LESSON TWO

Nodes, Networks and Digital Assets

To better understand Houdini, it is important to explore its workflow in the context of a real project. With Houdini's node-based workflow being such a key component of this exploration, it is important to start learning how to think and work procedurally. In this lesson, you will learn how to create your own custom tool using the procedural nodes and networks to define its function and interface.

LESSON GOAL

To create a custom tool that turns any given 3D shape into toy bricks.

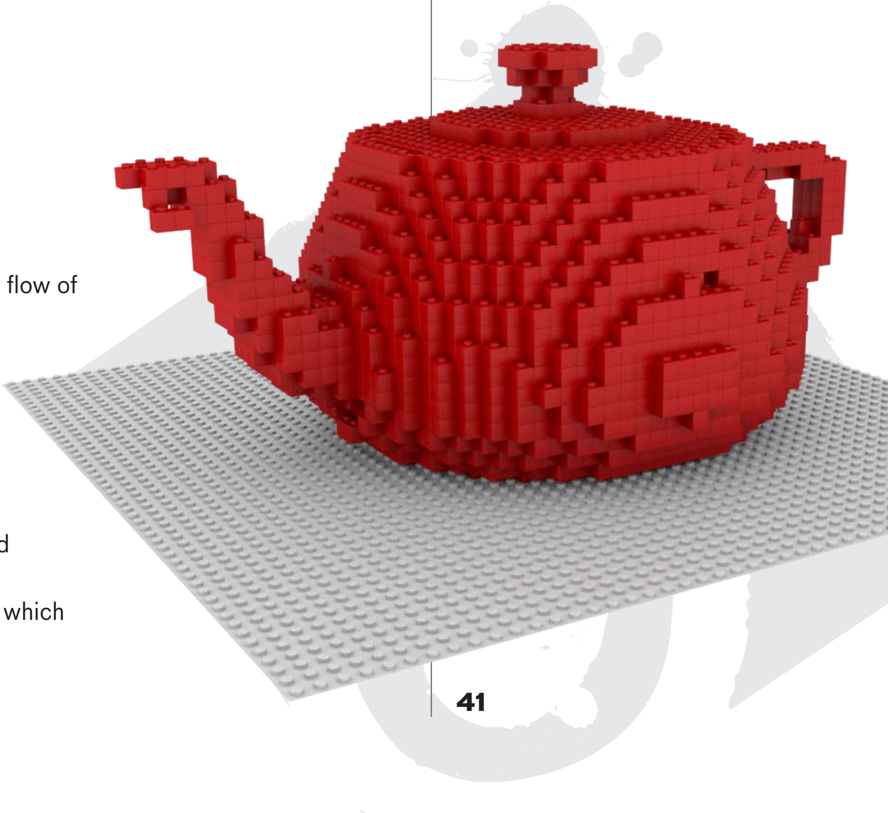
WHAT YOU WILL LEARN

- How to break down a default teapot shape into a grid of points.
- How to use instancing to make faster renderings.
- How to animate bricks appearing over time and how to create controls for adding color to all the bricks.
- How to work with **Houdini's Nodes and Networks** to control the flow of data then add and refine your work to get the look you want.
- How to use expressions to create dependencies within your network and how to work from prototype to finished solution.
- How Houdini's **Digital Asset Technology** can be used to package and share your solution with others.

Along the way you will get to use different aspects of Houdini's user interface. Be sure to refer to the overviews in the introduction to remind yourself of how these UI elements work together.

The lessons will then give you a chance to put your ideas into practice, which is one of the best ways to learn.

This lesson can be completed in its entirety using either Houdini Escape and Houdini Master.



Brickify a Teapot

In this part of the lesson, you are going to use a teapot model to define a grid of points then copy cubes to each of the points.

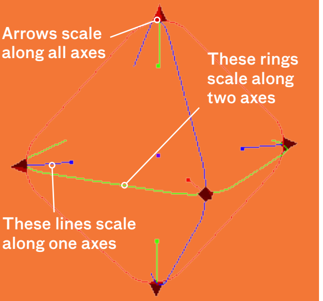
You will learn how to work in the Scene view to create objects and the Network view to make adjustments. You will also use expressions to help control the final solution.

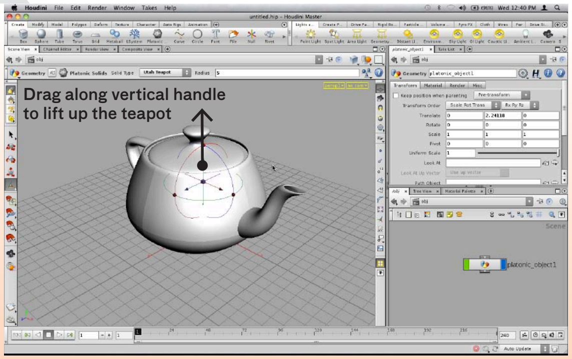
TIP

WORKING THE SCALE HANDLE

The *Scale* handle lets you manipulate the shape of your object in several different ways. You can scale all axes at once or restrict yourself to one or two axes depending on where you click and drag.

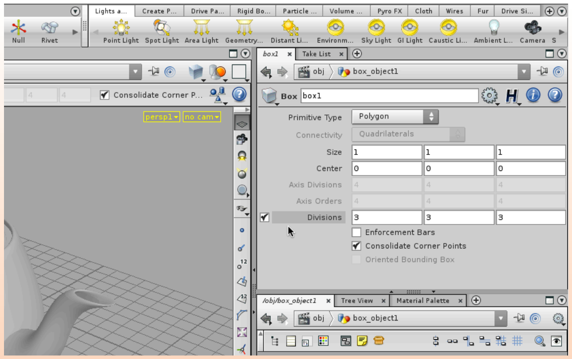
The following diagram shows you your options:


Arrows scale along all axes
These rings scale along two axes
These lines scale along one axes



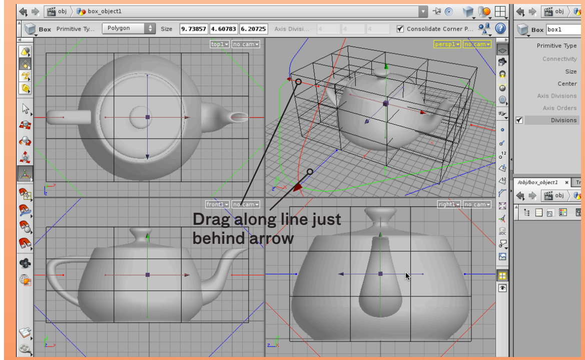
1 From the **Create** shelf, click on the **Platonic Solids** tool. Press **Enter** to place it at the origin. By default, this tool creates a Tetrahedron. From the **Operation Control** bar, set **Solid Type** to **Utah Teapot** and **Radius** to **5**.

Use the **transform** handle to lift the teapot up onto the ground. As you work, use **spacebar-LMB** to **tumble** around and make sure it is sitting on the ground. You may also want to track [**spacebar-MMB**] and Dolly [**spacebar-RMB**] in order to understand your scene from all angles.



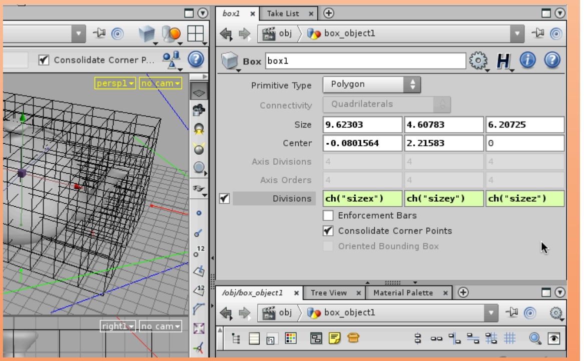
2 From the **Create** shelf, click on the **Box** tool. Press **Enter** to have it placed at the origin. From the **Operation Control** bar, click on the **Jump to Operator** button to dive into this object. You can also press **i** to dive into it. You are now at the geometry level of the box.

In the Parameter pane, click on the check box next to **Divisions** to create a 3D grid. You will later copy cubes to the grid points but first you need to encompass the teapot with the box then find out which points are inside the teapot.



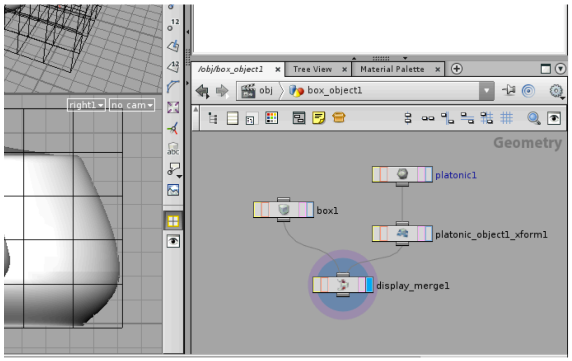
3 Set up multiple views using the Views menu then use the handle to position the box then scale it out to encompass the teapot in all three dimensions. The center handle is for positioning the box while the outer handles are for changing its size. **Tumble** around as you work.

The arrows on the **Scale handle** affect all dimensions while the outer rings change two dimensions at a time. The little lines behind the arrows resize in one dimension only. You can use any combination of these to resize the box.


Box box1
Primitive Type Polygon
Connectivity Quadrilaterals
Size 9.62303 4.60783 6.20725
Center -0.0801564 2.21583 0
Divisions ch("sizex") ch("sizey") ch("sizez")
Enforcement Bars
Consolidate Corner Points
Oriented Bounding Box

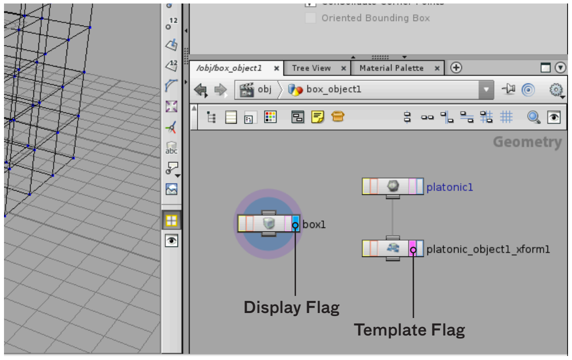
4 The divisions need to be set up in more of a cube-like grid therefore you can use the size of the box to define the divisions. In the parameter pane, **RMB-click** on the *Size* label and choose **Copy Parameter.** By clicking on the label, all three channels have been copied.

Next **RMB-click** on the *Divisions* label and select **Paste Copied Relative References**. This puts the size values in all three channels and makes a three dimensional grid that is about one unit by one unit.



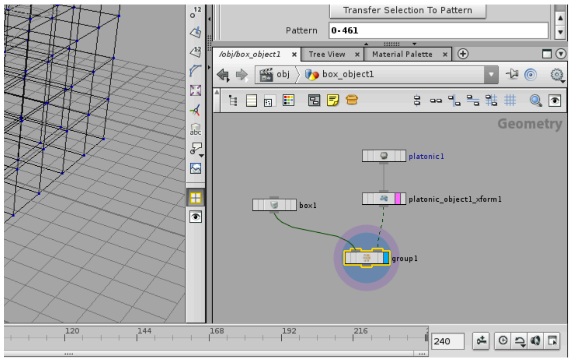
5 Press **u** to go back to the Object level. In the Network view, click on the *platonic* object then press the **shift** key and click on the *box* object. It is **important** to pick the box last so that the teapot nodes end up in the box object.

From the **Modify** tab, select **Combine** to bring these objects together. You are taken to the geometry level and the nodes are feeding into a merge node. The translation of the teapot up along the Y axis is represented by an **xform** node. You will use this later to position and size the teapot.



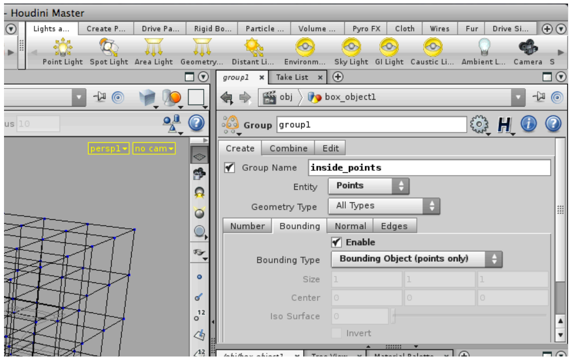
6 In the Network view, select the *display_merge* node and press the **delete** key. Now one node is active and displayed and the other is not. Houdini lets you choose which node you want to display which is the node that will be visible when you go back to the object level later.

Click on the *platonic* xform node's template flag to make it visible in light gray wire. You can use this as a guide to remind you where the original geometry sits. Click on the *box* node's display flag then select the *box* node to highlight it.



7 In the Scene view, click on the **Select** tool. Press **2** to go to point selection mode. Press **a** to select all the points. Press **tab** and select **Manipulate > Group Geometry**. This will assign a group attribute to all the selected points.

In the Network view, you will find a new node connected to the box node. Now connect the *platonic_xform* node's output to the new *group* node's right input. In the next step, you will use this as a bounding object to define the group.



Group group1
Create | Combine | Edit
✓ Group Name inside_points
Entity Points
Geometry Type All Types
Number | Bounding | Normal | Edges
✓ Enable
Bounding Type Bounding Object (points only)
Size 1 1 1
Center 0 0 0
Iso Surface 0
Invert

8 In the Parameter pane, set **Group Name** to *inside_points* then on the **Number** tab, turn off the **Enable** check box because this grouping method is not going to be used. Next, click on the **Bounding** tab and turn on the **Enable** check box. Next, set **Bounding Type** to **Bounding Object.**

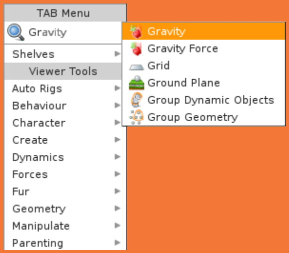
Now the teapot being fed into the second input is being used as a bounding object for the points. The points inside the teapot are highlighted because they are in the group.

FIND A TOOL BY TYPING ITS NAME

TIP

If you know the name of the tool you want to use, you can access it more quickly from the **tab** menu by spelling out its name.

As you enter each letter, the list of potential tools narrows until you find what you are looking for. For instance in step 7, if you started typing **G R** to get **Group Geometry** you would see this menu appear.

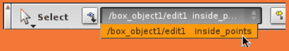


TIP

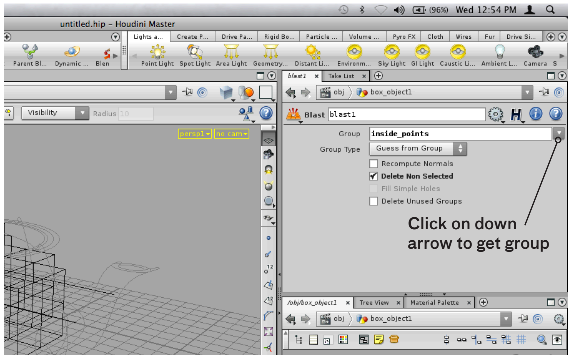
GROUPING GEOMETRY

Generally you will make a selection interactively and the points or faces you select will get assigned to the **Group** field of the node created by your current tool.

Groups can also be used to make selections. The **Select** tool has options in the Operation Control bar for adding or removing grouped geometry to or from your current selection.


Select
/box_object1/edit1 inside_p...
/box_object1/edit1 inside_points

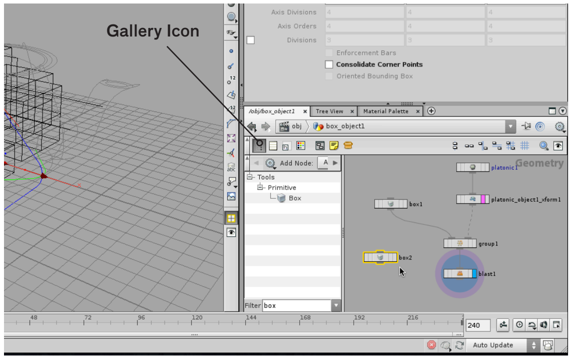
By defining a group beforehand using a group node, you can make it easier to change the selection by changing the contents of the group. This provides a more procedural solution which works well in cases such as the teapot where the contents of the group can change so easily.



