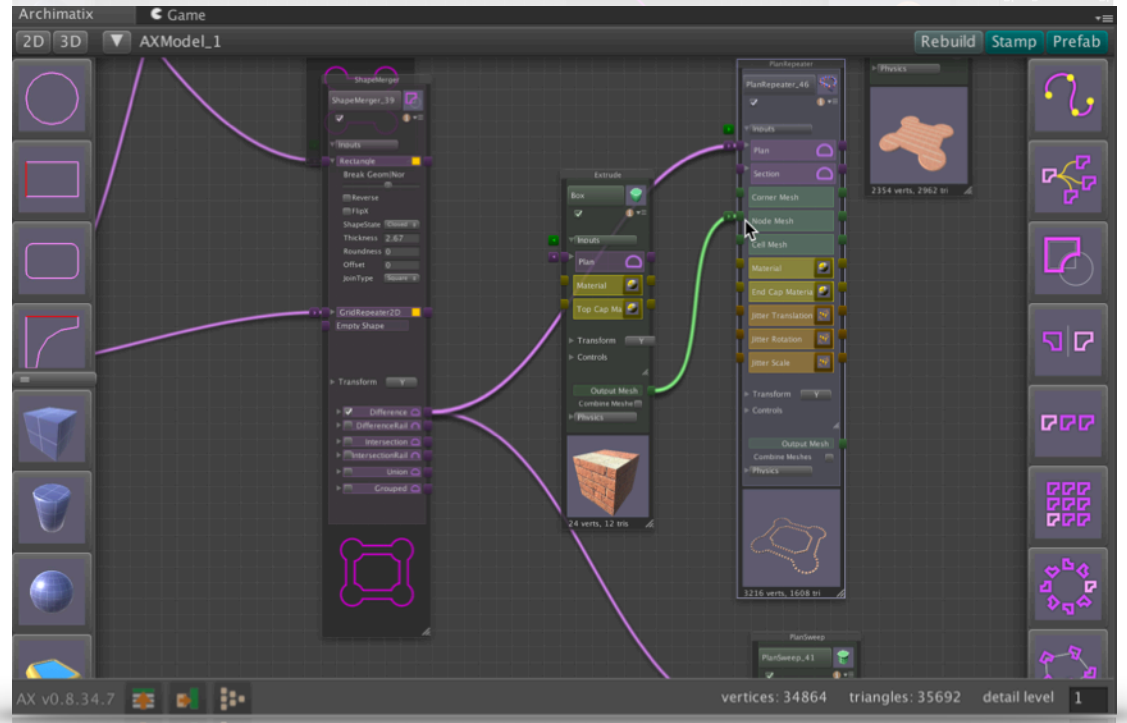


As a final step, let's add some battlements on top.

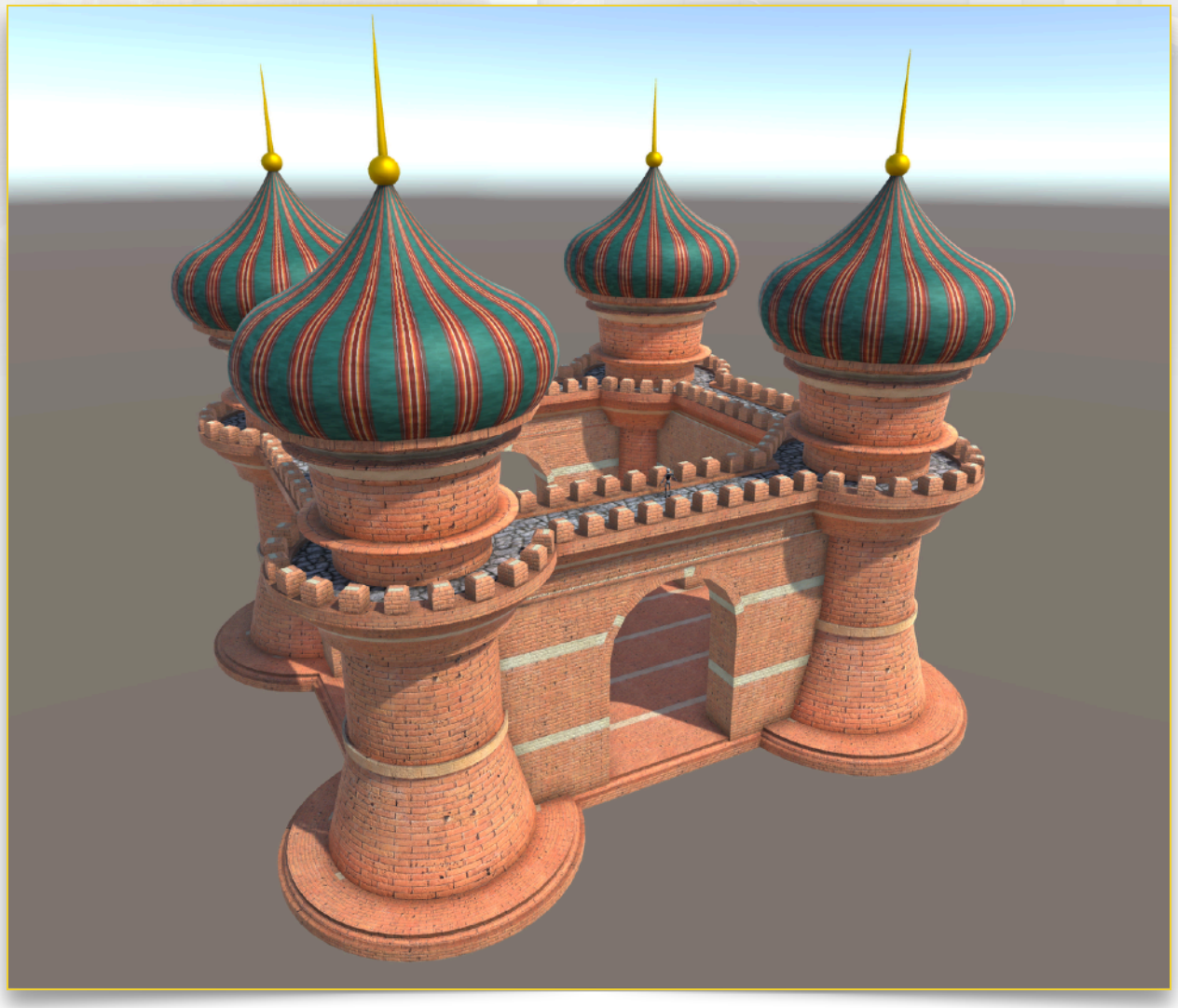
Step 6.59: Connect the output of the upper deck *ShapeMerger* to a *PlanRepeater_Plan* node in the right-hand sidebar menu.



