HOUDINI FOUNDATIONS

NODES, NETWORKS & ASSETS

A great way to understand Houdini's node-based workflow is to explore it in the context of a project. It is important to start learning how to think and work procedurally. In this lesson, you will learn how to create your own custom **brickify** tool using procedural nodes and networks to define its function and interface.

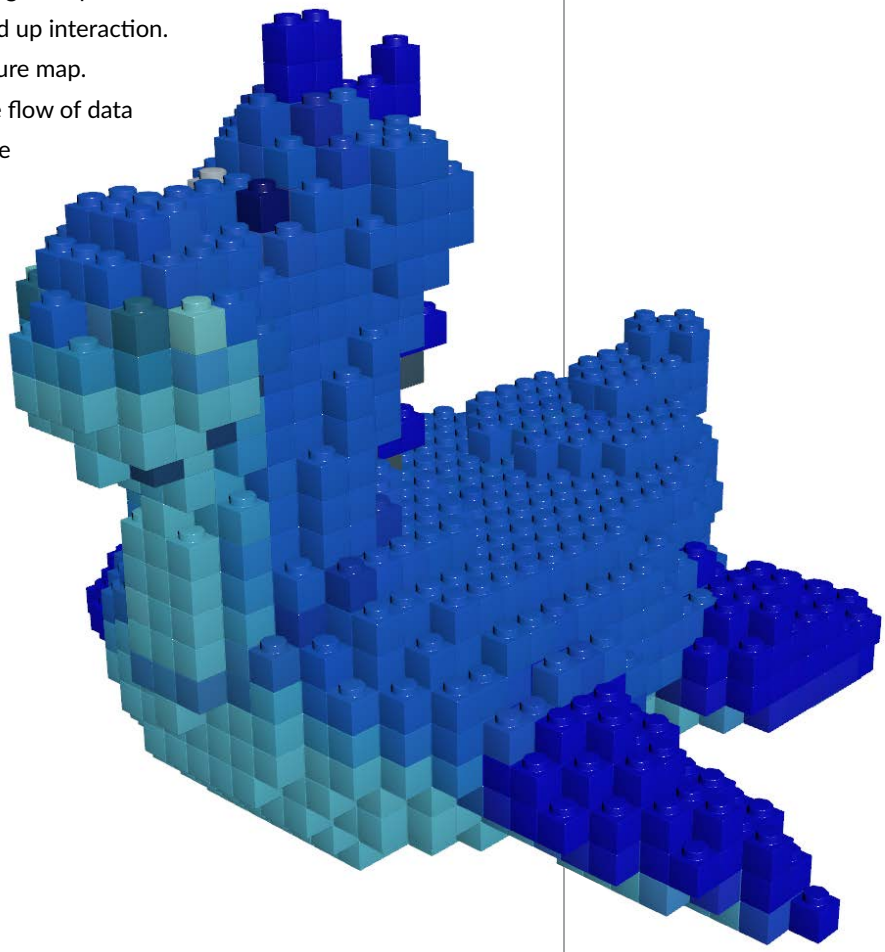
Along the way you will get to use different aspects of Houdini's workspace. By learning how to use these UI elements together, you will dive deeper into Houdini's node-based workflow. The lessons will then give you a chance to put your ideas into practice, which is one of the best ways to learn.

LESSON GOAL

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

WHAT YOU WILL LEARN

- How to model a plastic interlocking brick
- How to break down a default rubber toy shape into a grid of points.
- How to use packed primitives and instancing to speed up interaction.
- How to use attributes to color the bricks using a texture map.
- How to work with nodes and networks to control the flow of data
- How to create a Digital Asset to package up and share your solution with others.
- How to animate the bricks appearing over time.
- How to bring the scene into Solaris for lookdev
- How to render the shot using Karma XPU



LESSON COMPATIBILITY

Written for the features in Houdini 20.5+

The steps in this lesson can be completed using the following Houdini Products:

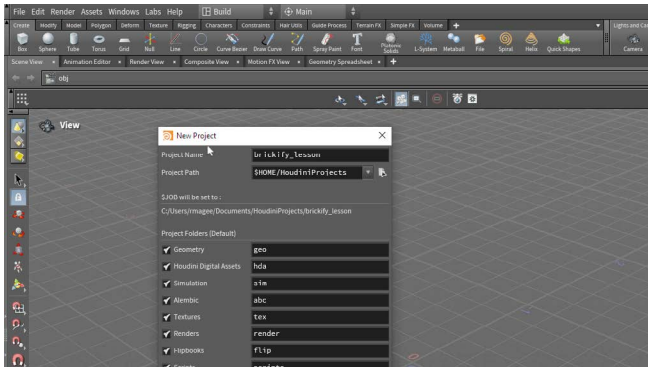
Houdini Core	✔
Houdini FX	✔
Houdini Indie	✔
Houdini Apprentice	✔
Houdini Education	✔

Document Version 4.0 | March 2025

PART ONE

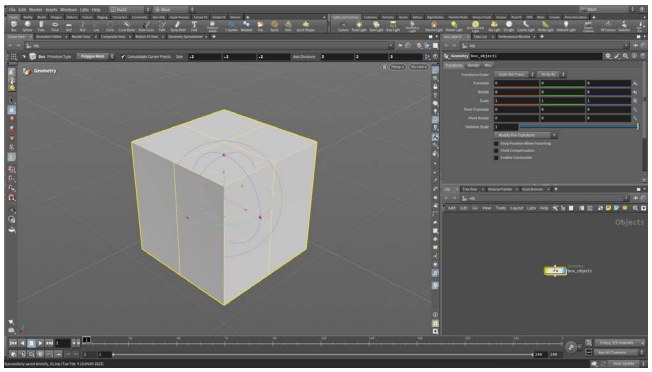
Create a Single Brick

To get started, you will build a model of a single brick that you will later copy onto points to create the brickified shape. You will create this brick using a combination of polygon modelling tools. Along the way, you will see how each action you take creates a node in Houdini. These nodes create a recipe of the steps that can be used to create the brick geometry.



01 Select **File > New Project.** Change the **Project Name** to *brickify_lesson* and press **Accept**. This creates a project directory with subfolders for all the files associated with this shot.

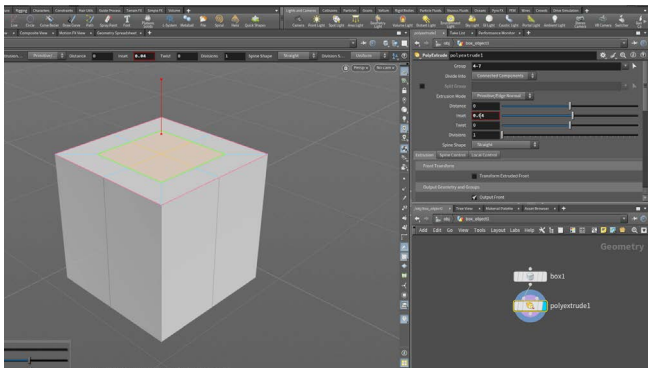
Select **File > Save As...** You should be looking into the new *brickify_lesson* directory. Set the file name to *bricks_01.hip* and click **Accept** to save.



02 In the viewport, **press c** to bring up a radial menu. From this menu, choose **Create > Geometry > Box**. Your cursor now shows the outline of a box waiting to be placed in the scene. **Press Enter** to place it at the origin. In the **Operation Controls** bar, set **Size** to **0.2, 0.2, 0.2** and **Axis Divisions** to **3, 2, 3**.

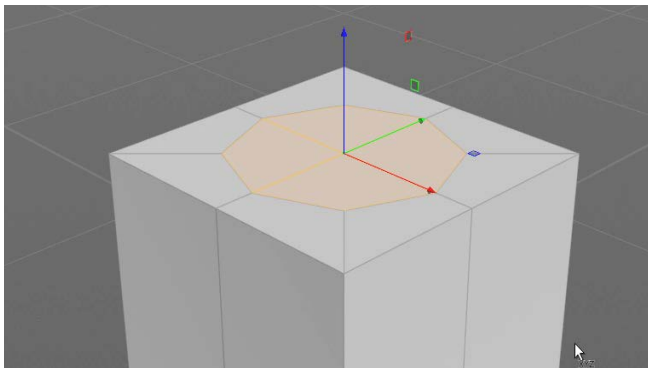
You can see that there is a *box_object* in the Network view. This object level node contains the transformation information for this shape. The **Operation Controls** show you parameters from another *box* node one level down.

Press **Spacebar - f** to focus on the box.



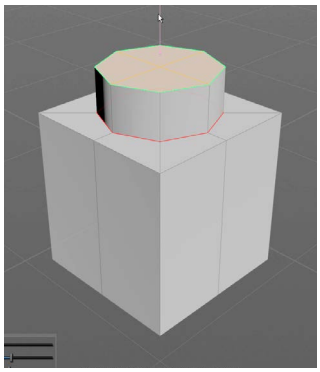
03 Click on the **Select** tool then **Press 4** to go to primitive selection mode. Press the **Shift** key and select the top four faces on the box. **Press c** to bring up a radial menu and choose **Model > Polygons > PolyExtrude**. In the Network view you can see the *box* node feeding into a *polyextrude* node.

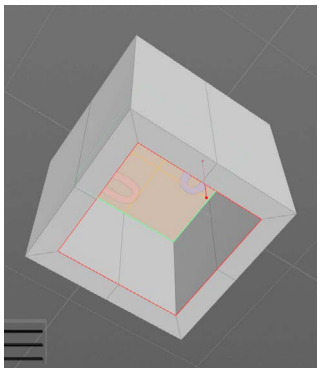
In the Parameter pane, use the slider to set **Inset** to **0.04** which creates new polygons on the top surface of the box. Each node contains parameters relevant to the purpose of that node. These are geometry nodes that are otherwise known as surface operators or SOPs.



04 Next, **press t** to call up the **Move** tool. This adds an *edit* SOP node into the network. In the viewport **RMB-click** in empty space to bring up a menu and select **Make Circle** to round out the selected polygons.

This menu is associated with the *edit* node. Every node has its own interface that you can access as long as a relevant tool is active. In this case, the Move tool gives you access to the handle. In other cases the **Handle** tool would be used to access the interactive handles for a node.



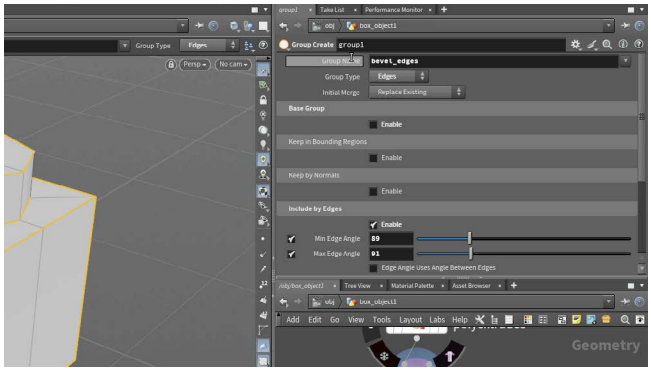


05

Press c to bring up a radial menu and choose **Model > Polygons > PolyExtrude**. In the Scene View pane, use the handle to drag up the polygons and set **Distance** to **0.05.**

Tumble around and **press s** to go into select mode then **shift-select** the bottom four polygons of the box. **Press q** to repeat the **PolyExtrude** tool. In the Parameter pane, use the slider to set **Inset** to **0.025.**

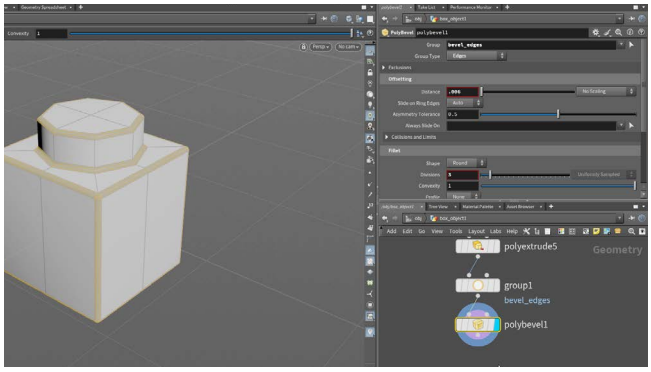
Press q to repeat that tool and set a **Distance** of **-0.175.** When you finish, tumble back to see the top of the brick.



06

Press 3 to go to edge selection and press n to select all the Edges. In the Scene view press tab and start typing **Group...** Select **Group** and in the Parameter pane, set the **Group Name** to *bevel_edges*.

Next, set **Enable** to **OFF** under **Base Group** and then set Enable to ON in the **Include by Edges** section. Turn on **Min Edge Angle** and set it to **89** and then turn on **Max Edge Angle** and set it to **91**.