Blast blast1
Group inside_points
Group Type Guess from Group
Recompute Normals
☑ Delete Non Selected
Fill Simple Holes
Delete Unused Groups

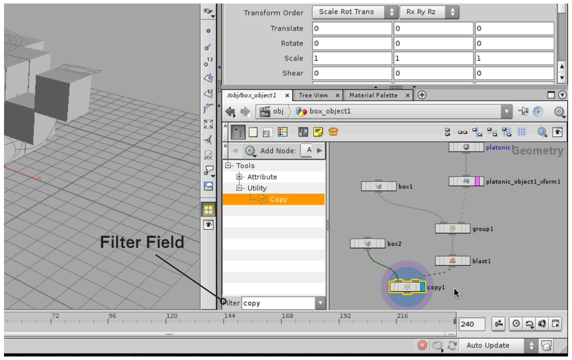
9 **RMB-click** on the output of the *group* node and select **Polygon > Blast.** Place the node then turn on its display node which at first blasts away all the points. Click on the down arrow next to **Group** and select *inside_points.*

Turn on **Delete Non-selected** to remove the outer points. Now you have a connected grid of points that lie inside the teapot. Now the procedural network uses the teapot's shape and size to define which points are in the group and therefore which will get blasted away.



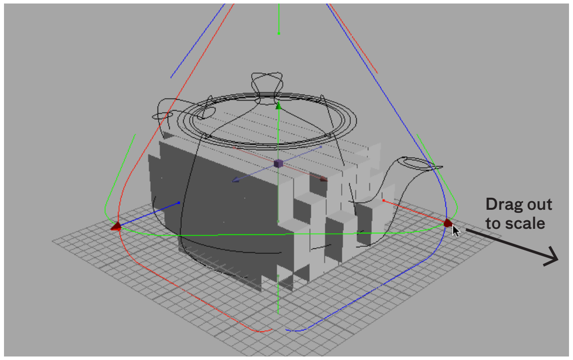
10 In the Network view, click on the Gallery icon. This opens a list of all the nodes that you can also access from the tab menu. In the Filter, type *box* then open up the **Primitive** folder and drag the **box** node into the workspace to the left of the blast node. This is another way of getting nodes into the Network view.

In the Parameter pane for *box2*, turn off the **Consolidate Corner Points** check box to create sharp edges on the box. By default the edges are being smoothed.



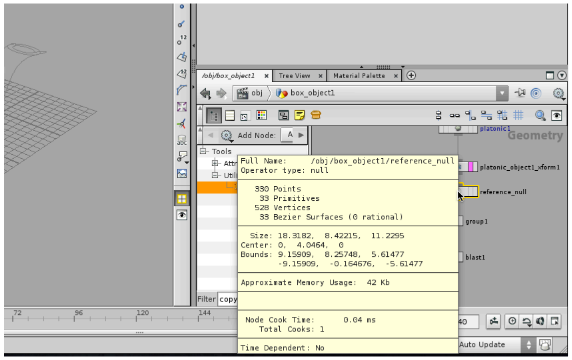
11 In the Filter field, type **copy** then open up the **Utility** folder and drag the *Copy* node below the new *box* node. Connect the **output** of the *box2* node into the **left input** of the *copy* node and the output of the blast node into the **right input** of the *copy* node. LMB-click on the box at the far right of the node to set its **Display flag.**

The boxes are copied to the points but the connecting bars are throwing off their orientation. Select the *box1* node and change its **Primitive Type** to **Points** to fix this.



12 Select the *platonic_object_xform* node. Move your cursor over the Scene view and press **Enter** to go to the handle tool. You can now translate the teapot and scale the teapot using the handles (**t** for translate, **e** for scale).

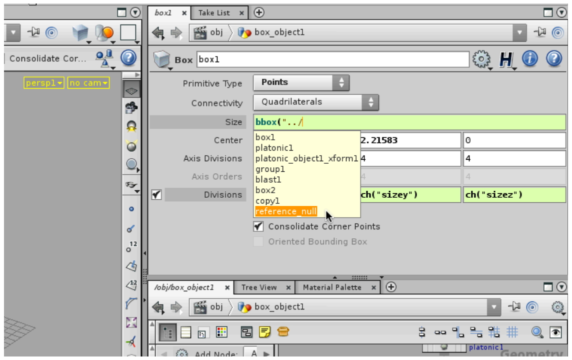
As you change the size and position of the teapot, the square grid updates. One problem is that the size of *box1* node which defines the grid, is cutting off the points so they can't fill the teapot anymore. You need to create a closer relationship between the box and the object it is bounding.



```
Full Name:    /obj/box_object1/reference_null
Operator type: null

  330 Points
   33 Primitives
  528 Vertices
   33 Bezier Surfaces (0 rational)

   Size: 18.3182,  8.42215,  11.2295
Center: 0,  4.0464,  0
Bounds: 9.15909,  8.25748,  5.61477
        -9.15909,  -0.164676,  -5.61477

Approximate Memory Usage:  42 Kb

Node Cook Time:     0.04 ms
   Total Cooks: 1

Time Dependent: No
```

13 In the Node view, **RMB-click** on the output of the *platonic_object_xform* node and select **Utility > Null**. Click to place the node between the *platonic_object_xform* and the *group* node. Change its name to *reference_null*.

You are going to use an expression to define the size and position of the bounding grid using the null. If you **MMB** on the null you will see that size and position information is flowing into the *null* node from the *teapot*. You will use the null so that later you can plug other objects into the network.

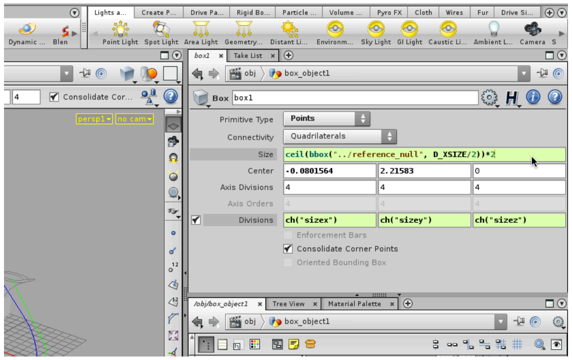


```
Box  box1
Primitive Type   Points
Connectivity     Quadrilaterals
Size             bbox("../
                 box1
                 platonic1
                 platonic_object1_xform1
                 group1
                 blast1
                 box2
                 copy1
                 reference_null
Center                      2.21583    0
Axis Divisions              4          4
Divisions                   ch("sizey")  ch("sizez")
[✓] Consolidate Corner Points
    Oriented Bounding Box
```

14 Select *box1.* In the Parameter pane, double-click on the **Size X** parameter Start typing `bbox` then add a **start bracket** and a **quote mark** Type **two dots** then a **slash** and you will see a list of nodes. Press the down arrow to select *reference_null* then press enter. Add another **quote**, a **comma**, a **space** then `D_XSIZE` then add an **end bracket**.

Here is the final expression:

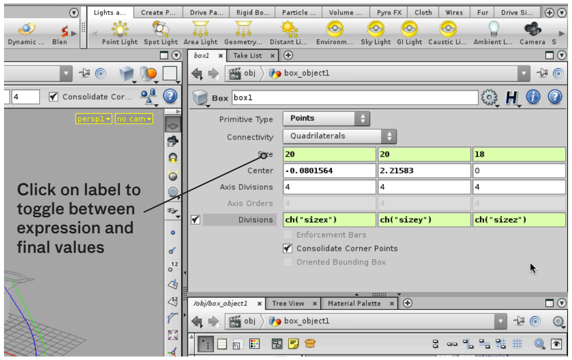
```
bbox("../reference_null", D_XSIZE)
```



```
Box  box1
Primitive Type   Points
Connectivity     Quadrilaterals
Size             ceil(bbox("../reference_null", D_XSIZE/2))*2
Center           -0.0801564   2.21583      0
Axis Divisions   4            4            4
Divisions        ch("sizex")  ch("sizey")  ch("sizez")
    Enforcement Bars
[✓] Consolidate Corner Points
    Oriented Bounding Box
```

15 The value needs to be an integer and an even number to make sure the grid is always positioned properly in 3D. Click on the Size X parameter again and add `ceil` and a **front bracket** in front and `/2`, an **end bracket** and `*2` to change it to the following expression:
`ceil(bbox("../reference_null", D_XSIZE)/2)*2`

Click on the **Size** label to verify that the size being returned by the expression is an even number. Dividing by 2, rounding up, then multiplying by 2 keeps it even.


Click on label to toggle between expression and final values

```
Box  box1
Primitive Type   Points
Connectivity     Quadrilaterals
Size             20           20           18
Center           -0.0801564   2.21583      0
Axis Divisions   4            4            4
Divisions        ch("sizex")  ch("sizey")  ch("sizez")
    Enforcement Bars
[✓] Consolidate Corner Points
    Oriented Bounding Box
```

16 RMB-click on the **Size X** parameter and select **Copy Parameter** from the menu. Press the tab key to go to the **Size Y** parameter and then RMB-click and select **Paste Copied Expressions.** Change `D_XSIZE` to `D_YSIZE` then press tab and **Paste Copied Expressions** again. This time change `D_XSIZE` to `D_ZSIZE`.

Click on the **Size** parameter label to toggle from expressions to values. You can see that the box is the same size as the teapot and the numbers have been rounded.

NULL NODES GIVE YOU FLEXIBILITY

TIP

In this lesson, the use of the Null node gives you flexibility down the line.

By referencing the null for size and position information, you can later feed another shape into the null node and the whole network will recook to reflect the new input.

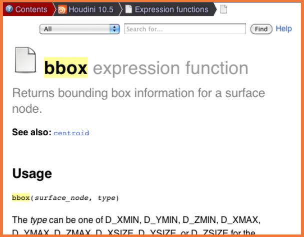
If you were to reference the teapot directly then a new shape wouldn't work properly.

BE SURE TO ACCESS ONLINE HELP

TIP

To get information on this expression, select **Help > Houdini Help**. In the search bar type *bbox* then click on **bbox** expression function.

This shows the function's syntax and in some cases shows you the function being used in an example expression.



bbox expression function

Returns bounding box information for a surface node.

See also: centroid

Usage

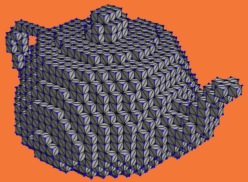
`bbox(surface_node, type)`

The *type* can be one of D_XMIN, D_YMIN, D_ZMIN, D_XMAX, D_YMAX, D_ZMAX, D_XSIZE, D_YSIZE, or D_ZSIZE for the

TIP

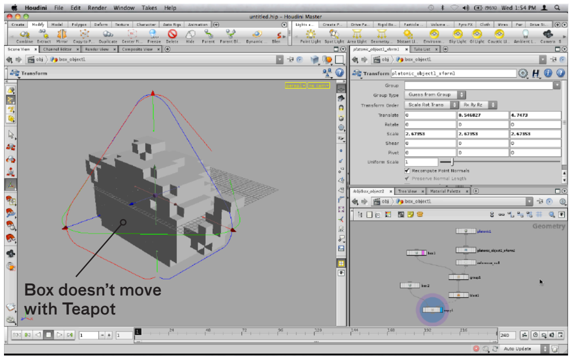
WHY DIDN'T WE USE ISO-OFFSET

To fill up an object with a 3D grid of cubes, the easiest way in Houdini is to use an **IsoOffset** node with **Output Type** set to **Tetra Mesh** and **Tetra Type** set to **Cubes.** You can control how many samples are visible and you quickly get your voxelized shape.



The reason this won't work for the brickify tool is that you need the grid to maintain specific dimensions to create a system and later you need to be able to output the points at the center of each cube to be used for instancing brick geometry.

Luckily in Houdini you can take a different approach which meets the requirements of this tool.

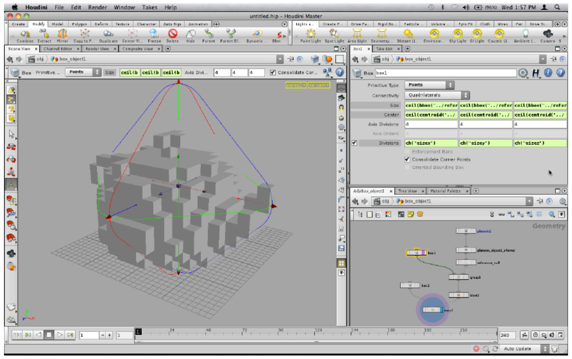


17 If you scale the teapot's *xform* node the cube density will increase. But if you move the teapot then the box doesn't move and the cubes start to get cut off. In *box1's* **Center X** parameter, enter this expression:

```
ceil(centroid("../reference_null", D_X))
```

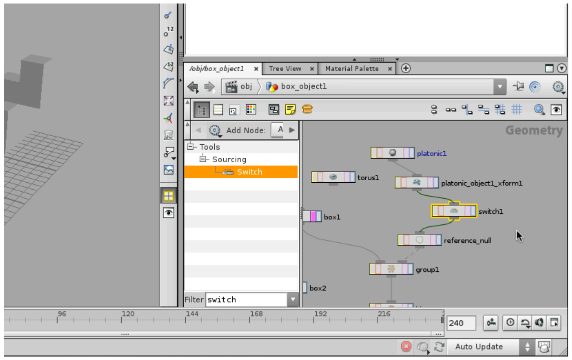
Copy and **Paste** this expression to Y and Z and change `D_X` to `D_Z` and `D_Y` accordingly. For Y change the expression to:

```
ceil(centroid("../reference_null", D_X))+ 0.5
```



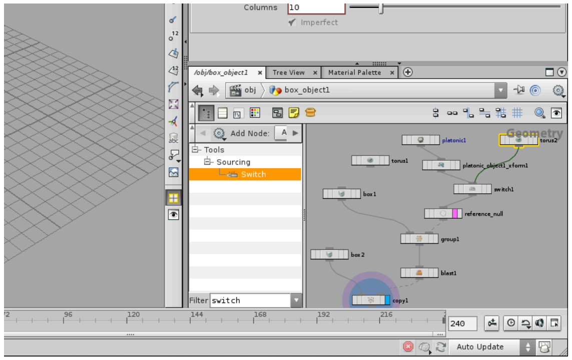
18 Now if you move or resize the teapot, the cubification will follow along. The *box1* node will position and size itself to match the geometry passing through the *reference_null* node. In fact, the network is robust enough to handle different geometry passing through the same network.

If you feed another model into the *reference_null* node then the network will cubify the new shape. But before adding a second shape into the network, you will add a **switch** node to make it easy to go back to the *teapot.*



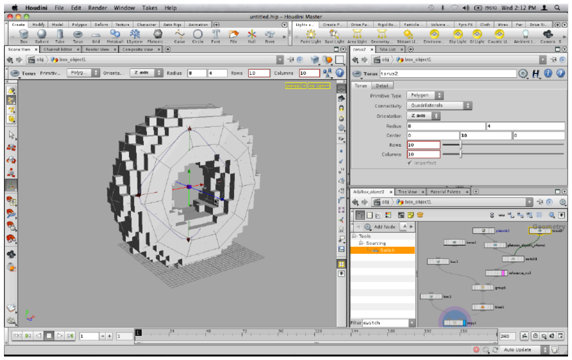
19 In the gallery filter, type the word **Switch** then from the **Sourcing** folder drag it into the Network view. Next, drag it over the line connecting the teapot's *xform* node to the *reference_null* node. This inserts it into the network between the two nodes.

This node will make it easier to switch between different incoming shapes.



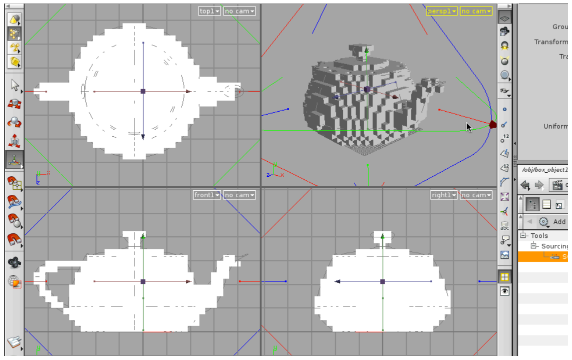
20 In the tool shelf, go to the Create menu and drag the **Torus** tool down to the network view. This places a torus node at the geometry level. Tools can be dragged into the network view as long as the node type makes sense for your current network level.

Connect the output of the *torus* node to the input of the *switch* node. Now you can select the *switch* node and change its **Select Input** to **1.** The torus has been cubified but you won't see this until you make it bigger.



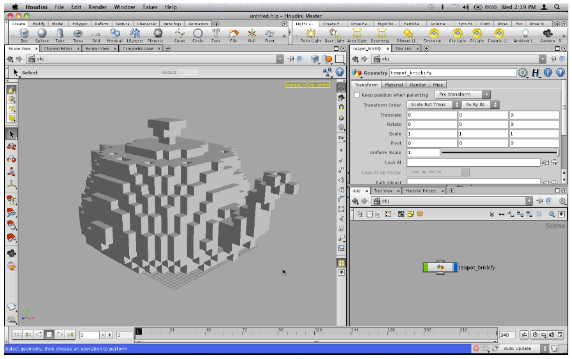
21 You can now size it and position the *torus* and the network works effectively. This is why the expressions on the box node use the *reference_null* node. Now other nodes can feed into the system and it works properly.