Step 6.60: Instantiate a *Cube* by clicking on it in the left-hand 3D Library menu.



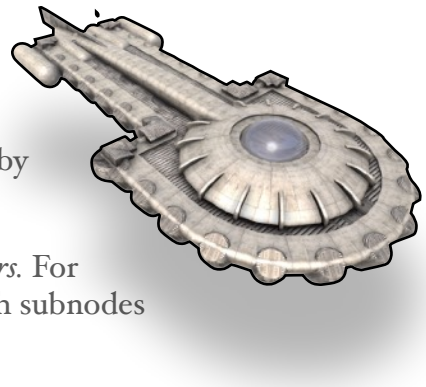
Step 6.61: Select the new *PlanRepeater* and lift it up in the Y-axis until it is sitting atop the upper deck. Open the *Plan* input foldout in the *PlanRepeater* node and adjust the *Offset* to move the battlements back a little. Select the *Cube* node and turn off *Bevels Unified,* then set the *Top Bevel* to about .15.



For the final shot above, *Ambient Occlusion* was added to the scene's *Main Camera* from Unity's *Cinematic Image Effects*, a free download from the Asset Store.

As our graphs become richer and more complex, it is clear that we could benefit from a node that groups and hides portions of the graph. Fortunately, Archimatix ships with a node that does just this. In the next chapter, we will take a look at the *Grouper* node.

7. Group Dynamics



As your parametric models become richer and more complex, you can use the powerful *Grouper* node to organize the Archimatix graph into a series of nested subgraphs. Each *Grouper* is a self-contained parametric object that has an interface defined by the *Grouper's* input parameters.

Many of the Library items you have worked with so far are *Groupers.* For example, the *RicketyStaircase* and the *ScifiPlatform* are *Groupers* with subnodes that have been saved to the Library.

A *Grouper* node may be thought of as a container that can hold other nodes, including other *Grouper* nodes, inside of it. Once a node has been placed inside a *Grouper*, it cannot have a direct connection with another node outside of the *Grouper*. Instead, a proxy parameter can be added to the list of the *Grouper*'s parameters to relay the connection between nodes inside and outside of the *Grouper*.

To initiate a *Grouper*, you can multi-select nodes and then click the *Grouper* icon in the right sidebar of the Node Graph Editor window. The selected nodes will disappear as the new *Grouper* node appears in the graph. To see the nodes inside, double-click on the *Grouper* node, expanding it. Nodes can be popped back out of the *Grouper* by dragging them to the left of the *Grouper* node. Nodes can be added to a *Grouper* by dragging and dropping them over the *Grouper* node.

Perhaps the best way to understand node group dynamics is to create a group.

Let's use the Node Graph Editor to create *GrayShip,* a model that is something like the *SciFiPlatform* we instantiated from the Library in Chapter 2 of this guide. In this example, we will be creating a parametric spaceship that can be used to stamp out dozens of variations to form an entire fleet.

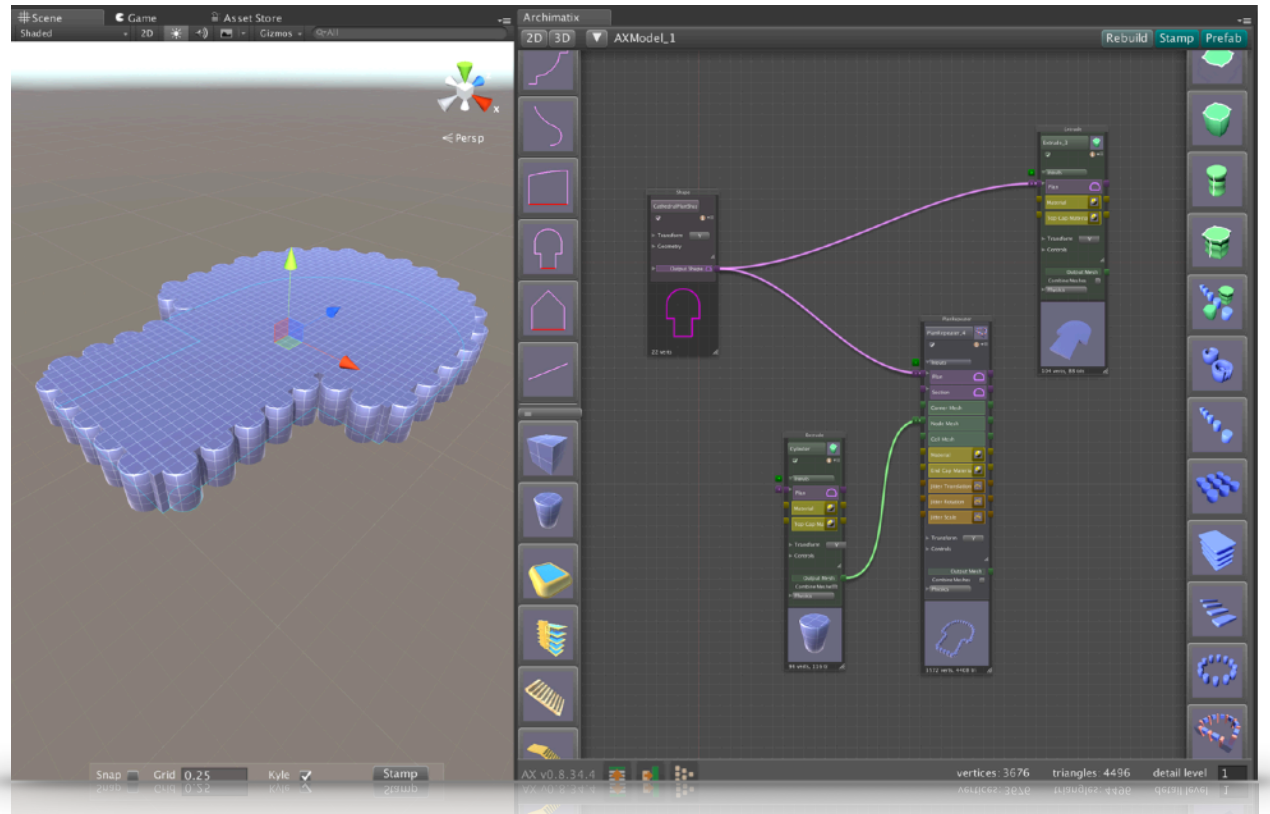
Step 7.1: In a new scene, instantiate the *CathedralPlan* from the 2D library by either clicking on its icon in the left sidebar of the Node Graph Editor window or by finding it in the Library window.