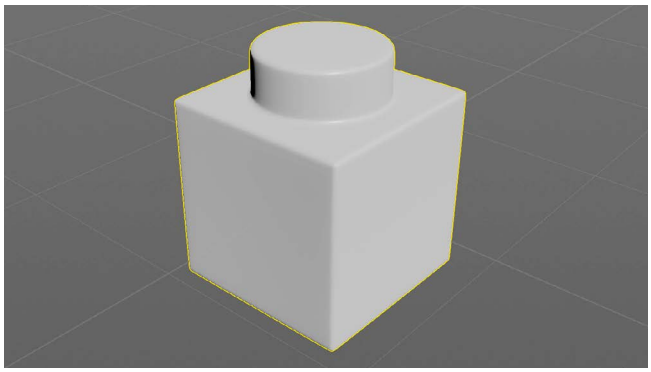
07

Press s to go to the **Select** tool. **Press 9** to turn on the "**Select Groups**" option. In the popup window, click on the *bevel_edges* group.

In the viewport, **press c** to bring up a radial menu. From this menu, choose **Model > Polygons > PolyBevel**. This adds a polybevel node and automatically fills in the **Group** field with *bevel_edges*.

Now set the following:

- **Distance** to **0.006**
- Under **Fillet**, set **Shape** to **Round**
- **Divisions** to **3**.



08

Add a **Null** node to the end of the chain and name it *BRICK_OUT*. This will make it easier to find later on.

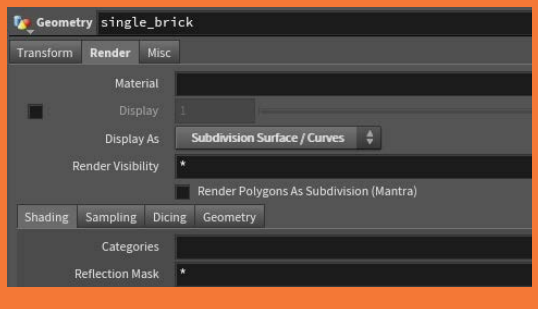
Go to the object level and in the Network view, rename the object to *single_brick*. With the brick selected, press **Shift +** to turn on subdivision surface display for this shape. Deselect the brick to see the model subdivided. If you see wire lines on your object, **press v** and from the radial menu, choose **Shading > Smooth Shading** to hide them.

Save your work.

SUBDIVISION DISPLAY

You can use **Shift +** and **Shift -** to turn subdivision display on and off on selected polygonal objects. This creates a viewport subdivide that lets you see what the shape would look like subdivided. These hotkeys set the **Display As** parameter which can be found at the object level on the object's **Render** tab.

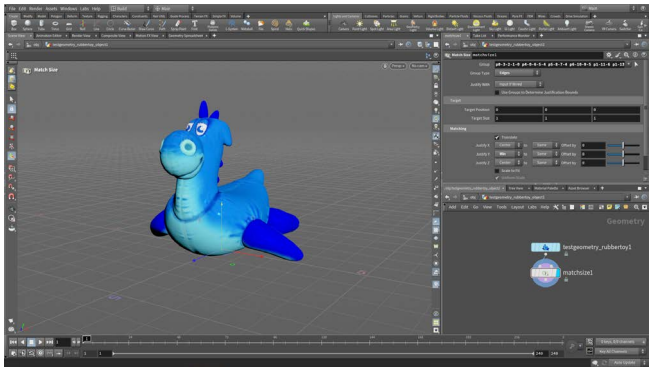
Your object will not render with subdivisions unless you turn on **Render Polygons As Subdivision Surfaces** on the same tab.



PART TWO

Copy Bricks to a Point Cloud

You are now going to use a Rubber Toy model to generate a cloud of points that fit within its boundaries. The brick you created in the last section will then be instanced to the 3D grid to create a brickified version of the model. The instancing is generated by packing the brick geometry before instancing them to the points.

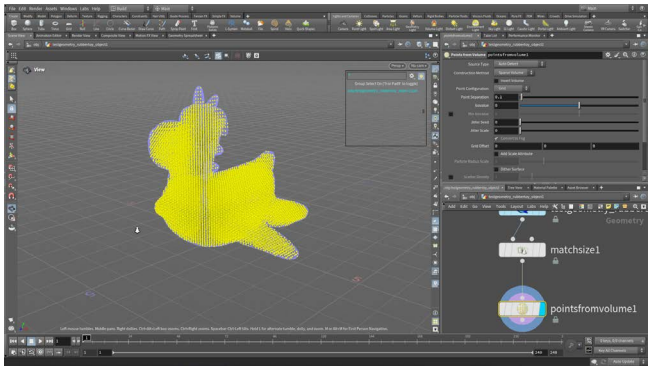


01 In the Scene View, press tab and start typing **Test...** then choose **Test Geometry: Rubber Toy.** Press **Enter** to place it at the origin.

Press i to dive into the *test geometry* object. Set the following:

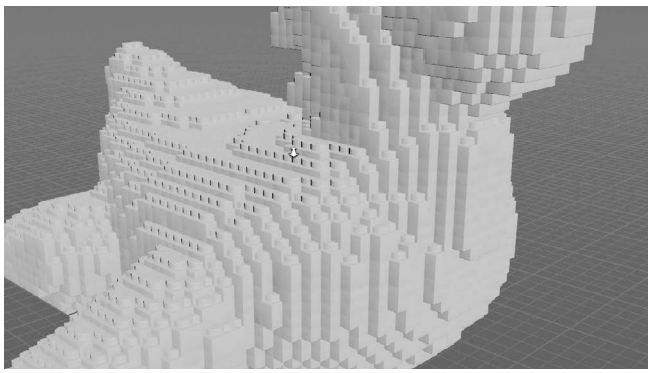
- **Uniform Scale** to **3**

Now add a **Match Size** node and set **Justify Y** to **Min**. This raises the toy so that it sits on the ground.



02 In the Network editor, **RMB-click** on the output of the *matchsize* node and type **Points...** then select **Points from Volumes** and place it's node in the network. Now set its **Display flag** to focus on the output of this new node.

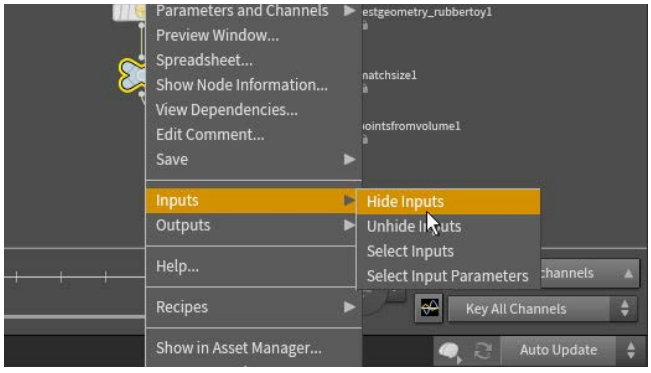
Press s to go to the select tool and **press 2** to get point selection and then **press n** to select all the points which will highlight in yellow. You will copy the bricks to these points.



03 Press **u** to go back to the Object level and shorten the name of the test geometry to just to *rubbertoy*. In the Network view, select the *single_brick*.

From the **Modify** shelf tab, select **Copy to Points** to bring these objects together. You are taken to the geometry level and the nodes are feeding into a *copytopoints* node.

Currently the bricks are overlapping. You will fix this in a couple of steps.

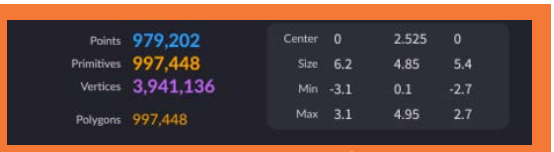


04 In the Network view, **RMB-click** on the *BRICK_OUT* node and choose **Inputs > Hide Inputs**. This will simplify the network and for now you don't need to work with the bricks that make up the single brick.

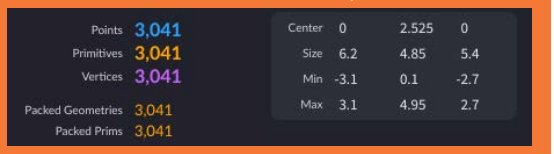
PACKED INSTANCES

If you leave **Pack and Instance** setting **OFF** on the copy to points node then you will end up with a large model with over a million points and primitives. This would make it very slow to manipulate in the Viewport because instancing is not being used.

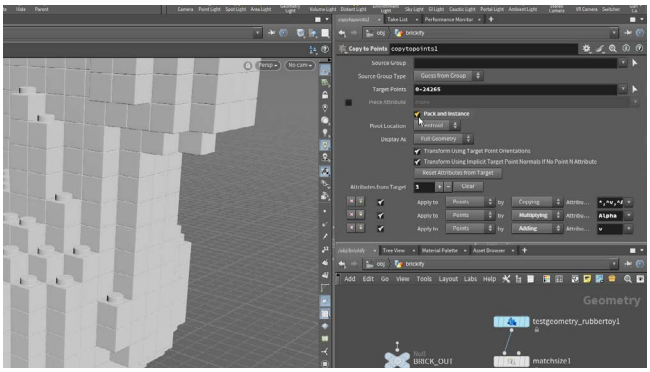
If you turn it **ON** then the **points** on the brick model are packed up and instanced which leaves a much more efficient point count for the copy to points node.



PACK AND INSTANCE | OFF



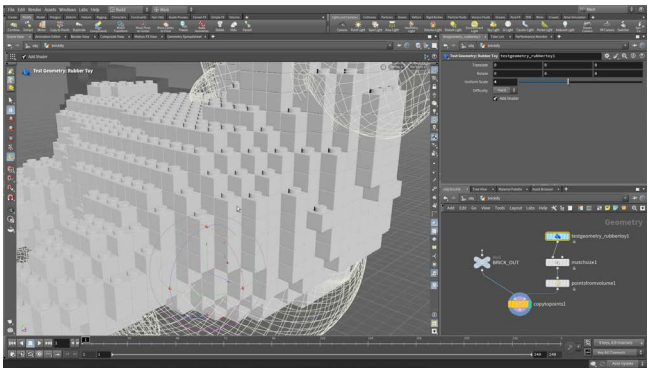
PACK AND INSTANCE | ON



05 On the copytopoints node, turn on the **Pack and Instance** option. This is important because it will display the copied bricks much faster than if this option is off. Remove all the Points from the **Target Points** section. When it is blank then it will find all the points.

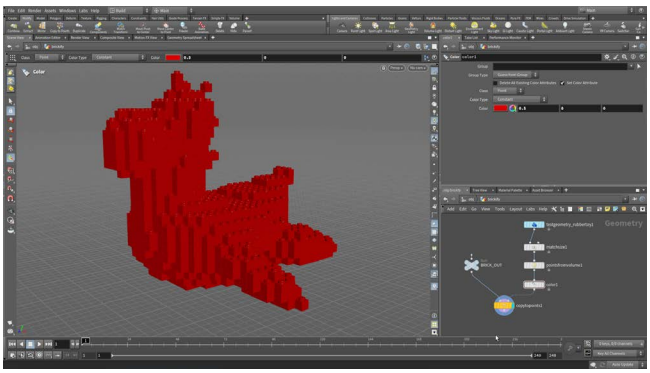
The bricks appear to be overlapping. Go back to the *pointsfromvolume* node and set **Point Separation** to **0.2.**

Now you are copying bricks to the updated points of the grid.

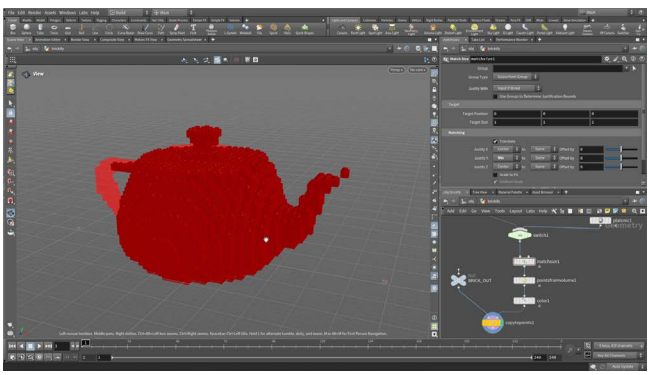


06 Select the *testgeometry_rubbertoy* node and set its uniform scale to 4. You can see that as the toy gets bigger, more bricks are added.

Change the Scale back to 3 to reduce the number of bricks.



07 Add a *color* node between the *pointsfromvolume* node and the *copytopoints* node. This will add color to the points which then get copied to the bricks. Change the **color** to **red** to help the bricks stand out against the background.



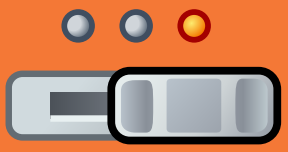
08 In the tool shelf, go to the **Create** menu and drag the **Platonic** tool down to the network view. This places a platonic node at the geometry level. Tools can be dragged into the network view as long as the node type makes sense for your current network level.