Later you will package up this network into a custom tool called a digital asset. As a **digital asset**, it will be easier to share the network with others and to manage the tool as it gets deployed in a studio environment where changes and updates are inevitable as the tool evolves.



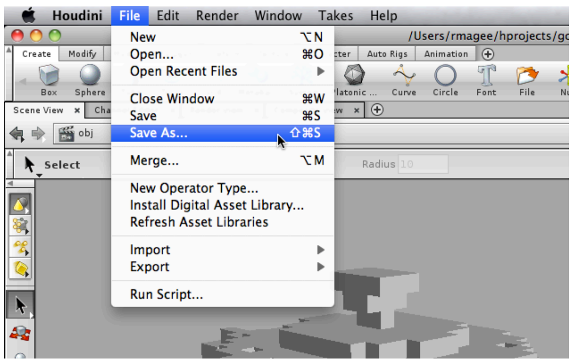
22 Use the *switch* node and set the **Select Input** to **0** to go back to the *teapot*. Select the teapot's *xform* node and scale it out and place it in a way where it sits directly on the ground. You might want to check a front or side view to make sure it is right on the ground.

You should choose enough cubes to update fast but still gives you some detail in the cubified model. If you middle-click on the *blast* node you will see how many points are in the group therefore how many cubes you have created.



23 On one of the network paths, click on **obj** to jump back up to the **Object** Level. Whichever node has its display flag set is visible at this level. If you don't see your cubified teapot then dive back down and make sure the display flag is set to the *copy* node.

In this lesson we didn't change the display flag very often but it is **Very Important** to be aware of which node has its display node set otherwise you may be confused when you go back to the object level.



24 Select **File > Save As...** and in the browser window, click on the *Home folder*. Find the folder called *gopro_guide* and then double-click on it to dive in. Double-click to go into the *lesson1* folder then name the file *cubify_01.hip*. Click on the **Accept** button to save.

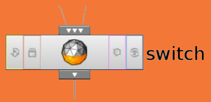
The next time you save, Houdini will copy the previously saved file into a back-up folder then over-write your scene file. This protects your work although the folder will get full fast and will need to be cleaned out once in a while.

TIP

THE SWITCH NODE

With Houdini's node-based networks, you can explore different paths by branching off nodes then testing each solution with the display flag at the end of that chain.

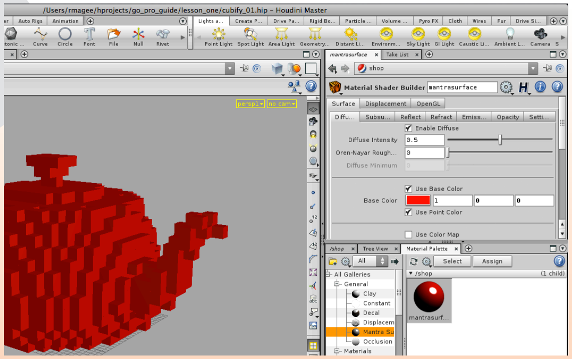
With the Switch node, you can feed different solutions into a single node then use that node to choose which solution is the best one.



You can also use the switch node to make this switch as you work or even as your scene animates. You could keyframe the input to switch on the fly or simply use the switch to define a checkbox or menu on your digital asset just like you are doing in this lesson.

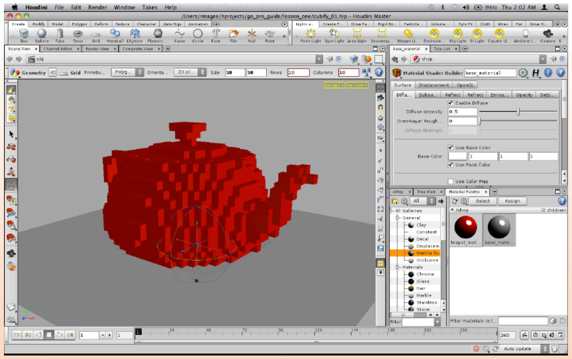
Render the Bricks

You are now going to create a rendered image of the bricks. This will let you see the scene with real lights and shadows. You will use a basic material to render the copied cubes then you will re-configure the object to use instanced cubes instead.



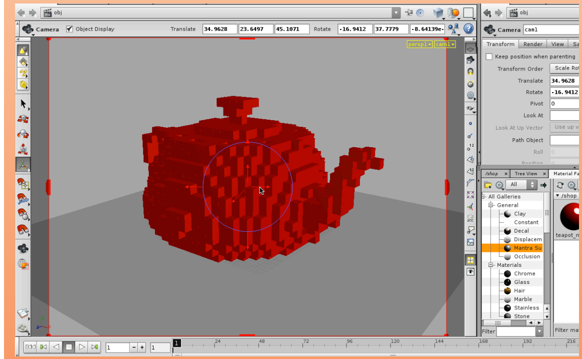
1 Click on the **Material Palette** in the lower right of the desktop. From the **General** section of the Material Gallery, select **Mantra Surface** and drag it onto the teapot. This assigns it to the teapot and adds it to the palette.

Click on the material in the palette and in the Parameter pane, click on the Color swatch next to **Base Color** to open up a color picker. Change it to a red color then close the picker. **RMB-click** on the material in the palette and select **Update Shader Ball Icon** to see the new color in the icon.



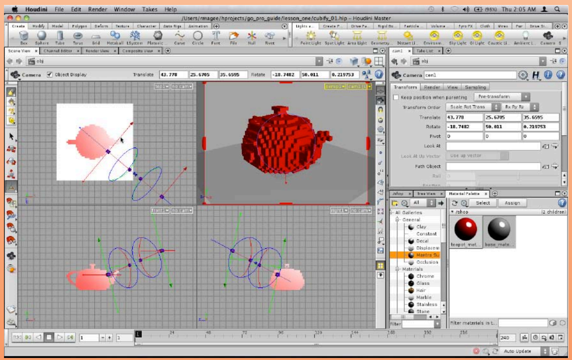
2 From the **Create** shelf, click on the **Grid** tool. Press **Enter** to put the grid at the origin. In the Operation Controls bar, change the **Size** to **50, 50**. This will give you a ground surface for casting shadows from the teapot.

In the **Material Palette**, drag a new **Mantra Surface** from the gallery to the new grid. Leave this material a light grey. Select the material and rename it *base_material* then select the first material and rename it *brick_material.*



3 In the perspective view, tumble zoom and pan frame a nice view of the teapot. When you are ready, **LMB-click** on the menu in the top right of the viewport which currently says **no cam** and select **New Camera**.

This sets up the view to look through a new camera which has been added to the scene. If you change your view then you will be knocked out of the camera and will have to reselect it from the view menu. To position and edit the camera view you need to lock the camera.



4 Click on the **Lock camera to View** button. Now you can use the spacebar view tools in the viewport to adjust the camera view. Once you have the view you like you should turn off the **Lock Camera** button.

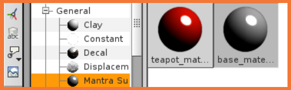
You can also choose **Select Camera** from the **view menu** then press the **Enter** key to go into the **Handle tool.** Now you can see handles in your view that can be used to change the view. If you go to multiple views **(spacebar-b)** then you can position the camera using 3D handles.

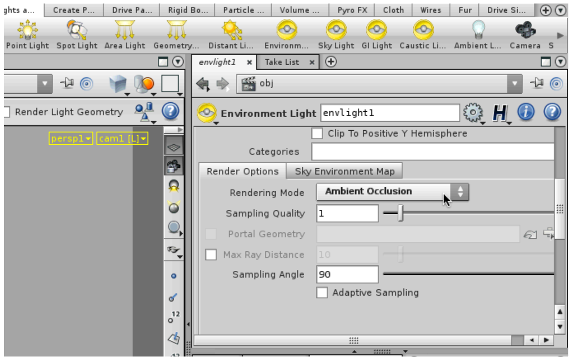
MANTRA SURFACE MATERIAL

TIP

There are lots of Materials in the Material Palette to choose from. The **Mantra Surface** material is a good starting point for most looks you are trying to achieve.

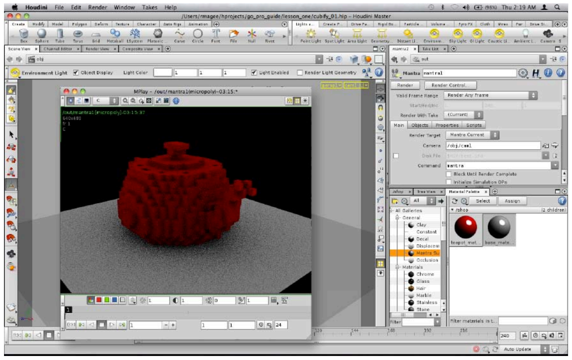
It can handle shader properties such as diffuse, specular, subsurface scattering and more. It also supports per-light exports when working with deep raster files. You will learn more about it in subsequent lessons.





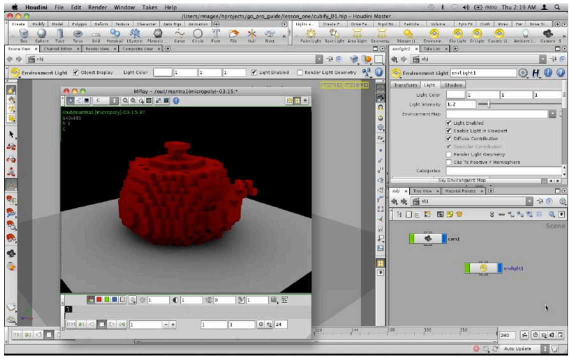
5 From the **Lights and Cameras** shelf, click on the **Environment light** tool. This places an environment light into the scene. This light will give you a nice soft shadow effect.

In the Parameter pane, click on the **Light** tab then under **Render Options,** set **Rendering Mode** to **Ambient Occlusion.** The default **Sampling Quality** is very low but you should keep it there for initial test renders. You can increase this later to get better looking renderings.



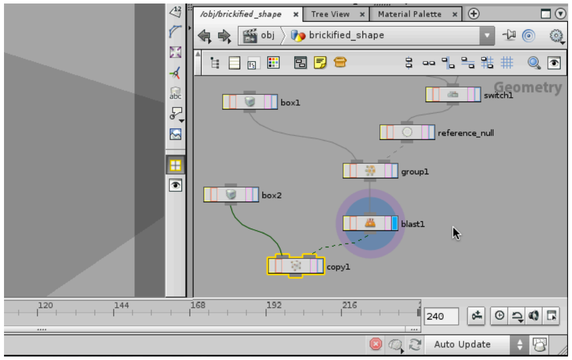
6 From the **Render** menu, select **Create Render Node > Mantra.** This render operator (ROP) node includes all of the render settings you will use to refine your rendering.

Click on the **Render** button in the Parameter pane. A rendering starts up using your current settings and opens Houdini's image viewer application called Mplay. Initially the rendering is very grainy because of the low **Sampling** on the environment light node.



7 Select the environment light node in the Network view and set the **Sampling Quality** to **32**. You could also up the sampling on the *mantra* node since this value and the mantra node's sampling are multiplied at render time to define the final look.

From the **Render** menu, select **Start Render > mantra1** to see how this affects the results. You can now tweak the lights and shaders. Try bumping the **Light Intensity** up to around **1.2.** You can then re-render to check out the results.



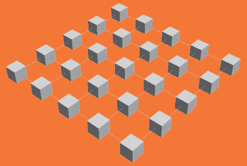
8 Click on the *box_object 1* which contains the teapot and rename it *brickified_shape*. Double-click on the *brickified_shape* object and in the network below, find the *Blast* node and turn on its Display flag. In the Render view the teapot disappears because you are now focusing on the points and not the copied cubes.

Because all the cubes are the same, you are going to instance geometry to the points to take advantage of Mantra's fast instance rendering.

TIP

COPY NODE VS INSTANCING

The **Copy node** is a versatile tool which provides you with a way of copying geometry in an array or to a cloud of points. You can run expressions to randomize and control the copies and the results look good. The one problem is that the node gives you tons of geometry because you are dealing with the resulting geometry at render time.



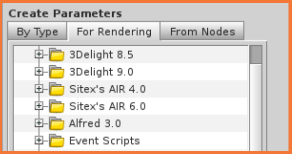
With **Instancing**, you take the point cloud and at the object level you assign geometry to all the points. At render time only the single copy is loaded into memory and the results are much more efficient.

HIDDEN RENDER PARAMETERS

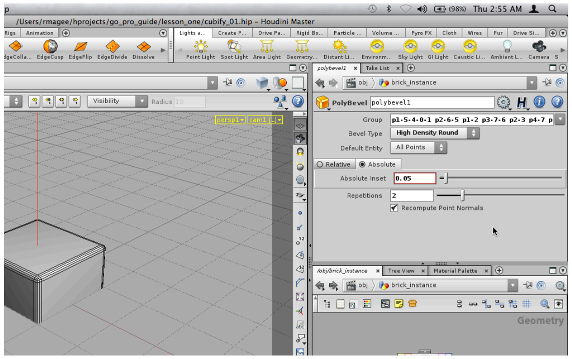
TIP

There are lots of rendering parameters available in Houdini for supporting Mantra and a number of other third party renderers.

Many of these are left off of typical Houdini nodes because they are only used in special cases and you wouldn't want them adding unwanted complexity to your nodes and potentially slowing down your renders.

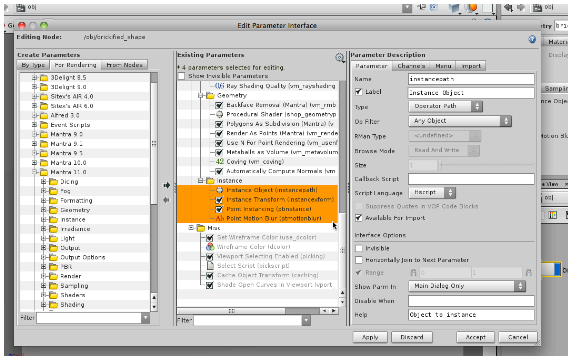


You can therefore add rendering parameters back to your nodes by selecting **Edit Rendering Parameters** from the gear icon. As you learn more about Houdini, you will learn more about these hidden capabilities.



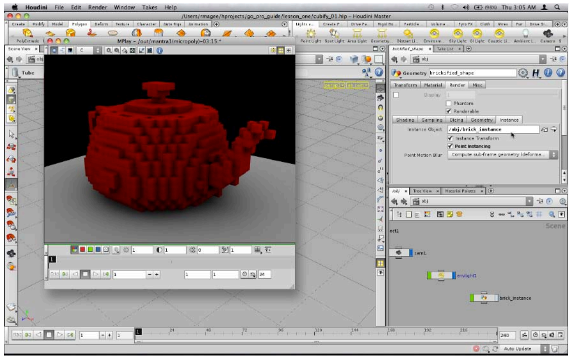
9 From the **Create** shelf, click on the **Box** tool then move your cursor over the Scene view and press **Enter** to place it at the origin. In the Parameter pane, rename this object *brick_instance*.

From the **Polygon** shelf, click on the **PolyBevel** tool to bevel the edges of the box. In the parameter pane, set the **Bevel Type** to **High Density Round**, click on the **Absolute** tab and set **Absolute Inset** to **0.05**. Now set **Repetitions** to **2** to add detail to the bevel.



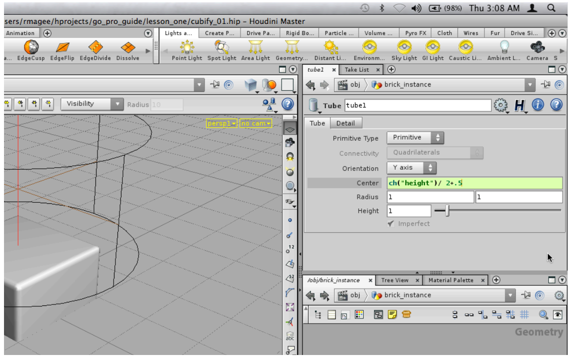
10 Press **u** to go back up to the object level. Select the *brickified_shape* object then in the Parameter pane click on the **Gear icon** and select **Edit Rendering Parameters**. Open up the **Mantra 11** tab and find the **Instance** folder.

Drag the **Instance** folder over to the **Render** folder in the **Existing Parameters** column. This adds new parameters to the object. Click **Accept** to add the parameters to the object. You will use these new parameters to set up the instanced geometry.