Step 7.2: Extrude the *CathedralPlan* by connecting it to the *Extrude Mesher* icon in the right sidebar.

Step 7.2: Make the *CathedralPlan* the *Plan* input of a *PlanRepeater* by connecting it to the *PlanRepeater* icon in the right sidebar.

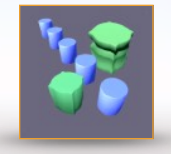
Step 7.4: Instantiate a *Cylinder* from the 3D Library and connect it to the *Node Mesh* input of the *PlanRepeater*.

At this point, your graph should look similar to the one in the image below.

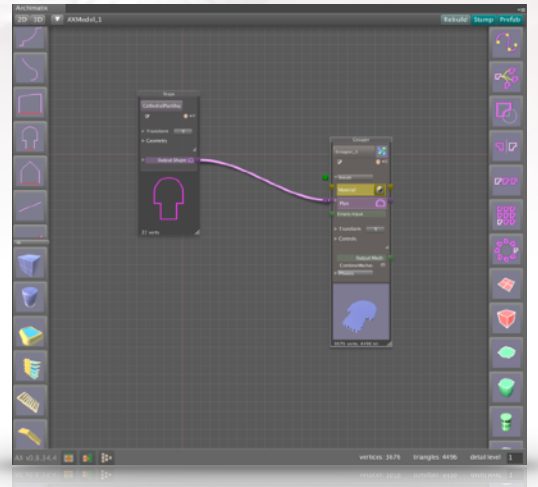


Lets jump in and group some of these nodes. It will be useful to leave the *CathedralPlan* outside of the group, allowing us to have a spaceship model that can take different *Shapes* as *Plan* input.

Step 7.5: Multi-select the *Cylinder*, *Extrude* and *PlanRepeater* nodes (but not the *CathedralPlan Shape*) and then click on the *Grouper* icon in the right sidebar. Multi-selection can be done by either shift-clicking the nodes or by dragging a rectangle that includes the nodes to select.



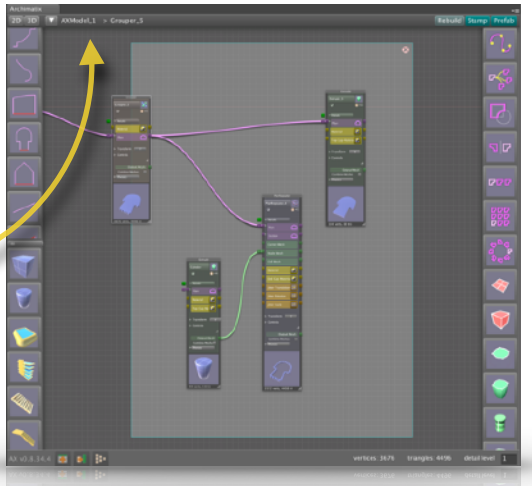
As soon as you clicked the *Grouper* icon, the node graph became visually simpler. All the nodes are still in the graph, but they are now organized in such a way as to logically hide a portion of the graph.



The *Grouper* serves as a parametric object whose internal implementation is encapsulated. If we replace the *CathedralPlan* with any other *Shape*, we will have an *Extrude* surrounded by *Cylinders* conforming to the new *Shape*.

The *Plan* input parameter was added automatically to the *Grouper* to serve as a proxy to relay the connection between the external *Shape* and any items inside the *Grouper* that may want to use it.

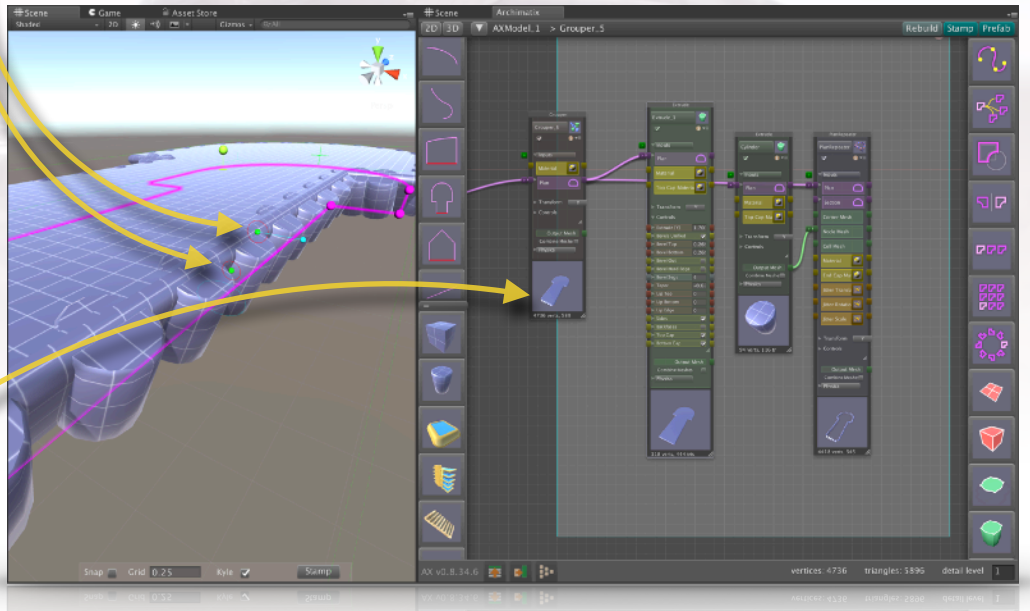
Step 7.6: Double-click the thumbnail at the bottom of the *Grouper* node to expand it.



Looking inside the *Grouper*, we can see that the *Plan* parameter has been automatically connected to the inputs of the nodes that were originally connected directly to the *CathedralPlan Shape*.