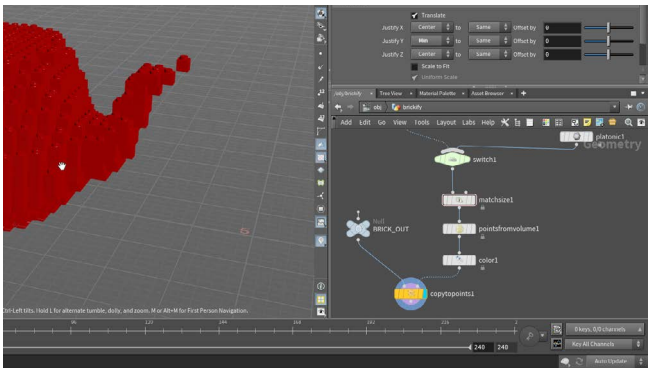
Connect the output of the *platonic* node to the input of the *switch* node. Set the platonic node's **Solid Type** to **Utah Teapot**. Set **Radius** to **4**.

SWITCH NODE

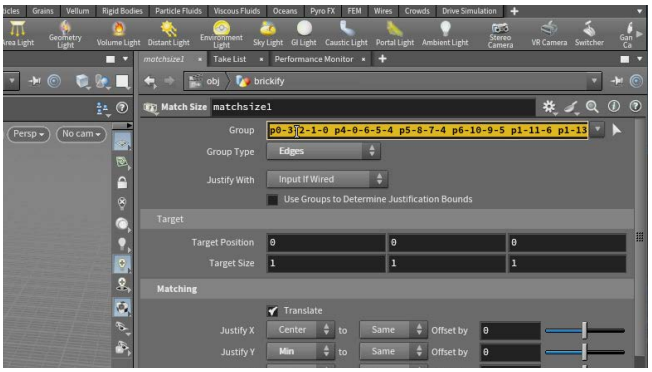
The **Switch** node is a great tool for providing options within a node network. This node lets you quickly explore multiple options without the need to wire and unwire different chains.

This node is also very useful when you wrap up your network into a **Digital Asset** because you can promote the switch to the asset as either a menu or a toggle for quick access to different options.



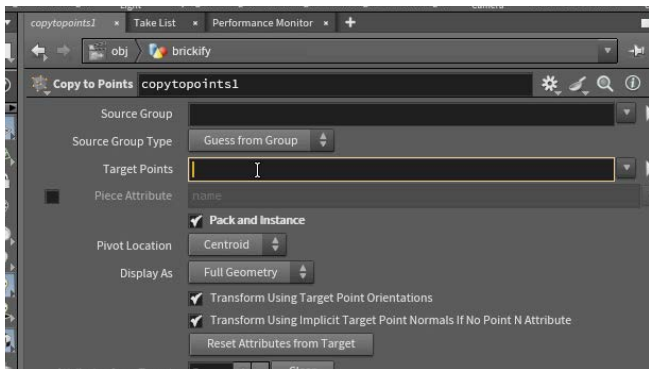


09 Select the *switch* node and change its **Select Input** to **1**. The platonic shape has been cubified using the same setup as the rubber toy. There are a couple issues that may crop up that can be quickly fixed.



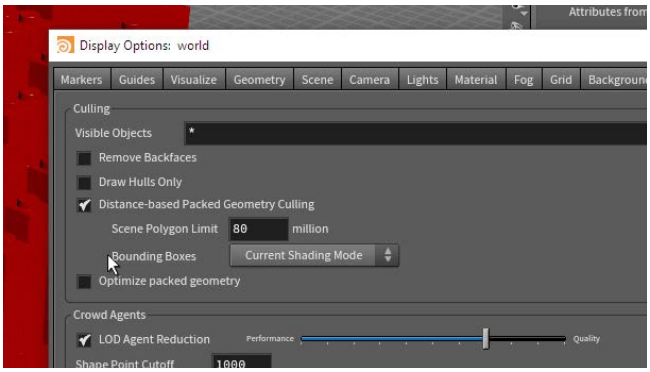
10 You may find that the teapot is not sitting on the ground. Go to mat*chsize* node and delete anything in **Group** field. When there is a selection in this field then only those parts will be selected. Then when you change inputs on the switch, this node doesn't know what to do with the new geometry.

When the field is empty then it will simply accept everything that comes into the node therefore changing inputs is no longer an issue.



11 Another issue is that not all of the points on the teapot have bricks. Select *copytopoints* node and clear the **Target Points** Field. When you used the shelf tool this field was filled up with the point numbers from the rubber toy. By clearing the field you will accept all the points even if the number of points change.

Keeping these group fields clear is important when trying to build a procedural system.

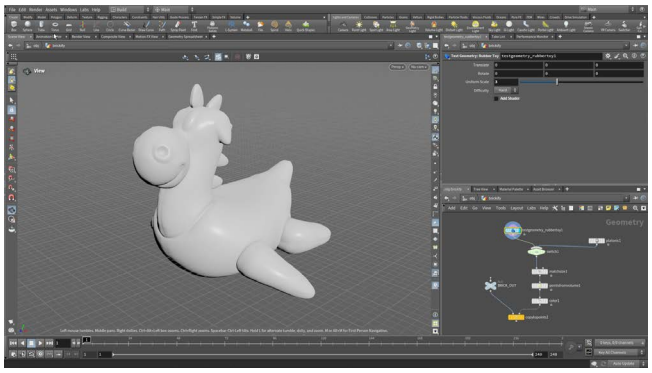


12 There may be some other display issues because there is too much geometry on the screen. In the Scene view, **press d** to bring up the Display Options. Click on the Optimize tab and set **Scene Polygon Limit** to **80** million. Now the whole teapot should have visible bricks.

PART THREE

Color the Points using a Texture

Earlier you added a color attribute to the points which affects the coloring of the brick instances. Instead of using a single color, you are now going to use a texture map to create a more interesting look for the bricks. This will involve some VOP nodes to convert the texture from vertex colors to point colors.

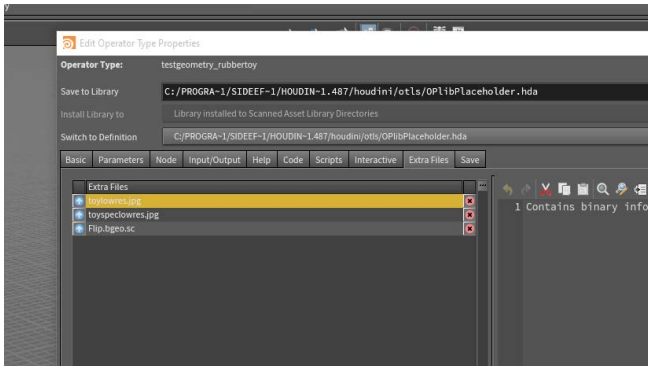


01

Turn on the **Display Flag** on the *testgeometry_rubbertoy*. Select the *rubbertoy* node and in the Parameter pane turn **Off** the **Add Shader** parameter.

To hide the UV display in the perspective view, go to the **Display Options** bar and turn **Off** the **Show UV Texture when UV's Present** button.

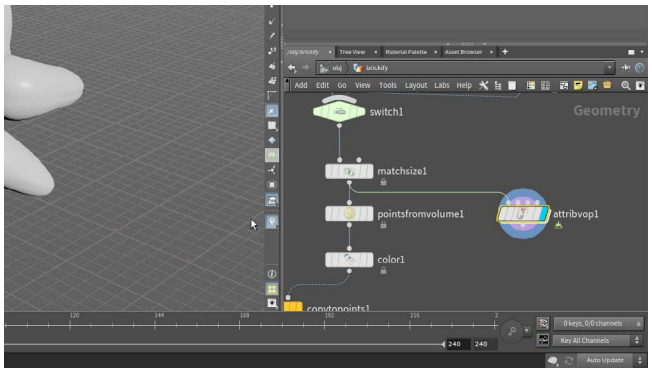
You will bring back the color on the bricks using a different method that involves pulling the color from a Texture Map on disk.



02

From the **Asset** menu, choose **Edit Asset Properties > Rubber Toy**. In the **Properties** window, click on the **Extra Files** tab and select *toylowres.jpg*.

Click the **Save as File** button and save it into the *tex* folder. The texture was stored in the digital asset so that it could be shared along with the asset. You will use the texture on disk to add color to the bricks.

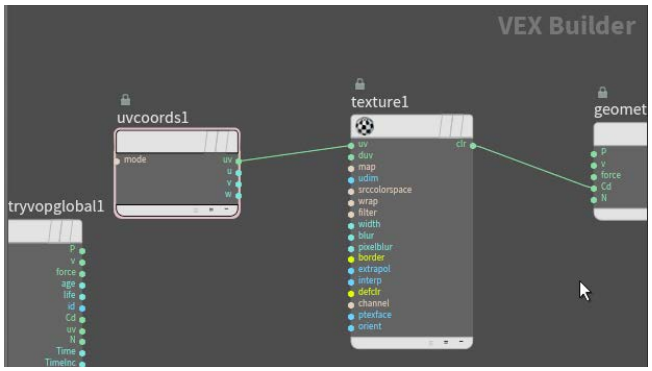


03

In the Network view, press **tab** and start typing **Attribute...** Select the **Attribute VOP** node and place it into the network beside the *pointsfromvolume* node.

Feed the output of the *matchsize* node into the first input of the *attributevop* node. Set the **Display Flag** on this new node.

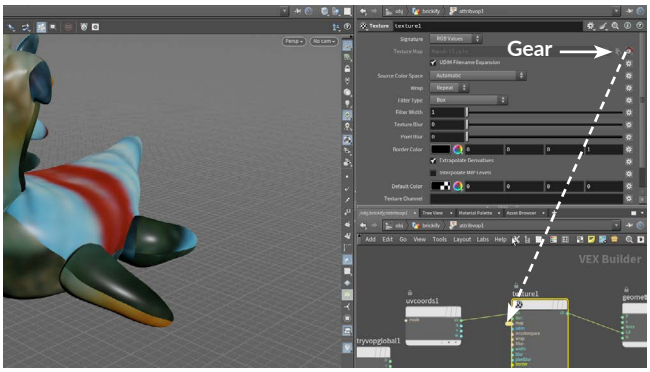
In the Parameter pane, set **Run Over** to **Vertices**.



04

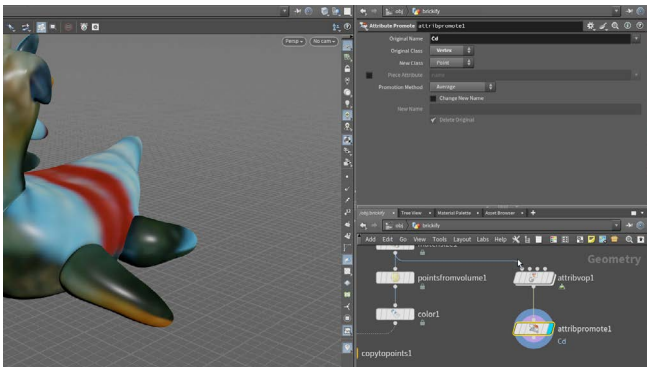
Double-click on the *attributevop* node to dive down and use the **tab** key to add a **Texture VOP**. Feed it into the **Cd** input of the *geometryvopoutput* node.

Add a **UV coordinate** node and feed it into the **UV** input on the *texture* node.



05 Select the *texture* node. Click on the **Gear** icon on the far right of the **Texture Map** parameter and from the menu, select **Promote Parameter**. This will add this parameter to the upper level of this node.

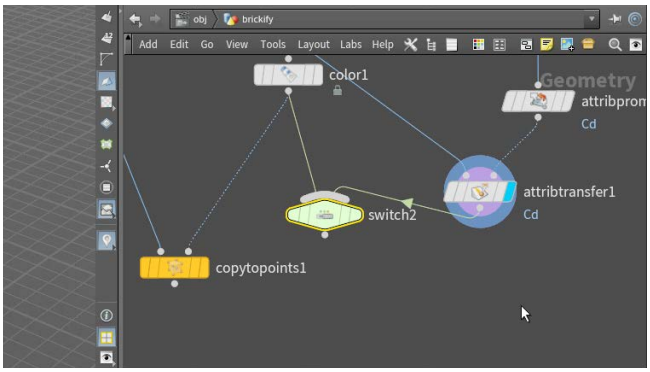
Click on the little knob that appears next to map. In the parameter pane, change the **Label** to **Texture Map.**



06 **Press u** to go up one level. You can see the **Texture Map** parameter. Leave it set to the default **Mandril.pic** texture for now. You will add the rubber toy's texture map at the end.

Add an **Attribute Promote** to the end of this chain. Set **Original Name** to **Cd** and **Original Class** to **Vertex**. Leave **New class** set to **Point**.

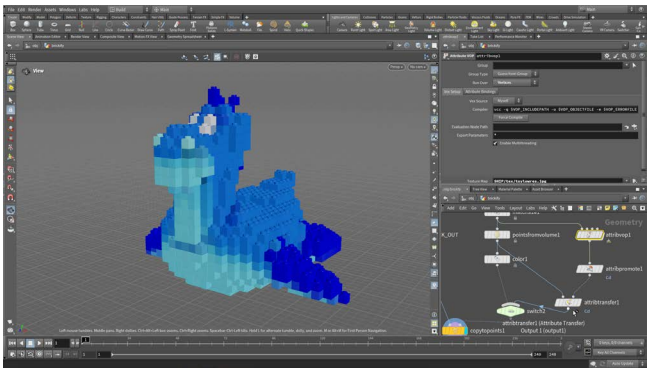
This is taking the information from the vertices and moving it to the points. You need it on the points so that it can be transferred to the bricks.