11 In the Parameter pane, click on the **Instance** tab then drag the *brick_instance* object from the Network view up to the **Instance Object** field. This adds the object to the field. Next, turn on **Point Instancing**. Now turn off the display flag on the *brick_instance* object so that it is hidden.

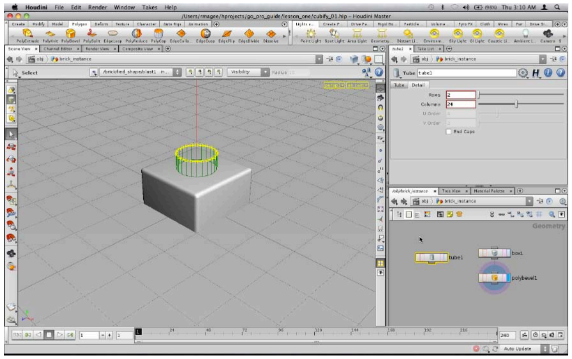
From the **Render** menu, select **Start Render > mantra1**. Now the *brickified_shape* is rendered using the instanced cubes. You can now add more detail to the bricks and these details will also be instanced.



12 Click on the Scene View tab. Select the *brick_instance* object and press **i** to dive into it. In the Network view, press **tab > Primitive > Tube.** Place the node in the view.

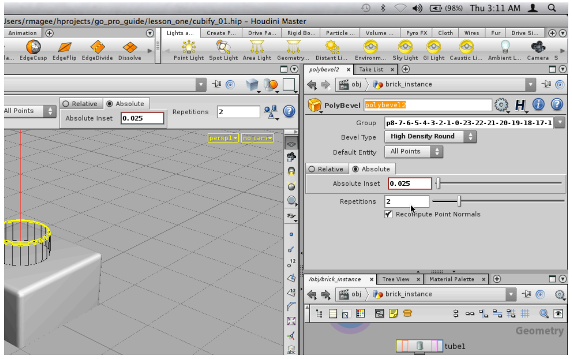
In the Parameter pane, RMB-click on the *Height* parameter and select **Copy Parameter.** RMB-click on the **Center Y** channel and select **Paste Copied Relative References.** Now divide by **2** and add **0.5** to change this expression to:

```
ch("height")/2+0.5
```



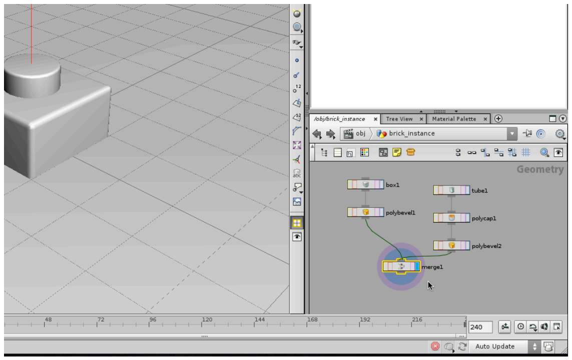
13 This expression makes sure the peg stays on top of the box no matter which height you choose. now change the **Radius** to **0.25** and **0.25** and the **Height** to **0.2**. Set the **Primitive Type** to **Polygon** then click on the **Detail** tab. Set the Columns to 24.

Now press **3** to choose Edge Selection then click on one of the top edges of the peg. **RMB-click** in empty space and select **Select Loop** to highlight the top edge of the tube. You will use this to cap the end.



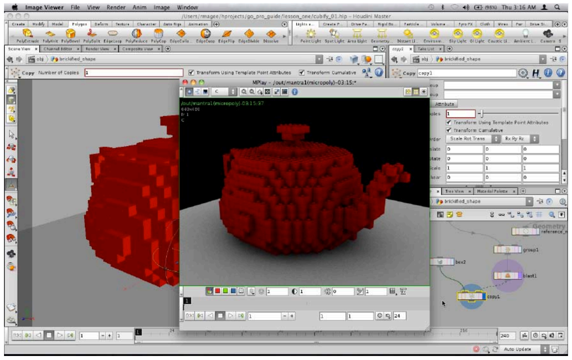
14 From the Polygon shelf, select **PolyCap**. This adds a surface to the top of the peg. The points at the edge of the cap have been fused to the tube. Now press **PolyBevel** to bevel the same edge.

In the Parameter pane, set the **Bevel Type** to **High Density Round**, then click on the **Absolute** tab. Set the **Absolute inset** to **0.025** and the **Repetitions** to **2**. Now you need to add this peg to the box to create the toy brick.



15 In the Network view, press **tab > Sourcing > Merge** to add a **Merge** node. Click on the output of the first *polybevel* node and then on the input of the merge node then repeat this for the second *polybevel*. Set the display flag for the *merge* node to make this the node that will be rendered.

In the Scene view you can see the finished peg. This is now the geometry that will be instanced. Now that the geometry for each brick is more complex, instancing is the best route to go for rendering.



16 One problem is that with this instancing setup the teapot is not visible in the scene view. Go to the object level and select the *brickified_shape* object. Press **i** to dive in. Set the Display flag back on the *copy* node to see the cubes.

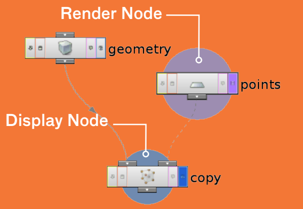
Now **Ctrl-click** on the blast node's display flag to set the Render flag there. Now select **Render > Start Render > mantra1** to create a non-IPR rendering of the teapot. It will come up in **MPlay** and you can save it to disk if you want to. Now select **File > Save** to save your work.

TIP

DISPLAY AND RENDER FLAGS

It is possible with a Houdini network to view one node while rendering out another. This can be used to view low-res geometry then render out high-res geometry or in this case to see copied geometry but render out the points for instancing.

The **Display flag** is used to set both display and render. You can **Ctrl-click** on this flag to set only the render flag or you can select a node and press **2** to make it the **Render flag**. Press **1** to set the **Display flag** on its own.



The display node is highlighted with a small circle and the render node with a larger circle. When the cirlces are together then the node serves both of these functions.

Create a Brickify Tool

The technique used to brickify the teapot would apply equally well to other objects. Instead of redoing all the steps taught in this lesson for each new object, you can re-purpose the existing network into a custom tool.

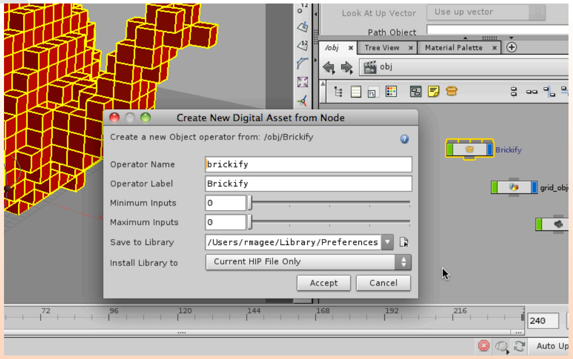
With Houdini this tool is easy to create and doesn't require any scripting or programming skills. You will turn your existing network into a digital asset and build an artist-friendly UI for the asset to make it easy to use.

TIP

WHAT IS A DIGITAL ASSET?

Digital assets are custom nodes which you can create by encapsulating your networks inside a subnetwork node. You can then build a top-level interface by promoting parameters and handles from the nodes inside your asset.

Assets reference their definition which is stored in an Operator Type Library [.otl] file on disk. You can swamp out libraries, share them with colleagues and integrate them into a wider asset management system. You will learn a lot about Digital Assets as you learn and work with Houdini.


Create New Digital Asset from Node

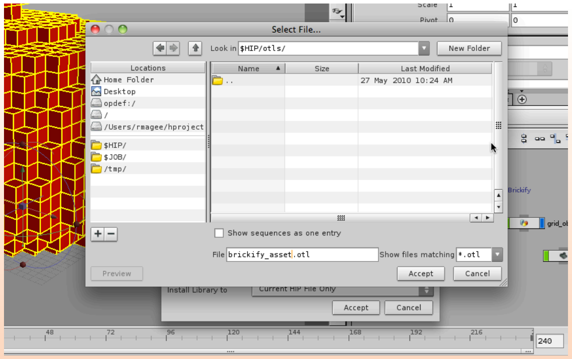
Create a new Object operator from: /obj/Brickify

Operator Name: brickify
Operator Label: Brickify
Minimum Inputs: 0
Maximum Inputs: 0
Save to Library: /Users/rmagee/Library/Preferences
Install Library to: Current HIP File Only

Accept | Cancel

1 In the Network view, drag the *brickify_instance* and *brickified_shape* objects off to the side. Select them both then click on the **Create Subnet from Selected** button.

Rename the subnet *Brickify* then RMB-click on the subnet and select **Create Digital Asset...** This opens up a panel where you can set the **Operator Name** to brickify, the **Operator Label** to *Brickify* then click on the button next to **Save to Library** to save the asset to a library file on disk.


Select File...

Look in: $HIP/otls/ | New Folder

Locations: Home Folder, Desktop, opdef:/, /, /Users/rmagee/hproject, $HIP/, $JOB/, /tmp/

Name	Size	Last Modified
..		27 May 2010 10:24 AM

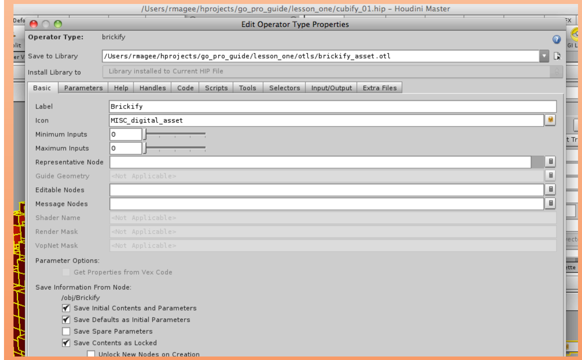
Show sequences as one entry

File: brickify_asset.otl | Show files matching: *.otl

Preview | Accept | Cancel

2 In the File Browser, click on **$HIP** which puts you in the same folder as your saved scene file. Click the **New Folder** button and then create a file called *brickify_asset.otl*. The .otl stands for Operator Type Library. Click **Accept**.

This library file can contain one or more asset definitions. Every time you change an asset you save your changes to this library file. In production, you could have different versions of an asset saved in different library files. You could then easily switch between them to test different ideas.


/Users/rmagee/hprojects/go_pro_guide/lesson_one/cubify_01.hip – Houdini Master

Edit Operator Type Properties

Operator Type: brickify
Save to Library: /Users/rmagee/hprojects/go_pro_guide/lesson_one/otls/brickify_asset.otl
Install Library to: Library installed to Current HIP File

Basic | Parameters | Help | Handles | Code | Scripts | Tools | Selectors | Input/Output | Extra Files

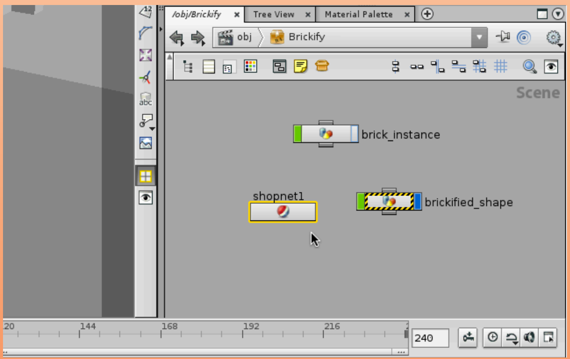
Label: Brickify
Icon: MISC_digital_asset
Minimum Inputs: 0
Maximum Inputs: 0
Representative Node:
Guide Geometry: <Not Applicable>
Editable Nodes:
Message Nodes:
Shader Name: <Not Applicable>
Render Mask: <Not Applicable>
VopNet Mask: <Not Applicable>

Parameter Options:
Get Properties from Vex Code

Save Information From Node:
/obj/Brickify
✓ Save Initial Contents and Parameters
✓ Save Defaults as Initial Parameters
☐ Save Spare Parameters
✓ Save Contents as Locked
☐ Unlock New Nodes on Creation

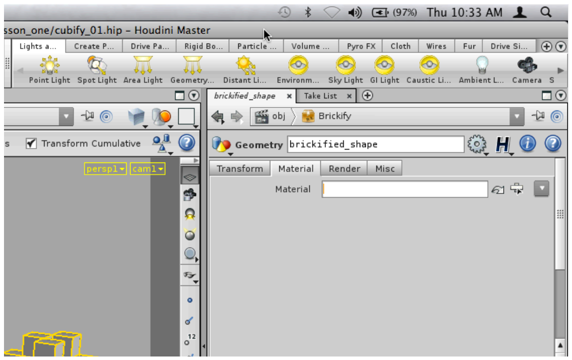
3 You are told in a dialog box that the nodes in your asset reference another node which lies outside the subnet. Click OK to proceed, you will correct this later. Now the Operator Type interface shows up which you can use to create the top-level interface for your asset.

Click **Accept** to close this window then click **OK** again. You are going to tweek your network then later you will important parameters to the Type Properties window to build a user interface for the **Brickify** tool.



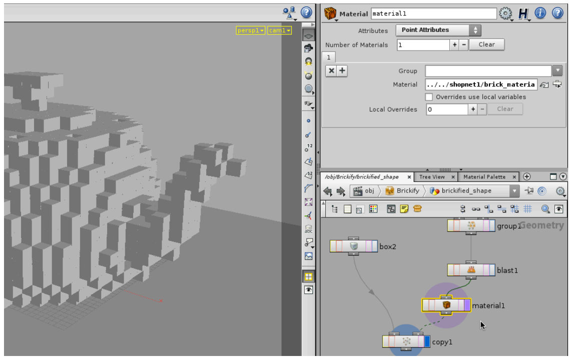
4 First lets fix that material reference by moving the material inside the asset. In the Network View, double-click on the *brickify* node. Press **tab >Managers > SHOP Network** and place the network node into the view. You can now move the material to this location which is nested inside the *brickify* asset.

Click on the material palette and click on *brick_material*. Click on the Network view which is now pointed at the shop level. **RMB-click** in empty space and choose **Cut**.



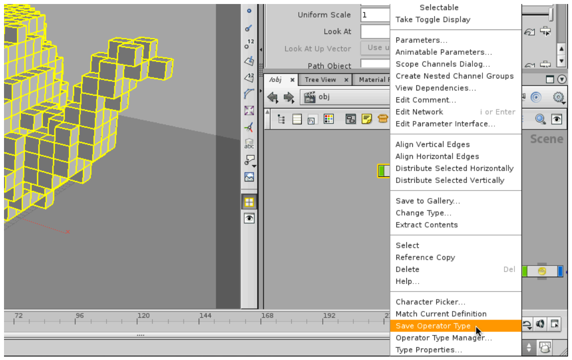
5 Click on the back arrow on the Network view to go back to the brickify network. Double click on the *shopnet1* node and then **RMB-click** and select **Paste** to put the material node here.

Press **u** to go up one level and select the *brickified_shape* node. Click on the **Material** tab and delete any text next to the **Material** parameter. This is no longer needed because the nested material is now being used.



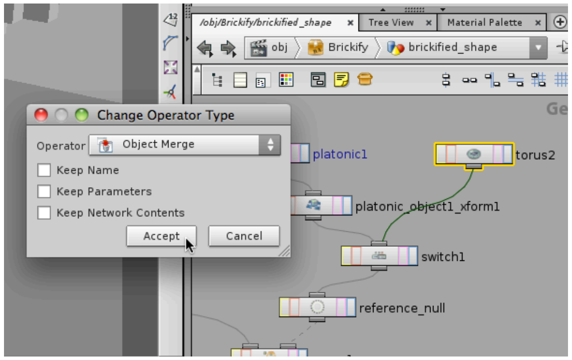
6 Double-click on the *brickified_shape* node to go to its geometry level. **RMB-click** on the blast1 node's output and select **Material > Material** to add a new node. Click to place the node down then press **Ctrl** and click on the *material* node's display flag to make it the render node.

On the Material node, set **Attributes** to **Point Attributes**. Click on the node selecter button next to **Material** and from the pop-up menu select the *brick_material* from inside *shopnet1*. Choose **Export Relative Path** and click **Accept**.



7 Press **obj** then **RMB-click** on *brickify* node and select **Save Operator Type**. This will save the changes you just made to the asset's definition in the .otl file. The pop-up window warning you about references lying outside your asset should now be gone.

In order to share assets with others, it is important that these references are fixed by either nesting all the required nodes into the asset or putting parameters such as the **Material** parameter onto the asset itself.



8 Navigate back into the *brickified_shape* node. Find the *torus* node which is feeding into the *switch* node and **RMB-click** on it. From the menu, select **Change Type...** In the pop-up window, change the **Operator** from **Torus** to **Object Merge** and turn off Keep Parameters. Press **Accept**.