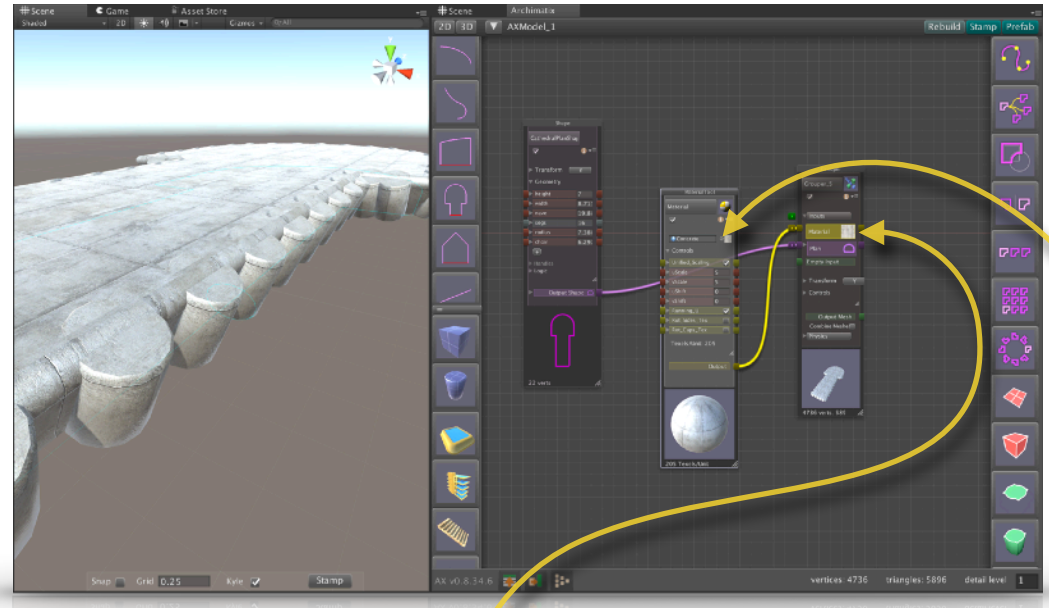
Notice the breadcrumb trail at the top of the window. Since *Groupers* can be nested, these links will allow you to see where you are and to click on any "upstream" *Groupers* to pop up to the previous grouping level. You can also pop back up a level by double-clicking again on the *Grouper* node thumbnail.

Step 7.7: In the scene view, select the *Extrude* and find the green point handles along the side. Adjust the top handle to give the *Extrude* a taper. Use the lower one to create a bevel. The cyan handle coming out from the top green point adjusts the number of segments in the bevel.



Step 7.8: Select any instance of the *Cylinder* in the scene.Rotate it until it is parallel to the tapered side of the hull *Extrude*.

Step 7.9: Double-click the *Grouper* thumbnail to pop up a group (in this case, back to the model).



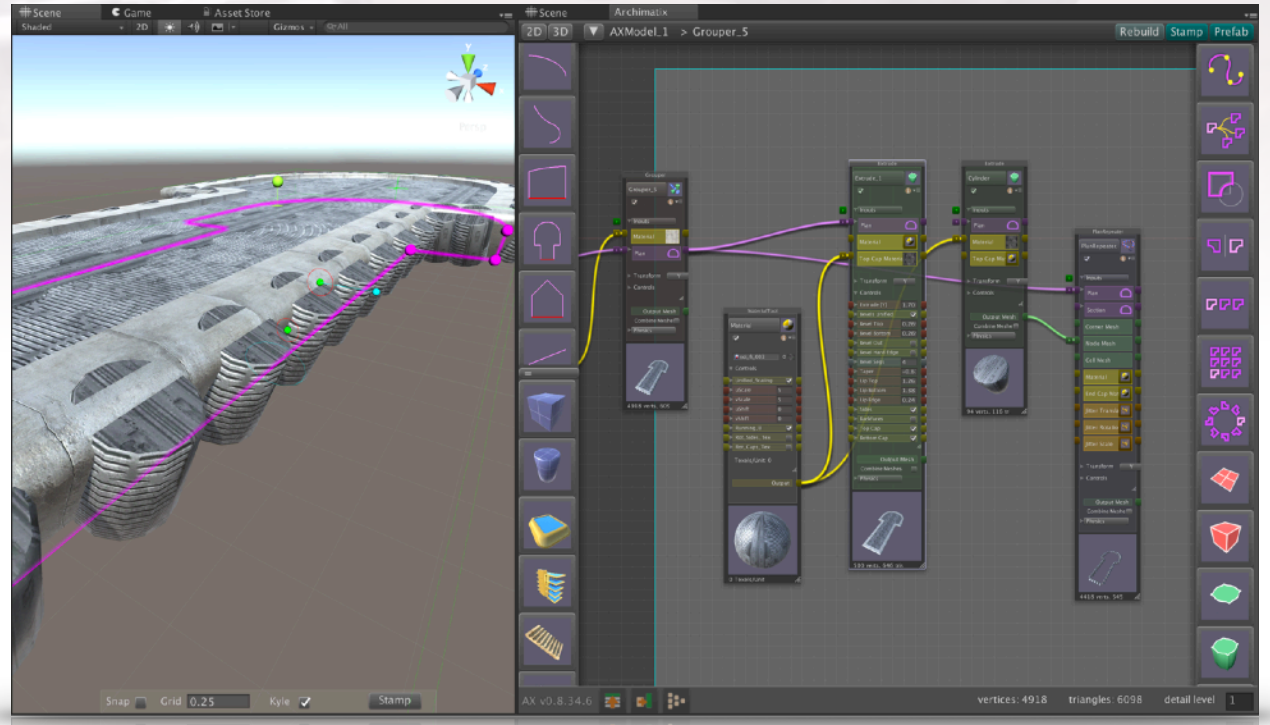
Step 7.10: Click on the *MaterialTool* button on the *Material* input for the *Grouper* to generate a *MaterialTool* node. Choose the Concrete material.

Step 7.11: Click on any *Cylinder* and hit the "f" key to frame the *Cylinder* in the scene and in the Node Graph Editor. Notice that this has automatically opened the *Grouper*.

Step 7.12: Add a *MaterialTool* node to the *Cylinder Extrude* and choose sci_fi_003.