07 Add an **Attribute Transfer** node. Wire the *pointsfromvolume* into the first input and *attribpromote* into the second. In the **Attributes** tab click on arrow on the right side of the **Points** field and choose **Cd**.

Add a switch node after the *color* node and wire the *attribtransfer* node into the switch. Rename this node *texture_switch*.

Set the **Switch** to **1**. Set the **Display Flag** on the *copytopoints* node. Now you can see the colors from the texture map being transferred to the copied points.



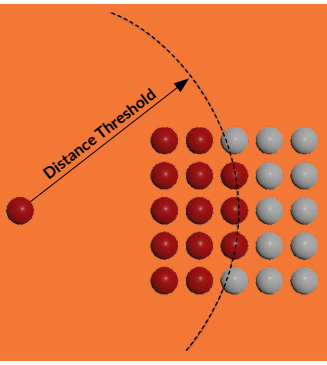
08 Wire the *texture_switch* node into the *copytopoints* node. Select the *attribvop* node and click on the **File** button on the far right of the **Texture Map** parameter and navigate to the *toylowres.jpg* texture and click **Accept**. You can see that the texture map colors are now being assigned to the vertices.

Save your work.

ATTRIBUTE TRANSFER

When you want to get attributes from one piece of geometry to another, you can use Attribute Transfer which uses a **Distance Threshold** along with other parameters to get the attributes copied over.

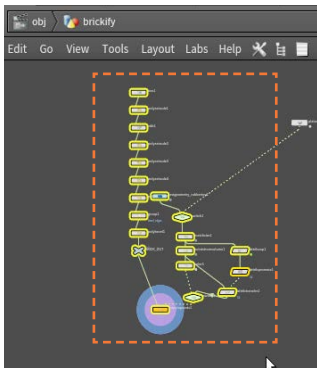
In the case of the rubber toy, you are transferring the **Cd** attribute from the points on the geometry to the point cloud that is used to copy the bricks. You could also copy attributes such as UVs or Capture Weights.

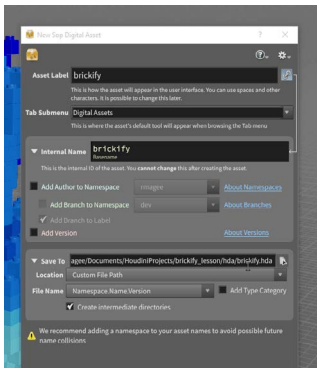


PART FOUR

Create a Brickify Digital Asset

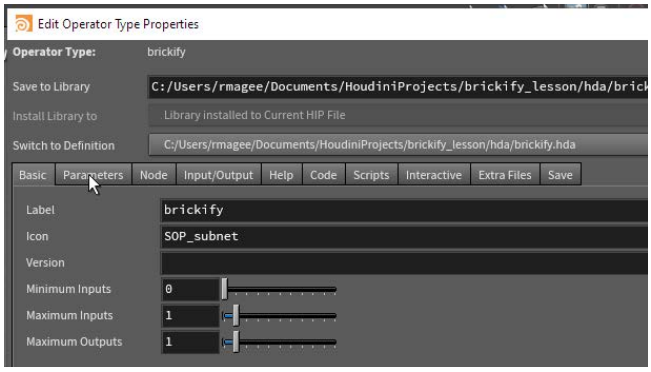
Now that the brickify recipe is working and the nodes are wired together properly, you are going to wrap up the nodes to create a single Houdini Digital Asset [HDA] node. Now you can share parameters from within the network to the top level to create an interface that can generate unique results each time the asset is used.





01 In the Network view, **RMB-click** on the *BRICK_OUT* node and choose **Inputs > Show Inputs**. These nodes need to be visible to add them to the asset. Drag the *platonic* node off to the side then select all the other nodes and from the **Assets** menu, select **New Digital Asset From Selection...**.

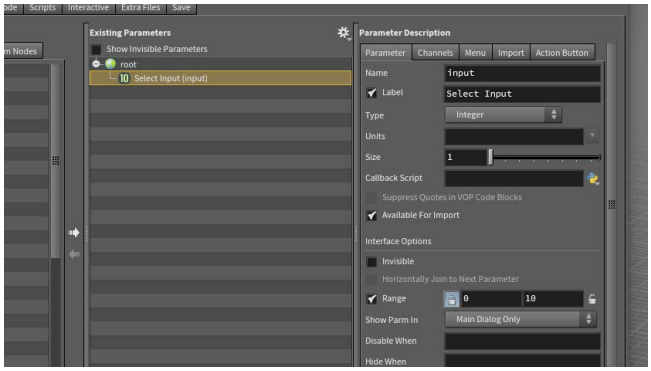
Set the **Asset Label** to *brickify* then turn off **Add Author to Namespace, Add Version** and **Add Type Category**. Click on the button on the far right of **Save to Library**. Select $HIP in the Location sidebar and then double click on the *HDA* directory. Press **Accept** then click **Create** in the **New Sop Digital Asset** dialog.



02 This brings up the **Type Properties** window and make sure the **Basic** tab is visible. Set **Minimum Input** to **0** so that you don't "require" an input for the asset to work. The **Maximum Inputs** parameter is set to 1 which defines how many inputs nodes you will allow.

This is the input that is currently connected to the platonic node. When you use this digital asset later, you will use this input to point it to a different shape.

Press **Apply**. **DON'T** press **Accept** because it will close the window.



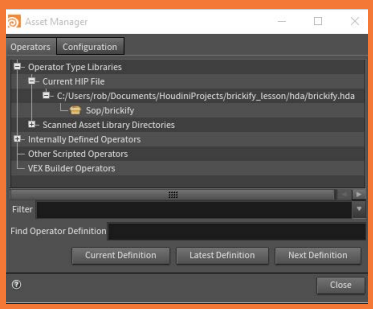
03 In the Network view, **rename** the new asset node *brickify_asset* and **double-click** to dive into it. Click on the *switch* node that is switching between the *testgeometry_rubbertoy* and the *Subnetwork Input* node. The input node is bringing in the platonic teapot shape from the level above.

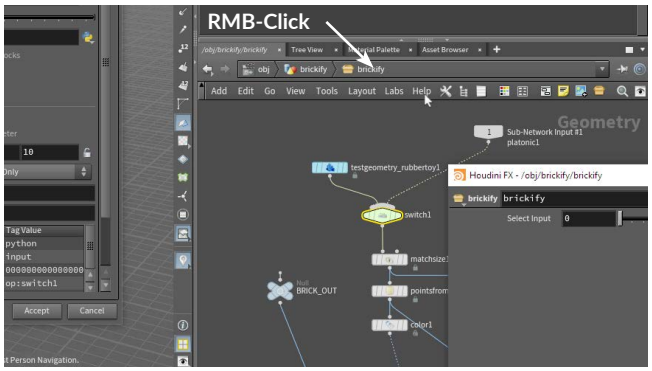
In the **Type Properties** window, click on the **Parameters** tab. Click on the *switch* node's **Select Input** parameter name and **LMB-drag** it to the **Existing Parameters** list in the **Type Properties** window. Drop it on the root to add it to the UI. Click **Apply** which adds the parameter but doesn't close the Type Properties window.

WHAT IS AN HDA FILE?

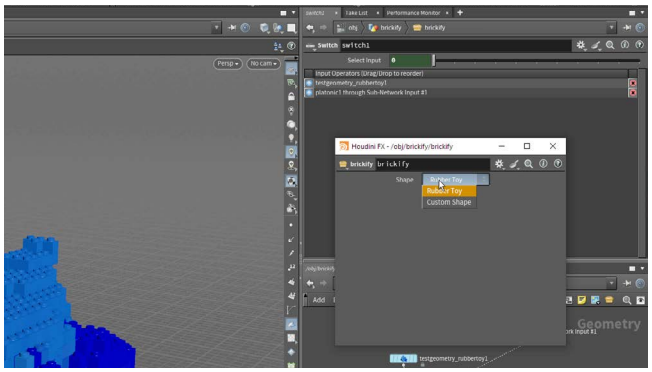
When you saved your asset, it creates a **.hda** file on disk. HDA stands for **Houdini Digital Asset** and the asset definition is stored in this file then referenced into your scene. This file contains information about the nodes, the promoted parameters, UI elements and more. This file can be referenced by multiple people into different shots for a shared experience.

The assets being referenced into your scene can be managed by going to the **Assets** menu, selecting **Asset Manager...** and opening **Current HIP File** .





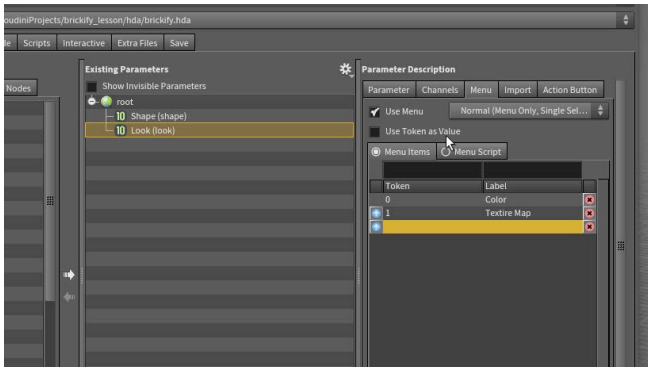
04 **RMB-click** on *brickify_asset* in the Network View's path bar and choose **Parameter and Channels > Parameters**. Now you have a floating Parameter window with the new *brickify* parameter. This parameters will be available to anyone who uses this asset. Change its value from **0 to 1** and back 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.



05 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*.

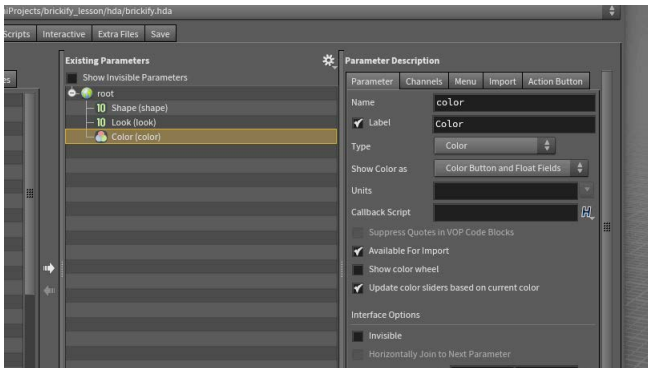
Now click on the **Menu** tab and turn on **Use Menu**. Now under menu items, type **0** under **Token** and *Rubber Toy* under **Label** then press **Enter**. Next type **1** under **Token** and *Custom Shape* under **Label**. Press **Apply**.

Now in the floating parameter pane, you see a **Shape** parameter with a menu. Try it out to see how it works.



06 Select the second *switch* node that sits just under the *color* and *attributetransfer* nodes and promote the **Select Input** parameter to the parameter list. Change its **Name** to *look* and its **Label** to *Look*.

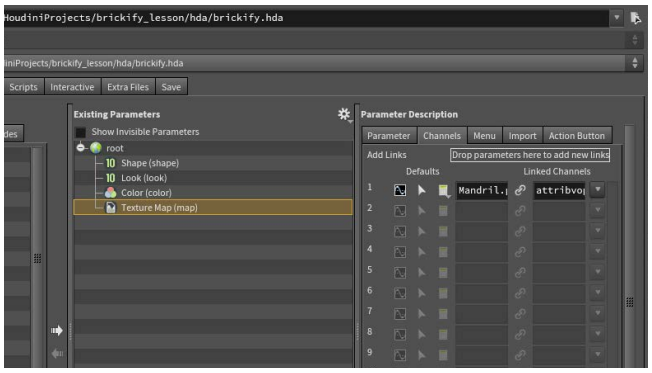
Now click on the **Menu** tab and turn on **Use Menu**. Now under menu items, type **0** under **Token** and *Color* under **Label** then press enter. Next type **1** under **Token** and *Texture Map* under **Label**. Press **Apply**.



07 Now select the *color* node in the network and promote the **Color** parameter to the parameter list. This brings over the color widget and the color wheel as well as the RGB fields.

Press **Apply** in the Type Properties window to see what this parameter looks like in the *brickify_asset* parameter interface.