The *object_merge* node will let you reference another object in your scene for brickification. You will now set up the UI on the brickify asset to make this node and the teapot node available at the top level.

ABSOLUTE VS RELATIVE REFERENCES

TIP

As you work in Houdini, you will often create references and dependencies to other nodes in your scene or files on disk.

In both cases, you can either reference the complete or **Absolute path** or a partial or **Relative path** which uses the scene file or node's existing location as a starting point to find the reference. In these cases, you often see two dots and a slash [../]which represent stepping up one level.

In production, relative references are much less likely to break. Digital Assets specifically prefer relative references and it is good practice to use them in your work.

For Scene files, you can use **Render> Preflight Scene** to manage your various references to files on disk.

TIP

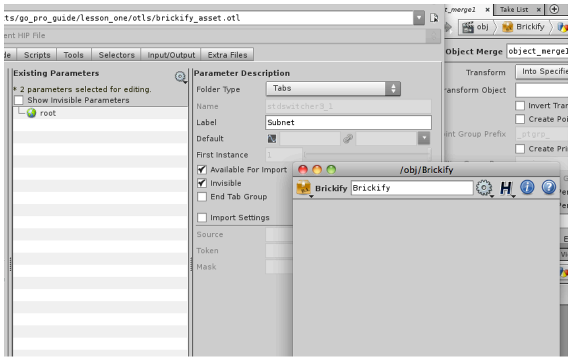
BUILDING A CUSTOM UI FOR YOUR ASSET

One of the advantages of working with Digital Assets is the ability to take a solution as defined by a complex network of nodes and present it to your artists as a single node.

In order for that node to be usable you need to promote parameters and add handles to allow the node to function as its own tool.

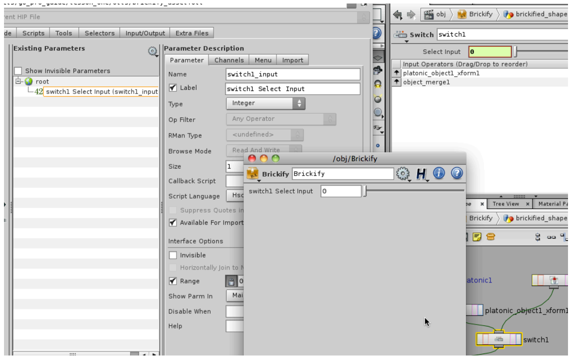
When parameters are promoted, you can either leave their name the same as it was inside the asset or if you think that parameter name is not very artist friendly then you can change it at the top level of the asset.

As you develop an asset you can send it to artists for testing and update the UI based on their feedback. Updated assets can also be swapped into your pipeline if you make changes during production.



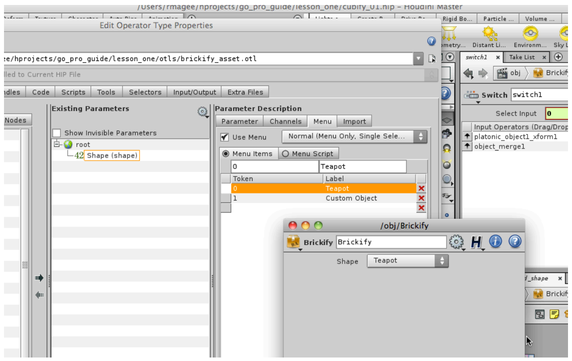
9 **RMB-click** on *brickify* in the path bar and choose **Type Properties...** Click on the **Parameters** tab in this window and then select both the **Transform** and **Subnet** folders. Turn on the checkbox next to **Invisible** and press the **Apply** button.

RMB-click on *brickify* in the path bar and choose **Parameters.** Now you have a floating window with the *brickify* parameters. Right now it is empty. You will add parameters to build up an artist-friendly UI.



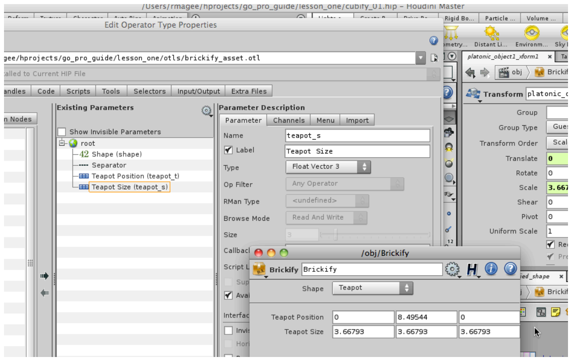
10 In the Network view, click on the *switch* node. Click on the **Select Input** parameter name and drag it to the **Existing Parameters** list in the **Type Properties** window. Drop it on the root to add it to the UI. Click **Apply**.

The *brickify* node now has a new parameter. Change its value from 0 to 1 and back to see how it affects your scene. The problem is the name isn't very appropriate and a menu would work much better than a slider in this case, so you are going to refine the UI using the Type Properties.



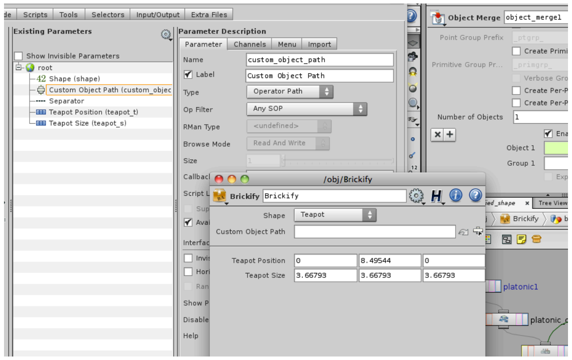
11 In the parameter list, click on the *Select Input* parameter. To the left are options for refining how people see it. Change its **Name** to *shape* and its **Label** to *Shape*.

Now click on the **Menu** tab and turn on **Use Menu**. Now under menu items, type **0** under **Token** and *Teapot* under **Label** then press enter. Next type **1** under **Token** and *Custom Object* under **Label.** Press Apply. Now in the floating parameter pane, you see a Shape parameter with a menu. Try it out to see how it works.



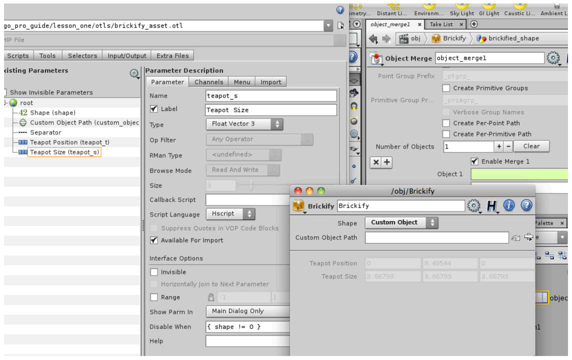
12 When you choose Teapot, you may want to control the size and position of this default shape. From the **Create Parameters** field, drag a **separator** to root. This will add it just under the *Shape* parameter.

In the Network view, select the *platonic_xform* node . Drag the **Translate** and **Scale** parameters over to the parameter list. Change **Name** to *teapot_t* and *teapot_s* and **Label** to *Teapot Position* and *Teapot Size.* Press **Apply** to add the parameters to the asset's UI.



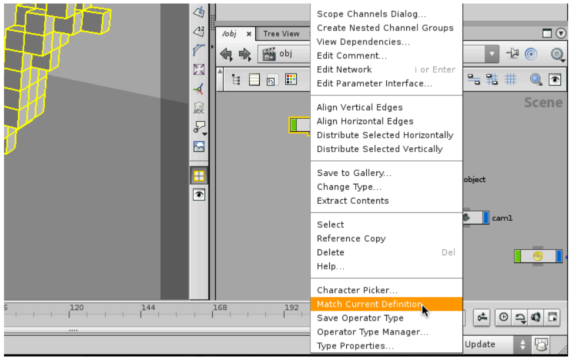
13 When you choose **Custom Object**, nothing shows up because you haven't set that up yet. You are going to use the object merge node to let you reference a custom object from your scene.

In the Network view, select the *object_merge* node. Under Transform Object place a dot (.) then drag **Object 1** over to the parameter list. Change its **Name** to *custom_object_path* and its **Label** to *Custom Object Path*. Press **Apply** to add the parameter.



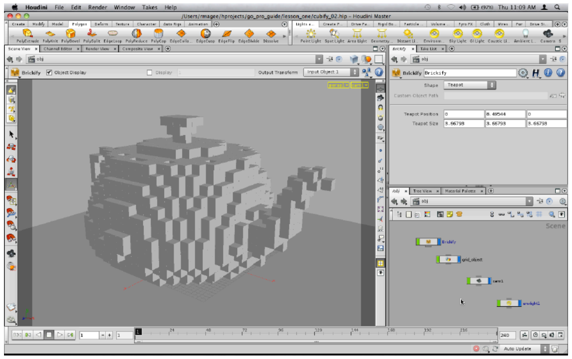
14 To make it clear which parameters are associated with which shape, you can disable and enable them based on the menu choice. Click on the **Custom Object Path** parameter and in the **Disable When** field, enter `{ shape != 1 }`.

This tells this parameter to disable whenever **Custom Object** has not been chosen in the menu. Next, click on **Teapot Position** then shift click on **Teapot Scale** and in the **Disable When** field enter `{ shape != 0 }`. Press **Apply** and test the results using the **Shape** menu.



15 Now press **Accept** to save the changes to your asset and close the Type Properties panel. You can continue to make changes to make your asset more versatile.

In one of the network paths, click on obj to go back up to the object level. **RMB-click** on the *brickify* node and select **Match Current Definition.** Now the inner workings of the asset are locked. If you dive down and select any of the nodes inside the brickify asset, you will see that their parameters and grayed out and cannot be changed.



16 Now **Save** your scene file to preserve the work you have done so far. You now have the scene file and the .otl file which is being referenced into your scene to create the asset. You can use this library to create other instances of the asset in this scene or to add the asset to another scene.

In the next section of this lesson, you will test out your asset and find out if it is working properly. It is always good to have one working asset and one for testing to make sure it is doing what you want it to do.

TIP

THE POWER OF OBJECT MERGE

Since objects in Houdini live within their own discreet networks, the **Object Merge** node is needed to grab geometry from one network and bring it into another. This connection remains live and changes to the original network are reflected in this node.

You can now bring geometry from one location and work on it in a new network. This technique is often used to grab geometry to act as emitters for particles or some other VFX.

One way to quickly create an object merge is to select an object then from the **Modify** shelf, choose the **Extract** tool.

If you select multiple objects then **Extract** will give you multiple *objectmerge* nodes which are merged together in a new object and if you select faces at the geometry level **Extract** will give you an *objectmerge* node which grabs those faces.

Refining the Asset

When you create a digital asset, you are creating a tool that can be shared with other artists and used in multiple shots. With the brickify asset, the Custom object parameter will let you bring in any piece of geometry and turn it into toy bricks.

Using Houdini's default toon character you will test the *brickify* asset then add features such as surface offset and color controls. The changes made to one asset in your scene will be automatically picked up by other instances of the asset.

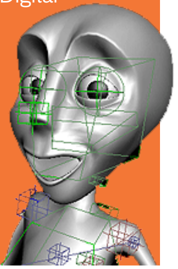
DEFAULT TOON RIG

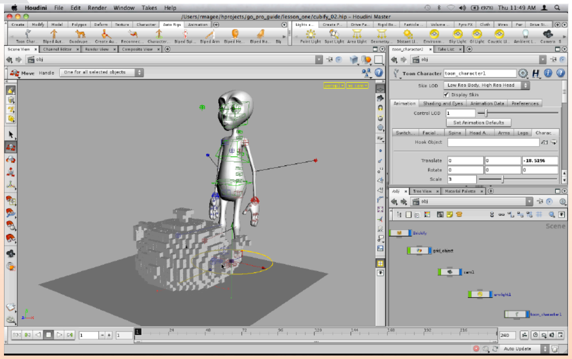
TIP

Houdini comes with a ready-to-animate **Toon Character** which you can find on the **Auto Rigs** shelf. This rig can be posed and keyframes can be set to bring him to life.

The Toon rig is a Digital Asset which all of the character controls built into the asset.

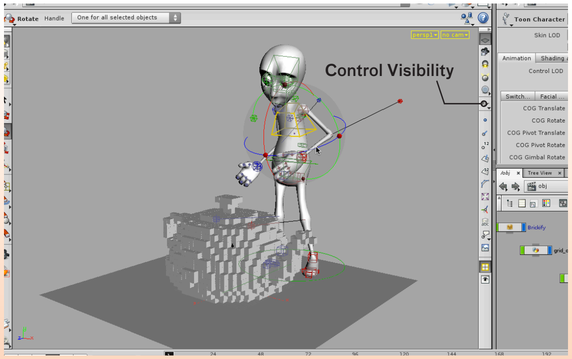
Working with it is a great way to learn how characters work in Houdini.





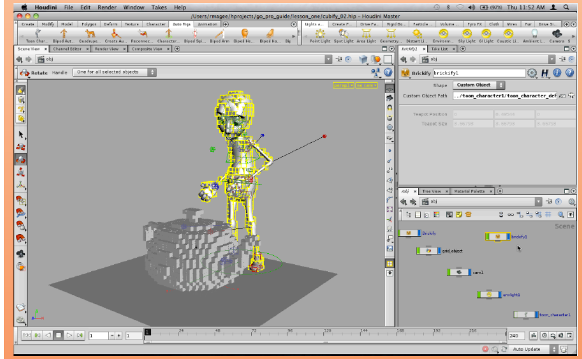
1 From the **Auto Rigs** shelf, click on the **Toon Character** tool. Press **Enter** to place it at the origin. Select the **Move** tool and click on the circle handle at the base of the character. Move the character to the side of the teapot.

In the Parameter pane, click on the **Character Placer** tab under **Animation** and drag the **Scale** slider to around **3** to make the character bigger. You will be using the brickify tool on this character.



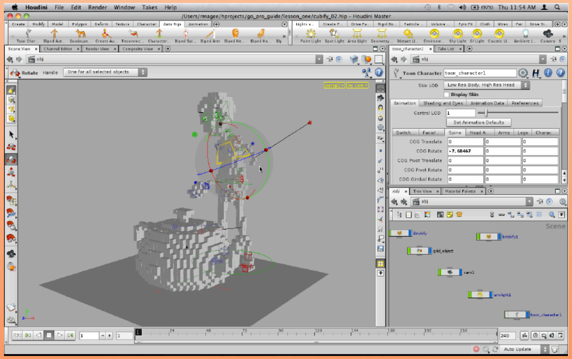
2 To pose the character, click on any of the control handles and then either use **Move (t)** or **Rotate (r)** to pose that body part. You can move around the character tweaking the pose to get the look you want.

Once you have the pose you want, you can use the **Control Visibility** button to turn off **Nulls** which hides the controls and lets you see the bricks better. Later if you want to pose the character some more then you can turn on these controls using the same button.



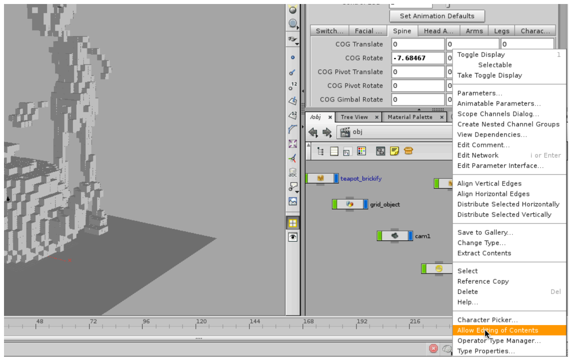
3 In the Network view, press **tab > Digital Assets > Brickify**. Place this node down. At first it creates another teapot. Change its **Shape** to **Custom Object** then click on the node icon next to **Custom Object Path**.

In the **Choose Operator** window, click on the **plus[+] sign** next to *toon_character* then again for *toon_character_deform*. Select the *skin* node then turn on **Export Relative Path** and press **Accept.** Now you can see bricks filling up the character's surface.



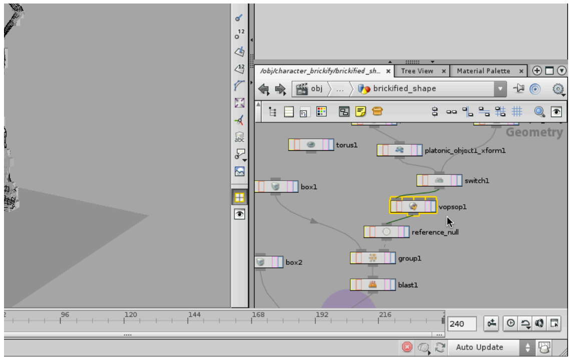
4 Click on the toon_character node and in the parameter pane, turn off the Display skin check box. Now you can see the character displayed as bricks and the control handles are still available.