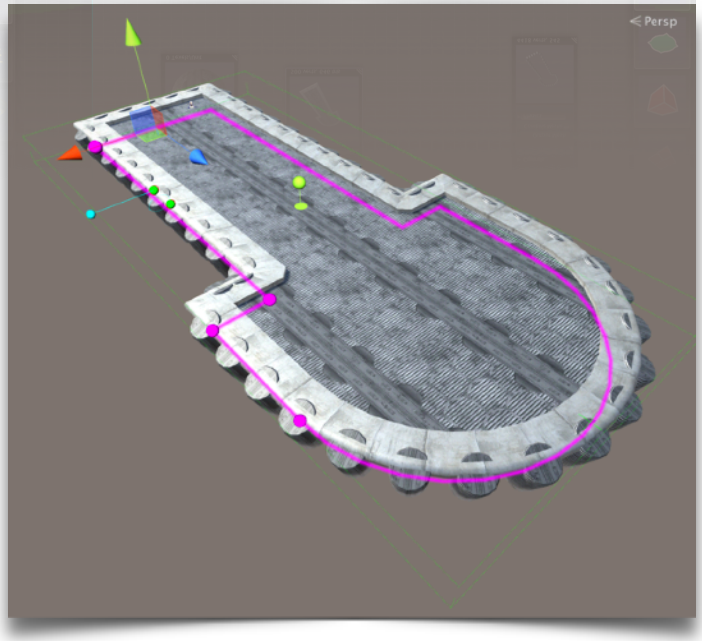
Step 7.13: Connect the new *MaterialTool* to the *Extrude's Top Cap Material*.

Step 7.14: Set the *Lip Top* of the *Extrude* to 1.2 and the *LipEdge* to 0.24.



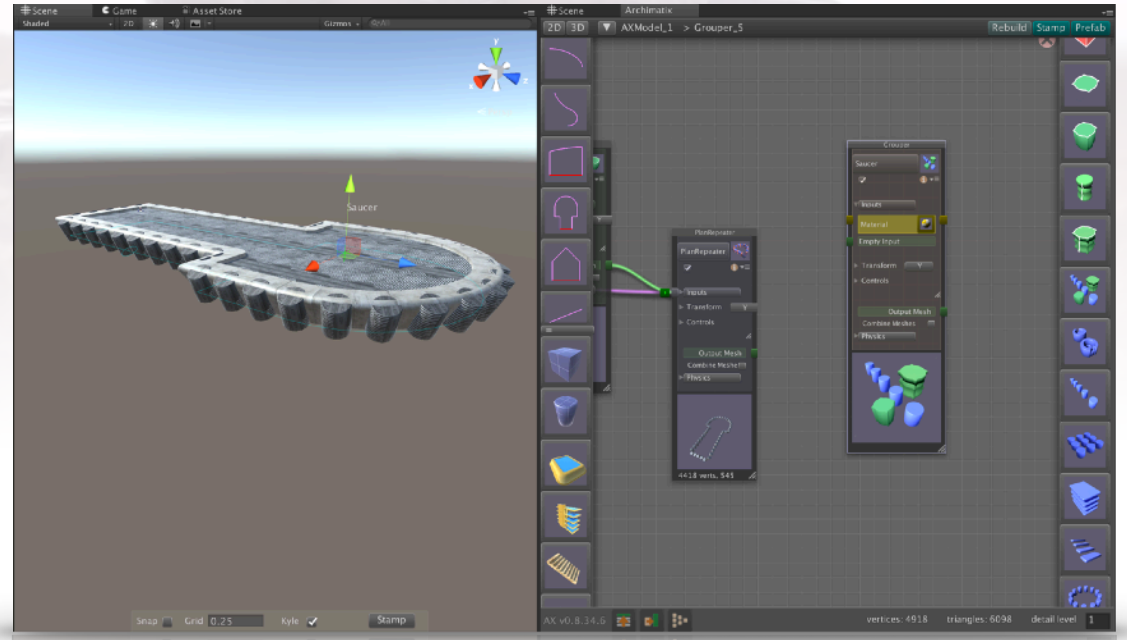
To make the round saucer superstructure, we will need to make several pieces that share a center. This shared center will be displaced from the origin of the model, which is currently at the tail of the ship.

Rather than making each part of the saucer and translating it, we can establish a new origin by creating a *Grouper* inside our first *Grouper* and translating that. Once we do that, any new nodes we create will be located relative to the position of the new *Grouper*.

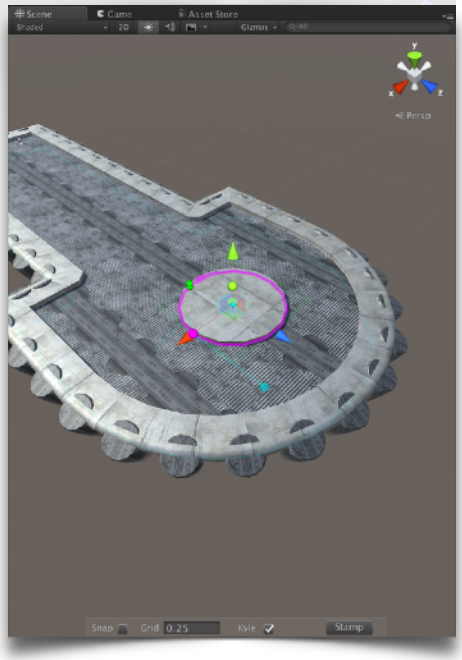


Step 7.15: With the first *Grouper* for the hull open, but with **nothing selected**, click on the *Grouper* icon in the right sidebar to create a new *Grouper* inside the first *Grouper*. Rename the new *Grouper* "Saucer."

Step 7.16: With the new *Saucer Grouper* selected, translate it to the front of the ship.



Step 7.17: Double-click the *Saucer Grouper*'s thumbnail to open it and then instantiate a *Cylinder* from the 3D Library sidebar. Decrease the *Extrude Height* of the *Cylinder* and increase its *radius* to make it serve as the deck for the bridge.