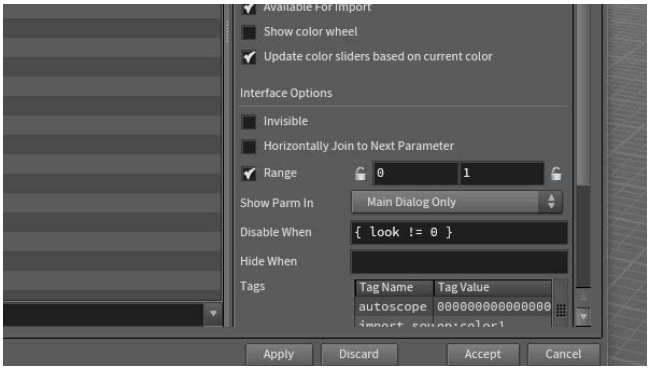
Note: If you press **Accept** by mistake the new parameter is saved to the asset but you will need to use the **Asset** menu and choose Open **Edit Asset Properties... > brickify** to reopen it.



08 Now select the *attributevop* node and promote the **Texture Map** parameter to the parameter list.

In the Parameter description section, click on the **Channels** tab and change the **default** value to **Mandril.pic**. This is a default texture map that doesn't require a path to load and is a more reliable default. Later you will reload the *toylowres.jpg* texture map.

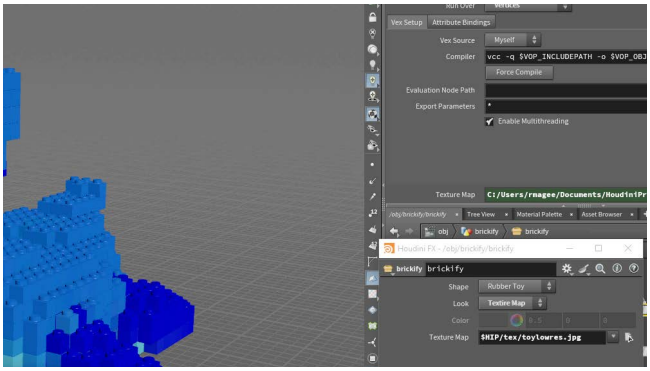
Press **Apply** in the Type Properties window and you will see this parameter in the *brickify_asset* parameter interface and the Mandril texture map is now coloring the bricks.



09 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 **Color** parameter and in the **Disable When** field, enter `{ look != 0 }`.

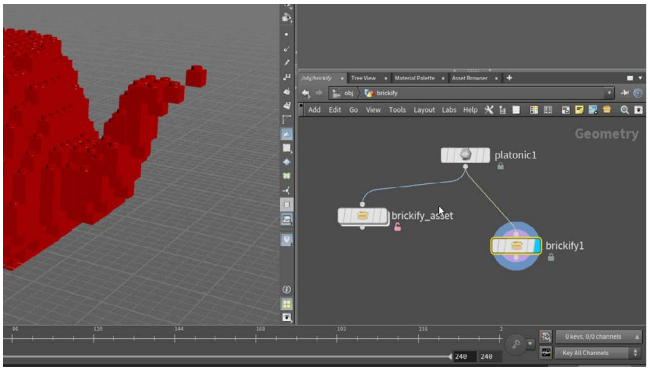
This tells this parameter to disable whenever **Color** has not been chosen in the Look menu. Next, click on **Texture Map** and in the **Disable When** field enter `{ look != 1 }`.

Press **Apply** and test the results using the **Shape** menu. You can see the parameters disabling when they are not needed. You could also hide them with the **Hide When** option but disabling is fine for now.



10 The *brickify_asset* **texture map** parameter now defaults to **Mandril.pic** which is a more versatile default because it is a texture map that will always be available. To go back to the toy map, you would need to click on the file selector next to **Texture Map** and again click on *$HIP* then dive into the *tex* directory and select the *toylowres.jpg* file.

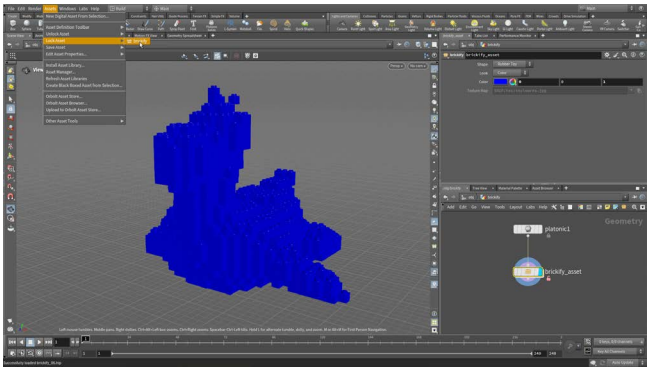
The asset is now using the *Mandril.pic* because it is the default.



11 Now press **Accept** to save the changes to your asset and close the Type Properties panel.

In the Network view, **press u** to go back up one level. With the *brickify_asset* node selected, choose **Assets > Lock Asset > Brickify** from the main menu. Press **Save Changes i**f prompted.

If you **double-click** on the *brickify_asset* node to dive down, you can see that the network is greyed out and these nodes are protected. You can only manipulate this asset using its parameters. You will have to unlock the asset to make changes to its inner workings.



12 Go to the Object level. Rename the object to *rubbertoy*. Now **Save** your scene file to preserve the work you have done so far.

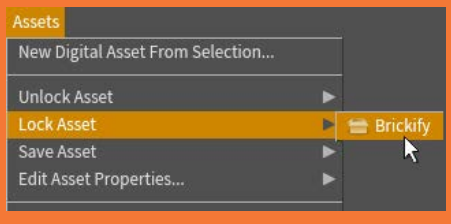
You now have the scene file and the .hda file which is being referenced into your scene to create the asset. You can use this library to create other instances of the asset in this scene or to add the asset to another scene.

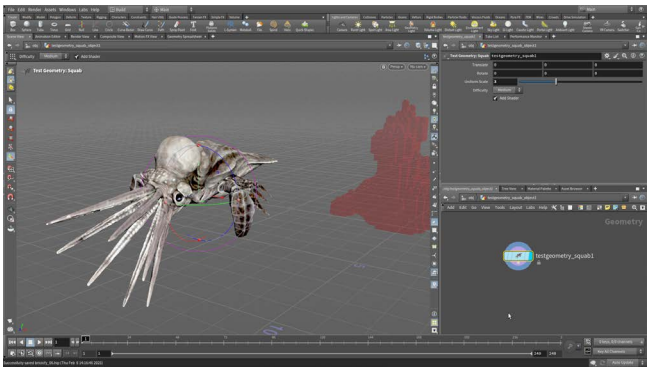
In the next section, you will test out your asset and find out if it is working properly. It is always good to have one working asset and one for testing to make sure it is doing what you want it to do.

LOCKING AND UNLOCKING ASSETS

You can lock and unlock selected assets using the **Assets** menu. When the asset is **locked**, it references the HDA file to determine how the asset behaves. If the asset is **unlocked** the active definition is in your scene file. When you lock it you will be prompted to save if there are changes.

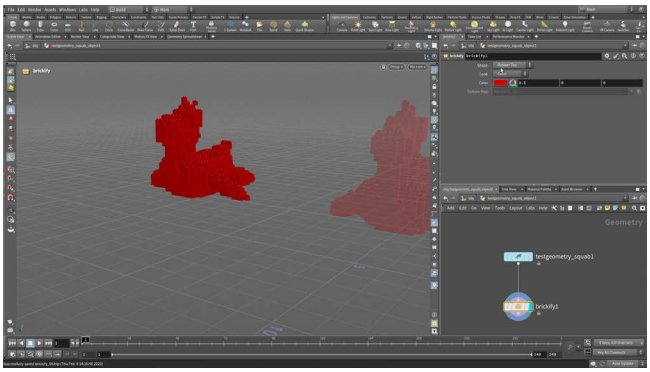
If you **RMB-click** on the asset node you can **Allow Editing of Contents** to unlock the asset or **Match Current Definition** to lock the asset but be careful because there is no prompt to save and changes might be lost.





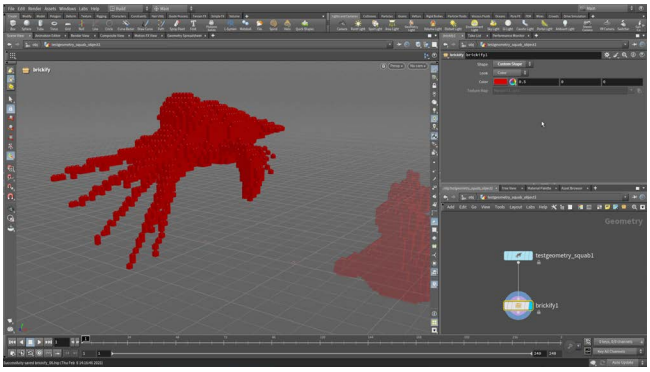
13 Use the **tab** key to get the **Test Geometry: Squab**. Press **Enter** to place it at the origin then use the handle to move it to the side away from the rubber toy. Rename this object to *squab*.

Double-click on the new object node to dive down to the geometry level. Select the node and set **Scale** to **3**. This will make the Squab a bit bigger than the rubber toy.



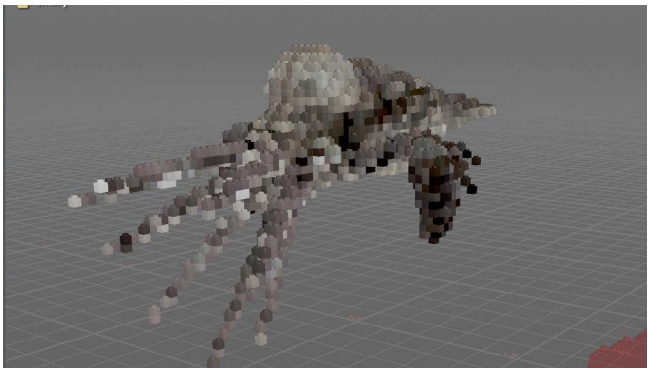
14 **RMB-click** on the output of the test geometry node and start typing **brickify...** then select the **brickify** asset from the menu. This places the asset into this new network.

Set its **display flag** and you will see another Rubber Toy colored **Red**. This is because these are the defaults for this asset.



15 On the *brickify* asset node, set the **Shape** parameter to **Custom Shape** and how the squab is being brickified. The new shape is running through the node network inside the asset to create a unique result. The shape will be lifted above the ground some more because of the **Match Size** node inside the asset.

This is how digital assets become tools that you can put into your pipeline to package up multiple actions into a single node. This is an approach that speeds up your workflow and helps you achieve more consistent results.



16 Select the *testgeometry_squab* node. From the **Asset** menu, choose **Edit Asset Properties > Squab**. In the **Properties** window, click on the **Extra Files** tab and select *squab_diffuse.jpg*. Click the **Save as File** button and save it into the *tex* folder. The texture was stored in the digital asset so that it could be shared along with the asset.

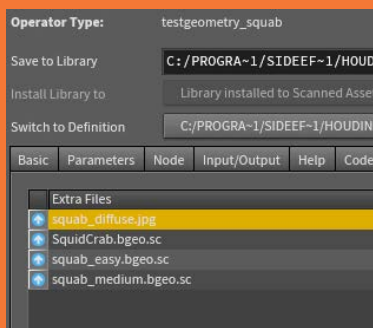
Now use this texture on disk to add color to the bricks using the *brickify* node's **Texture Map** parameter.

When you are finished, go to the object level and **name** this object *squab* and the other one *rubbertoy*. **Save** your work.

EXTRA FILES

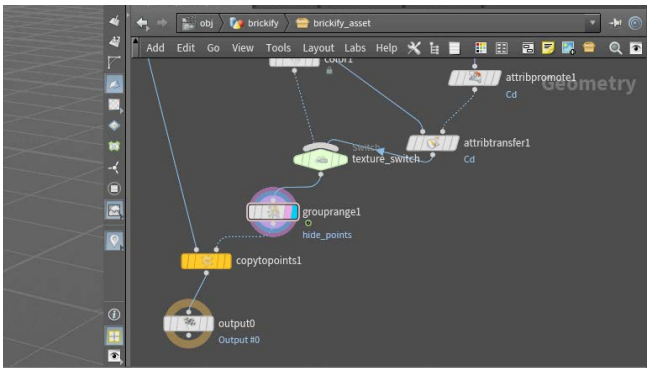
In order to make digital assets shareable, you sometimes need to include outside files into the asset definition. This can be done on the extra files tab. In this lesson you grab texture files from the rubber toy and squab to use in the brickify tool.

You can see that those assets have embedded geometry and texture files that stay with the hda file as it gets distributed. This avoids the need for maintaining references external files.



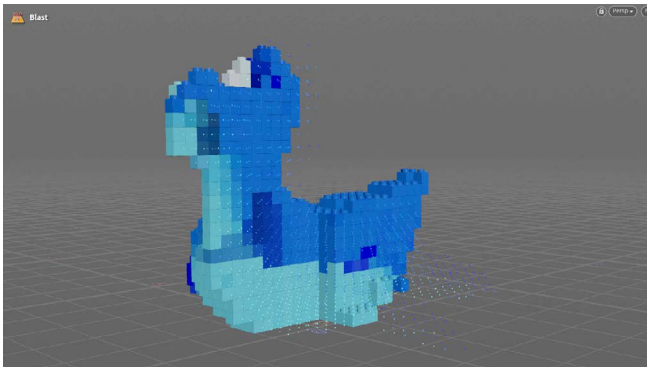
PART FIVE
Animate the Bricks

To continue adding features to the asset, you will create an automated animation of the bricks building up from the ground. This will involve adding more nodes to the network to make sure the asset has this new functionality. Once the new nodes are saved to the digital asset file and parameters have been promoted, the features will be available for use by anyone using this asset in their work.