Now you can select and manipulate the control handles with either use **Move (t)** or **Rotate (r)** tools. As you do the brickify effect will update to reflect the change. Later if you choose to animate the character then the bricks would update over time.



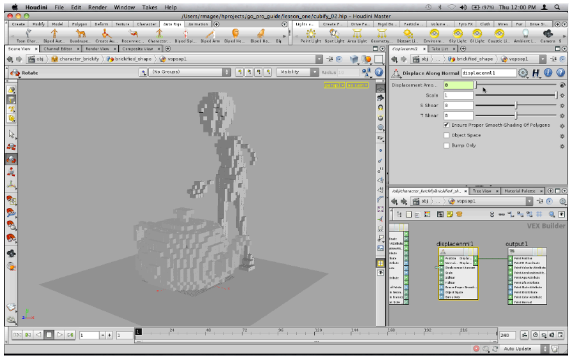
5 Rename the new *brickify* as *character_brickify* then rename the original *brickify* as *teapot_brickify*. **RMB-click** on *character_brickify* and select **Allow Editing of Contents.** Now you can dive in and make changes to the asset. Later when you save these changes to the .otl file, the teapot's *brickify* node will update to reflect the new controls.

Since the character has skinny arms and legs, the brickify effect is leaving gaps. These gaps don't look right and you need a way to solidify the character using the *brickify* tool.



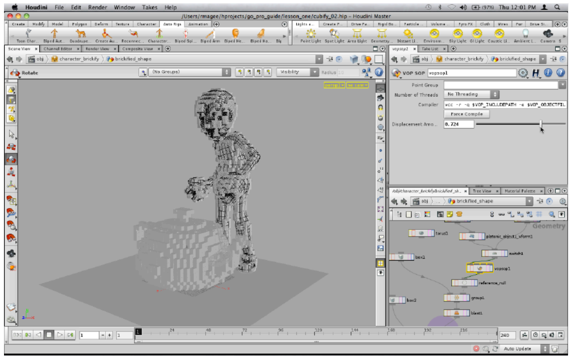
6 **Double-click** on the *character_brickify* node to dive in then **double-click** again on the *brickifed_shape* node. You are now going to modify this network to let you add more points to the effect by offsetting the surface.

Since Houdini doesn't have a surface offset tool, you will create a custom node using Houdini's vector expression tools known as VEX. **RMB-click** on the output of the *switch* node and select **Manipulate > VOP SOP**. Place this node down between the *switch* and the *reference_null* nodes.



7 Double-click on the *vopsop* node to dive in. Move your cursor over the network view and press **h** to find the global and output nodes. **RMB-click** on the **Point Position** input on the output node and select **Displace > Displace along Normal.** This will affect the surface at the geometry level.

Click on the **gear** icon next to **Displacement Amount** and from the menu select **Promote Parameter**. This will make this parameter available at the geometry level. Press **u** to jump up one level. Select the *vopsop* node.



8 Change the **Displacement amount** to a see how this affects the tool. You can see the character getting chubbier and more bricks appearing in the arms and legs. This is because you are offsetting the surface and more points are being added to the *inside_points* group.

You are now going to promote this parameter up to the object level so that it becomes part of the *character_brickify* asset's interface. This way artists can work at the top level and don't have to dive down to get what they want.

TIP

VEX AND VOPS

VEX is a high-performance vector expression language which is used in Houdini to write shaders and create custom nodes. It is loosely based on C but takes ideas from C++ as well as the RenderMan shading language.

Before you are scared off by all this talk about programming, you should know that Houdini also has an interactive node-based workflow for creating VEX code using VEX Operators, otherwise known as **VOPs**.

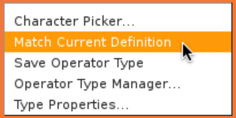
As you can see in this lesson, custom geometry nodes can be created using VOPs by simply wiring a couple of nodes together. There are many other uses you will encounter as you continue to work with Houdini. VEX is a powerful tool which is often used in production.

SYNCING DIGITAL ASSETS

TIP

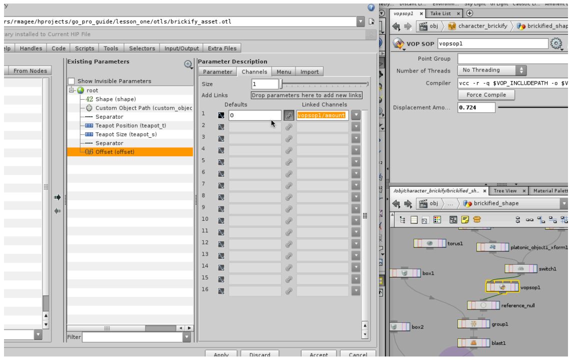
Digital Assets in your scene can either be matched with the asset definition in the .otl file or available for editing. If you are editing it then the asset in your scene is out of sync with the definition.

Once you edit your Digital Asset, you can save your changes by either pressing **Apply** in Type Properties or by **RMB-clicking** on your asset and choosing **Save Operator Type**.



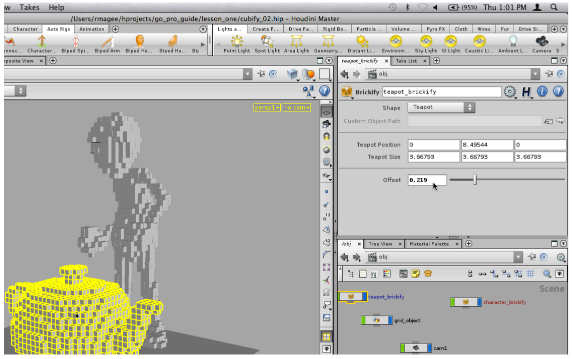
Once you know that the definition in the .otl file and the asset in your scene are in sync then you can use Match Current Definition to keep them synced. If someone else changes the definition it will automatically load into your scene and update your working asset.

Therefore it is important to use an asset management system when working with Digital Assets in production to define the rules for creating and sharing assets.



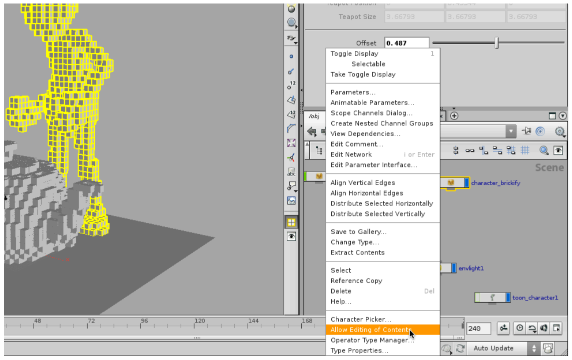
9 **RMB-click** on *character_brickify* in the path bar and choose **Type Properties...** Click on the **Parameters** tab then drag the *vopsop's* **Displacement Amount** from the Parameter pane to just under the *shape* parameter in the Type properties window. Change its **Name** to *offset* and its **Label** to *Offset*. Under **Parameter description**, click on the **Channels** tab and set the **Defaults** to **0**.

Click on **Accept**. This saves the contents of the asset and adds the new parameter to the asset's UI.



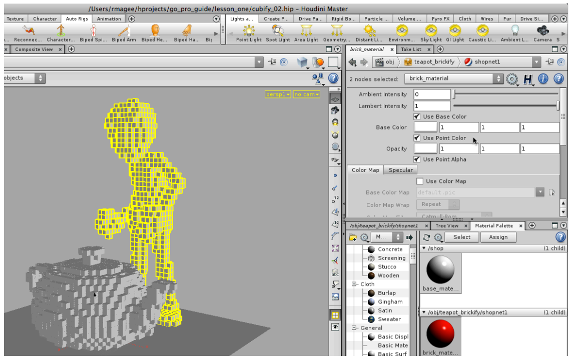
10 Click on **obj** in one of the path widgets to jump up to the Object level. Click on the *character_brickify* node and you can see the **Offset** parameter in place. You can now tinker with this to thicken up the bricks on the arms and legs.

Select the *teapot_brickify* node. It also has the **Offset** parameter. When you updated the *brickify* asset definition in the the .otl file it was automatically picked up by the *teapot_brickify* node. Brickify assets in other scene files which reference the same .otl file would also pick up these changes.



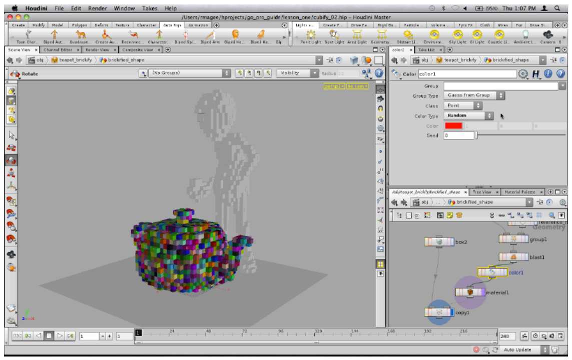
11 **RMB-click** on the *character_brickify* node and select **Match Current Definition.** This make sure that the asset and the saved definition are the same. This asset is now locked and you will make your next set of changes to the teapot_brickify node. This way you can see that you can use either asset to make changes as long as the other one has its defintion matched.

RMB-click on the *teapot_brickify* node and select **Allow Editing of Contents.**



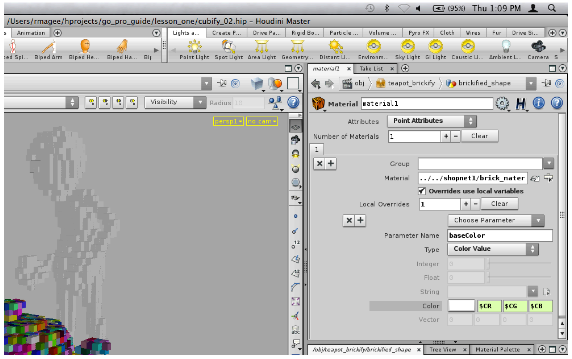
12 Bring up the Material palette. Open up the */obj/teapot_brickify...* folder and select the *brick_material* node. Set the **base color** to **1, 1, 1** and make sure that **Use Point Color** is turned **on.**

In the Network view, press **u** to back up one level then double-click in *brickified-shape.* You will now add a color node to control the color of the bricks then you will promote the relevent parameters to the asset.



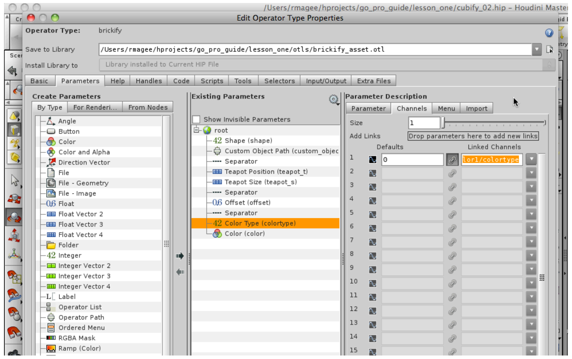
13 RMB-click on the *blast* node's output and select **Material > Color.** Place the node down. In the parameter pane, change the **Color** to red - **1, 0, 0**. Right now this will work if you render the teapot but you can't see it in the viewport.

Select the *copy* node. Click on the Attribute tab and turn on **Use Template Point Attributes.** Now you can see the color in the viewport. To test this, select the *color* node again and set **Color Type** to **Random.** You will now make sure that these points will be used when rendering.



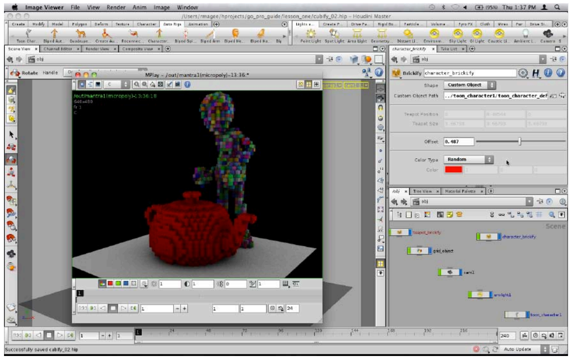
14 Click on the *material* node. In the Parameter pane, turn **on** the **Overrides use local variable** option then click on the **plus sign [+]** next to **Local Overrides.** From the **Choose Parameter** menu, select **Base Color.**

In the **Color** parameter, enter **$CR, $CG, $CB** in the three fields to use the point color as your local override. This way each instance of the brick will use a unique material color. Without this override, the point colors would not be used when you render the shapes using Mantra.



15 **RMB-click** on *teapot_brickify* in the path bar and choose **Type Properties...** Click on the **Parameters** tab then on the **gear** icon next to **Existing Parameters** and turn off **Prefix Linked Parameter Name...** and **Prefix Linked Parameter Label...**

Drag the *color* node's **Color Type** parameter from the Parameter pane to the bottom of the parameter list. Next, drag the *color* node's **Color** parameter. You should also add a **separator** between just before the **Color Type** parameter.



16 Press **Accept** to save your changes. Jump up to the object level and you can see that both the *teapot_brickify* and the *character_brickify* have the new color controls controls at the top level of the assets.

If you want to test the look of the two brickified objects, go back to the camera and adjust the view then re-render using the **Render > Render > mantra.** The various asset controls are now playing a part in creating your final image.

TIP

DON'T FORGET TO SAVE

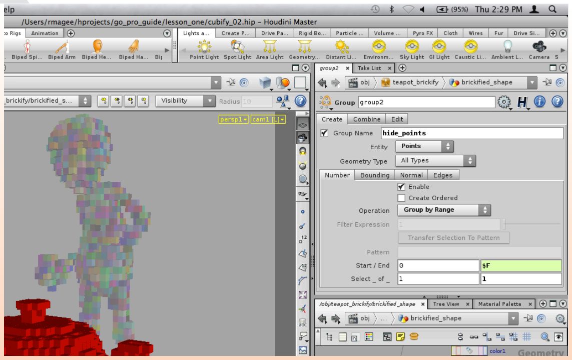
From now on, the responsibility to save your scene files will be left to you. Use **File > Save** or choose **Edit > Auto Save** then set the save interval in the preferences. The default is every 1 minute.

When you save your scene, a backup file is automatically created. When you save a digital asset, a backup of the .otl file is also made. This can take up lots of disk space so be sure to keep an eye out for these extra files. You can turn the backup feature off in the preferences.

Animating the Bricks

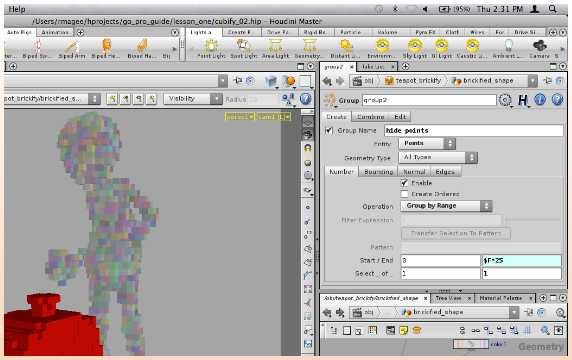
You are now going to add some animation controls to animate the bricks being built over time.

Just like the color controls, you will add some nodes to the network of the brickify asset then promote parameters and save. Other instances of the asset will then benefit from the new controls


```
Group  group2
Group Name    hide_points
Entity        Points
Geometry Type All Types
Number  Bounding  Normal  Edges
[✓] Enable
[ ] Create Ordered
Operation     Group by Range
Start / End   0    $F
Select _ of _ 1    1
```

1 Make sure you are inside the *teapot_brickify* asset and inside the *brickified_shape* object. In the Network view, **RMB-click** on the output of the *color* node and select **Manipulate > Group Geometry.** Set the Group Name to *hide_points* and **Entity** to **Points.**

Go to the Number tab and change **Operation** to **Group by Range.** Now change the **End** expression to `$F` and **Select _ of _** to **1** and **1.**


```
Group  group2
Group Name    hide_points
Entity        Points
Geometry Type All Types
Number  Bounding  Normal  Edges
[✓] Enable
[ ] Create Ordered
Operation     Group by Range
Start / End   0    $F*25
Select _ of _ 1    1
```

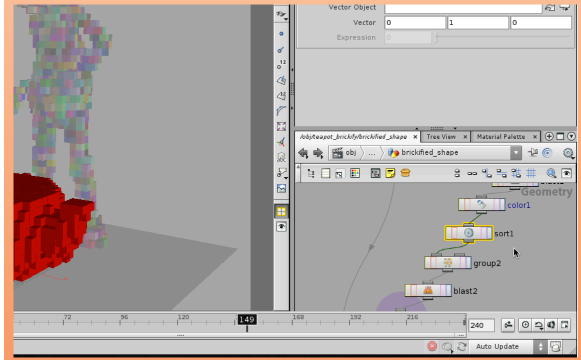
2 **RMB-click** on the output of the *group* node and select **Polygon > Blast.** Place this node down then using the arrow next to **Group**, select the *hide_points* group. Now turn on **Delete Non Selected** to delete points outside the group. Press play to watch as the teapot starts with one brick then grows a brick every frame.