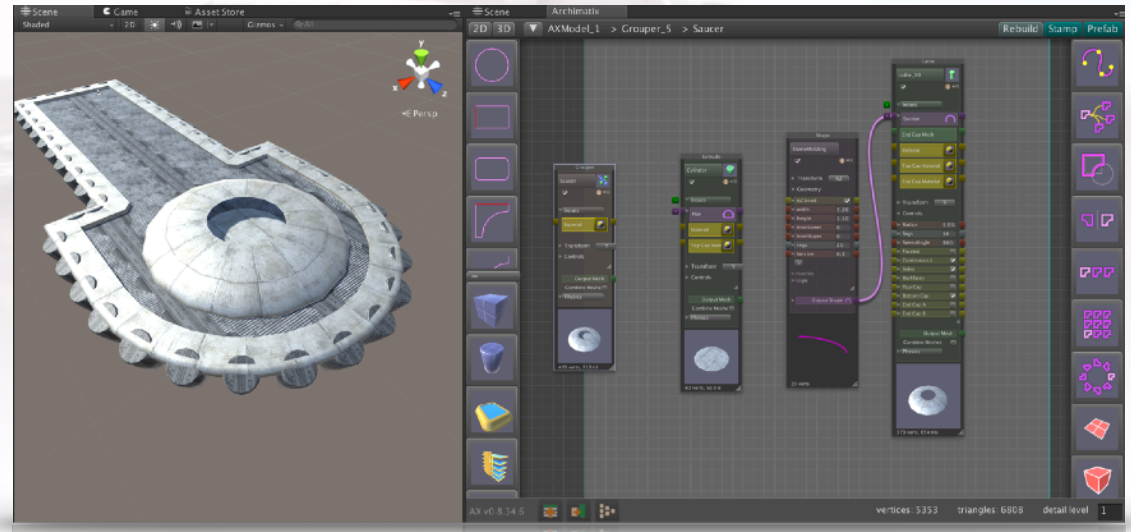


You will find the new *Cylinder* at the *Saucer Grouper*'s center. If you were to pop the *Cylinder* out of the Group by dragging and dropping the *Cylinder* node to the left of the *Saucer Grouper* node, then the *Cylinder* would immediately move to the center of the next *Grouper*, at the tail of the ship.

Thus we can see that *Groupers*, in addition to encapsulating subnode graphs, also serve as local reference planes to model in situ.

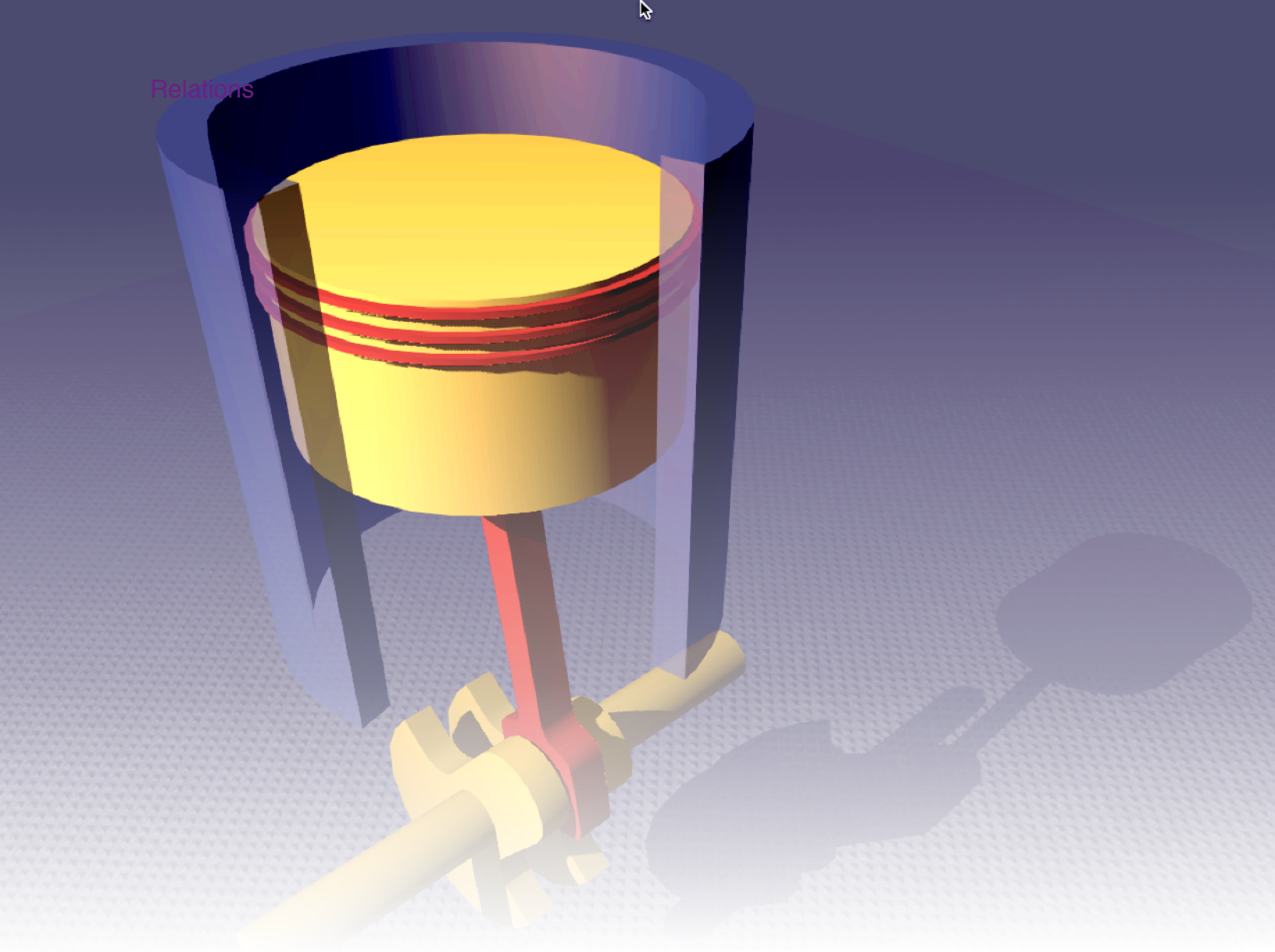
Let's add another piece of the saucer superstructure.

Step 7.18: Instantiate a *DomeMolding Shape* from the 2D Library sidebar. Connect it to a *Lathe Mesher* in the right-hand node sidebar. Turn the *Top Cap* of the *Lathe* to off.



This is as far as we'll go with *GrayShip* in this User Guide. While *GrayShip* helped us understand the *Grouper* node, it has more to teach us! To complete *GrayShip*, please visit the full tutorial video at the Archimatix support site.

Robot Kyle can hardly wait to take the helm of his new GrayShip*!*



8. Getting to Know Your Relations

Parts of a parametric model often need to be related to each other in mathematical terms. The ability of such expression-based relations is important for defining logic that governs the behavior of a model as you vary parameters. Architectural proportions and the interrelationships of machine parts depend on such logic.