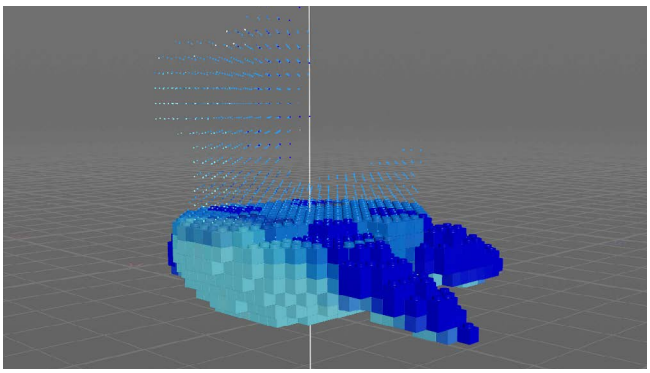
01 Hide the *Squab* object and dive into the *rubbertoy* object. Select *brickify_asset* and choose **Assets > Unlock Asset > Brickify**. **Double-click** on *brickify_asset* node then **RMB-click** on the output of *texture_switch* and select **Group by Range**. Place the node and set its **Display flag** then set the following parameters:

- **Group Name** to *hide_points*
- **Group Type** to **Points**
- **Range Type** to **Start and Length**
- **Length** to *($F-1)*20*
- Under **Range Filter**, leave **Select** to **1** and **Of** to **1**



02 **RMB-click** on the output of the *grouprange* 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. Set the **Display flag** on the *blast* node.

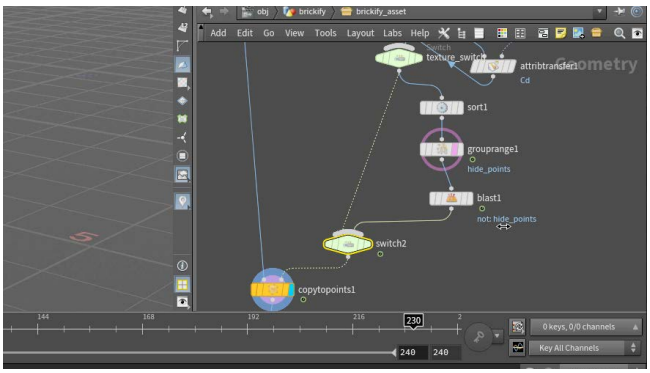
Press play to watch as the points grow with every frame. Now set the **Display flag** back to the *copytopoints* node at the end of the chain and watch the bricks grow over time.



03 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.

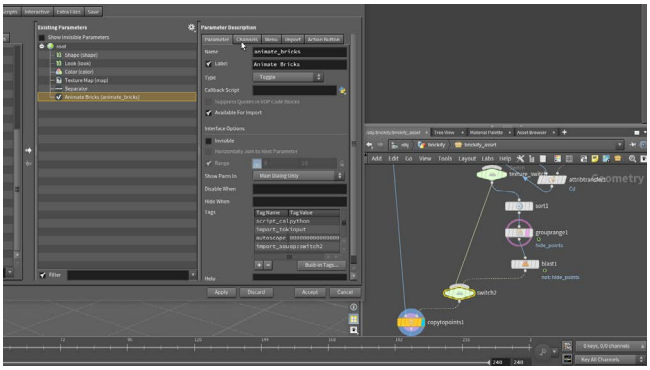
RMB-click on the output of the *texture_switch* node and type **Sort** then select the **Sort tool.** Place this node down and change **Point Sort** to **Along Vector**. With this set to **0, 1, 0**, the points start at the bottom and go up.

Playback to see this result. Test out different vectors to see how it affects the animation.



04 To let you choose whether you want to animate the brickify effect, you can add another switch node. In the Network view, **Press tab** and start typing **Switch...** Choose **Switch** and place the node down. Rename it *animation_switch*.

Click on the output of the *texture_switch* node and feed it into the input of the *switch* node. Repeat for the *blast* node. This makes the original shape the first option and the animated effect the second option. Changing **Select Input** to **1** will reveal the animated bricks but for now keep it at **0**.



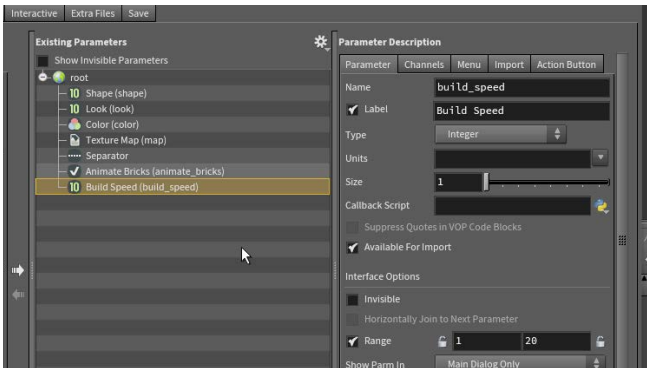
05

From the **Assets** menu, select **Edit Asset Properties > Brickify.** Go to the **Parameters** tab and drag a **separator** from the **Create Parameters** section to the bottom of the list.

Next, drag the *animation_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.

In the **Parameter Description** section, click on the **Channels** tab and set the default value to **0 [off]**.

Click **Apply** to save changes.



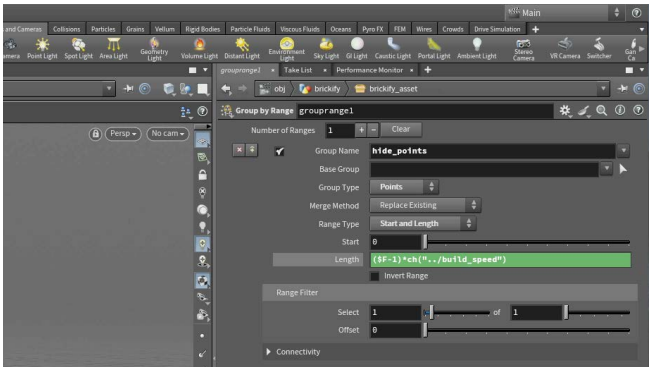
06

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 **20**. Click on the **lock** next to 1 to make sure the number never gets smaller than 1.

In the **Parameter Description** section, click on the **Channels** tab and set the default value to **10**.

Press **Accept** to save and close the window.

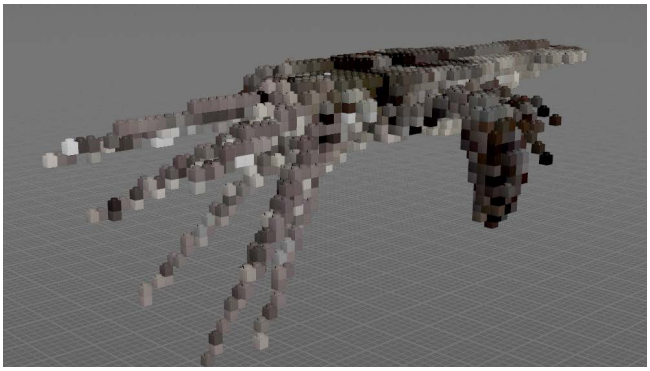


07

This parameter isn't attached to anything yet but you will now use it to drive the *grouprange* node's **Length** expression. Select the new *grouprange* node and change the **Length** expression to: `($F-1)*ch("../build_speed")`

At the end of the network there is an **output** node. This will ensure that you get the right output for your asset even if the **Display flag** is on another node in the network.

With the *brickify_asset* node selected, choose **Assets > Save Asset > Brickify** from the main menu. This lets you save this expression to the .hda file without re-opening the Type Properties window.



08

With the *brickify_asset* node selected, choose **Assets > Lock Asset > Brickify** from the main menu. You now have the brickify effect wrapped up into a custom tool which can be used on different shots by different artists.

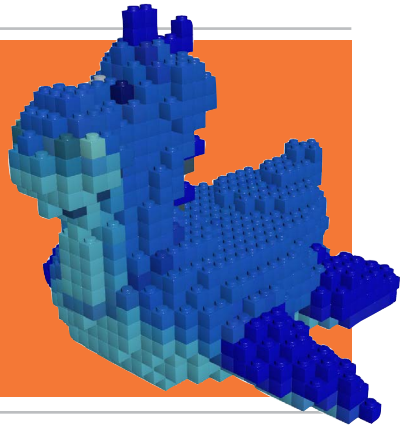
Go to the Squab network where the new features are available on the *brickify* node. Turn on the **Animate Bricks** toggle and set the **Build Speed** which might need to be set to around **30** because of the large number of bricks in this shape.

Playback to see the results.

SHARABLE TOOLS

You have now created a shareable tool which was built without writing scripts using a node-based workflow which is easily accessible to artists. By saving the HDA to disk, it becomes an asset on disk that can be referencing into multiple shots.

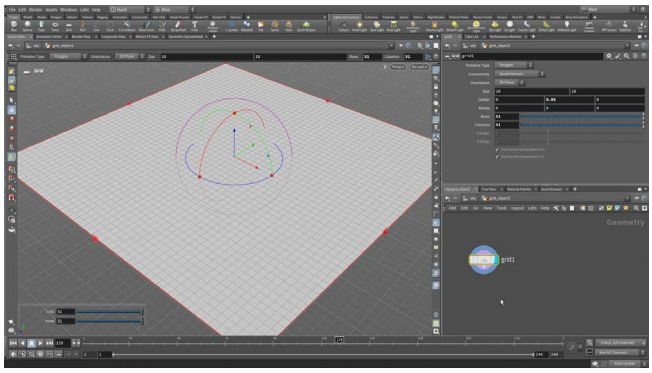
Houdini Digital Assets offer a powerful way for artists to share these kinds of tools in support of a studio-level production. These procedural assets make it easy to automate repetitive tasks and to stay focused on the creative needs of your project.



PART SIX

Create a Peg Board

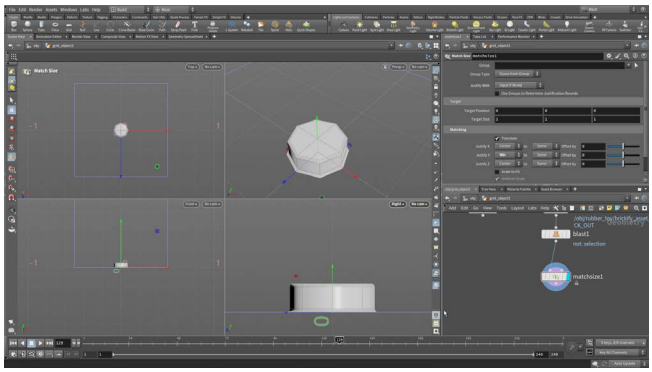
To prepare for rendering, create a peg board for the brickified shapes to sit on. You will borrow some of the geometry from the single brick and use that to copy to the points of a grid. These will be combined with a box that you bevel to create rounded edges.



01 Go to the Object level and hide the *squab* and *rubbertoy* objects. Create a new **Grid** and place it at the origin. Double click to go to the geometry level and set:

- **Rows and Columns** to **51, 51**
- **Center Y** to **0.05**

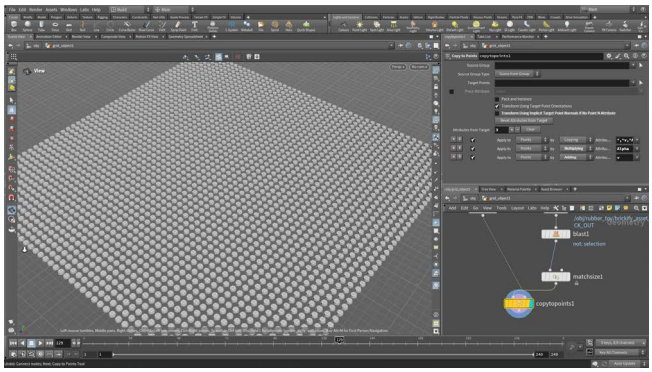
You are now going to use this to copy the pegs for a peg board base. You can get the peg from the brick you modelled earlier.



02 Press **tab > Object Merge**, place the node down and set its Display flag. Click on the **Node selector** button next to **Object 1** and navigate into the *rubbertoy/brickify* and select *BRICK_OUT*. This will import the single brick into this network.