To speed up the effect and to start with no bricks, change the **End** expression on the *group* node to `($F-1)*25`. This will create 25 bricks every frame. Press play to test.

TIME-BASED EXPRESSIONS

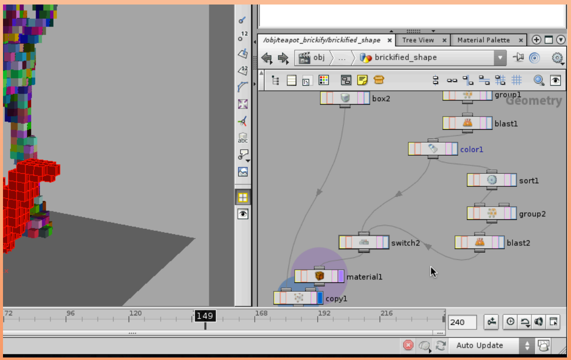
To animate the bricks, you are going to use a global variable called `$F` which represents the current frame. You can use this to add time into your expression. You will also use `$F` when you are saving out image or geometry sequences.

You can also use the variable `$T` which measures time in seconds. Since there are 24 frames per second then this will measure time 24 times slower. If you had changed the frames per second to 30 then it would be even slower than that.


```
Vector  0  1  0
```

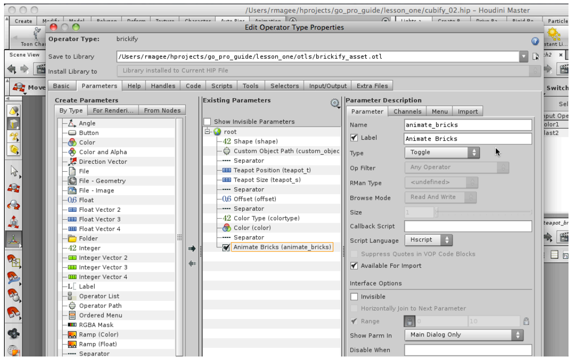
3 Right now the bricks are coming in from one side instead of from the ground. This is because the points are appearing based on their point numbers. To control this, you need to reorder the points to create the look you want.

RMB-click on the output of the *color* node and select **Attribute > Sort.** Place this node down and change **Point Sort** to **Along Vector.** With this set to **0, 1, 0**, the points start at the bottom and go up. Playback to see this result. Test out different vectors to see how it affects the animation.



4 To let you choose whether you want to animate the brickify effect, you can add a switch node. **RMB-click** on the output of the new *blast* node and select **Sourcing > Switch.** Place the node down then click on the output of the *color* node and feed it into the input of the *switch* node.

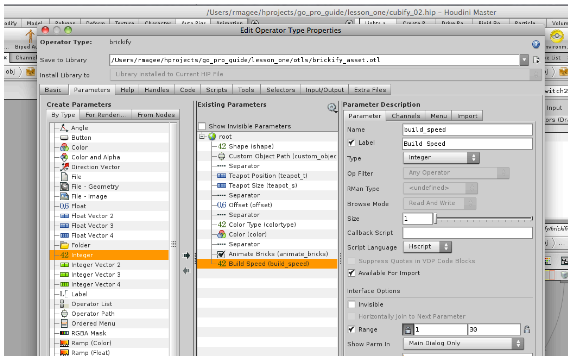
In the Parameter pane, click the up arrow next to *color* to move that input operator to the top of the list. The animated effect will be you second option. Changing **Select Input** will bring it back but for now keep it at **0.**



Name: animate_bricks
Label: Animate Bricks
Type: Toggle

5 **RMB-click** on *teapot_brickify* in the path bar and choose **Type Properties...** Click on the **Parameters** tab then from the **Create Parameters by Type** tab, add a **separator** just after *Color*.

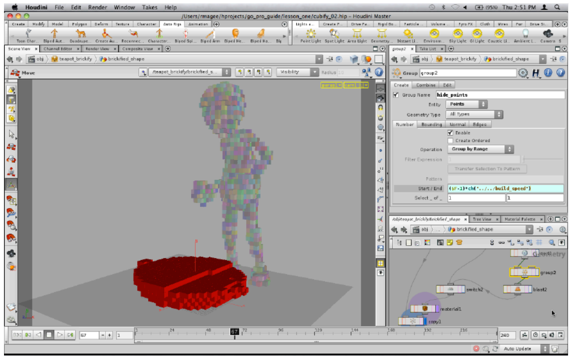
Next, drag the *switch* node's **Select Input** from the Parameter pane to just under the new separator. Set its **Name** to *animate_bricks* and its **Label** to *Animate Bricks*. Next, change its **Type** to **Toggle** which limits to you an on[0]/off[1] setting. Click **Apply** to save changes.



Name: build_speed
Label: Build Speed
Type: Integer
Range: 1 30

6 From the **Create Parameters** section in Type Properties, drag an **Integer** parameter under the *animate_bricks* parameter. Set its **Name** to *build_speed* and its **Label** to *Build Speed*. Turn on the **Range** option then set the first value to **1** and the second value to **30**. Click on the **lock** next to 1 to make sure the number never gets smaller than 1.

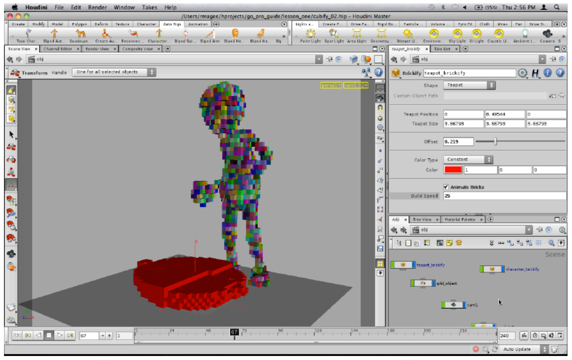
Press **Accept** to save these changes and close the window. This parameter is not currently linked to any of the nodes inside the asset but you will add it to the **End** expression.



Group Name: hide_points
Start / End: ($F-1)*ch("../../build_speed")

7 Select the new *group* node and change the End expression to: `($F-1)*ch("../../build_speed")`

Now you can change the speed of the animation using this parameter. Click on obj in any path widget to go up to the top level then **RMB-click** on the teapot_brickify node and select **Save Operator Type**. This lets you save this expression to the .otl file without opening up the Type Properties window. Don't forget this step or else the expression won't be part of the asset's definition.



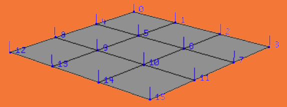
8 **RMB-click** on the teapot_brickify node and select **Match Current Definition** to lock it. You now have the brickify effect wrapped up into a custom tool which can be used on different shots by different artists.

Any changes or enhancements made to the asset will be picked up automatically. If you want to take this further you could unlock one of the assets and try promoting the **Vector** attributes from the *sort* node. The key is that you can continue to make the tool better.

TIP

EVALUATING GEOMETRY ATTRIBUTES

In 3D apps, geometry is stored using information such as point numbers, normal directions and more. In most apps this information is hidden from you and you are not able to use it in your work.



In Houdini, you can not only display these attributes but you can also manipulate them in a number of ways. In this lesson, you are actually sorting the point numbers while in other cases you might be redirecting the point normals.

This low level control over your geometry is a key feature of Houdini which you will turn to often in your projects.

Create a Peg Board

The brickified objects now need a peg board to act as a base. This will be set up as a second digital asset which will be stored in the same .otl file. Each operator type library can contain multiple assets.

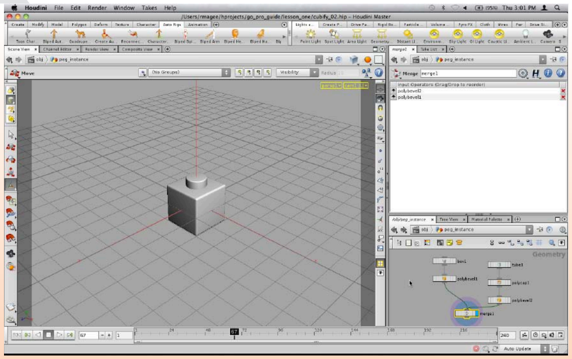
Along the way you will borrow some of the work you did on the brickify asset and utilize the same instancing approach to create the pegs.

COPY & PASTE

TIP

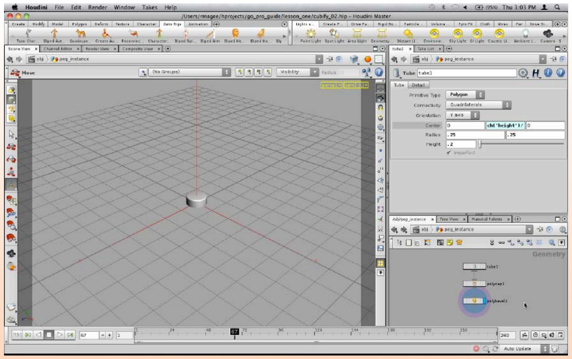
When working with nodes, one powerful tool is the ability to simply **Copy** (Ctrl[Cmd]-c) and **Paste** (Ctrl[Cmd]-v) nodes.

Whether you are copying them inside the same network, over to another network or even to another version of Houdini, this is one of the easiest ways of experimenting by re-working and re-purposing your nodes and networks.



1 In the Network view, **double-click** on the *teapot_brickify* asset to dive down one level. Select the *brick_instance* object then **RMB-click** in empty space and select **Copy** from the menu. Press **u** to go up one level and then **RMB-click** in empty space and select **Paste**.

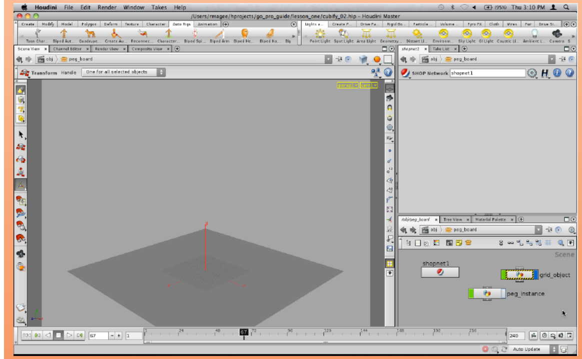
Rename the node *peg_instance* then **double-click** on it to dive down to the geometry level. From the visibility menu in the Scene view, select **Hide Other Objects**. You can see that brick with the peg on top.



2 Select and delete the *box*, its *polybevel* and the *merge* nodes. Now the peg is floating above the ground. Select the tube node and remove the `+ 0.5` for the **Center Y** expression. It should now read:

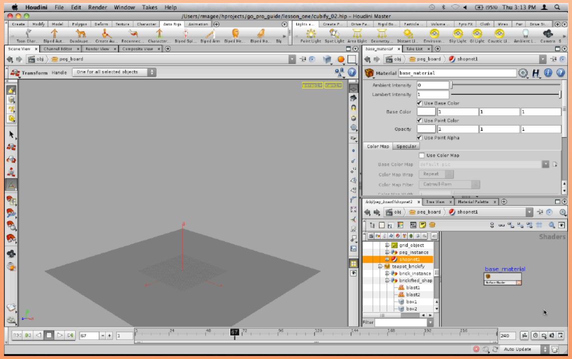
```
ch("height")/2
```

You will now instance this to a grid to create the pegboard. Press **u** to go back up to the object level and make sure that the display flag on *peg_instance* is off so it is hidden.



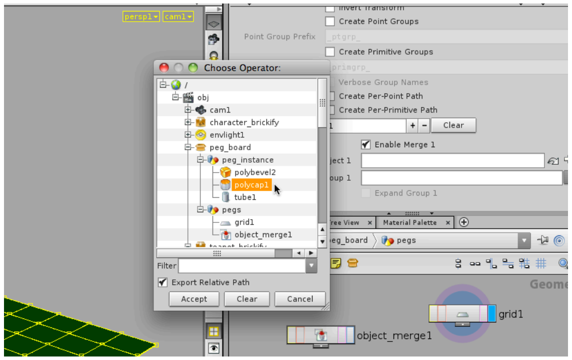
3 Find the grid node which you set up earlier as a ground surface and move it beside the *peg_instance* node. Select the two nodes then click on the **Create Subnet from Selected** button.

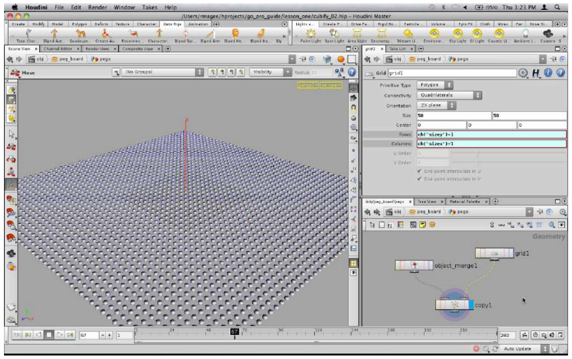
Rename the new node *peg_board* and then double click on it to dive down one level. Press **tab > Managers > SHOP network** then place this node down. You will move the base_material you created earlier to this location so that you can make an enapsulated digital asset of all the nodes.



4 Press **w** to bring up a tree view. Click on *shop* then select the *base_material* node and then **RMB-click** in empty space and select **Cut**. Navigate to the new shopnet then **RMB-click** in empty space and select **Paste**.

Press **w** to close the tree view then **u** to go up a level. Select the *peg_instance* and ground *grid* nodes. In the Parameter pane, select the **Material** tab then click on the **node selector** button. Select *base_material* inside the *shopnet* then turn on **Export Relative Path** and click **Accept**.



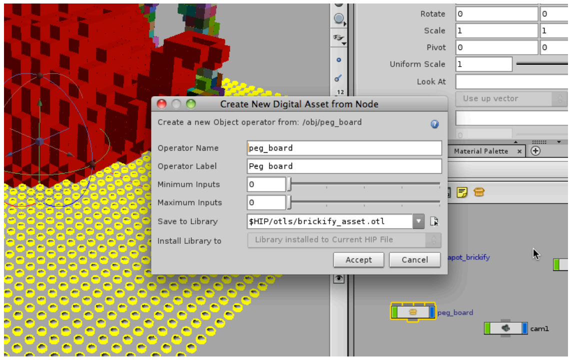


5 Select the ground grid object and rename it *pegs*. Double-click to dive in then press **tab > Sourcing > Object Merge.** Place the node object then in the Parameter pane, set **Transform Object** to a **single dot [.]** then click on the node select button next to **Object 1.**

From the selector, find the *peg_instance* object and open it up using the plus sign [+]. In order to keep the geometry light for the viewport, select the *polycap* node. This merged peg will be used for viewport visualization only.

6 **RMB-click** on the output of the *object_merge* node and choose **Utility >Copy.** Place the node down then feed the output of the *grid* node into the second input of *copy*. Turn off **Transfer Using Template Point Attributes.**

Drag the **Size X** parameter to the **Columns** parameter. Select **Relative References** then add `+1` to the expression. Repeat with **Size Y** copied to **Rows** parameter and add `+1`. Press **Ctrl** and click on the grid node's display flag to set it as the Render node.

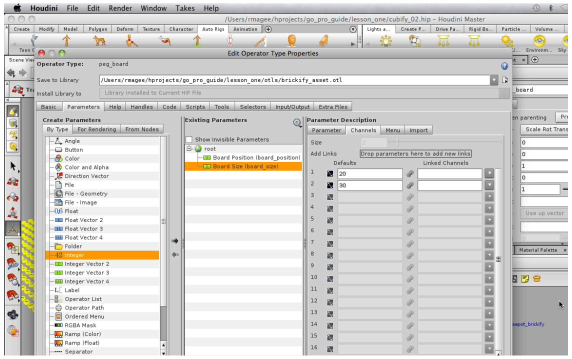


Create New Digital Asset from Node

Create a new Object operator from: /obj/peg_board

Operator Name: peg_board
Operator Label: Peg board
Minimum Inputs: 0
Maximum Inputs: 0
Save to Library: $HIP/otls/brickify_asset.otl
Install Library to: Library installed to Current HIP File

Accept Cancel



Operator Type: peg_board
Save to Library: /Users/rmagee/hprojects/go_pro_guide/lesson_one/otls/brickify_asset.otl
Defaults: 20, 30

7 Jump up one level and set up instancing on the pegs object. Use **Edit Parameters** like when you set up the **brickify** solution and reference the *peg_instance* node.