Archimatix lets you relate any parameter to any other parameter in a graph using a mathematical expression. For example, if you wanted to set the vertical position of a *Cylinder* to always equal the height of a *Box*, then you would essentially be saying that the *Cylinder* should sit on top of the *Box* as you vary the *Box* height.

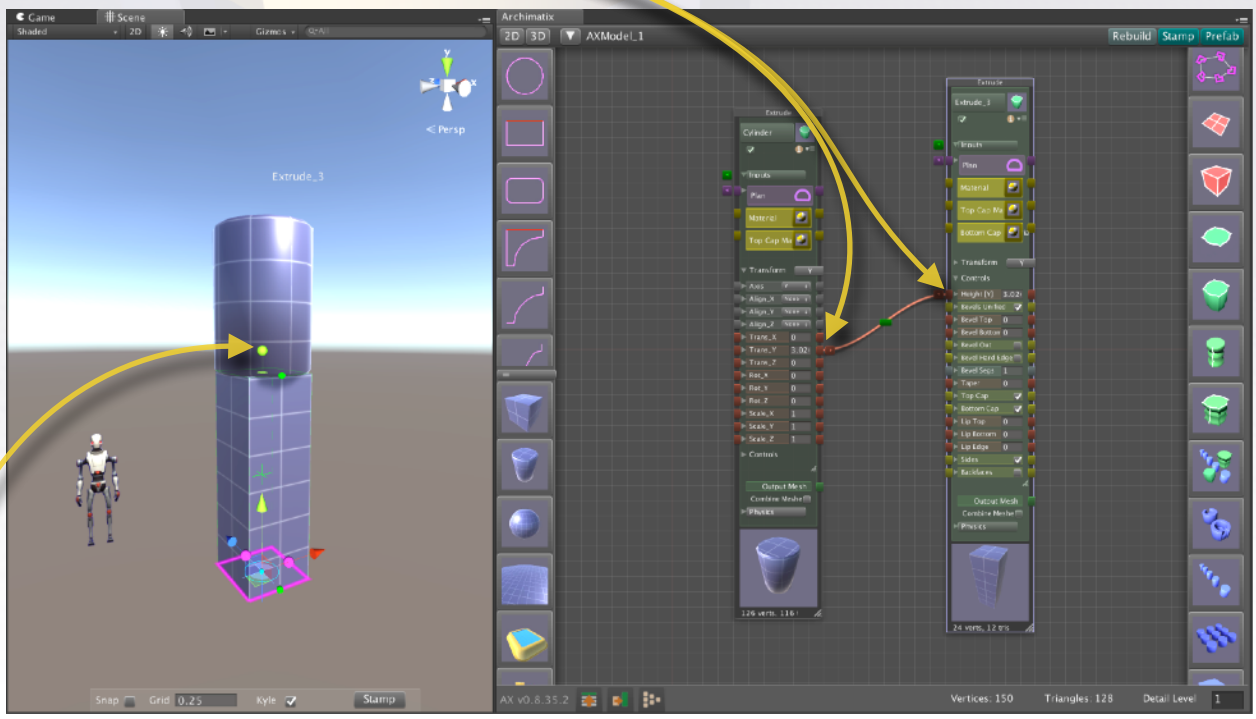
Relations in Archimatix are bi-directional. This means that, in the above example, you can not only alter the height of the *Box* and automatically translate the *Cylinder*, but you could also translate the *Cylinder* and automatically alter the height of the *Box*.

Why don't we go ahead and try this?!

Step 8.1: Start a new scene and add a *Cylinder* node and a *Box* node.

Step 8.2: Open the Transform foldout on the *Cylinder* and the Controls foldout on the *Box*.

Step 8.3: Click on the *Trans_Y* parameter of the *Cylinder* to start a new connector. Move the mouse to the *Height* parameter of the *Box* and click the input to complete the Relation connection.



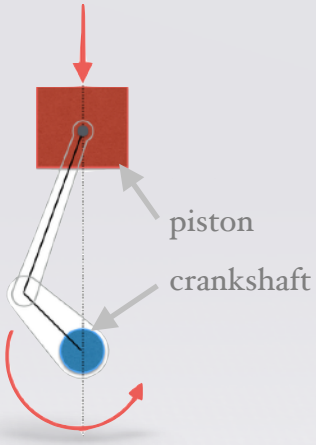
Step 8.4: Adjust the height of the Box either by sliding its *Height* parameter or by dragging the *Height* handle in the scene.

Remember, relations are bi-directional, meaning that the connector socket on either side of the parameter in the node palette can be chosen when connecting parameters. The bi-directionally of relations also means that you can adjust either end of the connection and the parameter on the other side of the connection will be modified according to the expression in the Relation.

The default Relation expression is equivalency, or "=". In the above example, the Relation equation is *Cylinder.Trans_Y=Box.Height*.

Let's create a Relation with a slightly more interesting expression.

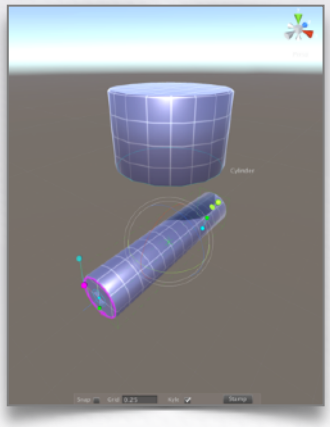
Cranking Up Relations



A good example of a relation between two objects is that of a piston to a crankshaft in an internal combustion engine.

Typically, as the piston moves up and down, the crankshaft rotates in a constant direction. Conversely, if the crankshaft rotates, it moves the piston up and down in a sinusoidal motion. Such a relationship calls for the use of a sine function in its Relation expression.

To make a simple model of an internal combustion engine, we will exclude the piston rod, representing the piston and crankshaft by two cylinders.