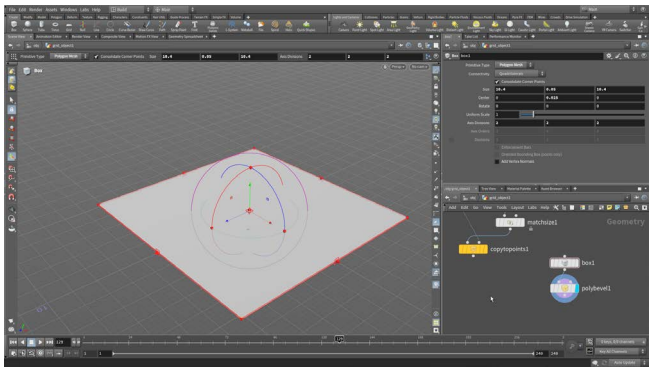
Get the **Select** tool and **press 4** to get primitive selection. Go to a **Right** view and box select the peg geometry including the bevel that connects the peg to the brick. Press **Delete** which creates a blast node. Turn on **Delete Non Selected**.

Now add a Match Size node set **Justify Y** to **Min to Same**. This places the peg on the ground.



03 Press **tab > Copy to Points**, place the node down and set its Display flag. Wire the *matchsize* node into the first input of the *copytopoints* node and the *grid* into the second input.

In the Parameter pane, turn **ON** the **Pack and Instance** option and Turn **OFF** the **Transform Using Implicit Target Point Normals...** option. Now all the pegs are in place.



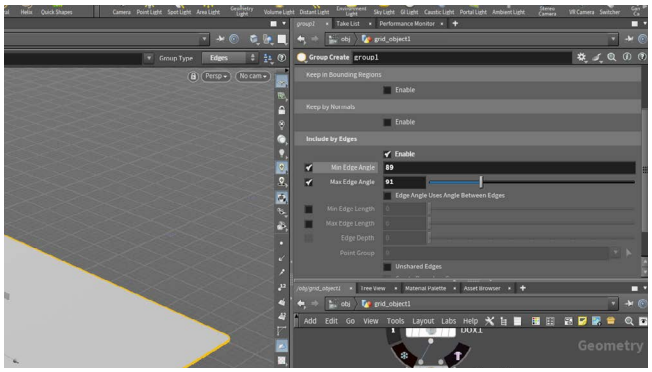
04 Press **tab > Box**, place the node down and set its Display flag. Set **Axis Divisions** to **2, 2, 2.**

Press 3 to get Edge selection and select the four vertical edges of the box. Press **tab > PolyBevel** and set:

- **Distance** to **0.2**
- **Divisions** to **7**

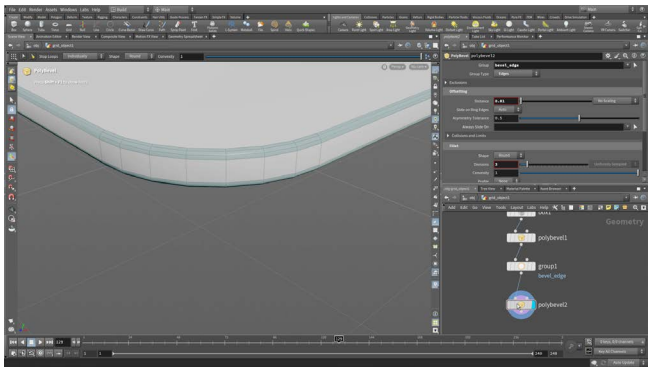
It was easier to select the edges before making the board thin. Go back to the box node and set the following:

- **Size** to **10.4, 0.05, 10.4**
- **Center Y** to **0.025**



05 **RMB-click** on the *polybevel* output and choose **Group**, then place the node down and set its Display flag. Set:

- **Group Name** to *bevel_edge*
- **Group Type** to **Edge**
- Under **Base Group** set **Enable** to **OFF**.
- Under **Include by Edges**, turn on **Min Edge Angle** and set it to **89** and **Max Edge Angle** and set it to **91**.



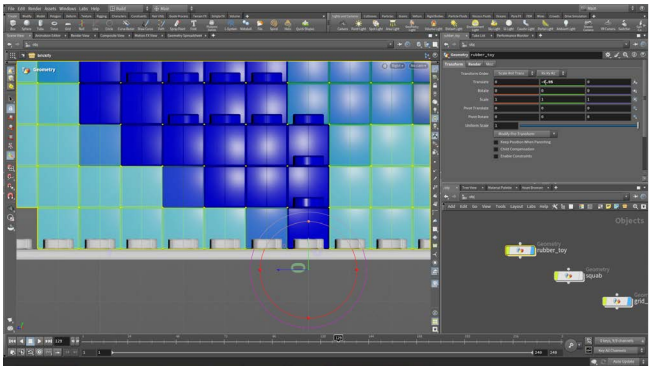
06 **RMB-click** on the *group* output and choose **Polybevel** then place the node down and set its Display flag.

Set the following:

- **Group** to *bevel_edge*
- **Distance** to **0.01**
- **Divisions** to **3**.

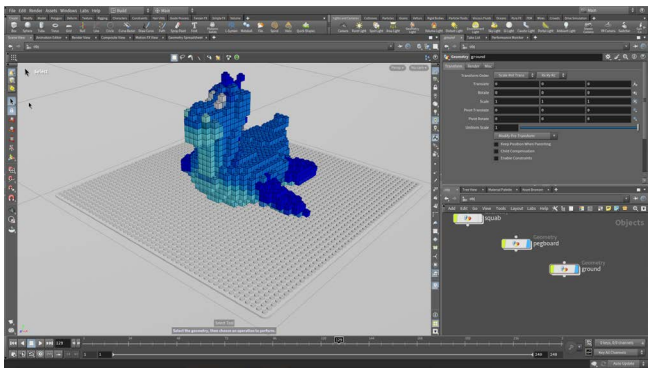
Now only the edges from the group will be bevelled.

Add a **Merge** node to the network and feed both the *copytopoints* node and the new *polybevel* node into it. Add a **Null** node after the *merge* and name it *PEGBOARD_OUT*. Set its Display flag.



07 Go to the Object level and turn on the Display flag on the *rubbertoy* object**. Press w** to go into wireframe mode and check the top view to see that the pegs are aligned.

In the Front view you can see that the *rubbertoy* bricks are floating. Set **Translate Y** on this object to **-0.05** and this will drop the toy down to the board.



08 Press **tab > Grid** and press **Enter** to place it at the origin. Rename it *ground*. Make sure you have the **Handle** tool active then in the Operation Control bar, set **Size** to **100, 100**.

This will create a surface for the area beyond the peg board. You will bring all three of these items into the Solaris lighting context for rendering.

WHAT'S NEXT: SOLARIS

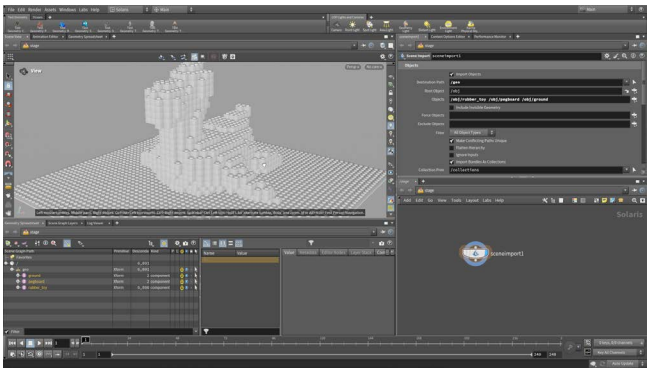
You have used object nodes and dived into them to create geometry using Surface Operators or SOPS. You are now going to move your work to Solaris which is the Lighting Operator or LOP context in Houdini. The Solaris environment can be found in the `/stage` network.

Here you will place nodes for bringing in geometry, assigning materials and adding lights and cameras. The Solaris environment converts everything into USD [**Universal Scene Description**] which is an open source format created by PIXAR. The Solaris/LOPS context allows you to work with USD which is needed to render to the Karma XPU renderer.

PART SEVEN

Render the Final Shot

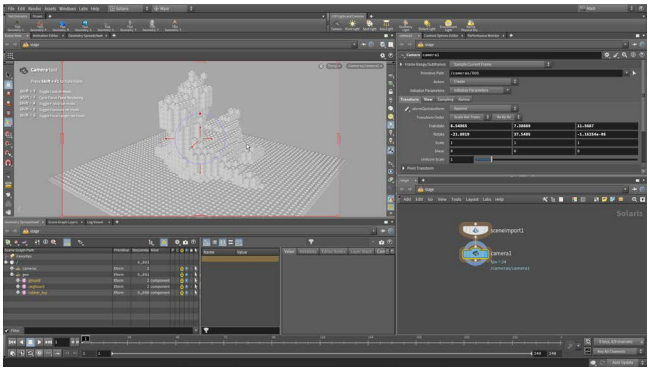
To create the final shot, you are going to bring the geometry into the Solaris or LOPS context of Houdini. LOPS are used to prepare the scene for rendering. You will set up materials that make use of the point colors on the bricks and then set up lights and a camera for the final rendering. You will render a still image then an animated sequence using the brickify tool's animate feature.



01 Go to the Object level and turn on the Display of the rubbertoy object. Dive in and on the *brickify* asset make sure that **Animate Bricks** is **OFF.**

Change the desktop to **Solaris**. Make sure you are looking at the **Stage** in the path bar. In the Network view, press **tab > Scene Import All** and click to place the node down. Name this node *brickify_scene*. Set the **Destination Path** to */geo*.

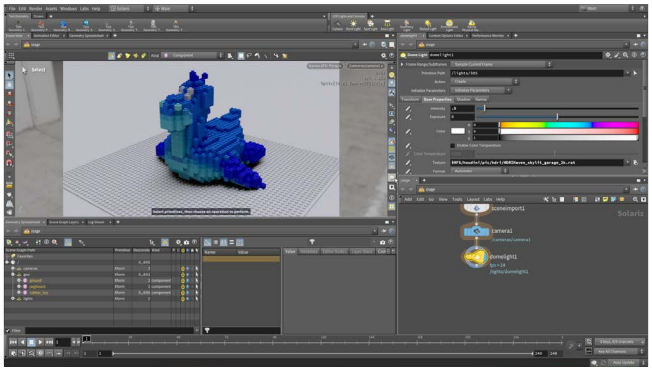
In the **Scene Graph** you can see */geo*. If you expand it then you will see the three objects that you imported.



02 In the Scene View, use your view tools such as **spacebar-f** and **spacebar-h** for homing the view to get a better look at the scene.

In the **LOP Lights and Cameras** shelf, click on **Camera.** In the Scene View, you will see camera handles at the origin. Zoom out and look down at the scene then adjust the handles so the camera is looking at the rubber toy from the left.

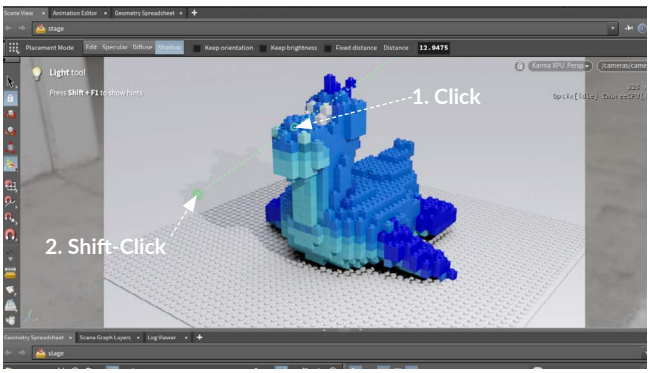
Click on the **Lock camera to view** button. Use the **View tools [Spacebar-LMB/MMB/RMB]** to reposition the camera. When you finish, toggle off the **Lock Camera** button.



03 From the **LOP Lights and Camera** shelf, **click** on the **Environment Light** tool. Click on the **File selector** next to **Texture** and in the locations section, choose *$HFS/houdini/pic/hdri/* then choose the *skylit_garage_2k.rat* file. Set **Intensity** to **0.5**.

Now click on the menu just to the left of the camera menu, and set it to **Karma XPU**. Now you are using the Karma renderer in the viewport. The rubbertoy renders in color because the point colors are affecting the look of the bricks.

If you have an Nvidia graphics card with the latest drivers installed, turn on the Optix Denoiser in the **Display Options** bar.



04 From the **LOP Lights and Camera** shelf, **click** on the **Point Light** tool then press **Enter** to place it at the origin. You are looking through the light. Set the view back to *camera1*.

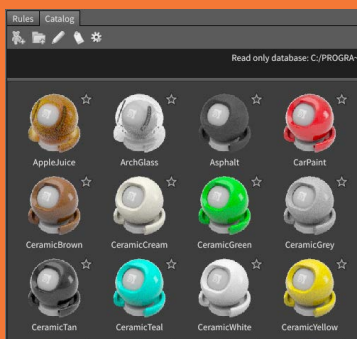
With the node active, press **Shift-F** to turn on the **Shadow** mode. You can also click on it in the **Operation Control** bar. Now **click** on the rubber toy's head to set a pivot then **Shift-click** to place a target on the ground.