Go up another level and **RMB-click** on the *peg_board* object and select **Create Digital Asset...** Set its **Operator Name** to *peg_board* and its **Operator Label** to *Peg Board*. Click on the **file selector** next to **Save to Library.** Click on *$HIP* then double go to the *otls* folder. Select the *brickify_asset.otl* file and click **Accept** twice.

8 Now the new *peg_board* asset is in the same library file as the *brickify* asset. In the Type Properties panel, make the **Transform** and **Subnet** folders **Invisible.** Drag an **Integer Vector 2** parameter to the *root* of the **Parameters** list. Set its **Name** to *board_position* and its **Label** to *Board Position*.

Drag another **Integer Vector 2** parameter from the **Create Parameters** list just below the *Board Position* and Set its **Name** to *board_size* and its **Label** to *Board Size*. Click on **Channels** and set its defaults to **20** and **30.** Click **Accept.**

TIP

OPERATOR TYPE LIBRARIES

When you create a Digital asset, you are storing the asset's definition in a special file on disk called an operator type library file [.otl]. This file can contain more than one Digital asset and can even hold other files such as scripts, textures or geometry.

You can view the contents of a .otl file by going to **Window > Operator Type Manager.** Here you can see which libraries have been loaded into your scene and what assets can be found inside.

At the end of this lesson, you will start a new scene and install the *brickify_asset.otl* file so that you can use the *brickify* asset. This will begin to show you how assets can be shared between scenes and with your colleagues.

TIP

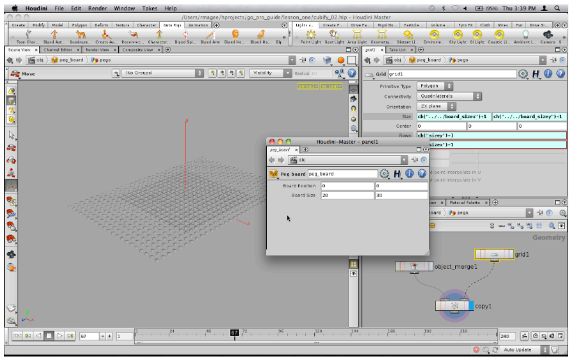
NON-LINEAR THINKING

In steps 11 and 12, you do something very simple that demonstrates a non-linear though process. In step 11 you set **Size Y** very big even though you ultimately want the pegboard to be very thin.

By keeping the board thick at this point, it is easier to select the four edges and bevel them. If the box had been very thin then selecting those edges would have required lots of zooming and tumbling.

You are then able to go back to the box node and set **Size Y** to a much smaller value and the pegboard is all set.

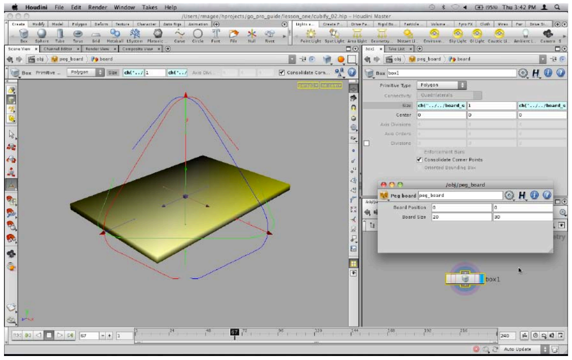
As you work with Houdini, be sure to use its node-based structure to think non-linearly.



9 Click on arrow on the right side of the parameter pane and choose **Tear Off Pane Copy.** Click on the **pin** icon.

Dive into the *peg_board* node then the *pegs* node. In the floating pane, **RMB-click** on the **Board Size** parameter and select **Copy Parameter**. **RMB-click** on the *grid* node's **Size** parameter and select **Paste Copied Relative References**. Edit the expressions to make them even:

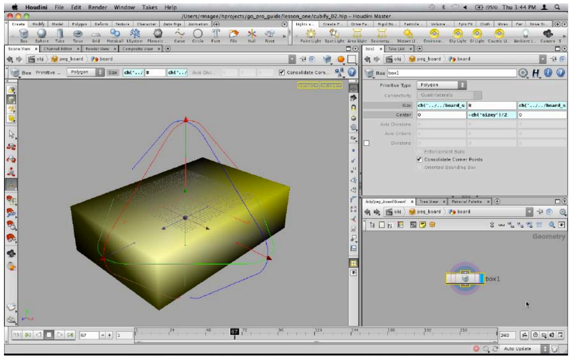
```
ceil("../../board_sizex"/2)*2
```



10 Jump up one level. Click on the **Box** tool in the tool-shelf and press **Enter** to place it at the origin and its node in the *peg_board* subnetwork. Rename it *board* then go into it.

From the floating pane, **RMB-click** on **Board Size X** and select **Copy Parameter. RMB-click** on the *box* node's **Size X** parameter and select **Paste Copied Relative References.** Edit the expression to make it even then add one to make it bigger. Do the same with **Board Size Y** and **Size Z**:

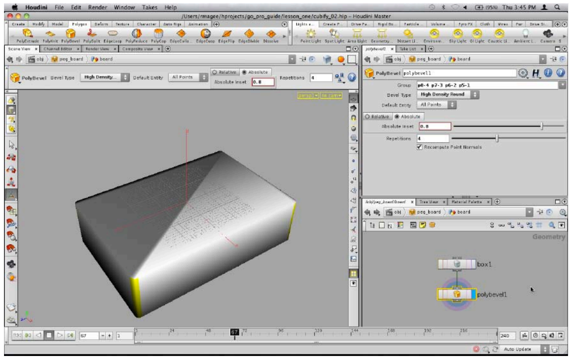
```
ceil("../../board_sizex"/2)*2 + 1
```



11 **Double-click** on the *box* node's **Size Y** parameter. Now drag the selected number down to the **Center Y** parameter. From the **Paste** window, select **Relative Channel Reference.** Now add a negative sign [-] in front of the channel expression and /2 at the end to get the following expression which puts the board just under the pegs:

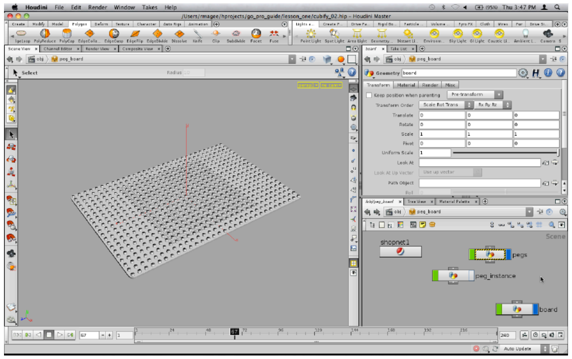
```
-ch("sizey")/2
```

Set the **Size Y** to **8** to make the box thicker for now.



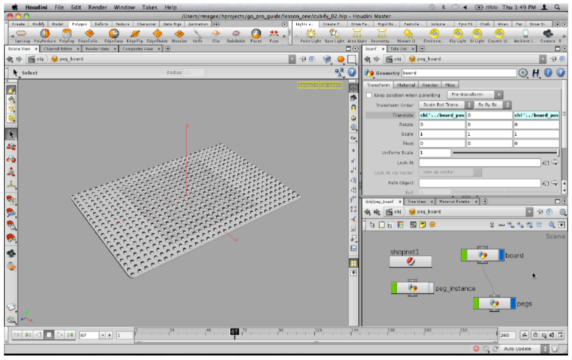
12 In the viewport, tumble the view to see the box from the side. Press **select** then **3** to turn on edge selection. select all the side edges of the box then from the **Polygon** shelf click on the **PolygBevel** tool.

Tumble back then in the parameter pane, click on the **Absolute** tab and set the **Absolute Inset** to **0.8**. Next, Set the **Bevel Type** to **High Density Round** and **Repetitions** to **4.**



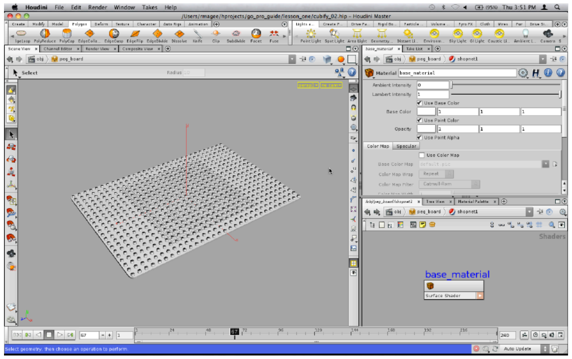
13 In the Scene view, press **a** to select all the edges then from the **Polygon** shelf, select **Facet**. In the Parameter pane, turn on **Cusp Polygons** and set the **Cusp Angle** to **80**. Now select the *box* node and set the **Size Y** to **0.25**. Now you have a thin board with rounded edges.

Press **u** to jump up one level. To save your changes to the asset, **RMB-click** on the *peg_board* asset in the path widget and select **Save Operator Type**. Jump up one level to see the pegs sitting nicely on the board.



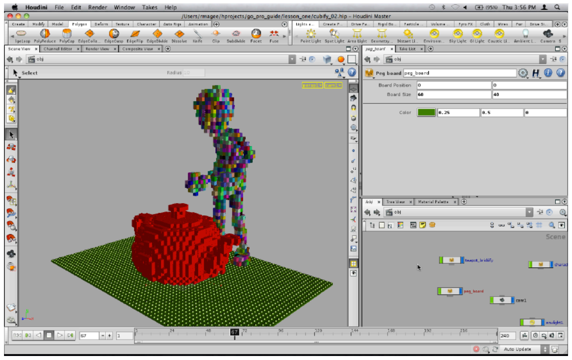
14 Next connect the output of the *board* node to the input of the *pegs* node. This will parent the pegs to the board.

From the floating *peg_board* parameter pane, **RMB-click** on the **Board Position X** parameter and select **Copy Parameter**. Then **RMB-click** on the *board* node's **Translate X** parameter and select **Paste Copied Relative References.** Next copy the **Board Position Y parameter** and paste it using relative references to **Translate Z.** Now you can position and size the board.



15 Select the *board* nodes and in the Parameter pane, select the **Material** tab then click on the **node selector** button. Select *base_material* inside the *shopnet* then turn on **Export Relative Path** and click **Accept**. Now all three of these objects are textured using the *base_material* shader.

Double-click on the *shopnet* node and select the actual *base_material* node. You are now going to add one of its parameters to the interface of the peg board asset.



16 Open up the Type Properties for the *peg_board* asset and drag the **Base Color** parameter from the *base_material* to the **Parameter list.** Change its **Name** to *color* and its **Label** to *Color*. Add a separator between this parameter and the Board Size parameter. Click **Accept** to save the changes to the asset definition.

Go back to the object level to see the asset sitting under the brickified teapot. You may want to change its size to better suite the size of the teapot.

TIP

ENDLESS LEVELS OF CONTROL

As you create your digital assets, you can promote as many parameters as you like which can be both a good thing and a not so good thing.

Be sure to design your assets beforehand and include all the controls the artist will need in production. While it is easy to add more control later, it is always a good idea to think ahead.

Planning will help you avoid "featuritis" where you add more and more parameters without considering what the artist really needed. If the UI of your asset is too cluttered then it may diminish the artist's ability to use it effectively.

Re-using the Asset

You have now built two tools which use Houdini's procedural networks to create a flexible way to turn any object into a brickified shape. You have already seen that you can use the asset within this scene. Now you will use it in another scene.

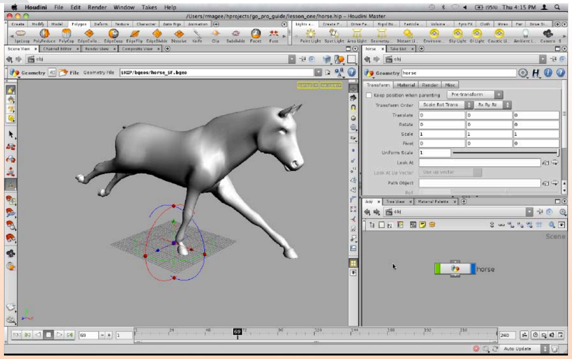
Digital assets make it easy to quickly share your networks with others and let them use your solutions in their work. To finish up, you will brickify a galloping horse model.

TIP

SHARING & RE-USING ASSETS

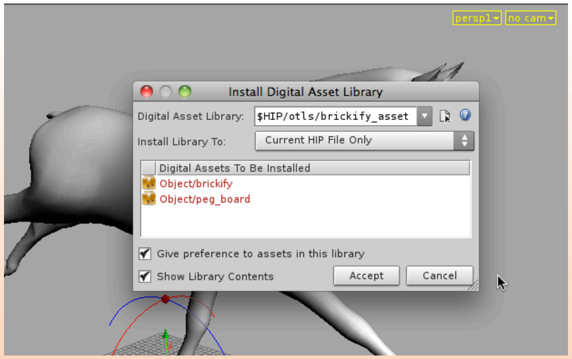
One of the biggest benefits of Houdini's node-based workflow is the ability to re-use your work for new shots or even new projects. And with Digital Assets, you have a means to package up and deliver these networks in a production pipeline setting.

Individual artists can also maintain libraries of assets for use in their projects and at any time you can dive into the asset and update it with new nodes and parameters.



1 You can now compose your scene and either render a still image or render out an animation of the bricks being built up. Lets do that with a new scene. **Save** your scene file and then select **File > New.**

Select **File > Import Geometry** and from the window, navigate to the *go_pro_guide/lesson_one/geo* directory. Select the *horse_$F.bgeo* listing to bring in a sequence of geometry files representing a horse galloping in place. Change the **End frame** to **30** to match the animation.



Install Digital Asset Library

Digital Asset Library: $HIP/otls/brickify_asset

Install Library To: Current HIP File Only

Digital Assets To Be Installed

Object/brickify

Object/peg_board

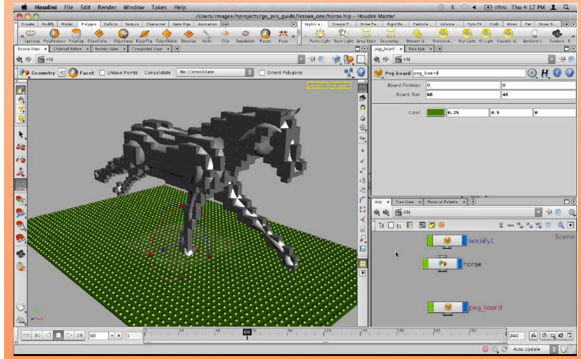
☑ Give preference to assets in this library

☑ Show Library Contents

Accept Cancel

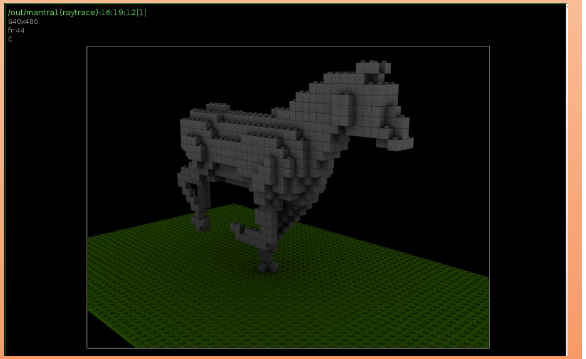
2 In the scene view, you can press play to watch the gallop. You are now going to test the *brickify* tool on an animated piece of geometry.

Select **File > Install Digital Asset Library**. Click on the **file selector** button next to **Digital Asset Library** and navigate to *go_pro_guide/lesson_one/otls/brickify_asset.otl*. Click **Accept** and you will see the two assets listed. Click **Accept** again. Now the two assets are loaded in your scene and you can create them using the tab key.



3 Press **tab > Digital Assets > Brickify** and press **Enter** to place the asset at the origin. Change its shape to **Custom Object** then set the **Custom Object Path** to the horse. Next, press **tab > Digital Assets > Peg Board** and press **Enter**. Change the peg boards **Board Size** to suit the horse then change its **Color** to a green.

You can now play back the animation to see how the procedural networks affect the geometry. As the geometry sequence plays the bricks get rebuilt every frame.



4 You can then add lights and cameras. To render an animation, you can select **Render > Edit Render Node > mantra1** and set the **Valid Frange Range** to **Render Frame Range** and then set your **Start** and **End** frame. Under **Output Picture** create a file name such as `brickify_$F.pic`. and put the files in a folder.

Press **Render** to save out an image sequence. Choose **Render > Mplay > Load Disk Files** to grab the images and view the rendered sequence.