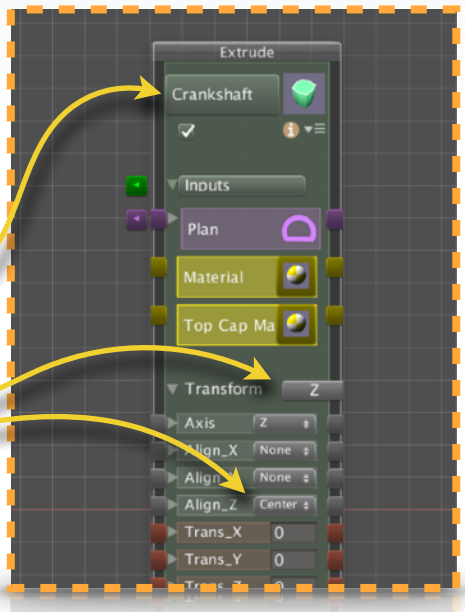


Also, for the purposes of a quick start, instead of using the trigonometry needed to find the more accurate relation of the piston and crankshaft based on the length of the piston rod, we will approximate the Relation with a variation of:

Piston.Trans_Y = sin(Crankshaft.Rot_Z)

For a more elaborate scenario using the actual geometry of a piston and crankshaft, see this tutorial at the Archimatix support site:

http://www.archimatix.com/tutorials/piston-and-crankshaft



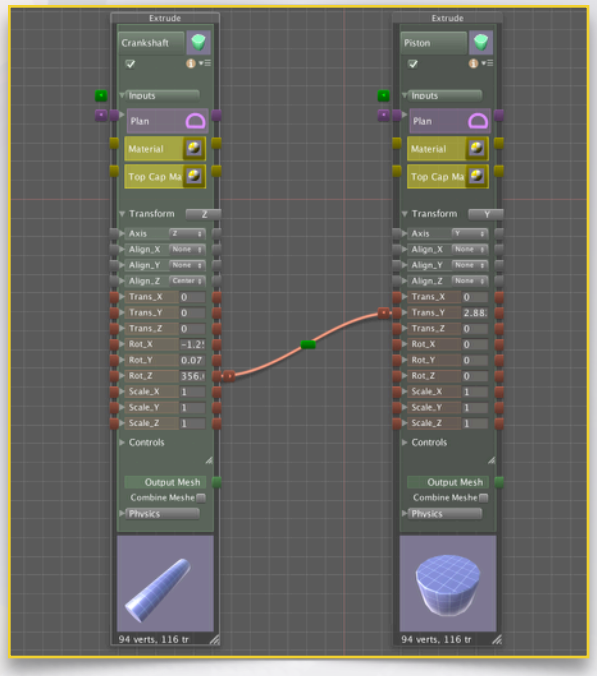
Step 8.5: In a new scene, choose the *Cylinder* item from the left sidebar menu. Click on the node name button and change the name to ***Crankshaft***.

Step 8.6: Click the axis orientation button until it is set to Z. Open the Transform foldout and set the *Align_Z* to Center.

Step 8.7: Change the material of the model to AX_PurpleGrid. In the Scene View, use the handles to adjust the length of the Crankshaft and its *radius* so that it looks like the image above right.

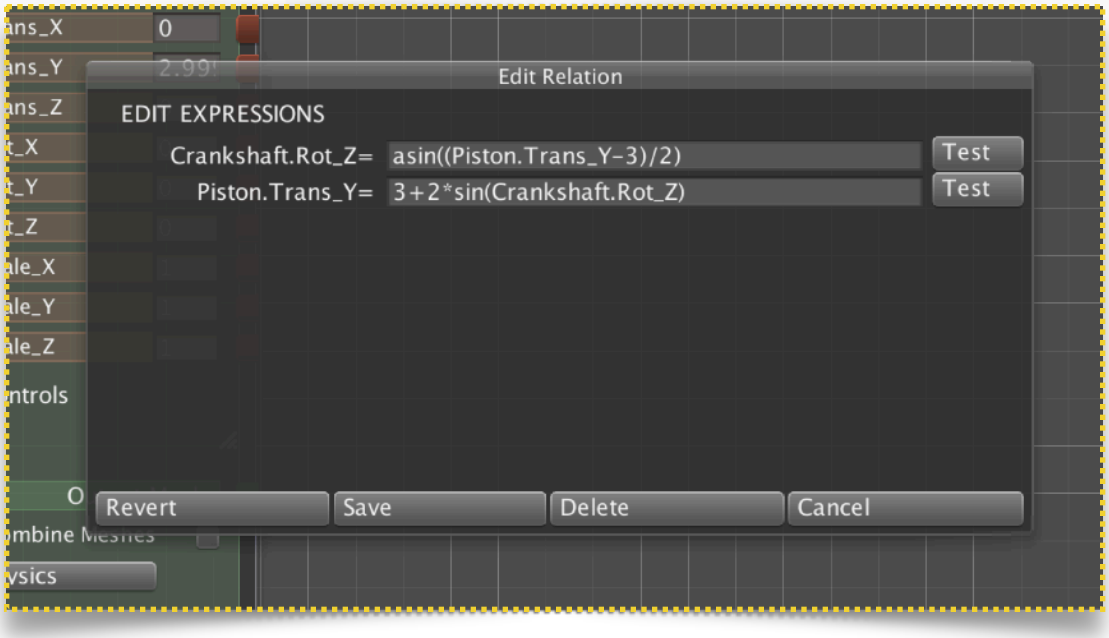
Step 8.8 Create another *Cylinder* and rename it Piston. Adjust its *radius* until it looks like the image on the previous page.

Step 8.9: Open the Transform foldouts for both nodes and create a connection between the *Rot_Z* of the Crankshaft and the *Trans_Y* of the Piston.



As mentioned previously, the default Relation expression is "=". If you like, you can drag the *Piston* in the Y-axis to see the *Crankshaft* rotate, but the sensitivity is not very high. In other words, as you drag the *Piston* through 2 units, the *Crankshaft* will rotate only 2 degrees.

Step 8.10: Open the Edit Relation window by clicking on the green button at the midpoint of the connection, or by clicking on the connection cable to select it and then hitting the Spacebar. Change the expressions to those in the image below.



The expressions relating the two parameters are the inverse of each other. In the second expression, the sine of the *Crankshaft.RotZ* is multiplied by 2 to give it more sensitivity, and 3 units are added to lift the sinusoidal motion of the *Piston* to be centered at 3 units above the

Crankshaft. Again, this is fudged geometry designed to produce quick, rudimentary results. With it, you can now rotate the *Crankshaft* in the *Z*-axis indefinitely, and the *Piston* will cycle up and down; if you move the *Piston* up and down, however, the *Crankshaft* will only respond while the *Piston* is just above the *Crankshaft* (in its first cycle).

Taking Archimatix to the Next Level

I hope you found this guide useful. For more documentation and tutorials, please visit the Archimatix support site at http://www.archimatix.com.

And come join the community at http://community.archimatix.com!

If you have specific technical questions, email us at support@archimatix.com.