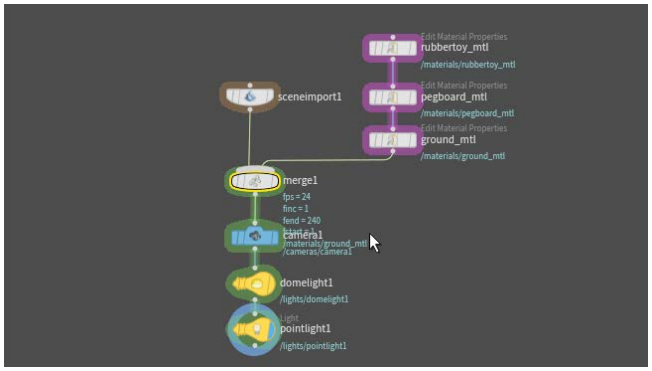
Set the **Intensity** to something like **500** . If that is too strong then try a lower number. If it is not strong enough then try a higher number until you get the look you want.

MATERIALS IN SOLARIS

The **Quick Surface Material LOP** gives you access to a shader that has been pre-built using **Material X** then stored in the **USD** format. A **Quick Surface Material** is actually an **Edit Material Properties** node that loads this USD file. You can then rename it and edit its properties for use in your scene.

You can also build materials using a **Material Library** node and a **Karma Material builder** but the **Quick Surface Material** will meet your needs in many situations. You can also access a **Catalog** of materials on the **Material Linker** node.

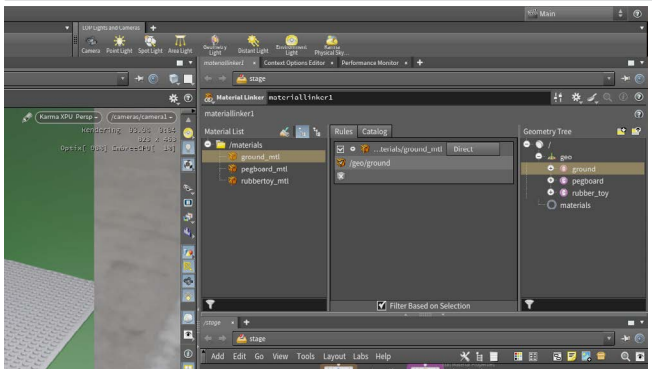




05 Press **tab > Quick Surface Material** and place the node to the right of the other nodes. Call it *rubbertoy_mat*. **Alt drag** that node to create two more. Name them *pegboard_mat* and *ground_mat*. Set the **Base > Color** on the *pegboard_mat* to grey and on the *ground_mat* to dark green.

Add a **Merge** node between the *sceneimport* and *camera* nodes. Wire the three **quick material** nodes into the merge node.

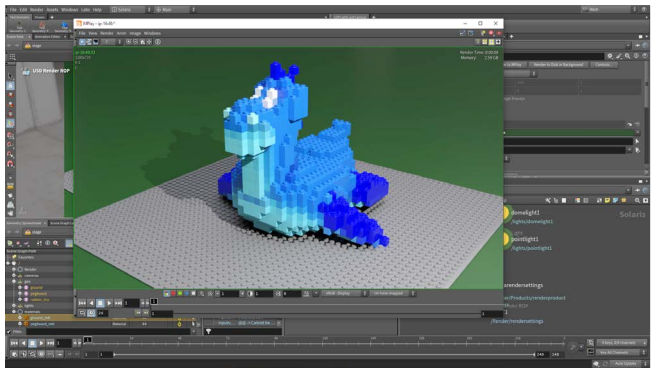
Nothing changes in the Scene view because these materials haven't been assigned yet. For this you need the **Material Linker** node.



06 Press **tab > Material Linker** and place the node in between the *merge* and the *camera* nodes. Keep the **Display Flag** set on the light at the end of the chain.

In the Parameter pane, Drag the *rubbertoy_mat* from the **Material List** to the **Rules** section. Open up the geo folder under **Geometry Tree** and drag *rubbertoy* to the shader part of the rule.

Repeat these steps for the *pegboard_mat* and the *ground_mat*. Your XPU preview rendering will update with the new materials.

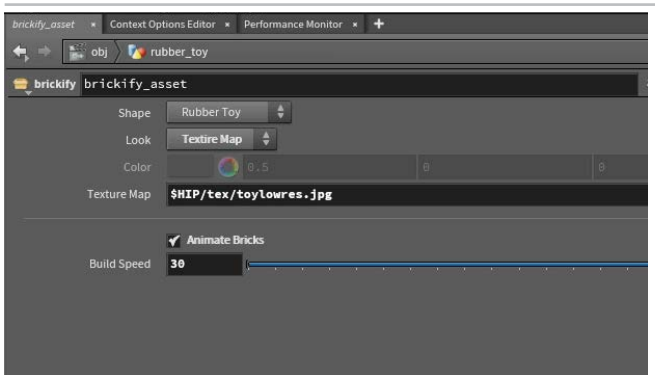


07 In the **Network View**, press **tab > Karma** to add a **Karma Render Settings** and **USD Render ROP** node.

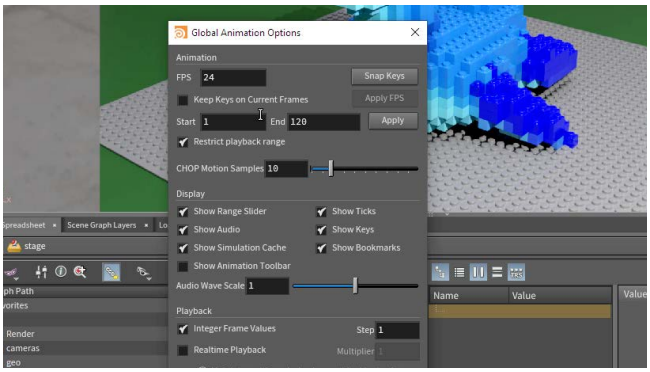
Wire them into the end of the chain. Select the *karmarendersettings* node and on the **Image Output > Filters** tab set **Denoiser** to **nvidia Optix Denoiser** to turn the denoiser back on.

Rename the *usdrender_rop* to *SHOT_1*.

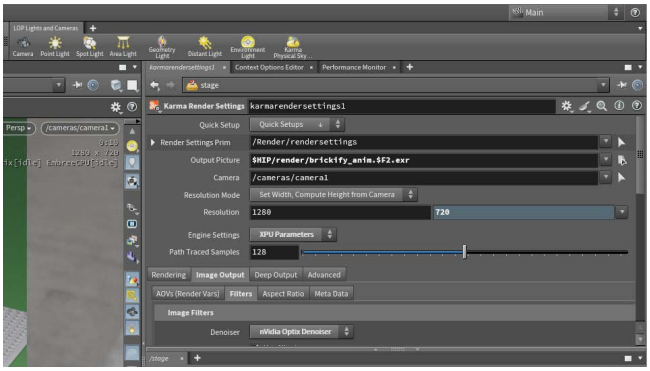
On the *SHOT_1* node, click **Render to Mplay**. This will bring up an Mplay window with your rendering.



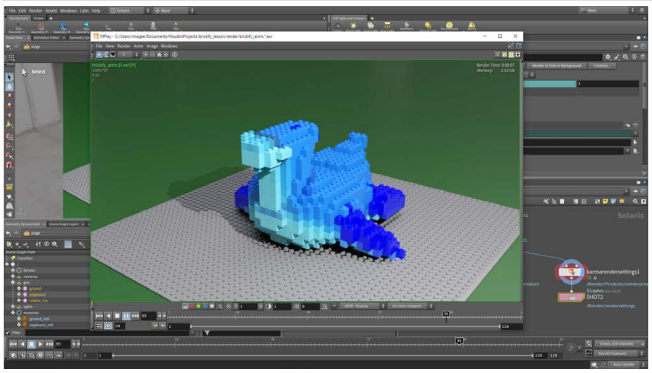
08 Go to the Object level then dive into the *rubbertoy* object. On the *brickify* asset set **Animate Bricks** to **ON** and set **Build Speed** to **30**.



09 At the bottom left edge of the **Timeline**, click on the **Global Animation Options** button. Set the *End* to **120** and click **Close**. This will set the timeline range to 120 frames.



10 Go back to */stage*. On the *SHOT_1* node, set **Valid Frame Range** to **Frame Range** and set the **Output Picture** to *$HIP/render/anim/brickify_anim_$F2.exr*. The *$F2* adds frame numbers to the renderings with a padding of two and the */anim/* creates a directory to hold these frames.



11 On the *usdrender_rop* node, click **Render to Disk**.

When you finish, choose **Render > Mplay > Load Disk Files** and open up the rendered images to review the final sequence.

Save your work.

CONCLUSION

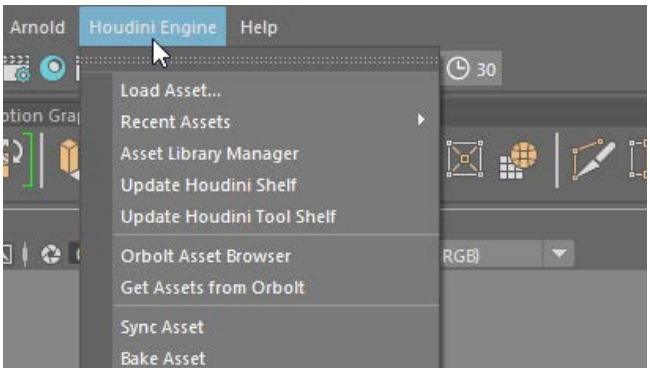
You have now created a shareable tool which was built without writing scripts using a node-based workflow which is easily accessible to artists. By saving the HDA to disk, it becomes an asset on disk that can be referencing into multiple shots.

Houdini Digital Assets offer a powerful way for artists to share these kinds of tools in support of a studio-level production. These procedural assets make it easy to automate repetitive tasks and to stay focused on the creative needs of your project.

BONUS

Loading HDAs into other Applications

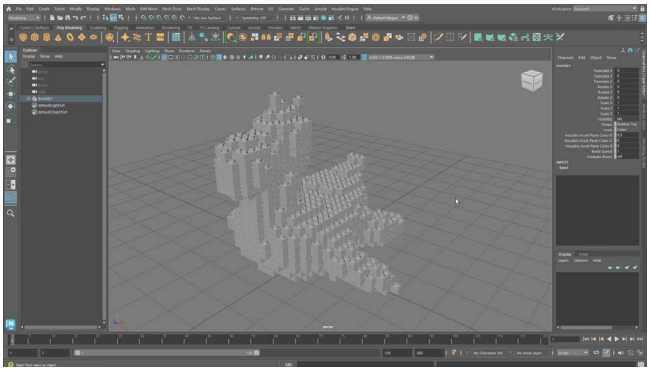
Once you have a Houdini Digital Asset [HDA] saved on disk, it is possible to load that asset into a host application using the Houdini Engine plug-ins. These let you share assets with colleagues who can load them directly into 3D apps such as Autodesk Maya or 3DS Max or into game editors such as Unity or Unreal Engine.



01 To use the Houdini Engine in a host application, you need to first install the plug-ins using the Houdini installer or Launcher. This will make the plug-ins available but you may need to take further steps to make the Houdini Engine available within your session.

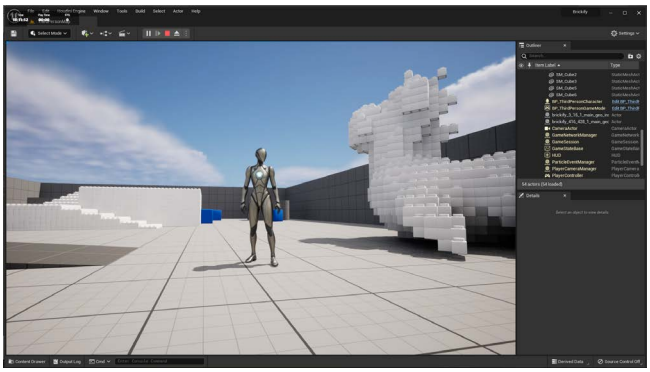
Visit the following page for details: **SideFX.com/engine**

Click on the Engine Plug-ins tab and then click on the desired plug-in for more information. Once installed, you will find a Houdini Engine menu in the host application. You can use this menu to load assets.



02 Once you have the plug-in installed, you can load the asset using the **Houdini Engine** menu. This will bring the *brickify* asset into the viewport while the asset parameters become available for manipulation.

You can also turn on animation for Maya or 3ds Max and play it the sequence using the timeline.



03 In Unreal, you can use the **Import** button to bring digital assets into your scene. You can also set **Shape** to **Custom Shape** and connect the asset to geometry within the host application and the brickification will be applied to that object.

WHAT HAPPENED TO THE BRICK COLORS?

You may notice that none of the brickified shapes are showing the brick colors within the various host applications. This is because the plug-ins don't always process information the same way as Houdini would. The point colors are still part of the asset but the host application is not receiving this information.

And while the animation works in Maya and 3ds Max, it would not make sense in Unity and Unreal Engine because Houdini assets can not become part of the runtime experience of a game therefore the built-in animation would be ignored. It is important to tailor your assets for the capabilities of the host application.